Collège Doctoral du Languedoc-Roussillon

THÈSE

Pour obtenir le grade de Docteur

Délivré par l'**Université Montpellier II**

Préparée au sein de l'école doctorale **I2S***
Et de l'unité de recherche **LIRMM, UMR 5506**

Spécialité: **Informatique**

Présentée par **Rémi Watrigant**

Approximation et complexité paramétrée de problèmes d'optimisation dans les graphes : partitions et sous-graphes

Soutenue le 02/10/2014 devant le jury composé de :

Frédéric Havet	DR	CNRS, INRIA Sophia-Antipolis	Rapporteur
Vangelis Th. Paschos	Prof.	LAMSADE, U. Paris-Dauphine	Rapporteur
Mathieu Liedloff	MdC	LIFO, U. d'Orléans	Examinateur
Christophe Paul	DR	CNRS, LIRMM, U. Montpellier II	Examinateur
Stéphan Thomassé	Prof.	École Normale Supérieure de Lyon	Examinateur
Vassilis Zissimopoulos	Prof.	Kapodistrian U. of Athens	Examinateur
Marin Bougeret	MdC	LIRMM, U. Montpellier II	Encadrant
Rodolphe Giroudeau	MdC	LIRMM, U. Montpellier II	Directeur

UNIVERSITÉ MONTPELLIER 2 SCIENCES ET TECHNIQUES

* **I2S :** École doctorale Information Structures Systèmes

Ce travail a été financé par l'ANR 2010 BLAN 021902.

Table des matières

Remerciements

Mes premiers remerciements concernent Frédéric Havet et Vangelis Paschos, pour avoir accepté d'être rapporteurs de cette thèse. Merci pour vos remarques et conseils qui ont permis d'améliorer sa compréhension. Merci également à Mathieu Liedloff, Christophe Paul, Stéphan Thomassé et Vassilis Zissimopoulos d'avoir accepté d'être membres de mon jury.

Merci à Rodolphe de m'avoir proposé un sujet, et fourni des conditions plus qu'idéales pour effectuer cette thèse. Merci également de m'avoir accompagné, en ayant trouvé l'équilibre entre une liberté d'investigation qui m'a permis de m'épanouir dans ce domaine (et dans le monde de la recherche en général), et les quelques coups de fouets nécéssaires qui m'ont aidé à être plus productif et rigoureux quand il le fallait.

Merci Marin de m'avoir fait partager ta connaissance dans toutes les étapes techniques qu'on a pu rencontrer. Sans ton expérience et ton abnégation, un grand nombre de résultats de cette thèse n'aurait pas pu voir le jour. Ce fut un honneur d'avoir été ton premier étudiant, et un plaisir d'avoir participé à mes premières conférences à tes côtés, d'avoir organisé tes missions, ainsi que d'avoir été humilié à chaque fois au squash et au mini ping-pong. Je vous souhaite beaucoup de bonheur avec Laïla et la petite Nora qui vient d'arriver.
D'un peu plus loin, merci à Jean-Claude, Marianne et Roland, pour les discussions, qu'elles furent de recherche ou non. Merci aussi aux permanents de l'équipe MAORE, en particulier Vincent, Sylvain, Miklos et Jérôme.

Merci également à tous les doctorants du Lirmm que j'ai pu cotoyer pendant ces années. En premier, mes co-bureaux : Danh et Massi, j'admire votre courage d'être loin de vos proches ; Guillerme, bon courage pour supporter le climat Montpellierain. N'oublie pas de t'occuper des missions de Marin. Ensuite, les doctorants de l'équipe AlGCo, en particulier Marthe, dont les recettes uniques et le fructivorisme n'ont d'égal que sa culture scientifique, et Nico, avec qui j'ai partagé la majorité de mes pauses (et également pas mal de questions de recherche). Bonne chance à Sandrine et toi pour la recherche des caribous. Merci aux membres du conseil des doctorants, plus particulièrement Flavien, Fabien, Manu, Nam, Romain, Florent.

Merci à tout le personnel administratif qui est là pour nous guider dans les

méandres des procédures auxquelles nous devons faire face. Mention spéciale à Nico, dont le dynamisme nous permet souvent de débloquer des situations compliquées.

Merci aux membres du Lamsade qui m'ont accueilli plusieurs fois : Cristina, Florian, Édouard, et bien sûr Vangelis, merci pour ta gentillesse et tes conseils, là aussi qu'ils furent à propos de recherche ou non.

Je souhaiterai terminer ces remerciements « recherche » par toutes les personnes qui ont participé à ma formation et mon expérience, à commencer par les enseignants en Mathématiques et en Informatique de l'Université de Nîmes, les membres du LGI2P, les enseignants en Informatique de l'UM2, et l'équipe AlGCo, en particulier Christophe (désolé encore de t'avoir aveuglé de mon ~~talent~~ coup droit au squash), Ignasi, Stéphane, Stéphan, Daniel et Alex. Merci également à Marc de m'avoir accompagné dans mon monitorat.

Je n'ai évidemment pas fait que de la recherche durant ces trois années, et je voudrais maintenant remercier toutes les personnes qui ont pu me faire dévier d'elle.

En premier lieu merci à ma famille. Mes parents tout d'abord, merci de m'avoir toujours supporté, et laissé libre dans mes choix d'orientation, et ce même en ne comprenant pas toujours les intitulés des matières que je choisissais. J'espère que ce manuscrit rendra plus concret mes tentatives d'explication de mes journées. Merci à mon frère sur qui je peux toujours compter pour des choses sérieuses et (surtout) moins sérieuses, je vous souhaite beaucoup de bonnes choses pour le futur avec Laetitia. Merci à ma grand-mère de m'avoir accueilli et copieusement nourri à chaque fois que je venais à Nîmes. Merci à ma belle-famille pour sa gentillesse, sa générosité et son accueil en toutes circonstances. Une mention spéciale au petit Maë.

Impossible d'oublier mes amis d'enfance et leurs pièces rapportées. Gui, Gilles, Nico, que de chemin parcouru gastronomiquement et œnologiquement parlant grâce à nos rendez-vous hebdomadaires ! Le verre est plein, il faut que je soutienne ! Nanou évidemment, sur qui on peut toujours compter pour partager les petits tracas mais surtout les meilleurs moments. Les irréductibles Nîmois Laurie et Arnaud, toujours partant pour faire la fête, et enfin Stely, avec qui c'est toujours un plaisir de se retrouver malgré la distance. Peu de gens peuvent se vanter d'avoir un tel noyau d'amis, merci énormément à tous les 7 pour tous ces moments passés ensemble, et merci d'avance pour les prochains qui suivront sans nul doute.

J'allais oublier Yoda, merci pour ton calme et ta sérénité à toute épreuve, et surtout pour ne jamais porter de jugement sur qui que ce soit, en toutes circonstances.

Enfin je termine cette longue liste par Élody, qui m'a accompagné au plus près pendant toute cette thèse, et plus généralement ces 8 dernières années. Te savoir à mes côtés m'a évidemment aidé à affronter les épreuves du quotidien, mais m'a surtout permis d'apprécier encore plus les bons moments. Cette fin de thèse n'est que le début d'autres péripéties, et je suis impatient de les découvrir avec toi. Enfin, merci d'avoir, corrigé, toutes les, virgules, mal placées ;-)

Introduction

Durant les dernières décennies, l'émergence et l'évolution de la technologie ont placé l'informatique au centre de la plupart des questions de recherche. L'idée fondamentale de ce domaine est la notion d'algorithme, une suite d'instructions élémentaires (calculs) permettant, étant donnée une liste de paramètres d'entrée donnés, de résoudre un problème. L'une des principales façons de comparer deux algorithmes effectuant la même tâche est de comparer leur complexité temporelle, définie comme étant le nombre d'instructions élémentaires nécessaires dans le pire des cas pour résoudre un problème, en fonction de la taille de l'entrée. À partir de là, il est naturel de se demander, pour un problème donné, quel sera le meilleur algorithme possible pouvant le résoudre. Pour la plupart des problèmes intéressants, il se trouve qu'il est toujours possible de concevoir un algorithme le résolvant. Cependant, la complexité souvent obtenue est une fonction exponentielle en la taille de l'entrée du problème. Autrement dit, le temps de calcul de l'algorithme augmente de manière exponentielle en fonction de la taille de l'entrée, et ces algorithmes ne permettront pas d'obtenir en pratique des solutions dans un temps raisonnable. Pour certains problèmes en revanche, il est possible de trouver un algorithme dit « efficace » : c'est à dire dont la complexité n'augmente que de manière polynomiale. On qualifiera alors un tel algorithme de polynomial. Une séparation semble alors pouvoir s'effectuer entre les problèmes dits « faciles », pour lesquels on peut trouver un algorithme polynomial, et les problèmes « difficiles », pour lesquels il n'en existe pas. L'une des problématiques majeures de l'informatique est alors de pouvoir classer les problèmes selon cette difficulté. Bien que cette question paraisse simple à première vue, elle reste à ce jour l'une des principales motivations de l'algorithmique théorique. Face à l'impossibilité apparente de pouvoir prouver qu'un problème ne pourra jamais être résolu par un algorithme efficace, la théorie s'est alors orientée vers la possibilité de montrer que certains problèmes étaient « au moins aussi difficiles » à résoudre que d'autres, en montrant que l'existence d'un algorithme efficace pour l'un impliquerait l'existence d'un algorithme efficace pour chacun des autres. Ceci a ainsi permis de regrouper un ensemble de problèmes plus difficiles que les autres : les problèmes $\mathcal{NP}$-difficiles.

Concernant les problèmes, on en distingue deux types :

- ceux qui, étant donnée leur entrée (on parlera d'instance), consistent à répondre simplement par « oui » ou « non » à une question. Ces problèmes sont appelés problèmes de décision,
- ceux qui, étant donnée leur instance d'entrée, demandent à retourner en sortie une solution de valeur optimale (minimum ou maximum). Ces problèmes sont appelés problèmes d'optimisation.

La plupart des problèmes considérés dans ce manuscrit appartiennent à la deuxième catégorie : les problèmes d'optimisation. La théorie de la $\mathcal{NP}$-difficulté, mentionnée plus haut, nous apprend ainsi qu'il est vain d'espérer, pour un grand nombre de problèmes, un algorithme à la fois efficace et optimal. À partir de ce constat, deux compromis peuvent être réalisés :

- rester efficace, mais perdre en optimalité,
- rester optimal, mais perdre en efficacité.

Cependant, étant donnée une instance d'entrée, il est souvent facile de trouver un algorithme efficace retournant une solution quelconque (non optimale), et il est également facile de trouver un algorithme (non efficace) retournant une solution optimale, en énumérant toutes les solutions possibles (qui sont souvent en nombre exponentiel, d'où le caractère non efficace de l'algorithme). Ainsi, ces deux approches se doivent de garantir une certaine performance, soit sur le côté de la qualité des solutions pour le premier, soit sur la qualité de la complexité de l'algorithme pour le second. Pour cela, deux cadres théoriques ont vu le jour permettant d'étudier les problèmes difficiles sous ces angles différents. Le premier porte le nom d'approximation polynomiale, et le second, apparu ensuite, de complexité exacte et/ou paramétrée.

Les travaux réalisés dans cette thèse sont de deux natures. Dans un premier temps, nous avons appliqué et étudié les différentes techniques connues pour établir des résultats positifs et négatifs pour des problèmes d'optimisation combinatoires précis, d'un point de vue de la résolution approchée ou paramétrée, dans le but de mieux comprendre leur complexité intrinsèque et de proposer des solutions pratiques. Nous avons principalement appliqué ces techniques sur deux problèmes d'optimisation dans les graphes, l'un appartenant à la famille des problèmes de partition de graphes : le problème Sparsest k-Compaction, et l'autre appartenant à la famille des problèmes de recherche de sous-graphes avec une contrainte de cardinalité : Sparsest k-Subgraph.

Dans un second temps, nous avons étudié les possibilités de combinaisons des techniques de la théorie de l'approximation et ceux de la complexité paramétrée, dans le but de proposer de nouveaux outils algorithmiques afin de traiter les problèmes difficiles. Ce concept a pris forme récemment sous le nom d'approximation

paramétrée. Nous proposons un état de l'art sur les différents résultats sur ce sujet, et étudions un cas particulier de cette combinaison, entre l'approximation et les algorithmes de noyaux (algorithmes issus de la complexité paramétrée).

Le plan de cette thèse est le suivant : dans le Chapitre 1, nous introduisons les notions de base ainsi que les notations générales nécessaires à la bonne compréhension du reste du document. Dans le Chapitre 2, nous définissons formellement les deux paradigmes utilisés que sont la théorie de l'approximation et la complexité paramétrée, et proposons un bref état de l'art sur les récents travaux concernant l'approximation polynomiale. Nous présentons également plus en détails un lien entre approximation et algorithmes de noyaux. Les contributions majeures de cette thèse sont ensuite abordées dans les deux chapitres suivants. Le Chapitre 3 sera consacré aux résultats algorithmiques sur les problèmes de compaction de graphes, en particulier sur le problème Sparsest k-Compaction. Le Chapitre 4 sera quant à lui dédié aux résultats sur les problèmes avec contrainte de cardinalité, et plus précisément sur Sparsest k-Subgraph. Chaque chapitre sera conclu indépendamment par des pistes de recherche et des questions ouvertes, et une conclusion générale terminera enfin le manuscrit.

Ces travaux ont donné lieu aux publications suivantes :

[114] *Approximating the Sparsest k-Subgraph in Chordal Graphs*, avec Marin Bougeret et Rodolphe Giroudeau. Theory of Computing Systems, à paraître. (version journal de [113]).

[116] *On the Sum-Max Graph Partitioning Problem*, avec Marin Bougeret, Rodolphe Giroudeau et Jean-Claude König. Elsevier Theoretical Computer Science, vol. 540-541, pp. 143-155, 2014. (version journal de [115]).

[21] *Parameterized Complexity of the Sparsest k-Subgraph Problem in Chordal Graphs*, avec Marin Bougeret, Nicolas Bousquet et Rodolphe Giroudeau. Proceedings of the 40th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2014). Springer LNCS 8327, pp. 150-161.

[113] *Approximating the Sparsest k-Subgraph in Chordal Graphs*, avec Marin Bougeret et Rodolphe Giroudeau. Proceedings of the 11th Workshop on Approximation and Online Algorithms (WAOA 2013). Springer LNCS 8447, pp. 73-84.

[115] *Sum-Max Graph Partitioning Problem*, avec Marin Bougeret, Rodolphe Giroudeau et Jean-Claude König. Proceedings of the 2nd International Symposium on Combinatorial Optimization (ISCO 2012). Springer LNCS 7422, pp. 297-308.

Chapitre 1

Préliminaires

Plan du Chapitre :

Ce chapitre pose les bases des notions abordées dans cette thèse. Nous commencerons par introduire la notion de graphe, et les notations communément utilisées autour de ceux-ci. Puis, nous aborderons l'idée centrale de ce manuscrit : les algorithmes, et plus généralement la complexité algorithmique des problèmes combinatoires. Nous finirons ce chapitre par une section sur la définition de quelques classes de graphes utilisées dans la suite du document ainsi que quelques unes de leurs propriétés. Nous nous concentrerons notamment sur une classe importante en théorie des graphes : la famille des graphes parfaits, et plus précisément sur quelques sous-classes. Des indications à l'attention du lecteur seront enfin données, afin de pouvoir poursuivre la lecture de cette thèse sans ambiguïté.

Pour la suite du document, nous supposerons que sont acquises les définitions et notions de base concernant les mathématiques discrètes, en particulier les notations ensemblistes ainsi que celles relatives à la théorie des nombres.

1.1 Graphes : origines et définitions

1.1.1 Origines

L'objet mathématique principalement utilisé dans ce document pour appliquer les principes algorithmiques qui seront présentés dans les prochaines sections sont les *graphes*. Ceux-ci ont été introduits dans le but de modéliser des problématiques de la vie courante, et trouvent leurs origines dans un article de 1741 du mathématicien suisse L. Euler [53], traitant du problème désormais classique des *sept ponts de Königsberg*.
La ville de Königsberg (maintenant Kaliningrad, en Russie) est traversée par le fleuve Pregolia, à l'intérieur duquel se trouvent deux îles reliées entre elles par un pont. Puis, plusieurs ponts relient les rives droite et gauche et les deux îles, comme décrit dans la Figure 1. L. Euler se demanda alors s'il était possible de faire un tour de la ville en passant une et une seule fois par chaque pont de celle-ci, sans avoir besoin de traverser le fleuve à la nage. Le mathématicien schématisa alors le problème en représentant chaque bout de terre de la ville (chacune des deux rives, et les deux îles) par un point (ou sommet), et en reliant deux points à l'aide d'une arête s'il existe un pont entre les deux endroits correspondants. Le but du problème est alors de trouver un chemin passant une et une seule fois par chaque arête. En fait, comme le remarqua L. Euler, l'existence d'un tel chemin (appelé *chemin eulérien* en hommage à ce problème) est impossible dans ce cas précis en raison du fait que certains sommets sont les extrémités d'un nombre impair d'arêtes (trois). Ces travaux sont considérés par beaucoup comme l'origine de la théorie des graphes.

L'utilité des graphes est leur aptitude à pouvoir modéliser un grand nombre de problématiques combinatoires appliquées. De manière générale, ils permettent de représenter des relations (binaires) entre les entités d'un système. Plus récemment

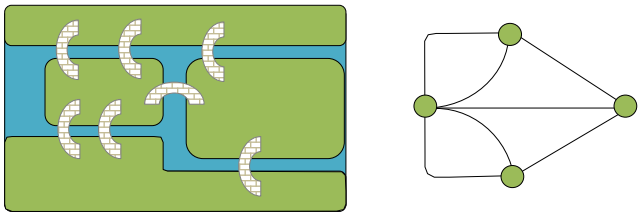


FIGURE 1 – Schéma du problème des sept ponts de Königsberg, et sa modélisation en graphe[1]. Source : [78]

par exemple, ils sont largement utilisés dans le cadre de l'étude de réseaux : réseaux d'ordinateurs, de personnes (réseaux sociaux)... etc. Nous verrons également dans le Chapitre 3 comment un problème issu du génie logiciel peut être résolu à l'aide d'une modélisation en graphes.

Plus formellement, la définition la plus simple d'un graphe utilise deux ensembles : un ensemble de *sommets*, souvent noté V (pour *vertices* en anglais), et un ensemble d'*arêtes*, souvent noté E (pour *edges* en anglais), composé de paires non ordonnées de sommets de V. On dit que ces graphes sont simples, à l'inverse des multigraphes qui peuvent comporter plusieurs arêtes entre deux paires d'arêtes, et non-orientés, à l'inverse des graphes orientés dont les arêtes sont des paires ordonnées. Par exemple, pour le graphe représenté par la Figure 2, son ensemble de sommets est $V = \{v_1, v_2, v_3, v_4\}$, et son ensemble d'arêtes est $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_3\}, \{v_3, v_4\}\}$. Il est possible de munir un graphe d'une fonction de poids sur ses sommets $w_V : V \longrightarrow \mathbb{N}$ et/ou ses arêtes $w_E : E \longrightarrow \mathbb{N}$. On parlera alors de graphe pondéré. Sauf mention du contraire, dans la suite du manuscrit un graphe désignera toujours implicitement un graphe simple non orienté et non pondéré. De plus, pour un graphe $G = (V, E)$, on notera toujours $n = |V|$ son nombre de sommets et $m = |E|$ son nombre d'arêtes.

1.1.2 Définitions

Définitions de base

Nous décrivons maintenant quelques notations couramment utilisées en théorie des graphes utiles pour la suite de ce manuscrit. Nous ne présentons ici qu'une liste non exhaustive des notions, et celle-ci ne saurait évidemment se substituer à des ouvrages de référence du domaine, tel que [47].

Soit $G = (V, E)$ un graphe. On dira que deux sommets u et v sont *voisins* ou *adjacents* si $\{u, v\} \in E$, et ils seront dits *indépendants* si $\{u, v\} \notin E$. On dira que

[1] En fait, le graphe représenté ici est un multigraphe, avec plusieurs arêtes entre certaines paires de sommets.

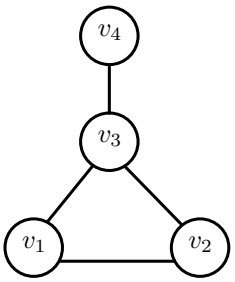


FIGURE 2 – Exemple de graphe à quatre sommets et quatre arêtes.

deux arêtes $\{u, v\}, \{x, y\} \in E$ sont adjacentes si $u = x$ ou $u = y$ ou $v = x$ ou $v = y$. Pour un sommet $v \in V$, nous notons $N_G(v) = \{x \in V : \{v, x\} \in E\}$ l'ensemble des voisins de v dans G, et $N_G[v] = N(v) \cup \{v\}$ son voisinage fermé. Nous notons $d_G(v) = |N_G(v)|$ le nombre de ses voisins, aussi appelé *degré* du sommet v. Le degré maximum de G est noté $\Delta_G = \max_{v \in V} d_G(v)$, et le degré minimum $\delta_G = \min_{v \in V} d_G(v)$. Pour toutes ces notations, il pourra nous arriver d'omettre l'indice G s'il n'y a aucune ambiguïté sur le graphe considéré.
On définit le graphe complémentaire de G, que l'on note $\bar{G}$ comme le graphe constitué des mêmes sommets V de G, et des arêtes $\{u, v\}$ tels que $\{u, v\} \notin E$. Autrement dit, $\bar{G}$ est consitué des non-arêtes de G et inversement.

Opérations sur les graphes

Il est possible d'effectuer des transformations sur un graphe afin d'en obtenir un autre. Pour la suite de cette partie, nous considérons un graphe $G = (V, E)$.

1. *suppression d'un sommet* : un sommet $u \in V$, le graphe ainsi obtenu est le graphe dont l'ensemble de sommets est $V \setminus \{u\}$, et dont l'ensemble d'arêtes est $E \setminus \{\{u, x\} : \{u, x\} \in E\}$.

2. *suppression d'une arête* : étant donnée une arête $e \in E$ de G, le graphe obtenu est simplement $G' = (V, E \setminus \{e\})$.

3. *contraction d'une paire de sommets* : soient $u, v \in V$. Cette modification consiste à supprimer u et v, à ajouter un sommet uv, et à relier uv avec $N(u) \cup N(v)$.

4. *contraction d'une arête* : même principe que précédemment, sauf que dans ce cas u et v sont tels que $\{u, v\} \in E$. Dans le cas où nous ne considérons que des graphes simples, la contraction d'une arête n'implique pas une boucle sur le sommet uv.

Ces transformations de graphes sont souvent utilisées dans le but de définir différentes notions d'inclusion de graphes. En particulier, soient G et H deux graphes. On dit que :

- H est un sous-graphe induit de G si H peut être obtenu à partir de G à l'aide de l'opération 1.
- H est un sous-graphe partiel de G si H peut être obtenu à partir de G à l'aide des opérations 1 et 2.
- H est un mineur de G si H peut être obtenu à partir de G à l'aide des opérations 1, 2 et 4.
- H est une compaction de G si H peut être obtenu à partir de G à l'aide des opérations 1 et 3. Cette dernière définition est moins commune, mais sera le cœur des problèmes étudiés au Chapitre 3.

Ces différentes définitions sont résumées dans la Figure 3.

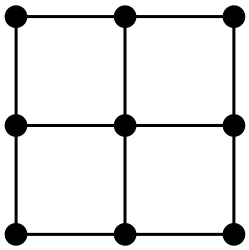
Étant donné un graphe $G = (V, E)$ et $S \subseteq V$, on notera $G[S]$ le sous-graphe de G induit par S, autrement dit $G[S] = (S, E \cap \{\{u, v\} : u, v \in S\}$. Par extension, pour $F \subseteq E$, on notera $G[F]$ le sous-graphe induit par F, autrement dit $G[F] = (V \cap \bigcup_{e \in F} e, F)$.

Structures et paramètres de graphes

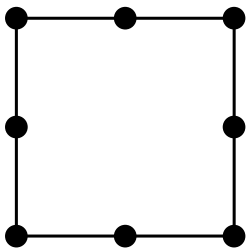
Soit $G = (V, E)$ un graphe tel que $E = \{\{u, v\} : u, v \in V\}$. Autrement dit, G comporte chacune des $\binom{|V|}{2}$ arêtes possibles[2]. On dit alors que G est une *clique*. Au contraire, si $E = \emptyset$, alors G est dit indépendant. Par extension, étant donné un graphe G, on dira qu'un ensemble $S \subseteq V$ est une clique (resp. un ensemble indépendant) de G si $G[C]$ induit une clique (resp. un indépendant). On remarquera facilement qu'une clique est le complémentaire d'un ensemble indépendant, et inversement.

On dira que $S \subseteq V$ est une clique *maximale* (resp. un ensemble indépendant *maximal*) si pour tout $v \notin S$, $S \cup \{v\}$ n'est pas une clique (resp. un ensemble indépendant). Cette notion est différente de celle de clique ou d'ensemble indépendant maximum. En effet une clique (resp. ensemble indépendant) maximum d'un graphe est un ensemble S tel qu'il n'existe pas de clique (resp. ensemble indépendant) de taille $|S| + 1$. Il est clair qu'une clique de taille maximum est également une clique maximale, mais l'inverse n'est pas vrai en général (idem pour les ensemble indépendant).

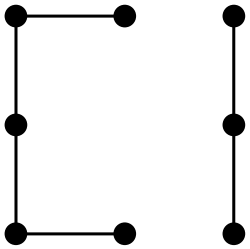
[2]Dans tout le manuscrit, nous préfèrerons utiliser la notation anglaise $\binom{n}{k}$ à la notation française C_n^k pour désigner le nombre de combinaisons de k parmi n.



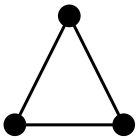
Graphe de départ G.



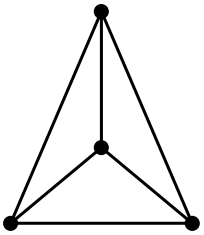
Sous-graphe induit de G.



Sous-graphe patiel de G.



Mineur de G.



Compaction de G.

FIGURE 3 – Différentes définitions de sous-graphes d'un graphe

On note respectivement $\omega(G)$ et $\alpha(G)$ (ou simplement ω et α s'il n'y a pas d'ambiguïté) la taille maximum d'une clique et d'un ensemble indépendant de G.

Étant donnée une partition $P = \{V_1, \cdots, V_{|P|}\}$ de V, (avec $V_i \neq \emptyset$ pour tout $i \in \{1, \cdots, |P|\}$), on dira que P est une $|P|$-coloration propre de G si pour tout $i \in \{1, \cdots, |P|\}$, V_i induit un ensemble indépendant. On note $\chi(G)$ le plus petit entier k tel que G admet une k-coloration propre.

Un chemin dans un graphe est une suite de sommets $C = (u_1, \cdots, u_k)$ tels que $u_i \neq u_j$ pour tout $i \neq j$ et $\{u_i, u_{i+1}\} \in E$ pour tout $i \in \{1, \cdots, k\}$. La taille d'un chemin est son nombre d'arêtes, et on dit que u_1 et u_k sont les extrémités de C. Un plus court chemin entre $u, v \in V$ est un chemin dont les extrémités sont u et v, et qui est de taille minimum parmi tous les chemins dont les extrémités sont u et v. L'excentricité maximum d'un sommet u est la plus grande longueur d'un plus court chemin entre u et un autre sommet $v \neq u$ du graphe. Le diamètre d'un graphe G,

noté $D(G)$ est le plus grand entier tel qu'il existe un sommet u d'excentricité $D(G)$.

Enfin, un cycle est un chemin dont les extrémités sont égales. Nous noterons respectivement P_k et C_k le chemin et le cycle à k sommets[3].

Un ensemble $S \subseteq V$ est dit connexe si pour tout $x, y \in S$, il existe un chemin dont les extrémités sont x et y et n'utilisant que des sommets de S. Par extension, un graphe est dit connexe si son ensemble de sommets est connexe, et il est dit déconnecté s'il n'est pas connexe.

1.2 Problèmes, algorithmes et complexité

1.2.1 Problèmes

Les notions de *problème* et d'*algorithme* sont deux idées centrales de ce manuscrit. D'une manière générale et informelle, un problème est une question dépendant d'un certain nombre de paramètres formels (ou variables). Une instance d'un problème consistera à donner des valeurs précises à ces variables, et on cherchera à répondre à la question munie de ces valeurs.

Un exemple de problème classique est le problème du Traveling Salesman Problem (voyageur de commerce), qui consiste en pratique à savoir quel est le tour le plus court possible pour visiter une et une seule fois un ensemble de villes, sachant que chaque paire de villes est reliée par une route d'une longueur donnée. Une instance de ce problème est illustrée Figure 4. De manière formelle, les variables de ce problème consistent en un ensemble de n villes $V = \{v_1, \cdots v_n\}$, avec, pour chaque paire (v_i, v_j), une distance $d(i, j)$ entre elles. La question est alors de trouver un ordre $(v_{\phi(1)}, \cdots, v_{\phi(n)})$ (avec ϕ une permutation de $\{1, \cdots, n\}$) tel que son coût, donné par la formule

$$\left(\sum_{i=1}^{n-1} d(\phi(i), \phi(i+1))\right) + d(\phi(n), \phi(1))$$

est minimum.

Formellement, les instances d'un problème sont en fait exprimées sous forme de chaînes construites à partir d'un alphabet Σ donné, permettant d'encoder les différentes variables du problème. On rappelle que pour un alphabet Σ, l'ensemble des chaînes finies pouvant se construire à partir de Σ est noté Σ^*. Pour plus de

[3]On remarquera que dans l'ouvrage de référence de R. Diestel [47], P_k est défini comme le chemin de longueur k, c'est à dire à $k+1$ sommets. Cependant P_k est plus communément défini comme le chemin à k sommets, et nous préfèrerons ainsi cette notation. Concernant les cycles, le problème ne se pose pas puisque son nombre de sommets est égal à son nombre d'arêtes.

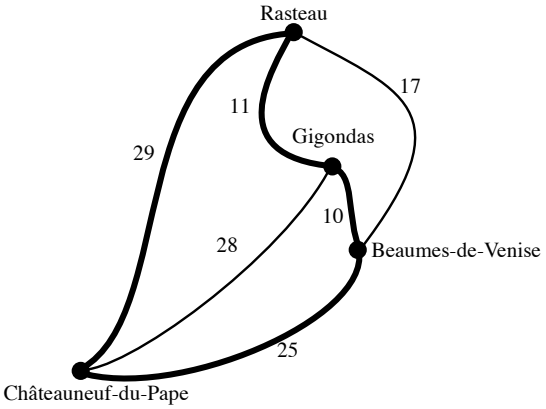


FIGURE 4 – Exemple d'instance du problème TRAVELING SALESMAN PROBLEM avec un ensemble de villes de la vallée du Rhône à parcourir, des distances entre chacune d'elles, et en gras une solution de coût 75, qui est le plus petit possible dans ce cas.

précisions sur les fonctions d'encodage, nous renvoyons le lecteur vers [61]. Ainsi, un problème Π sera défini sur un alphabet Σ par un triplet $(sol, cost, goal)$, où :

- pour tout $x \in \Sigma^*$, $sol(x) \subseteq \Sigma^*$ est l'ensemble des solutions pour x,
- pour tout $x \in \Sigma^*$ et $s \in sol(x)$, $cost(s) \in \mathbb{N}$ est la valeur de la solution s pour l'instance x,
- $goal \in \{\min, \max\}$.

Et, étant donnée une instance $x \in \Sigma^*$, on notera $opt_\Pi(x) = goal_{s \in sol(x)} cost(s)$ le coût d'une solution optimale, en omettant l'indice lorsqu'il n'y aura aucune ambiguïté. Pour le problème précédent TRAVELING SALESMAN PROBLEM, on aura par exemple :

- Une instance $x \in \Sigma^*$, une fois décodée, représentera l'ensemble des villes V ainsi qu'une distance $d(i, j)$ pour toute paire de villes $i, j \in \{1, \cdots, |V|\}$, $i \neq j$.
- L'ensemble $sol(x)$ sera l'ensemble des permutations de $\{1, \cdots, |V|\}$.
- sol sera le coût d'une solution, donné par la formule précédente.
- $goal$ sera min, puisqu'on cherche un tour de coût minimum.

Nous noterons tous les noms des problèmes en petites majuscules, et présenterons généralement un problème de la façon suivante :

Π
Entrée : $x \in \Sigma^*$
Sortie : $y \in sol(x)$
But : Optimiser $cost(y)$ (selon $goal$)

Pour le problème précédent par exemple :

Traveling Salesman Problem
Entrée : Un ensemble de villes $V = \{v_1, \cdots, v_n\}$, une distance $d(i,j)$ pour toute paire $i, j \in \{1, \cdots, n\}$, $i \neq j$
Sortie : Une permutation ϕ de $\{1, \cdots, n\}$
But : Minimiser $\left(\sum_{i=1}^{n-1} d(\phi(i), \phi(i+1))\right) + d(\phi(n), \phi(1))$

Un autre exemple de problème d'optimisation classique est le problème Clique :

Clique
Entrée : Un graphe $G = (V, E)$
Sortie : Un ensemble C de sommets deux à deux adjacents
But : Maximiser $|C|$

Alors que les problèmes précédents consistent à trouver une structure qui optimise une certaine fonction objectif, certains problèmes demandent simplement à répondre par « OUI » ou « NON » à une question. En fait, ces derniers s'appellent généralement *problèmes de décision*, alors que ceux vus précédemment s'appellent *problèmes d'optimisation*.
Ainsi, un problème de décision sera simplement défini sur un alphabet Σ par un sous-ensemble d'instances positives $L \subseteq \Sigma^*$. Puis, étant donnée une instance $x \in \Sigma^*$, le problème consistera à décider si cette dernière est une instance positive ($x \in L$) ou négative ($x \notin L$). On le présentera de la manière suivante :

L
Entrée : $x \in \Sigma^*$
Sortie : $x \in L$?

Comme nous allons le voir, la théorie de la complexité s'intéresse plus précisément aux problèmes de décision. La raison à cela est que ceux-ci ont une structure plus simple, tout en permettant d'exprimer au moins autant de problèmes que les problèmes d'optimisation. En effet il est possible, à partir d'un problème d'optimisation Π défini par $(sol, cost, goal)$, de définir le problème de décision Π^{dec} suivant :

Π^{dec}
Entrée : $x \in \Sigma^*, c \in \mathbb{N}$
Sortie : est-ce qu'il existe $y \in sol(x)$ tel que $cost(x) \leq c$?

Par exemple, pour le problème Clique précédent, on aura :

CLIQUE^{dec}
Entrée : Un graphe $G = (V, E)$, $c \in \mathbb{N}$
Sortie : Existe-t-il un ensemble C de sommets deux à deux adjacents tel que $|C| \geq c$?

Souvent, par abus de langage, il pourra nous arriver de confondre les notations CLIQUE et CLIQUE^{dec}.

On peut facilement voir que trouver une solution au problème d'optimisation Π pour une instance $x \in \Sigma^*$ permet de répondre au problème de décision Π^{dec} pour les instances (x, c), pour tout $c \in \mathbb{N}$. On dit que le problème de décision n'est pas plus difficile que le problème d'optimisation. Il est en fait également possible de montrer l'inverse pour la plupart des problèmes, *i.e.* que le problème d'optimisation n'est pas plus difficile que le problème de décision (nous y reviendrons à la Section 2.5).

1.2.2 Algorithmes

Un algorithme est une suite d'instructions permettant de résoudre un problème. Généralement, il prend en entrée des variables, effectue des opérations et des calculs (dictés par la suite d'instructions), et retourne en sortie un résultat. On dira qu'un algorithme résoud un problème s'il prend en entrée l'ensemble des variables d'entrée d'un problème, et retourne en sortie une solution à ce problème. Dans l'exemple précédent, un algorithme résolvant TRAVELING SALESMAN PROBLEM prendra donc en entrée un ensemble de villes ainsi qu'une distance pour chacune des paires de villes, et retournera en sortie un ordre sur ces villes, représentant le tour à effectuer. On dira qu'il le résoud exactement s'il retourne un ordre de coût minimum.

Pour le problème précédent, il est facile d'élaborer un algorithme trouvant une permutation engendrant un tour de coût minimum. En effet, il suffit pour cela d'énumérer toutes les permutations possibles, et de retourner celle dont le coût est minimum. Cet algorithme est présenté de façon formelle par l'Algorithme 1, écrit dans un langage théorique appelé « pseudo-code » que nous utiliserons tout au long de ce manuscrit. Le gros désavantage de l'algorithme présenté est le nombre d'instructions qu'il devra effectuer avant de trouver une solution optimale. En effet, le nombre de permutations d'un ensemble à n éléments étant égal à $n!$, cet algorithme devra tester un nombre de solutions gigantesque pour des valeurs de n de quelques dizaines. La finalité des algorithmes étant d'être exécutés sur des ordinateurs, ce dernier a peu de chance d'être utile en pratique.

Ainsi, l'efficacité d'un algorithme sera mesuré, dans cette thèse, par leur complexité temporelle[4]. Nous exprimerons cette dernière à l'aide d'une fonction exprimant le nombre d'instructions élémentaires nécessaires dans le pire des cas, en

[4]D'autres mesures existent, comme la complexité spatiale : la quantité d'information qu'un algorithme doit garder en mémoire pour effectuer ses calculs.

Algorithme 1 Algorithme brute-force pour le problème Traveling Salesman Problem.

Entrée :

- un ensemble de villes $V = \{v_1, \cdots, v_n\}$,
- une distance $d(i,j)$ pour tout $i, j \in \{1, \cdots, n\}$, $i \neq j$

Algorithme :
$meilleurOrdre \leftarrow (v_1, \cdots, v_n)$
$valMeilleurOrdre \leftarrow cost(v_1, \cdots, v_n)$
pour tout ordre $(v_{\phi(1)}, \cdots, v_{\phi(n)})$ **faire**
  si $(cost(v_{\phi(1)}, \cdots, v_{\phi(n)}) < valMeilleurOrdre)$ **alors**
    $meilleurOrdre \leftarrow (v_{\phi(1)}, \cdots, v_{\phi(n)})$
    $valMeilleurOrdre \leftarrow cost(v_{\phi(1)}, \cdots, v_{\phi(n)})$
  fin si
fin pour
retourner $meilleurOrdre$

fonction de la taille de l'instance. Pour le problème précédent par exemple, la taille d'une instance sera son nombre de villes n. De plus, on ne donnera généralement qu'une borne supérieure sur cette fonction. Par exemple, on dira que l'algorithme précédent (Algorithme 1) a une complexité de $\mathcal{O}(n!)$.

1.2.3 Complexité

Nous l'avons vu précédemment, écrire un algorithme retournant une solution optimale pour toutes les instances d'un problème est parfois simple : il suffit d'énumérer toutes les solutions possibles, et retourner la meilleure d'entre elles. C'est l'idée de la classe de problèmes $\mathcal{NP}$, regroupant la plupart des problèmes combinatoires qui vont nous intéresser dans ce manuscrit. La définition formelle, donnée ci-après, utilise la notion de *certificat polynomial* que nous ne détaillerons pas ici (voir [61] pour plus d'informations).

Définition 1. *La classe $\mathcal{NP}$ est l'ensemble des problèmes de décision $L \subseteq \Sigma^*$ pour lesquels il existe un algorithme polynomial $\mathcal{V}$ tel que pour tout $x \in \Sigma^*$, on a :*
$x \in L$ si et seulement s'il existe $y \in \Sigma^*$ de taille polynomiale en $|x|$ tel que $\mathcal{V}$ retourne « oui » sur l'entrée (x, y).
La chaîne y est alors appelé certificat polynomial.

Intuitivement, un certificat est un objet à fournir afin de vérifier, en temps polynomial, que l'instance est bien une instance positive. Pour le problème précédent Traveling Salesman Problem par exemple, un certificat sera une permutation de $\{1, \cdots, n\}$, puisque, étant donné une instance (du problème de décision) et une

	Taille de l'instance (n)				
Fonction	20	30	40	50	60
n	0,00002 sec.	0,00003 sec.	0,00004 sec.	0,00005 sec.	0,00006 sec.
n^2	0,0004 sec.	0,0009 sec.	0,0016 sec.	0,0025 sec.	0,0036 sec.
n^3	0,008 sec.	0,27 sec.	0,064 sec.	0,125 sec.	0,216 sec.
n^5	3,2 sec.	24,3 sec.	1,7 min.	5,2 min.	13 min.
2^n	1 sec.	17,9 min.	12,7 jours	35,7 jours	366 siècles
3^n	58 min.	6,5 années	3855 siècles	2×10^8 siècles	$1{,}3 \times 10^{13}$ siècles

TABLE 1 – Comparaison de temps d'exécution selon la complexité de l'algorithme, pour une machine effectuant un million d'opérations par seconde.

permutation des villes, il est facile de vérifier en temps polynomial si le tour correspondant a un coût inférieur à celui demandé. Ainsi, on dit souvent que $\mathcal{NP}$ regroupe la classe des problèmes « vérifiables » en temps polynomial.

Comme dit précédemment précédent, les algorithmes se contentant d'énumérer toutes les solutions d'une instance donnée auront, en général, une complexité exponentielle, les rendant inutiles en pratique. En effet, le Tableau 1 (issu de [61]), donne une idée du temps d'exécution des algorithmes selon l'ordre de grandeur de leur complexité temporelle. Partant de ce constat, il est naturel de chercher, pour un problème donné, des algorithmes avec des temps d'exécution les plus faibles possibles, le meilleur des cas étant lorsqu'il est possible d'obtenir un algorithme dont le temps d'exécution est un polynôme en la taille de l'instance d'entrée (on appellera ce dernier un *algorithme polynomial*. L'ensemble des problèmes pour lesquels il existe un tel algorithme est capturé par la classe $\mathcal{P}$.

Définition 2. *La classe $\mathcal{P}$ est l'ensemble des problèmes de décision $L \subseteq \Sigma^*$ tels qu'il existe un algorithme polynomial décidant pour tout $x \in \Sigma^*$ si $x \in L$.*

Alors que $\mathcal{NP}$ regroupe les problèmes « vérifiables » en temps polynomial, $\mathcal{P}$ regroupe ceux « résolubles » en temps polynomial. Ainsi, on a clairement $\mathcal{P} \subseteq \mathcal{NP}$. Afin de montrer qu'un problème appartient à $\mathcal{P}$, il suffit donc d'exhiber un algorithme polynomial. Pour montrer qu'un problème de $\mathcal{NP}$ n'appartient pas à $\mathcal{P}$ en revanche, les choses sont beaucoup moins évidentes. En effet, il n'a pas été possible jusqu'à présent de montrer que $\mathcal{NP} \setminus \mathcal{P} \neq \emptyset$, autrement dit, de montrer qu'un problème n'admet pas d'algorithme polynomial, malgré de forts soupçons pesant sur tout un ensemble de problèmes. La question de savoir si $\mathcal{P} \neq \mathcal{NP}$ est d'ailleurs un des problèmes ouverts majeurs de l'informatique théorique. Pourtant, il serait très important d'un point de vue pratique et théorique de pouvoir distinguer les problèmes dits « faciles », pour lesquels il existe un algorithme polynomial, des problèmes dits « difficiles », pour lesquels il n'en existe pas. Cette faiblesse de la théorie a été contournée par le biais de réductions, permettant de montrer que tout un ensemble de problèmes sont équivalents entre eux.

Définition 3. *Soient $L_1, L_2 \subseteq \Sigma^*$ deux problèmes de $\mathcal{NP}$. Une réduction polynomiale de Karp de L_1 vers L_2 est une fonction $f : \Sigma^* \longrightarrow \Sigma^*$ telle que :*

- *il existe un algorithme polynomial calculant $f(x)$ pour tout $x \in \Sigma^*$,*
- *pour tout $x \in \Sigma^*$, $x \in L_1$ si et seulement si $f(x) \in L_2$.*

On note dans ce cas $L_1 \propto L_2$.

L'idée de ces réductions vient du résultat suivant :

Théorème 1. *Soient $L_1, L_2 \subseteq \Sigma^*$ deux problèmes de $\mathcal{NP}$ tels que $L_1 \propto L_2$. Alors $L_2 \in \mathcal{P}$ implique $L_1 \in \mathcal{P}$.*

La preuve est immédiate, et consiste à appliquer la réduction suivie de l'algorithme polynomial pour L_2.

Comme dit précédemment, ces réductions permettent de montrer qu'il existe dans $\mathcal{NP}$ des problèmes plus difficiles que les autres :

Définition 4. *Un problème Π est dit $\mathcal{NP}$-difficile si pour tout $L \in \mathcal{NP}$, on a $L \propto \Pi$.*

Définition 5. *Un problème Π est dit $\mathcal{NP}$-complet si $\Pi \in \mathcal{NP}$ et si Π est $\mathcal{NP}$-difficile.*

Ainsi, les problèmes $\mathcal{NP}$-complets sont les problèmes de $\mathcal{NP}$ tels que s'il existait un algorithme polynomial pour résoudre l'un d'entre eux, alors on serait capable de résoudre tous les problèmes de $\mathcal{NP}$ en temps polynomial. Ce cas de figure est en fait peu probable d'arriver, puisqu'il s'avère que l'ensemble des problèmes $\mathcal{NP}$-complets capture un très grand nombre de problèmes combinatoires intéressants et variés. Le premier problème a être prouvé $\mathcal{NP}$-complet est le problème SAT.

SAT
Entrée : Un ensemble de variables U, et une collection C de clauses sur U
Sortie : Existe-t-il une affectation positive de U pour C ?

On appelle ce résultat le Théorème de Cook-Levin, du nom de leurs auteurs.

Théorème 2. *[40, 87]* SAT *est $\mathcal{NP}$-complet.*

Ainsi, l'ensemble des travaux de ce manuscrit se placent sous l'hypothèse que $\mathcal{P} \neq \mathcal{NP}$. Un schéma résumant la situation est disponible Figure 5

1.3 La famille des graphes parfaits

Cette section vise à détailler les définitions et propriétés de base de la classe des graphes parfaits et certaines de ses sous-classes, telles que les graphes chordaux ou les graphes d'intervalles.

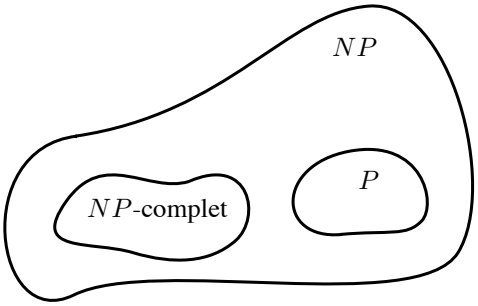


FIGURE 5 – Schéma simplifié de la situation, sous l'hypothèse $\mathcal{P} \neq \mathcal{NP}$.

1.3.1 Introduction, définition

Les graphes parfaits forment une famille importante en théorie des graphes. Introduits principalement par C. Berge dans les années 60 [4], ils sont définis comme étant les graphes pour lesquels le nombre chromatique est égal à la taille maximum d'une clique pour tous leurs sous-graphes. Un exemple simple de graphe qui n'appartient pas à cette famille est le cycle de longueur cinq : son nombre chromatique est de trois, alors que sa clique maximale est de taille deux. Ils ont été rendus célèbres en partie grâce à une conjecture concernant leur caractérisation émise par C. Berge dans les années 60, et démontrée en 2002 par M. Chudnovsky, N. Robertson, P. Seymour et R. Thomas [38]. Cette conjecture (devenue donc un théorème), stipule qu'un graphe est parfait si et seulement si ni lui ni son complémentaire ne contiennent de cycle impair de longueur au moins cinq. Une autre raison pour laquelle les graphes parfaits ont une place importante en théorie des graphes provient des travaux de V. Chvátal [39], permettant de déduire des algorithmes polynomiaux pour des problèmes tels que CHROMATIC NUMBER, CLIQUE ou INDEPENDENT SET (et donc VERTEX COVER) lorsque le graphe d'entrée est parfait, alors que ces problèmes sont $\mathcal{NP}$-difficiles dans le cas général. Enfin, une dernière raison (celle-ci va particulièrement nous intéresser) est qu'un grand nombre d'autres classes de graphes importantes (pour des raisons pratiques ou théoriques) sont des sous-classes des graphes parfaits. Parmi ceux-ci, citons les graphes chordaux, les graphes d'intervalles, les graphes bipartis, les split graphes, les graphes de comparabilité, les cographes, les graphes de permutation. Nous donnons maintenant les définitions formelles de certaines de ces classes, utiles pour la suite du manuscrit.

Définition 6. *Un graphe $G = (V, E)$ est un graphe biparti si $V = A \cup B$, où A et B sont chacun des ensembles indépendants.*

Une définition équivalente des graphes bipartis consiste à dire que ce sont exactement les graphes qui n'admettent pas C_k en tant que tout graphe induit, pour tout $k \geq 3$ impair. En interdisant maintenant tous les cycles de taille au moins quatre, nous obtenons la classe des graphes chordaux.

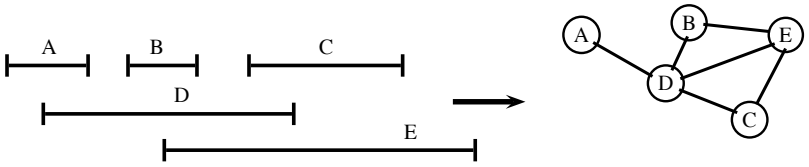


FIGURE 6 – Exemple de modèle d'intersections et le graphe d'intervalles associé.

Définition 7. *Un graphe est un graphe chordal s'il ne contient pas C_k en tant que sous-graphe induit, pour tout $k \geq 4$.*

Ainsi, on remarquera que l'intersection des graphes bipartis et des graphes chordaux est la classe des arbres[5], puisque cela revient à interdire tous les cycles induits de taille au moins trois.
Les graphes chordaux ont un rôle important en théorie des graphes, d'un point de vue pratique et théorique. En effet, ils permettent d'une part de modéliser beaucoup de problèmes combinatoires pratiques [67], et se caractérisent à partir d'outils algorithmiques très utiles, dont nous verrons une partie dans les sous-sections suivantes. Nous présentons maintenant deux sous-classes des graphes chordaux : les split graphes et les graphes d'intervalles. Un schéma des inclusions des classes de graphes sus-mentionnées est représenté par la Figure 7.

Définition 8. *Un graphe $G = (V, E)$ est un split graphe si $V = C \cup S$, où C induit une clique, et S induit un ensemble indépendant.*

Définition 9. *Un graphe $G = (V,E)$ est un graphe d'intervalles, avec $V = \{v_1, \cdots, v_n\}$ s'il existe un ensemble $\mathcal{I} = \{i_1, \cdots, i_n\}$ de n intervalles de $\mathbb{R}$ tels que pour tout $p, q \in \{1, \cdots, n\}$, on a $\{v_p, v_q\} \in E$ si et seulement si les intervalles i_p et i_q s'intersectent. Dans ce cas, on dit que $\mathcal{I}$ est un modèle d'intersections pour G.*
Un graphe d'intervalles G est un graphe d'intervalles propres si son modèle d'intersections $\mathcal{I}$ ne contient que des intervalles propres, autrement dit, pour tout $i, i' \in \mathcal{I}$, on a $i \not\subseteq i'$ et $i' \not\subseteq i$.

Un exemple de graphe d'intervalles et de son modèle d'intersections est représenté par la Figure 6.

Comme dit précédemment, les split graphes et les graphes d'intervalles sont des sous-classes des graphes chordaux. En effet, il est facile de voir qu'il est impossible de construire un cycle de longueur quatre ou plus à l'aide d'intervalles ou bien à l'aide d'une clique et d'un ensemble indépendant. Enfin, les graphes chordaux sont eux-mêmes des graphes parfaits. Ceci peut facilement se voir à l'aide du théorème des graphes parfaits mentionné précédemment. En effet, tout d'abord un graphe chordal G ne peut avoir de cycle impair de taille cinq ou plus, par définition. Puis,

[5] À ne pas confondre avec la classe des chordaux bipartis, définis comme les graphes bipartis n'ayant pas de cycle de taille 6 ou plus.

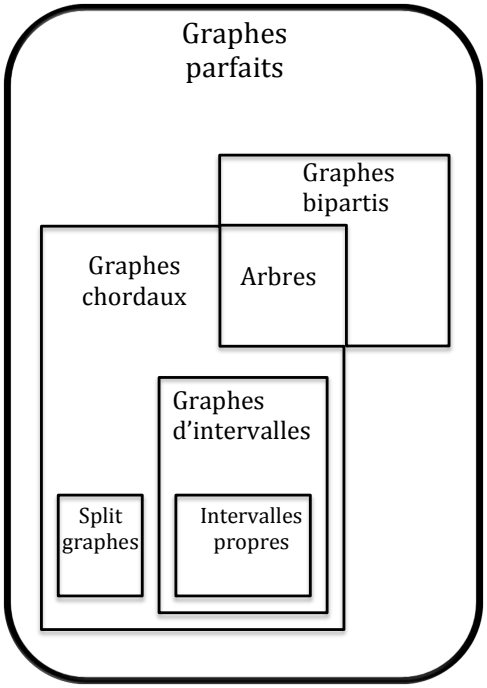


FIGURE 7 – Schéma des inclusions des classes de graphes présentées. Ce dernier est légèrement simplifié, puisque nous avons omis les intersections de classes lorsque celles-ci sont finies et/ou triviales (par exemple, le graphe P_2 à deux sommets est contenu dans toutes les classes).

on peut observer que son complémentaire $\bar{G}$ n'en admet pas non plus, car si tel était le cas, G contiendrait soit $\bar{C_5}$, soit $\bar{P_5}$, qui ne sont pas chordaux, une contradiction (il est facile de voir que la chordalité est héréditaire par sous-graphes induits). Dans la partie suivante, nous donnons deux caractérisations des graphes chordaux à l'aide d'outils algorithmiques puissants, qui nous seront utiles dans la suite du manuscrit.

1.3.2 Ordre d'élimination simplicial

Le premier outil utilisé est celui d'ordre d'élimination simplicial[6]. Un sommet d'un graphe est dit simplicial si son voisinage est une clique.

Définition 10. *Soit $G = (V, E)$ un graphe, avec $V = \{v_1, \cdots, v_n\}$ classés selon un ordre total. On dit que cet ordre est un ordre d'élimination simplicial de G si pour tout $i \in \{1, \cdots, (n-1)\}$, le sommet v_i est un sommet simplicial de $G[\{v_i, v_{i+1}, \cdots, v_n\}]$.*

[6]On trouvera également dans la littérature le terme d'*ordre d'élimination parfait* pour décrire un tel ordre.

Dans la suite, lorsqu'on manipulera un graphe possédant un ordre d'élimination simplicial, nous utiliserons la notation $x < y$ pour signifier que x apparaît avant y selon cet ordre. Il est facile d'observer qu'un graphe admettant un tel ordre est forcément un graphe chordal. En effet, si un graphe G admet à la fois un ordre d'élimination simplicial et un cycle C de longueur quatre ou plus, alors il suffit de prendre le premier sommet $c \in C$ dans l'ordre pour arriver à une contradiction, puisque par définition de l'ordre, $C \setminus \{c\}$ doit former une clique. D. J. Rose prouva en 1970 [104] la réciproque, caractérisant ainsi exactement les graphes chordaux en termes d'ordres d'élimination simpliciaux :

Théorème 3. *[104] Soit $G = (V, E)$. Il y a équivalence entre les deux assertions suivantes :*

- *G est un graphe chordal.*
- *G admet un ordre d'élimination simplicial.*

Cet outil est très utile puisqu'il permet très rapidement de montrer que des problèmes combinatoires $\mathcal{NP}$-difficiles dans le cas général deviennent polynomiaux dans le cas des graphes chordaux. Des exemples de tels problèmes sont INDEPENDENT SET et COUVERTURE PAR DES CLIQUES :

INDEPENDENT SET
Entrée : Un graphe $G = (V, E)$
Sortie : Un ensemble de sommets S indépendants
But : maximiser $|S|$

COUVERTURE PAR DES CLIQUES
Entrée : Un graphe $G = (V, E)$
Sortie : Une partition $\mathcal{C} = \{C_1, \cdots, C_q\}$ de V telle que $G[C_i]$ est une clique $\forall i \in \{1, \cdots, q\}$
But : minimiser $|\mathcal{C}|$

Il est à noter que le résultat de V. Chvátal [39] mentionné plus haut implique le théorème suivant, puisque les chordaux sont des graphes parfaits. Cependant, celui-ci est moins facile à obtenir pour les graphes parfaits directement. En fait, ces résultats ont d'abord été remarqués par F. Gavril en 1972 [62].

Théorème 4. INDEPENDENT SET *et* COUVERTURE PAR DES CLIQUES *sont polynomiaux dans les graphes chordaux.*

Preuve. Nous montrons ceci par le principe de relation min-max entre ces deux problèmes. En effet, il est facile de voir qu'une couverture par des cliques ne peut être de taille plus petite que celle d'un ensemble indépendant du graphe, puisqu'autrement deux sommets indépendants devraient se retrouver dans une clique, ce qui est

impossible.
Considérons alors l'algorithme suivant qui, étant donné un graphe $G = (V, E)$, où $V = \{v_1, \cdots, v_n\}$ est ordonné selon un ordre d'élimination simplicial, construit à la fois un ensemble indépendant S et une couverture par des cliques $\mathcal{C} = \{C_1, \cdots, C_k\}$ de manière gloutonne. L'algorithme commence par mettre v_1 dans S, définir $C_1 = N[v_1]$, retire $N[v_1]$ du graphe, et recommence : il prend le premier sommet v_i de l'ordre restant, le met dans S, construit une autre clique à l'aide de son voisinage fermé, et supprime cette clique du graphe, ceci jusqu'à ce que le graphe ne contienne plus de sommet. À la fin, il est facile de voir que S contient autant de sommets que ce que $\mathcal{C}$ contient de cliques. Ainsi, par la remarque précédente, il ne peut pas exister d'ensemble indépendant de taille strictement plus grande que $|S|$, ni de couverture par cliques de taille strictement plus petite que $|\mathcal{C}|$. □

1.3.3 Décomposition arborescente des graphes chordaux

Un second outil utile pour les graphes en général, et plus particulièrement pour les graphes chordaux, est la *décomposition arborescente*. Cet outil est l'une des briques (au même titre que la notion de mineur de graphes, présentée plus haut) de la théorie des mineurs de graphes de N. Robertson et P. Seymour, détaillée dans leur série de 23 articles *Graph Minors* publiés de 1983 jusqu'à 2012 pour le dernier[7].

Définition 11. *Soit $G = (V, E)$ un graphe. Une décomposition arborescente de G est un arbre $T = (\mathcal{X}, A)$ tel que :*

- *$\mathcal{X} \subseteq 2^V$ (les nœuds[8] de T sont des sous-ensembles de sommets de G),*
- *pour tout $x \in V$, l'ensemble des nœuds de T contenant x, noté $\mathcal{X}_x = \{X \in \mathcal{X} : x \in X\}$ est un arbre (connexe, non vide),*
- *pour tout $\{u, v\} \in E$, il existe $X \in \mathcal{X}$ tel que $u, v \in X$.*

La largeur de T est notée et définie par $w(T) = \max_{X \in \mathcal{X}} |X|$. La largeur arborescente (ou tree-width) de G est notée et définie par $tw(G) = \min\{w(T) : T$ est une décomposition arborescente de $G\} - 1$.

On remarquera ainsi facilement que les graphes de tree-width 0 sont exactement les ensembles indépendants, les graphes de tree-width 1 sont exactement les arbres. Concernant les graphes tree-width 2, il est possible de montrer que ce sont exactement les graphes dont les composantes 2-connexes sont des graphes série-parallèles (des graphes pouvant être construits à partir d'opérations simples, que nous ne détaillerons pas ici).

[7] En fait, *Graph Minors XXIII* fut publié en 2010, mais *Graph Minors XXII* fut publié en 2012

[8] Afin de lever toute ambiguïté, les sommets d'une décomposition arborescente seront plutôt appelés *nœuds*, le mot *sommet* étant réservé au graphe que celle-ci représente.

Un des intérêts des décompositions arborescentes est leur utilisation dans des algorithmes, plus précisément dans les algorithmes de programmation dynamique, qui reposent principalement sur le fait que chaque nœud de l'arbre est un séparateur du graphe (un ensemble $X \subseteq V$ d'un graphe est un séparateur si $G[V \setminus X]$ est déconnecté). Ainsi, lorsqu'on considère un nœud de l'arbre X, avoir la solution de chaque sous-graphe correspondant aux sous-arbres enracinés à chaque nœud fils permet, relativement facilement selon les problèmes, de construire la solution au sous-graphe correspondant au sous-arbre enraciné en X. Cette étape nécessite souvent de résoudre le problème sur les sommets de X, et donc entraine souvent une complexité de l'algorithme exponentielle en la taille des nœuds, et donc exponentielle en la tree-width du graphe. Ainsi, un très grand nombre de problèmes deviennent polynomiaux lorsque la tree-width du graphe d'entrée est constante. Un résultat classique dans ce sens est le Théorème de B. Courcelle stipulant que tous les problèmes exprimable en logique monadique du second ordre sont polynomiaux lorsque la tree-width du graphe d'entrée est constante [42] (ils sont même $\mathcal{FPT}$, une notion que nous verrons dans le chapitre suivant).

Malheureusement, la tree-width des graphes chordaux n'est en général pas bornée (en effet, pour tout $p \in \mathbb{N}$, la clique K_p de taille p est bien un graphe chordal, et celle-ci est de tree-width $p-1$). Cependant, les graphes chordaux possèdent des décompositions arborescentes très particulières, pouvant être exploitées algorithmiquement, comme nous le verrons par la suite. Cette particularité est décrite par [63] et [106] :

Théorème 5. *[63, 106] Soit $G = (V, E)$ un graphe chordal. Il existe une décomposition arborescente $T = (\mathcal{X}, A)$ de G où $\mathcal{X}$ est en bijection avec l'ensemble des cliques maximale de G. Autrement dit :*

- *pour tout $X \in \mathcal{X}$, X induit une clique maximale dans G*
- *pour tout clique maximale $X \subseteq V$ de G, on a $X \in \mathcal{X}$.*

Ce résultat illustre bien une autre définition équivalente des graphes chordaux : celle de *graphe d'intersections de sous-arbres d'un arbre*. Étant donné un arbre $T = (X, A)$, et un ensemble $\mathcal{X} = \{t_1, \cdots, t_n\}$ de sous-arbres (connexes) de T, le graphe d'intersection consiste à créer un sommet v_i par sous-arbre t_i, reliant deux sommets v_i et v_j si les sous-arbres correspondants s'intersectent. Il est facile de voir que l'arbre T ainsi que l'ensemble des sous-arbres constituent finalement une décomposition arborescente du graphe, où chaque nœud de T correspond à une clique du graphe. La Figure 8 illustre ces deux représentations.

Le Théorème 5 implique immédiatement le fait que pour un graphe chordal G, sa tree-width est égale à la taille de sa clique maximale moins un.

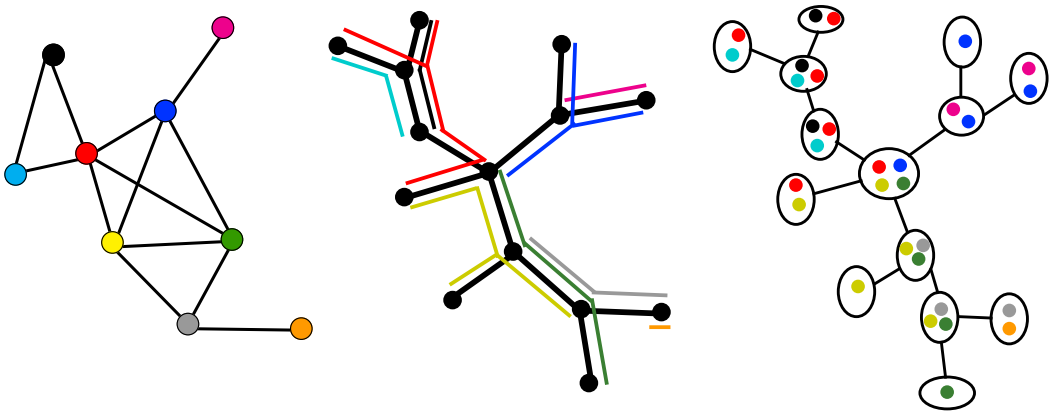


FIGURE 8 – Exemple de graphe chordal (à gauche), avec sa représentation en tant qu'intersection de sous-arbres d'un arbre (au milieu), et sa représentation équivalente en décomposition arborescente où chaque nœud de l'arbre est une clique du graphe (à droite).

De la même manière, étant donné la définition des graphes d'intervalles en tant que graphes d'intersection de sous-chemins d'un chemin, il est possible de représenter ceux-ci à l'aide d'une décomposition arborescente qui est en fait un chemin, avec une bijection entre les nœuds de la décomposition et les cliques maximales du graphe.

1.4 Notes de lecture

Nous finissons ce chapitre par quelques notes de lectures nécessaires à la bonne compréhension du manuscrit. Tout d'abord, même si la langue de ce document est le français, nous avons conservé un certain nombre de mots techniques en anglais. Dans le même esprit, nous avons également conservé tous les noms des problèmes en anglais, afin de permettre au lecteur de les retrouver dans la littérature. Enfin, certains chapitres traitant à la fois d'approximation et de complexité paramétrée, nous omettrons souvent de préciser si le problème traité est en fait sa version *décision* ou *optimisation*, le contexte permettant de le déduire dans tous les cas. Dans le même esprit, puisque les problèmes traités dans ce document appartiennent tous à la classe $\mathcal{NP}$, l'expression $\mathcal{NP}$-complet fera toujours référence au problème de décision associé au problème d'optimisation traité qui, lui, sera $\mathcal{NP}$-difficile.

Chapitre 2

Algorithmes Approchés et/ou Paramétrés

Plan du Chapitre :

Nous avons vu au chapitre précédent que pour un grand nombre de problèmes combinatoires (les problèmes $\mathcal{NP}$-difficiles), il est vain d'essayer de trouver un algorithme retournant une solution exacte en un temps polynomial. Partant de ce constat, ce chapitre présente deux alternatives classiques à la résolution de ces problèmes : les algorithmes approchés et les algorithmes exacts et paramétrés. Nous présentons brièvement l'idée derrière chacun de ces principes ainsi que les définitions qui y sont rattachées. Puis nous nous concentrerons sur une combinaison de ces deux paradigmes ayant pris forme ces dernières années : l'approximation paramétrée, en donnant notamment un court état de l'art sur les résultats connus sur le sujet. Dans une dernière section, nous proposons d'étudier cette combinaison de manière plus précise, entre l'approximation et un sous-domaine de la complexité paramétrée : les noyaux. Nous reprenons ainsi la définition de noyau fidèle déjà définie dans la littérature et proposons de l'étudier plus en détails sur des exemples concrets. Ces travaux sont toujours l'objet de recherches en cours, et plusieurs problèmes ouverts et pistes de recherche seront donnés en fin de chapitre.

2.1 Résolution de problèmes $\mathcal{NP}$-difficiles

Comme nous l'avons vu précédemment, la $\mathcal{NP}$-difficulté d'un problème d'optimisation combinatoire nous indique qu'il est inutile d'espérer (sauf si $\mathcal{P} = \mathcal{NP}$) de pouvoir le résoudre à la fois *efficacement* (en temps polynomial) et de manière *exacte* (en trouvant une solution optimale pour chaque instance). Partant de ce constat, et du fait qu'il est malgré tout nécessaire de résoudre ce problème en pratique, deux compromis peuvent être trouvés :

- trouver un algorithme efficace, qui ne donne pas tout le temps une solution optimale, mais qui essaie à chaque fois de trouver une solution *proche* d'une solution optimale.

- trouver un algorithme qui détermine à chaque fois une solution optimale au problème, possédant une meilleure complexité que celle de l'algorithme naïf qui essaie toutes les solutions possibles.

Chacun de ces compromis donna en fait naissance à deux paradigmes différents : l'*approximation polynomiale* et la *résolution exacte et/ou paramétrée*. Outre le fait de contourner la $\mathcal{NP}$-difficulté, en proposant malgré tout des solutions algorithmiques, ils ont surtout permis de confirmer que tous les problèmes, dits « difficiles » au sens de la complexité classique, n'étaient finalement pas complètement équivalents, en raffinant la complexité de ceux-ci selon certains points de vue. Ainsi, de nouvelles classes de complexité furent créées afin de capturer les différents comportements des problèmes, selon les approches avec lesquelles ils sont étudiés. Nous décrivons un peu plus en détail chacun de ces deux paradigmes dans les deux sections suivantes, en donnant les notions formelles de base. Bien que ces sous-domaines furent pendant

longtemps étudiés séparément, certains travaux récents ont commencé à les combiner, donnant naissance au principe d'*approximation paramétrée* que l'on décrit en Section 2.4. Enfin, en Section 2.5, nous décrivons l'une de ces combinaisons possibles que nous avons étudiée en particulier, entre les algorithmes de kernelization et l'approximation.

2.2 Algorithmes approchés

Cette section vise à décrire les concepts de base de la théorie de l'approximation. Pour plus de précision, nous redirigeons le lecteur vers les ouvrages classiques du domaine, tels que [77, 99, 108].
Tout d'abord, il est à noter que les algorithmes approchés traitent de problèmes d'optimisation. Nous rappelons qu'un problème d'optimisation Π est défini, à partir d'un alphabet Σ, par un triplet $(sol, cost, goal)$, où :

- pour tout $x \in \Sigma^*$, $sol(x) \subseteq \Sigma^*$ est l'ensemble des solutions réalisables pour x,
- pour tout $x \in \Sigma^*$ et $s \in sol(x)$, $cost(s) \in \mathbb{N}$ est la valeur de la solution s pour l'instance x,
- $goal \in \{\min, \max\}$.

Le problème étant alors formellement donné par :

Π
Entrée : $x \in \Sigma^*$
Sortie : $s \in sol(x)$
But : Optimiser (selon $goal$) $cost(s)$

Et on note $opt_\Pi(x) = goal_{s \in sol(x)} cost(s)$ la valeur d'une solution optimale pour x, en omettant l'indice Π lorsqu'il n'y a aucune ambiguïté.

2.2.1 Bornes supérieures : algorithmes approchés

Comme dit précédemment, si Π est $\mathcal{NP}$-difficile, il est inutile d'espérer trouver un algorithme $\mathcal{A}$ s'exécutant en temps polynomial et déterminant une solution exacte pour chaque instance. Le compromis des algorithmes approchés concerne alors l'exactitude de la solution. Plus précisément, ce paradigme s'intéresse à des algorithmes polynomiaux retournant une solution proche d'une solution optimale ayant une garantie de performance. La problématique est de définir ces notions de proximité et de garantie de performance. Comme nous allons le voir, dans la définition la plus commune[1], une solution sera dîte proche si le rapport entre sa valeur et la valeur

[1] D'autres notions de proximité peuvent être définies, comme par exemple la mesure différentielle, qui compare la valeur de la solution retournée par l'algorithme par rapport à la valeur optimale et par rapport à la pire valeur possible. Pour plus de détails, voir [99].

d'une solution optimale peut être majoré.

Définition 12. *Étant donné un problème de minimisation* $\Pi = (sol, cost, \min)$*, on dit que* $\mathcal{A}$ *est un algorithme* ρ*-approché, pour une certaine fonction* $\rho : \Sigma^* \longrightarrow [1, +\infty]$ *si* $\mathcal{A}$ *est un algorithme polynomial, et si pour tout* $x \in \Sigma^*$*, on a*

$$cost(\mathcal{A}(x)) \leq \rho(x) \cdot opt(x)$$

Pour un problème de maximisation, cette dernière inégalité sera remplacée par

$$opt(x) \leq \rho(x) \cdot cost(\mathcal{A}(x))$$

On remarque que dans cette définition générale, le rapport d'approximation ρ est donné comme une fonction dépendant de l'instance. Cependant, pour des raisons évidentes, les algorithmes ρ-approchés les plus intéressants seront lorsque ρ est une fonction constante $\rho(x) = c$ pour tout $x \in \Sigma^*$. Dans ce cas là, on dira simplement que l'algorithme est c-approché. L'ensemble des problèmes admettant un algorithme ρ-approché avec ρ une fonction constante est appelé $\mathcal{APX}$. Par extension, et lorsque ρ n'est plus une constante, on note ρ-$\mathcal{APX}$ l'ensemble des problèmes admettant un algorithme ρ-approché, comme par exemple la classe log-$\mathcal{APX}$.

Il est évident de remarquer que plus la fonction ρ est petite, meilleur l'algorithme sera. À l'extrême, il est parfois possible d'obtenir des algorithmes avec des rapports d'approximation aussi petits que possible.

Définition 13. *[99] Étant donné un problème d'optimisation* $\Pi = (sol, cost, goal)$*, on dit que* Π *admet une PTAS (pour Polynomial-Time Approximation Scheme) si pour tout* $\epsilon > 0$ *il existe un algorithme polynomial* $\mathcal{A}_\epsilon$ *qui est* $(1 + \epsilon)$*-approché.*

Ainsi, un temps d'exécution typique de $PTAS$ est $\mathcal{O}^*(|x|^{\frac{1}{\epsilon}})$, celui-ci devenant « mauvais » plus la précision demandée diminue.
Quelques fois, la dépendance en $\frac{1}{\epsilon}$ peut être supprimée de l'ordre du polynôme en $|x|$, pour se placer dans un terme multiplicatif. On obtient alors un temps d'exécution de $\mathcal{O}(f(\frac{1}{\epsilon})q(|x|))$ pour une certaine fonction f et polynôme q, et on parle alors d'$EPTAS$ (pour *Efficient Polynomial-Time Approximation Scheme*).
Enfin, si la fonction f ci-dessus est un polynôme, on parlera de $FPTAS$ (pour *Fully Polynomial-Time Approximation Scheme*).

2.2.2 Bornes inférieures : « problèmes gap » et réductions

Il est évident que pour un problème d'optimisation donné, on cherchera des algorithmes ayant le plus petit rapport d'approximation possible. Cependant, pour certains problèmes, il est possible de montrer qu'il est inutile d'espérer obtenir un algorithme polynomial ρ-approché, pour un certain rapport ρ. Pour cela, nous utiliserons finalement les mêmes principes que pour montrer la $\mathcal{NP}$-difficulté de problèmes

de décision, à savoir des réductions. Cependant, comme nous l'avons vu précédemment, les algorithmes d'approximation traitent de problèmes d'optimisation, et le passage classique au problème de décision associé ne traduit pas le caractère approché que l'on recherche. Pour pallier à ce problème, la notion de « problème gap » fut introduite. Étant donné un problème d'optimisation $\Pi = (sol, cost, goal)$, et une fonction $\rho : I \longrightarrow [1, +\infty]$, le problème gap associé, appelé GAP$_\rho$-Π est défini de la manière suivante :

GAP$_\rho$-Π
Entrée : $x \in \Sigma^*$, $C \in \mathbb{N}$
Sortie : « OUI » $opt(x) \leq C$
« NON » si $opt(x) > \rho(x) \cdot C$

Comme on peut le voir, la définition de ce problème n'impose pas vraiment de répondre « OUI » ou « NON » à une question posée, et rien n'est imposé non plus sur la sortie de l'algorithme s'il existe $y \in sol(x)$ tel que $C \leq cost(y) \leq \rho(x) \cdot C$. Cependant, on peut en fait transformer ce problème en un problème de décision classique en restreignant l'entrée aux instances $x \in \Sigma^*$ telles que $cost(x) \leq C$ ou $cost(x) > \rho(x) \cdot C$. L'utilité des problèmes gap est leur lien avec l'approximation, comme le montre le résultat suivant :

Théorème 6. *Soient Π un problème d'optimisation, et* GAP$_\rho$-Π *son problème gap associé, pour un certain rapport d'approximation ρ.*
S'il existe un algorithme polynomial $\mathcal{A}$ ρ-approché pour Π, alors GAP$_\rho$-Π *est polynomial.*

Preuve. Considérons l'algorithme qui, étant donné $x \in \Sigma^*$ et $C \in \mathbb{N}$, calcule $z = \mathcal{A}(x)$, et retourne :

- « OUI » si $cost(z) \leq \rho(x) \cdot C$,
- « NON » sinon

Il est clair que puisque $\mathcal{A}$ s'exécute en temps polynomial, cet algorithme également. De plus :

- si $\exists y \in sol(x)$ tel que $cost(y) \leq C$, alors on a $opt(x) \leq C$, et puisque par définition de $\mathcal{A}$, on a $cost(z) \leq \rho(x) \cdot opt(x)$, on a également $cost(z) \leq \rho(x) \cdot C$, et l'algorithme retourne « OUI »,
- si $\forall y \in sol(x)$ on a $cost(y) > \rho(x) \cdot C$, alors forcément $cost(z) > \rho(x) \cdot C$, et donc l'algorithme retourne « NON ».

Autrement dit, cet algorithme résoud GAP$_\rho$-Π en temps polynomial. □

Ces problèmes gap permettent ainsi de pouvoir prouver qu'il est inutile d'espérer un algorithme ρ-approché pour un certain problème Π, en montrant que le problème de décision associé GAP_ρ-Π est $\mathcal{NP}$-difficile. Pour cela, il suffira d'utiliser, comme dans le cas général, des réductions de Karp à partir d'autres problèmes de décision $\mathcal{NP}$-difficiles. Dans la littérature, les termes de « gap preserving reduction » ou « gap introducing reduction » peuvent se rencontrer. En fait, une gap introducing reduction sera simplement une réduction depuis un problème $\mathcal{NP}$-difficile classique vers un problème gap. Dans ce cas, la réduction fait « apparaître » le gap, d'où le nom. Une gap preserving reduction sera, en revanche, une réduction entre deux problèmes gap, le gap du premier problème servant généralement à obtenir celui du second problème, on observe donc une « préservation » de ce gap. Ces idées de réduction ont été pour la première fois introduites par [55], avant d'être combinées avec une caractérisation probabiliste de la classe $\mathcal{NP}$ [8] (le fameux *théorème PCP*), afin d'obtenir, en améliorant ce dernier, le premier résultat d'inapproximabilité pour le problème CLIQUE [7]

Théorème 7. *[7] Il existe $\epsilon > 0$ tel que $\text{GAP}_{n^{1-\epsilon}}$-CLIQUE est $\mathcal{NP}$-difficile (où n est le nombre de sommets du graphe d'entrée).*

Quelques années plus tard, J. Håstad renforça ce résultat en montrant que pour tout $\epsilon > 0$, $\text{GAP}_{n^{1-\epsilon}}$-CLIQUE ne peut admettre d'algorithme polynomial, sauf si $\mathcal{NP} = \mathcal{ZPP}$ [74]. Enfin D. Zuckerman [118] dé-randomisa ce dernier pour obtenir le même résultat sous l'hypothèse $\mathcal{P} \neq \mathcal{NP}$ (plus précisément : $\text{GAP}_{n^{1-\epsilon}}$-CLIQUE est $\mathcal{NP}$-difficile). Ainsi, il est vain d'espérer, pour le problème CLIQUE, un algorithme polynomial avec une garantie de performance sous-linéaire en n. Depuis ces avancées, un grand nombre de résultats d'inapproximabilité ont pu être obtenus pour différents problèmes d'optimisation, par le biais notamment des réductions à base de gap vues précédemment.

2.3 Algorithmes exacts et paramétrés

Contrairement aux algorithmes approchés présentés ci-dessus, les algorithmes paramétrés trouvent généralement une solution exacte au problème. Plus précisément, ils ne traitent que des problèmes de décision, répondant donc de manière binaire à une question. Bien évidemment, comme nous l'avons vu précédemment, il est possible de transformer un problème d'optimisation en une version de décision, et puisque répondre à ce problème permet de trouver la valeur de la solution optimale (et souvent de construire une solution optimale), ces algorithmes résolvent de manière exacte les problèmes d'optimisation. Ainsi, le compromis des algorithmes paramétrés repose sur leur temps d'exécution : si le problème traité est $\mathcal{NP}$-difficile, on ne peut qu'espérer (sauf si $\mathcal{P} = \mathcal{NP}$) un temps d'exécution exponentiel en la taille de l'instance. Cependant, il est malgré tout possible, en un temps exponentiel, d'élaborer des algorithmes non triviaux, et souvent bien meilleurs en pratique que l'algorithme

brute-force naïf essayant toutes les solutions possibles. Comme nous l'avons vu au chapitre précédent, les algorithmes sont souvent analysés en donnant une borne supérieure de leur temps d'exécution dans le pire des cas, en fonction de la taille de l'entrée, que nous appellerons n dans la suite de cette section. Ici aussi deux cas de figure se présentent lors de l'analyse d'un algorithme exponentiel non trivial :

- exprimer sa complexité toujours en fonction de la taille de l'instance, n, en espérant obtenir une complexité strictement meilleure que l'algorithme brute-force,
- exprimer sa complexité en fonction d'un ou plusieurs paramètres de l'instance en plus de n, en espérant que ceux-ci soient suffisamment petits (bien que théoriquement de l'ordre de n dans le pire des cas) pour que l'algorithme soit efficace.

Dans la littérature, les algorithmes analysés à l'aide de la première approche portent souvent le nom d'*algorithmes exacts*, alors que les second sont appelés *algorithmes paramétrés* (bien que dans les deux cas, les algorithmes doivent trouver des solutions exactes).

2.3.1 Algorithmes exacts

D'un point de vue théorique, la première approche ne nécessite pas vraiment de nouveaux concepts ou de nouvelles définitions. En effet, si l'algorithme brute-force d'un certain problème s'exécute en un temps $\mathcal{O}(p(n) \cdot 2^n)$ pour un certain polynome p (ce qui est le cas de beaucoup de problèmes, par exemple pour des problèmes de recherche d'un sous-graphe particulier dans un graphe, où l'algorithme naïf essaie tous les sous-ensembles de sommets possibles), alors on pourra chercher des algorithmes s'exécutant en un temps $\mathcal{O}(c^n)$ avec une constante $c < 2$, ou bien des algorithmes s'exécutant en un temps $c^{o(n)}$ pour une certaine constante $c \in \mathbb{R}$ (ces algorithmes étant appelés des algorithmes sous-exponentiels).
L'intérêt de ce type d'algorithmes est à la fois théorique et pratique : d'une part ils aident à une meilleure compréhension de la complexité intrinsèque des problèmes $\mathcal{NP}$-difficiles (en essayant de réduire d'écart entre le meilleur algorithme connu et une éventuelle borne inférieure), et permettent le développement de nouveaux concepts algorithmiques, pouvant être utilisés dans d'autres cas. D'autre part, d'un point de vue pratique, même si améliorer un temps d'exécution de $\mathcal{O}(2^n)$ à $\mathcal{O}(1{,}2^n)$ peut paraître dérisoire, puisque toujours exponentiel, cela permet en fait dans ce cas précis, étant donné un temps d'exécution fixé, de multiplier par plus de 7 la taille des instances pouvant être résolues en ce temps (à titre de comparaison, utiliser un ordinateur x fois plus puissant n'améliorera la taille de ces instances que par un terme additif $\log_2(x)$). Une autre raison est que pour des valeurs de n relativement petites, un « bon » algorithme exponentiel sera dans certains cas meilleurs qu'un

« mauvais » algorithme polynomial : par exemple, pour $n < 100$ on a $n^4 > 1{,}2^n$. Il est à noter que toutes ces comparaisons dépendent des constantes cachées par la notation $\mathcal{O}(.)$ qui ont été omises ici.

À propos des bornes inférieures que nous mentionnions précédemment, il est en effet possible de montrer que, sous certaines hypothèses, certains problèmes n'admettent pas d'algorithme sous-exponentiel.

2.3.2 Algorithmes paramétrés : définitions de base

Concernant les algorithmes paramétrés, en revanche, il y a une nécessité de poser quelques bases formelles, que nous décrivons maintenant. Comme pour les définitions de la théorie de l'approximation, celles-ci ne sauraient se substituer à un ouvrage de référence, tels que [59, 97, 51]. Plus précisément, nous utiliserons les notations de [59].

Problèmes paramétrés, classe $\mathcal{FPT}$

Comme dit précédemment, les algorithmes paramétrés traitent de problèmes de décision. Toutes les instances seront ainsi des mots sur un alphabet Σ, et un problème Π sera un sous-ensemble de Σ^*.

Définition 14. *[59] Une paramétrisation de Σ^* est une fonction calculable κ : $\Sigma^* \longrightarrow \mathbb{N}$.*

Définition 15. *[59] Un problème paramétré Π est un couple (I, κ), avec $I \subseteq \Sigma^*$ un problème et κ une paramétrisation de Σ^*.*

L'objectif d'un problème paramétré $\Pi = (I, \kappa)$ est le même que pour un problème classique : étant donnée une instance $x \in \Sigma^*$, il consiste à décider si $x \in I$.

Prenons par exemple le cas du problème Vertex Cover, dont nous rappelons la définition (en version de décision) :

Vertex Cover
Entrée : Un graphe $G = (V, E)$, un entier $k \in \mathbb{N}$
Sortie : Existe-t-il un ensemble de sommets S tel que $G[V \setminus S]$ soit sans arêtes, et tel que $|S| \leq k$?

Dans ce cas, une instance $x \in \Sigma^*$, une fois décodée, représentera un couple[2] (G, k), où G est un graphe et $k \in \mathbb{N}$, et où l'on demande si G contient un ensemble de k sommets adjacents à toutes les arêtes du graphe. Ici, où nous avons affaire à la version de décision d'un problème d'optimisation, plusieurs paramétrisations peuvent être définies :

[2]Dans ce cas, l'ensemble des instances Σ^* peut être identifié avec l'ensemble des couples (G, k), avec G un graphe et $k \in \mathbb{N}$.

- La paramétrisation dîte *classique*, qui représente la valeur de la solution recherchée : $\kappa(G,k) = k$. C'est la paramétrisation la plus naturelle, car le paramètre est déjà présent dans l'instance d'entrée.

- Des paramétrisations plus *structurelles*, dépendant du graphe, par exemple $\kappa(G,k) = \Delta(G)$, ou $\kappa(G) = tw(G)$. Ces paramétrisations permettent le plus souvent d'exploiter la structure de l'instance considérée, en obtenant des algorithmes efficaces lorsques les instances ont des petites valeurs de paramètres, autrement dit, lorsque l'instance a une structure peu complexe. Un type de paramétrisation structurelle est la « distance à une classe ». Dans ce cas, le paramètre mesure une distance entre l'instance considérée et un type d'instances souvent faciles à résoudre. Ainsi, c'est ici aussi une mesure de complexité de l'instance.

Comme annoncé dans l'introduction, le but sera d'exprimer la complexité des algorithmes en fonction du paramètre $\kappa(x)$ en plus de la taille de l'instance $|x|$. Le but de ceci est d'obtenir, pour des petites valeurs du paramètre, des algorithmes efficaces en pratique. Une première étape est d'obtenir un algorithme s'exécutant en temps polynomial lorsque la valeur du paramètre est une constante.

Définition 16. *[59] $\mathcal{XP}$ est la classe des problèmes paramétrés $\Pi = (I, \kappa)$ pour lesquels il existe deux fonctions $f, g : \mathbb{N} \longrightarrow \mathbb{N}$ telles qu'il existe un algorithme décidant Π en un temps $\mathcal{O}(f(\kappa(x))n^{g(\kappa(x))})$ pour tout $x \in \Sigma^*$.*

Par extension un tel algorithme est appelé *algorithme $\mathcal{XP}$*.
Par exemple, si l'on considère un problème de graphes où le but est de savoir s'il existe un sous-ensemble de sommets de taille k remplissant certains critères (des exemples de tels problèmes sont Clique, Independent Set ou Vertex Cover), il est facile de voir que l'algorithme naïf est un algorithme $\mathcal{XP}$ dans le cas où le paramètre est la taille de la solution k : en effet, en testant tous les sous-ensembles de sommets de taille k, il est possible de répondre à la question, tout ceci s'exécutant en un temps $\mathcal{O}(n^k)$, où n est la taille du graphe d'entrée.
D'un autre côté, il est aussi facile de voir que certains problèmes paramétrés n'admettent très probablement pas d'algorithme $\mathcal{XP}$: le problème Chromatic Number, qui prend en entrée un graphe G et un entier k, et demande si G admet une k-coloration propre, est déjà $\mathcal{NP}$-difficile pour $k = 3$. Ainsi, sauf si $\mathcal{P} = \mathcal{NP}$, celui-ci ne peut pas admettre d'algorithme $\mathcal{XP}$ s'il est paramétré par k.
Nous avons vu que pour les algorithmes $\mathcal{XP}$, le degré du polynôme pouvait dépendre du paramètre choisi. Dans certains cas, il est en fait possible d'obtenir un algorithme avec une complexité où la partie exponentielle est séparée de la taille de l'instance pour être restreinte au paramètre : c'est le principe de la classe $\mathcal{FPT}$ (pour *Fixed-Parameter Tractable*).

Définition 17. *[59] $\mathcal{FPT}$ est la classe des problèmes paramétrés $\Pi = (I, \kappa)$ pour lesquels il existe une fonction calculable $f : \mathbb{N} \longrightarrow \mathbb{N}$ et une constante c tels qu'il existe un algorithme décidant Π en un temps $\mathcal{O}(f(k) \cdot |x|^c)$ pour tout $x \in \Sigma^*$.*

Ici également un tel algorithme sera appelé *algorithme* $\mathcal{FPT}$. Il est clair, d'après ce qui précède, que $\mathcal{FPT} \subsetneq \mathcal{XP}$ (avec $\mathcal{FPT} \neq \mathcal{XP}$ sauf si $\mathcal{P} = \mathcal{NP}$).
La fonction f de la définition étant quelconque, des exemples de temps d'exécution peuvent être $\mathcal{O}(\kappa(x)! \cdot |x|^2)$ ou bien $\mathcal{O}(2^{2^{\kappa(x)}} \cdot |x|)$, ou bien même

$$\mathcal{O}(\overbrace{2^{\cdot^{\cdot^{\cdot^{2^{2^k}}}}}}^{g(k) \text{ fois}} \cdot |x|)$$

cette dernière étant finalement peu intéressante en pratique.

2.3.3 Difficulté et $\mathcal{W}$-hiérarchie

De la même manière qu'il est possible de montrer que certains problèmes de $\mathcal{NP}$ n'admettent très probablement pas d'algorithme polynomial, il est possible, en utilisant les mêmes principes de réduction, de montrer que certains problèmes n'admettent très probablement pas d'algorithme $\mathcal{FPT}$. Il faut pour cela adapter la définition de réduction aux problèmes paramétrés, en prenant soin d'ajouter une contrainte sur les paramètres.

Définition 18. *[59] Soient (A, κ_A) et (B, κ_B) deux problèmes paramétrés, sur les alphabets Σ_A et Σ_B respectivement. On dit que (A, κ_A) est $\mathcal{FPT}$-réductible vers (B, κ_B), et on note $(A, \kappa_A) \leq_{fpt} (B, \kappa_B)$ s'il existe un algorithme $\mathcal{R} : \Sigma_A^* \longrightarrow \Sigma_B^*$ tel que :*

- *Pour tout $x \in \Sigma_A^*$, on a $x \in A$ si et seulement si $\mathcal{R}(x) \in B$.*
- *Il existe une fonction calculable $f : \mathbb{N} \longrightarrow \mathbb{N}$ et une constante $c > 1$ tel que $\mathcal{R}$ s'exécute en temps $\mathcal{O}(f(\kappa_A(x)) \cdot |x|^c)$ pour tout $x \in \Sigma_A^*$.*
- *Il existe une fonction calculable $g : \mathbb{N} \longrightarrow \mathbb{N}$ tel que $\kappa_B(\mathcal{R}(x)) \leq g(\kappa_A(x))$ pour tout $x \in \Sigma_A^*$.*

Comme on le voit, la différence entre une réduction polynomiale et une $\mathcal{FPT}$-réduction est que cette dernière peut s'exécuter en temps $\mathcal{FPT}$, mais oblige en revanche à ce que le paramètre de l'instance de sortie soit une fonction du paramètre de l'instance de départ.

L'utilité de ces réductions est formalisé par le résultat suivant, dont la preuve repose simplement sur le fait que combiner une $\mathcal{FPT}$-réduction et un algorithme $\mathcal{FPT}$ donne immédiatement un algorithme $\mathcal{FPT}$:

Théorème 8. *Soient (A, κ_A) et (B, κ_B) deux problèmes paramétrés tels que $(A, \kappa_A) \leq_{fpt} (B, \kappa_B)$. Alors $(B, \kappa_B) \in \mathcal{FPT} \Rightarrow (A, \kappa_A) \in \mathcal{FPT}$.*

Alors que du côté de la complexité classique, comme nous avons pu le voir, tous les problèmes « difficiles » parmi la classe $\mathcal{NP}$ sont équivalents via des réductions polynomiales, ce n'est malheureusement pas aussi simple concernant la complexité paramétrée. La raison de ceci provient essentiellement de la contrainte sur le paramètre dans les $\mathcal{FPT}$-réductions, créant finalement plusieurs classes de complexité. Il est cependant plus ou moins possible d'obtenir des analogues aux classes $\mathcal{NP}$ et $\mathcal{NP}$-complet, en se ramenant à des problèmes de satisfaction de formules booléennes.

Pour cela, définissons les analogues de SAT, CNF-SAT et p-CNF-SAT en complexité paramétrée, où, en plus de leur formule respective, les problèmes prennent en entrée un entier k, et la question est de savoir s'il existe une affectation des variables de poids au plus k, où le poids d'une affectation est défini comme le nombre de variables à vrai. Ces variantes s'appellent respectivement WEIGHTED-SAT, WEIGHTED-CNF-SAT et WEIGHTED-p-CNF-SAT pour $p > 0$. Il est à noter que le problème WEIGHTED-2-CNF-SAT est $\mathcal{NP}$-complet, alors que 2-CNF-SAT est polynomial, comme le montre la réduction suivante :

Théorème 9. WEIGHTED-2-CNF-SAT *est* $\mathcal{NP}$*-complet.*

Preuve. Le problème est clairement dans $\mathcal{NP}$. Pour montrer qu'il est $\mathcal{NP}$-difficile, nous réduisons depuis le problème $\mathcal{NP}$-complet VERTEX COVER. Soient $G = (V, E)$ un graphe avec $V = \{v_1, \cdots, v_n\}$ et $k \in \mathbb{N}$. Construisons la formule 2-SAT suivante :

$$\bigwedge_{\{v_i, v_j\} \in E} (x_i \vee x_j)$$

On peut facilement vérifier que cette formule a une affectation de poids k si et seulement si G contient une solution de taille k (en prenant dans la solution les sommets correspondant aux variables à vrai), et cette réduction peut clairement se réaliser en temps polynomial. □

Une intuition possible pour se rendre compte des différences entre problèmes vis-à-vis de la complexité paramétrée est d'observer la réduction classique entre les problèmes CNF-SAT et 3-CNF-SAT adaptée aux versions paramétrées. On rappelle qu'alors que les clauses du premier sont de tailles arbitraires, celles du second sont de taille constante trois. La réduction polynomiale classique de WEIGHTED-CNF-SAT transforme chaque clause de m littéraux en $(m - 2)$ clauses de trois littéraux, en ajoutant $(m - 3)$ nouvelles variables. Par exemple, elle remplace la clause

$$(l_1 \vee l_2 \vee l_3 \vee l_4 \vee l_5 \vee l_6)$$

en ajoutant trois nouvelles variables z_1, z_2 et z_3, et en remplaçant la clause par :

$$(l_1 \vee l_2 \vee z_1) \wedge (\bar{z_1} \vee l_3 \vee z_2) \wedge (\bar{z_2} \vee l_4 \vee z_3) \wedge (\bar{z_3} \vee l_5 \vee l_6)$$

Si une affectation de poids k existe pour WEIGHTED-CNF-SAT, alors le poids de l'affectation correspondante dépendra forcément de la taille maximale d'une clause de la formule, ce qui implique que cette réduction n'est pas une $\mathcal{FPT}$-réduction.
Ceci montre que la complexité paramétrée des problèmes de satisfaction de formules booléennes dépendent fortement de la structure de celles-ci. À partir de là, plusieurs structures peuvent être définies, chacune donnant naissance à une classe de complexité. Les structures communément utilisées reposent sur des alternances de quantificateurs logiques dans les formules, et donnent naissance à une hiérarchie de classes d'équivalence en complexité paramétrée. Cette hiérarchie porte le nom de *$\mathcal{W}$-hiérarchie*, et ses classes se notent $\mathcal{W}[t]$ pour $t \geq 1$. La première, $\mathcal{W}[1]$, est définie à partir de WEIGHTED-2-CNF-SAT. On rappelle que la paramétrisation utilisée pour WEIGHTED-2-CNF-SAT (de même que pour ses autres variantes) est toujours le poids d'une affectation (nombre de variables placées à « vrai »).

Définition 19. *[51] Un problème paramétré (A, κ_A) est dans la classe $\mathcal{W}[1]$ s'il est $\mathcal{FPT}$-réductible à* WEIGHTED-2-CNF-SAT.

On remarque ainsi, en utilisant la réduction du Théorème 9 (qui est également une $\mathcal{FPT}$-réduction, puisque la valeur du paramètre dépend bien du paramètre de départ), que le problème VERTEX COVER est dans $\mathcal{W}[1]$. Plus généralement, tous les problèmes $\mathcal{FPT}$ (dont VERTEX COVER) sont dans $\mathcal{W}[1]$, car il suffit de décider le problème en temps $\mathcal{FPT}$ et de retourner une instance de WEIGHTED-2-CNF-SAT triviale. Il est également possible de montrer que CLIQUE, INDEPENDENT SET $\in \mathcal{W}[1]$.
Observons maintenant un autre problème classique : DOMINATING SET, qui, rappelons-le, prend en entrée un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$, et demande s'il existe un ensemble S de k sommets tel que pour tout $v \in V$, on a soit $v \in S$, soit $\{v, s\} \in E$ pour un certain $s \in S$. La traduction naturelle en formule booléenne est la suivante :

$$\bigwedge_{v \in V} \bigvee_{u \in N[v]} x_u$$

Ici aussi il est facile de voir que le graphe possède un ensemble dominant de taille k si et seulement si cette formule admet une affectation de poids k. Remarquons cependant que cette formule n'est pas une instance de WEIGHTED-2-CNF-SAT, car la taille des clauses dépendent de la taille des voisinages des sommets du graphe. Ainsi, cette formule est une instance de WEIGHTED-CNF-SAT, et c'est à partir de ce problème qu'est définie la classe $\mathcal{W}[2]$.

Définition 20. *[51] Un problème paramétré (A, κ_A) est dans la classe $\mathcal{W}[2]$ s'il est $\mathcal{FPT}$-réductible à* WEIGHTED-CNF-SAT.

Et on a immédiatement, par ce qui précède, que DOMINATING SET $\in \mathcal{W}[2]$. En fait, nous remarquerons que la plupart des problèmes intéressants se trouvent dans les classes $\mathcal{W}[1]$ et $\mathcal{W}[2]$.

Les classes $\mathcal{W}[t]$ pour $t > 2$ sont ensuite définies de manière similaire à partir de problèmes généralisant WEIGHTED-CNF-SAT et dont la structure dépend de t. On a vu qu'une formule de WEIGHTED-2-CNF-SAT était une « grande » conjonction de « petites » disjonctions de littéraux, alors qu'une formule de WEIGHTED-CNF-SAT était une « grande » conjonction de « grandes » disjonctions de littéraux[3]. Informellement, il est possible de généraliser ces problèmes en alternant les conjonctions et les disjonctions, et on appelera *weft* d'une telle formule le nombre maximum d'alternances entre des « grandes » conjonctions et des disjonctions. Ainsi, la classe $\mathcal{W}[t]$ regroupera tous les problèmes $\mathcal{FPT}$-réductibles au problème de satisfaction avec une formule de *weft* t. Ceci explique dans le même temps l'origine du nom de la $\mathcal{W}$-hiérarchie, le $\mathcal{W}$ provenant du mot *weft*. Pour plus de précisions, nous renvoyons le lecteur à [59].
Enfin, *presque* au sommet de la hiérarchie, la classe $\mathcal{W}[SAT]$ regroupe les problèmes $\mathcal{FPT}$-réductibles à WEIGHTED-SAT, ce dernier étant la version paramétrée classique du problème SAT (*c.f.* plus haut). Il est en effet *presque* au sommet puisque, pour k constant, une instance de WEIGHTED-SAT étant décidable en temps polynomial, ce dernier appartient à la classe $\mathcal{XP}$, ce qui nous donne finalement la suite d'inclusions suivantes :

$$\mathcal{FPT} \subseteq \mathcal{W}[1] \subseteq \mathcal{W}[2] \subseteq \cdots \subseteq \mathcal{W}[SAT] \subseteq \mathcal{XP}$$

Ces inclusions étant supposées strictes. De la même façon que pour la classe $\mathcal{NP}$, on dira qu'un problème est $\mathcal{W}[1]$-difficile s'il est $\mathcal{FPT}$ réductible à partir d'un problème de $\mathcal{W}[1]$, et qu'il est $\mathcal{W}[1]$-complet s'il appartient à $\mathcal{W}[1]$ et qu'il est $\mathcal{W}[1]$-difficile, ceci s'étendant à toutes les classes de la $\mathcal{W}$-hiérarchie mentionnées ci-dessus.

2.3.4 Noyaux : définitions et bornes inférieures

Introduction et définition

La complexité paramétrée, en plus de permettre l'analyse plus fine d'algorithmes exponentiels exacts, fournit un cadre théorique à l'analyse d'un autre type d'algorithmes : les algorithmes de pré-traitement. En effet, depuis de nombreuses années en pratique, les personnes souhaitant résoudre un problème efficacement ont souvent eu recours à des algorithmes permettant de diminuer la taille de l'instance, notamment en retirant de celle-ci des parties triviales. Par exemple, la plupart des solveurs SAT disposent de règles de pré-traitement permettant de « nettoyer » l'instance, avant de lui appliquer un algorithme coûteux par la suite. Ainsi, la combinaison de pré-traitements et d'un algorithme donne bien souvent une procédure plus performante. Cette intuition est en fait vérifiée en théorie également. En effet, nous allons voir que certains algorithmes de pré-traitement peuvent s'analyser élégamment via des paramètres de l'instance, formant ce que l'on appellera un *noyau*. De plus, nous verrons que l'une des façons de montrer qu'un problème est $\mathcal{FPT}$ est d'exhiber un

[3]Ici les mots « petit » ou « grand » signifient de taille constante ou non.

tel noyau. Informellement, un noyau est un algorithme polynomial permettant de transformer une instance en une autre instance équivalente, mais dont la taille est bornée par le paramètre de la première instance.

Définition 21. *[51] Soit (A, κ_A) un problème paramétré sur un alphabet Σ_A. Un noyau pour (A, κ_A) est un algorithme polynomial $\mathcal{N} : \Sigma_A^* \longrightarrow \Sigma_A^*$ tel que :*

- *Pour tout $x \in \Sigma_A^*$ on a $x \in A$ si et seulement si $\mathcal{N}(x) \in A$.*
- *Il existe une fonction calculable $f : \mathbb{N} \longrightarrow \mathbb{N}$ tel que $|\mathcal{N}(x)| \leq f(\kappa_A(x))$ pour tout $x \in \Sigma_A^*$.*

La fonction f est appelée taille du noyau.

Il est facile de voir qu'appliquer un algorithme exact exponentiel quelconque (par exemple, un algorithme brute-force) sur une instance obtenue après un noyau donne un algorithme $\mathcal{FPT}$, puisque cet algorithme sera exponentiel en la taille de l'instance réduite, qui ne dépend que du paramètre de l'instance de départ. Il y a en fait équivalence entre noyau et $\mathcal{FPT}$.

Théorème 10. *Soit (A, κ) un problème paramétré. Alors (A, κ) admet un noyau si et seulement s'il est $\mathcal{FPT}$[4].*

Preuve. Il ne nous reste qu'à montrer qu'un algorithme $\mathcal{FPT}$ pour (A, κ) implique un noyau pour celui-ci. Soit $\mathcal{A}$ l'algorithme $\mathcal{FPT}$ pour (A, κ), s'exécutant en temps $\mathcal{O}(f(\kappa(x)) \cdot |x|^c)$ pour tout $x \in \Sigma^*$, pour une certaine fonction f et une constante c. Soit $x \in \Sigma^*$. Si $|x| \leq f(\kappa(x))$, alors l'instance est déjà un noyau.
Autrement, on a $|x| > f(\kappa(x))$, et l'algorithme $\mathcal{A}$ s'exécute alors en temps $\mathcal{O}(f(\kappa(x)) \cdot |x|^c) = \mathcal{O}(|x|^{c+1})$, en temps polynomial donc. On peut donc l'exécuter, et retourner une instance triviale négative ou positive de taille constante selon la réponse de l'algorithme. □

Bornes inférieures de noyaux

On remarquera que le théorème précédent donne, pour tous les problèmes $\mathcal{FPT}$, un noyau de taille exponentielle, puisque dépendant de la partie exponentielle du temps d'exécution de l'algorithme. Il est cependant naturel de se poser la question de savoir si un problème $\mathcal{FPT}$ admet un noyau polynomial ou non. C'est le cas de plusieurs problèmes classiques, tels que Vertex Cover ou d-Hitting Set pour tout d fixé. Malheureusement, pour certains problèmes $\mathcal{FPT}$, il se trouve qu'un noyau de taille polynomial semble difficile à obtenir. Ces dernières années, des techniques ont permis de prouver, sous des hypothèses de complexité relativement fortes, que certains problèmes ne pouvaient pas admettre de noyau polynomial.

[4]On rappelle que tous les problèmes considérés dans ce manuscrit sont supposé décidables.

Ce cadre théorique, créé par H. Bodlaender, R. G. Downey, M. R. Fellows et D. Hermelin [14] et reposant sur un résultat de complexité de L. Fortnow et R. Santhanam [60], utilise les notions d'algorithme de distillation pour les problèmes de $\mathcal{NP}$, et celle de composition pour les problèmes paramétrés.

Définition 22. *[60] Une distillation de* SAT *vers un problème* $L \subseteq \Sigma^*$ *est un algorithme qui, étant donnée une suite d'instances* $(x_1, \cdots, x_t)$ *de* SAT*, retourne, en un temps polynomial en* $\sum_{i=1}^{t} |x_i|$*, une instance* $y \in \Sigma^*$ *telle que :*

- $y \in L \Leftrightarrow$ *il existe* $i \in \{1, \cdots, t\}$ *tel que* $x_i \in$ SAT*,*
- $|y| \leq p(\max_{i=1}^{t} |x_i|)$ *pour un certain polynôme* $p : \mathbb{N} \longrightarrow \mathbb{N}$*.*

Le résultat fondamental étant qu'il est très improbable qu'un tel algorithme existe, comme le montre le théorème suivant.

Théorème 11. *[60] S'il existe une distillation de* SAT *vers un problème* $L \subseteq \Sigma^*$*, alors* $\mathcal{NP} \subseteq co\mathcal{NP}/poly$*.*

Nous ne rentrerons pas dans les détails de l'hypothèse utilisée ici, mais mentionnerons simplement que la réfutation de celle-ci impliquerait l'effondrement de la hiérarchie polynomiale au troisième niveau : $PH = \Sigma_3^p$ [44], or il est supposé que toutes les classes de la hiérarchie polynomiale sont distinctes.

Les méthodes initiales permettant de montrer la non-existence de noyau polynomial [14] ont depuis été généralisées par H. Bodlaender, B. M. P. Jansen et S. Kratsch [17], aussi nous n'utiliserons que ces dernières. Nous présentons d'abord les ingrédients nécessaires avant de présenter l'outil principal qu'est la *cross-composition* (Définition 26) ainsi que le résultat principal de leur technique (Théorème 12). Il est à noter que bien que les deux définitions suivantes (Définitions 23 et 24) ne soient pas nécessaires à la compréhension du Théorème 12, elles seront ré-utilisées par la suite, en Section 2.5.

Définition 23. *[17] Étant donné un problème* $L \subseteq \Sigma^*$*, on définit le problème* $OR(L)$ *comme l'ensemble des suites d'instances* $(x_1, \cdots, x_t)$ *telles qu'il existe* $i \in \{1, \cdots, t\}$ *avec* $x_i \in L$*.*

Définition 24. *[17] Étant donné un problème paramétré* (A, κ) *avec* $A \subseteq \Sigma^*$*, on note* $\tilde{A}_\kappa$ *sa version non paramétrée, définie par* $\tilde{A}_\kappa = \{x\#1^{\kappa(x)}, x \in A\}$*, où* $\#$ *est un nouvel élément de l'alphabet* Σ*, et un est un élément arbitraire de* Σ*.*

Définition 25. *[17] Une relation d'équivalence* $\mathcal{R}$ *sur* Σ^* *est appelée relation d'équivalence polynomiale si les deux conditions suivantes sont respectées :*

- *Étant donnés* $x, y \in \Sigma^*$*, il existe un algorithme polynomial permettant de décider si* x *et* y *appartiennent à la même classe d'équivalence.*

- *Il existe une constante $c \geq 1$ telle que pour tout $S \subseteq \Sigma^*$, la relation $\mathcal{R}$ partitionne les éléments de S en au plus $\mathcal{O}((\max_{x \in S} |x|)^c)$ classes.*

Définition 26. *[17] Soit $L \subseteq \Sigma^*$ un problème, et (A, κ) un problème paramétré. On dit que L se cross-compose en (A, κ) s'il existe une relation d'équivalence polynomiale $\mathcal{R}$ et un algorithme qui, étant donnée une suite $(x_1, \cdots, x_t)$ de t éléments de Σ^* appartenant à la même classe d'équivalence de $\mathcal{R}$, construit une instance $x^* \in \Sigma^*$ en un temps polynomial en la la taille de la suite d'instances telle que :*

- $x^* \in A \Leftrightarrow x_i \in L$ *pour un certain* $i \in \{1, \cdots, t\}$.
- $\kappa(x^*)$ *est polynomial en* $\max_{i=1}^{t} |x_i| + \log_{|\Sigma|} t$.

L'idée clé derrière la notion de cross-composition est d'obtenir une instance paramétrée dont le paramètre ne dépend pas[5] de t. En effet, informellement, le résultat suivant montre qu'exhiber une cross-composition d'un problème $\mathcal{NP}$-difficile vers un problème paramétré (A, κ) permet de prouver que ce dernier ne peut pas admettre de noyau polynomial.

Théorème 12. *[17] Soit $L \subseteq \Sigma^*$ un problème $\mathcal{NP}$-difficile. Si L se cross-compose vers un problème paramétré (A, κ), et que (A, κ) admet un noyau polynomial, alors il existe une distillation de* SAT *vers $OR(\tilde{A}_\kappa)$, et $\mathcal{NP} \subseteq co\mathcal{NP}/poly$.*

Idée de la preuve. La preuve consiste à construire explicitement un algorithme de distillation de SAT vers $OR(\tilde{A}_\kappa)$ en utilisant :

- la réduction polynomiale de SAT vers L, puisque L est $\mathcal{NP}$-difficile,
- la relation d'équivalence polynomiale,
- la cross-composition de L vers (A, κ),
- le noyau polynomial de (A, κ).

Étant donnée une suite d'instances de SAT de taille maximale n, nous utilisons le fait que L est $\mathcal{NP}$-difficile pour transformer chacune d'entre elles en une instance de L. Puis, nous utilisons la relation d'équivalence polynomiale afin de regrouper les instances équivalentes entre elles. Pour un ensemble d'instances d'une même classe d'équivalence, nous appliquons successivement la cross-composition suivie du noyau polynomial, obtenant ainsi pour chacune des classes d'équivalence une instance de (A, κ) dont la taille est bornée par un polynôme en n (puisque la cross-composition donne une instance dont le paramètre est borné polynomialement en n, et que le noyau donne une instance dont la taille est bornée polynomialement par ce paramètre). Ceci constitue finalement une instance de $OR(\tilde{A}_\kappa)$ qui est positive si et

[5] On pourra remarquer que le terme $\log t$ dans la définition est superflu, puisque si $\log_{|\Sigma|} t > \max_{i=1}^{t} |x_i|$, alors on a $t > |\Sigma|^{\max_{i=1}^{t} |x_i|}$, et donc forcément deux instances de la suite sont identiques. On peut donc toujours se ramener au cas $\log_{|\Sigma|} t = \mathcal{O}(\max_{i=1}^{t} |x_i|)$.

seulement si l'une des instances de SAT de départ est positive, formant de cette manière un algorithme de distillation de SAT vers $OR(\tilde{A}_\kappa)$.

Il est à noter que la définition de cross-composition (Definition 26) est en fait plus précisément une OU-cross-composition, en ce sens que l'instance de sortie doit être positive *s'il existe* une instance d'entrée positive parmi la suite d'instance. Il est également possible de définir de manière analogue le principe de AND-cross-composition, où l'on demande que l'instance de sortie soit positive *si et seulement si* toutes les instances de la suite sont positives. De la même manière, ce principe s'applique également aux algorithmes de distillation, qui peuvent être soit des algorithmes de OU-distillation ou de AND-distillation. La seule différence pour la suite réside dans le Théorème 11 qui montre qu'il est peu probable qu'un algorithme de OU-distillation existe depuis SAT. L'existence d'un théorème équivalent pour les AND-distillation a été ouverte pendant quelque temps, et fermée récemment positivement par A. Drucker [52] (preuve simplifiée par H. Dell [43]). Depuis lors, il est possible d'utiliser les outils de composition (et notament de cross-composition) dans leur « version AND ».

Nous n'avons détaillé ici qu'une manière d'obtenir des bornes inférieures de noyaux : celle que nous avons utilisé dans les travaux présentés dans cette thèse. Pour connaîtres d'autres méthodes, nous renvoyons le lecteur vers l'état de l'art de N. Misra, V. Raman et S. Saurabh [95]

2.4 Approximation paramétrée

Dans cette section, nous présentons un bref état de l'art sur les combinaisons de techniques venues de la théorie de l'approximation et de la complexité paramétrée. Alors que ces deux sous-domaines furent longtemps étudiés séparément, de plus en plus de travaux sont apparus ces dernières années afin d'utiliser les avantages de chacun pour traiter les problèmes difficiles. Comme nous allons le voir, certains travaux ont également proposé la définition d'un cadre formel pour l'élaboration de telles méthodes.
L'un des premiers liens remarqué entre approximation et complexité paramétrée concernent les $EPTAS$. Plus précisément, il est facile de voir que si un problème admet une $EPTAS$, alors sa version de décision est $\mathcal{FPT}$, munie de sa paramétrisation standard. Ce résultat, remarqué indépendamment par [11] et [34], est surtout utile pour établir des bornes inférieure (concernant l'existence d'$EPTAS$). Dans le même esprit, [30] montre qu'un problème est $\mathcal{FPT}$ s'il appartient à la classe $\mathcal{MAX} - \mathcal{SNP}$ ou $\mathcal{MIN}\ \mathcal{F}^{+}\Pi_1$, deux classes capturant un grand nombre de problèmes approximables à facteur constant.

Puis, l'idée de mélanger algorithmes d'approximation et paramétrés peut être

une solution plutôt naturelle pour palier à certaines situations face à certains problèmes. Par exemple lorsqu'un rapport d'approximation semble ne pas être réalisable en temps polynomial, ou parce qu'une solution exacte semble difficile à obtenir en temps $\mathcal{FPT}$ (soit parce que cela contredirait une hypothèse, soit parce qu'aucun tel algorithme n'existe encore à ce jour). Un algorithme exponentiel (mais paramétré) peut alors permettre d'atteindre un bon rapport d'approximation. Pour la définition formelle des problèmes cités ci-dessous, nous renvoyons le lecteur à l'article correspondant.

L'un des premiers problèmes concerné fut TreeWidth, avec plusieurs algorithmes permettant, étant donné un graphe G et un entier k, soit de certifier que $tw(G) > k$, soit de construire une décomposition arborescente de largeur $c \cdot k$ pour une certaine constante c en temps $\mathcal{O}(f(k) \cdot n^{\mathcal{O}(1)})$ [102, 103]. Quelques temps après, un algorithme $\mathcal{FPT}$ exact fut proposé pour ce problème [15], mais malheureusement impossible à implémenter en pratique. Ainsi, d'autres algorithmes paramétrés d'approximation continuent d'être développés, proposant des temps d'exécutions plus intéressants tel que simple exponentiel [16].

Cette approche a également fait ses preuves sur d'autres problèmes, qu'ils soient munis de la paramétrisation standard ou de paramétrisations structurelles. La plupart des résultats suivants sont issus de l'état de l'art de D. Marx [91].

- Pour le problème Minimum Sum Edge Coloring, un algorithme paramétré par la tree-width du graphe d'entrée a été proposé [90], retournant des solutions aussi proches que possible d'une solution optimale, alors que le problème est $\mathcal{NP}$-difficile pour les graphes de tree-width deux déjà, et $\mathcal{APX}$-hard en général.

- Pour le problème Chromatic Number, un algorithme $\mathcal{FPT}$ 2-approché paramétré par le genre du graphe d'entrée est proposé [46], alors que celui-ci reste $\mathcal{NP}$-difficile dans les graphes planaires (les graphes planaires étant de genre 0). Paramétré par le nombre de sommets à retirer pour obtenir un graphe planaire, il est également possible de construire une solution $\frac{7}{3}$-approchée, alors que le problème est $\mathcal{NP}$-difficile dans les graphes planaires [91].

- Dans le même esprit, le problème Independent Set, restant $\mathcal{NP}$-difficile pour les graphes planaires, admet un schéma d'approximation lorsque paramétré par le genre du graphe d'entrée, et lorsque paramétré par le nombre de sommets à retirer pour obtenir un graphe planaire [69, 45] (qui généralise la $PTAS$ dans les graphes planaires [10]).

- Pour le problème Minimum k-Center, $\mathcal{W}[1]$-difficile dans le cas général, possède également un schéma d'approximation en temps $\mathcal{FPT}$ [3].

- Le problème METRIC TSP WITH DEADLINES, muni de sa paramétrisation standard, admet un algorithme $\mathcal{FPT}$ 2,5-approché, alors qu'il est $\mathcal{NP}$-difficile lorsque son paramètre vaut un, et qu'un algorithme approché polynomial avec rapport constant est impossible sauf si $\mathcal{P} = \mathcal{NP}$ [27].

- Dans [37], les auteurs proposent un algorithme paramétré avec un rapport d'approximation constant pour le problème STRONGLY CONNECTED STEINER SUBGRAPH, alors que celui ci est $\mathcal{W}[1]$-difficile dans le cas général, et n'admet pas de tel algorithme d'approximation polynomial.

- Dans [57], les auteurs étudient plusieurs paramétrisations pour le problème MINIMUM LINEAR ARRANGEMENT, en donnant notament un algorithme $(1+\epsilon)$-approché paramétré par la taille d'un vertex cover du graphe (la question de l'existence d'un algorithme exact pour ce paramètre étant toujours une question ouverte)

- Concernant le problème MAXIMUM k-COVERAGE, celui-ci admet un algorithme polynomial 1,5-approché, et est $\mathcal{W}[1]$-difficile paramétré par k. Toujours avec cette paramétrisation, il est possible d'obtenir un schéma d'approximation paramétré [91] (nous reviendrons plus en détails sur ce problème au Chapitre 4).

- Dans [23], les auteurs présentent un algorithme paramétré pour VERTEX COVER avec un rapport d'approximation de $\frac{2s+1}{s+1}$ pour tout $s \geq 1$, avec un temps d'exécution en $\mathcal{O}^*(c^{f(s)})$ pour une certaine constante c et fonction f. En particulier, pour $s = 1$, leur algorithme donne une solution 1,5 approchée en temps $\mathcal{O}^*(1{,}0883^k)$.

Malgré tous ces résultats positifs, aucun cadre théorique ne fut réellement établi pour unifier toutes les notions d'algorithmes paramétrés d'approximation, jusqu'en 2006, où, de manière surprenante, trois articles proposèrent indépendamment une solution [31, 49, 36]. Dans chacun de ces trois articles, les auteurs proposent des définitions assez similaires d'algorithme d'approximation paramétré et de schéma d'approximation paramétré.
En outre, ils abordent l'idée d'établir des bornes inférieures sur les rapports d'approximation qu'il est possible d'obtenir en un temps $\mathcal{FPT}$ donné. En particulier :

- Dans [31], les auteur montrent que si un problème admet un algorithme d'approximation paramétré, alors il admet forcément un algorithme d'approximation polynomial pour les instances où la valeur de la solution optimale est bornée par une fonction de la taille de l'entrée. En utilisant la contraposée, ils sont capables de montrer que DOMINATING SET ne peut admettre de r-approximation en temps $\mathcal{O}(2^k n^{\mathcal{O}(1)})$ (sous certaines hypothèses de complexité). De plus, ils montrent que tous les problèmes de la classe $\mathcal{MAX} - \mathcal{SNP}$, une classe de problèmes d'optimisation définissables à l'aide d'une formule simple

logique (qui admettent tous, entre autres, un algorithme d'approximation polynomial à facteur constant) admettent un schéma d'approximation paramétré, muni de leur paramétrisation standard.

- Dans [36], les auteurs montrent notamment que pour tout $t \geq 1$, la version pondérée du problème SAT correspondant à la classe $\mathcal{W}[t]$ (*c.f.* Section 2.3.3) n'admet pas d'algorithme $\mathcal{FPT}$ approché.

- Dans [49], les auteurs étudient entre autres l'approximation avec facteur additif $k + c$ (avec $c \in \mathbb{N}$ constant) des problèmes Independent Set et Clique, en montrant qu'il est impossible d'obtenir, sauf si $\mathcal{FPT} = \mathcal{W}[1]$, de tels algorithmes s'exécutant en temps $\mathcal{FPT}$ (avec la paramétrisation classique). Ils montrent également, sous l'hypothèse $\mathcal{FPT} \neq \mathcal{W}[2]$, que Dominating Set ne peut admettre de schéma d'approximation (avec facteur additif, de type $k + \epsilon$ pour tout $\epsilon > 0$) en temps $\mathcal{FPT}$. Enfin, ils montrent que le problème Independent Dominating Set ne peut admettre d'algorithme d'approximation paramétré quel que soit le rapport d'approximation (multiplicatif cette fois-ci), sous l'hypothèse $\mathcal{FPT} \neq \mathcal{W}[2]$ (ici également le paramètre est la paramétrisation classique).

Plus récemment enfin, des travaux se sont plus précisément concentré sur l'obtention de bornes inférieures, notamment en appliquant les techniques connues pour montrer la non-existence d'algorithme d'approximation en temps polynomial. Notamment, [92] adapte le théorème PCP afin de caractériser les classes de complexité paramétrée. Dans [37], les auteurs établissent qu'il est impossible d'obtenir, sous certaines hypothèses de complexité, des algorithmes paramétrés d'approximation pour Clique et Set Cover, pour certains rapports d'approximation et certains temps d'exécutions précis. Dans [19], une version plus faible du théorème PCP est supposée (ainsi que l'hypothèse ETH) afin de montrer la non-existence d'algorithme paramétré (pour la paramétrisation classique) avec un rapport d'approximation r, pour tout $r \geq 1$ pour Independent Set, et pour tout $r < 2$ pour Dominating Set.

Dans la section suivante, nous nous concentrons sur un exemple de lien entre approximation et complexité paramétrée, et plus précisément entre approximation et noyaux. C'est dans cette section que nous donnerons notamment la définition formelle d'algorithme approché paramétré.

2.5 Application : noyaux et approximation

Dans cette section, nous décrivons un exemple d'application de techniques venues de l'approximation et de la complexité paramétrée, plus précisément de l'approximation et de la kernelization. Ces résultats sont toujours l'objet de recherches en cours, en

collaboration avec E. Bonnet, M. Bougeret, R. Giroudeau, V. Paschos, C. Paul et F. Sikora.

2.5.1 Travaux existants

Le lien entre noyaux et approximation a été initié par [58], en définissant la notion de *fidelity kernels* et *fidelity preserving transformations*. Leur but est de proposer un compromis entre exactitude de la solution et rapidité de l'algorithme, en développant une série d'algorithmes paramétrés d'approximation.
Leur problématique est la suivante : soit Π un problème d'optimisation tel que le meilleur rapport d'approximation atteignable en temps polynomial soit ρ_0. Est-il possible de développer des algorithmes $\mathcal{A}_\rho$ fournissant une solution ρ-approchée, tels que $\mathcal{A}_{\rho_0}$ s'exécute en temps polynomial, et que $\mathcal{A}_1$ (retournant une solution exacte) ait un temps d'exécution au moins aussi bon que le meilleur algorithme paramétré exact ?
Ils développent pour cela des algorithmes qui transforment une instance d'un problème paramétrée en une instance dont la valeur du paramètre est plus petite, et qui préserve de manière *approchée* les solutions optimales. En le combinant ensuite avec des noyaux connus, ils obtiennent les algorithmes désirés. Les problèmes concernés sont Vertex Cover et sa généralisation en d-Hitting Set, Connected Vertex Cover et Steiner Tree. Pour Vertex Cover par exemple, ils obtiennent, pour tout $\alpha \in [1,2]$, un algorithme α-approché s'exécutant en temps $\mathcal{O}^*(1{,}273^{(2-\alpha)k})$.
Un des ingrédients utilisé est la notion de noyau fidèle[6], que nous allons étudier plus précisément dans les sous-sections suivantes.

2.5.2 Définitions et premières propriétés

Introduction

Nous posons maintenant les définitions formelles concernant les algorithmes d'approximation paramétrés. On rappelle qu'un problème d'optimisation Π défini sur un alphabet Σ est un triplet $(sol, cost, goal)$, où le but est de développer un algorithme prenant en entrée une instance $x \in \Sigma$, et retournant en sortie une solution $y \in sol(x)$ telle que $cost(y) = opt_\Pi(x)$. Il existe plusieurs possibilités concernant la définition d'algorithme d'approximation paramétré. La manière la plus naturelle serait de conserver la même définition que pour les algorithmes d'approximation polynomiaux, en exprimant la complexité en fonction d'un paramètre $\kappa(x)$ de l'instance. Cependant, cette façon de faire a un inconvénient, celui de ne pas pouvoir définir correctement la notion de paramètre standard. Ainsi, la définition maintenant

[6] « fidelity kernel » en anglais.

communément admise d'algorithme d'approximation paramétré est la suivante[7].

Définitions

Toutes les définitions suivantes utilisent des problèmes de minimisation. Cependant, des définitions similaires peuvent facilement s'obtenir pour des problèmes de maximisation.

Définition 27. *Soient* $\Pi = (sol, cost, goal)$ *un problème de minimisation sur un alphabet* Σ, $\kappa : \Sigma^* \times \mathbb{N} \longrightarrow \mathbb{N}$ *une paramétrisation de* $\Sigma^* \times \mathbb{N}$, *et* $\rho : \Sigma^* \longrightarrow [1, \infty]$. *Un algorithme* $\mathcal{A}$ *est un algorithme* $\mathcal{FPT}$ ρ*-approché pour* (Π, κ) *s'il prend en entrée un couple* $(x, c) \in \Sigma^* \times \mathbb{N}$, *et :*

- *retourne une solution quelconque si* $opt_\Pi(x) > c$,
- *retourne* $y \in sol(x)$ *tel que* $cost(y) \leq \rho(x) \cdot c$ *sinon,*
- *l'algorithme s'exécute en un temps* $\mathcal{O}(f(\kappa(x, c)) \cdot |x|^{\mathcal{O}(1)})$ *pour une certaine fonction calculable* $f : \mathbb{N} \longrightarrow \mathbb{N}$.

Nous définissons maintenant le problème de décision lié au problème Π, que nous noterons Π^{dec} (on rappelle que Π est un problème de minimisation) :

Π^{dec}
Entrée : $x \in \Sigma^*, c \in \mathbb{N}$
Sortie : Est-ce que $opt_\Pi(x) \leq c$?

Ainsi, le langage associé à Π^{dec} est un sous-ensemble de $\Sigma^* \times \mathbb{N}$, et la paramétrisation standard de Π^{dec} sera la fonction $(x, c) \mapsto c$. Il semblerait facile d'admettre que si un problème Π^{dec} muni d'une paramétrisation $\kappa : \Sigma^* \times \mathbb{N} \longrightarrow \mathbb{N}$ est $\mathcal{FPT}$, alors (Π, κ) admet un algorithme 1-approché paramétré. Cependant, comme le remarquent [32, 31], une subtilité réside dans le principe de constructibilité d'une solution. En effet, alors qu'un algorithme $\mathcal{FPT}$ ne fait que répondre à une question, un algorithme approché nécessite de construire une solution. Ce problème est souvent résolu en ayant recours à des algorithmes similaires aux d'algorithmes de « self-réductibilité » pour les problèmes d'optimisation de $\mathcal{NP}$, qui permettent de montrer qu'un algorithme polynomial pour la version de décision permet de résoudre polynomialement la version d'optimisation (pour plus d'information, voir [64]). On a alors la définition suivante.

[7] Dans la littérature, les problèmes d'optimisation ont souvent un quatrième paramètre correspondant à l'ensemble des instances. Cependant, afin de lier proprement algorithmes paramétrés et algorithmes d'approximation, nous avons choisis d'exprimer toutes les instances comme des éléments de Σ^*.

Définition 28. *Soient* Π *un problème de minimisation sur un alphabet* Σ*, et* $\kappa : \Sigma^* \times \mathbb{N} \longrightarrow \mathbb{N}$ *une paramétrisation de* $\Sigma^* \times \mathbb{N}$*. Supposons qu'il existe un algorithme prenant en entrée* $(x, c) \in \Sigma^* \times \mathbb{N}$ *et décidant si* $opt_\Pi(x) \leq c$ *en temps* $T(|x|, \kappa(x, c))$ *pour une certaine fonction calculable* T*.*
On dit que (Π, κ) *est solution-constructible si pour tout* $(x, c) \in \Sigma^*, \mathbb{N}$*, une solution* $y \in sol(x)$ *telle que* $cost(y) \leq c$ *peut être construite en temps polynomial en* $|x|$ *et en* $T(|x|, \kappa(x, c))$

Comme mentionné dans [32, 64], il se trouve en fait que la plupart des problèmes sont solution-constructibles.
La définition suivante décrit la notion de noyau fidèle pour un problème de décision lié à un problème de minimisation Π, introduite par [58].

Définition 29. *Soient* $\rho \in [0, 1]$*,* Π^{dec} *le problème de décision lié à un problème de minimisation* Π *défini sur un alphabet* Σ*, et* κ *une paramétrisation de* $\Sigma^* \times \mathbb{N}$*. Un noyau* ρ*-fidèle pour* (Π^{dec}, κ) *est une fonction* $\mathcal{F} : \Sigma^* \times \mathbb{N} \longrightarrow \Sigma^* \times \mathbb{N}$ *telle que pour tout* $(x, c) \in \Sigma^* \times \mathbb{N}$ *:*

- $\mathcal{F}(x, c)$ *se calcule en temps polynomial,*
- $|\mathcal{F}(x, c)| \leq g(\kappa(x, c))$ *pour une certaine fonction calculable* $g : \mathbb{N} \longrightarrow \mathbb{N}$*,*
- $\kappa(\mathcal{F}(x, c)) \leq \kappa(x, c)$*,*
- $(x, c) \in \Pi^{dec} \Rightarrow \mathcal{F}(x, c) \in \Pi^{dec}$*,*
- $\mathcal{F}(x, c) \in \Pi^{dec} \Rightarrow (x, \rho(x) \cdot c) \in \Pi^{dec}$*.*

Propriétés

Pour résumer, la différence entre un noyau classique et un noyau ρ-fidèle réside dans l'équivalence des solutions, qui n'est plus vérifiée dans ce dernier. Un schéma d'un noyau fidèle est représenté Figure 9 (gauche). Un noyau ρ-fidèle permet toutefois de récupérer une solution ρ-approchée pour l'instance de départ si l'on résoud de manière exacte l'instance réduite. En fait, on peut établir, modulo le fait que le problème soit solution-constructible, la même équivalence entre noyau ρ-fidèle et algorithme approché $\mathcal{FPT}$ qu'entre noyau classique et algorithme $\mathcal{FPT}$:

Théorème 13. *Soient* Π^{dec} *un problème de décision lié à un problème de minimisation* Π*,* $\kappa : \Sigma^* \times \mathbb{N} \longrightarrow \mathbb{N}$ *une paramétrisation, et supposons que* (Π, κ) *soit solution-constructible et décidable. Alors* (Π^{dec}, κ) *admet un noyau* ρ*-fidèle si et seulement si* Π *admet un algorithme* ρ*-approché* $\mathcal{FPT}$*.*

Preuve.
Supposons que (Π^{dec}, κ) admette un noyau ρ-fidèle $\mathcal{F}$, et considérons $(x, c) \in \Sigma^* \times \mathbb{N}$. En temps polynomial, nous utilisons le noyau $\mathcal{F}$ pour obtenir une autre instance paramétrée (x', c'). Enfin, en temps exponentiel en $|x'| + c'$, nous décidons si $(x', c') \in \Pi^{dec}$ ou $(x', c') \notin \Pi^{dec}$, et :

- si $(x', c') \in \Pi^{dec}$ alors cela implique $(x, \rho(x) \cdot c) \in \Pi^{dec}$, autrement dit $opt(x) \leq \rho(x) \cdot c$. Dans ce cas nous utilisons le fait que le problème soit solution-constructible pour construire une solution $y \in sol(x)$ telle que $cost(y) \leq \rho(x) \cdot c$.

- si $(x', c') \notin \Pi^{dec}$, alors $(x, c) \notin \Pi^{dec}$, ce qui équivaut à $opt(x) > c$. Ainsi dans ce cas nous retournons « NON ».

Ainsi nous avons bien une solution ρ-approchée, et puisque le noyau retourne une instance (x', c') de taille bornée par une fonction de $\kappa(x, c)$, celui-ci s'exécute bien en temps $\mathcal{FPT}$.
Supposons maintenant que Π admette un algorithme ρ-approché $\mathcal{A}$ pour (Π, κ), s'exécutant en un temps $\mathcal{O}(f(\kappa(x, c)) \cdot |x|^{\mathcal{O}(1)})$ pour toute entrée $(x, c) \in \Sigma^* \times \mathbb{N}$. Soit $(x, c) \in \Sigma^* \times \mathbb{N}$. Si $|x| \leq f(\kappa(x, c))$, alors l'instance est déjà un noyau. Autrement, l'algorithme $\mathcal{A}$ s'exécute en temps $\mathcal{O}(f(\kappa(x, c)) \cdot |x|^{\mathcal{O}(1)}) = \mathcal{O}(|x|^{\mathcal{O}(1)})$ polynomial. Nous retournons une instance triviale positive dans le cas où celui-ci retourne une solution $y \in sol(x)$ telle que $cost(y) \leq \rho(x) \cdot c$, et nous retournons « NON » sinon.
Il est facile de remarquer que dans tous les cas, la taille de l'instance retournée est bornée par $f(\kappa(x, c))$, l'algorithme s'exécute en temps polynomial, et :

- si $(x, c) \in \Pi^{dec}$, alors il existe une solution $y \in sol(x)$ telle que $cost(y) \leq c$, ce qui implique $opt(x) \leq c$, et par définition de $\mathcal{A}$ celui-ci retourne une solution $y \in sol(x)$ telle que $cost(y) \leq \rho(x) \cdot c$ et nous retournons bien une instance positive.

- si nous retournons une instance positive, alors cela signifie que $\mathcal{A}$ a retourné une solution $y \in sol(x)$ telle que $cost(y) \leq \rho(x) \cdot c$, autrement dit $(x, \rho(x) \cdot c) \in \Pi^{dec}$, ce qui termine la preuve.

□

Noyaux fidèles et noyaux pour problème gap

Lorsque l'idée d'associer noyaux et approximation apparaît, la manière la plus naturelle de la formaliser serait simplement d'appliquer la définition de noyau aux problèmes gap. Un tel noyau prendrait ainsi (dans le cas d'un problème de minimisation) une instance (x, C) d'un problème gap, retournerait en temps polynomial une instance (x', C') de taille bornée par une fonction du paramètre de l'instance de départ, et telle que :

- si $opt(x) \leq C$, alors $opt(x') \leq C'$,

- si $opt(x) > \rho(x) \cdot C$, alors $opt(x') > \rho'(x) \cdot C'$.

Pour deux certains rapports d'approximation $\rho, \rho' : \Sigma^* \longrightarrow [1, \infty]$.
Cependant, on peut voir que ces deux notions sont différentes, et même qu'un noyau

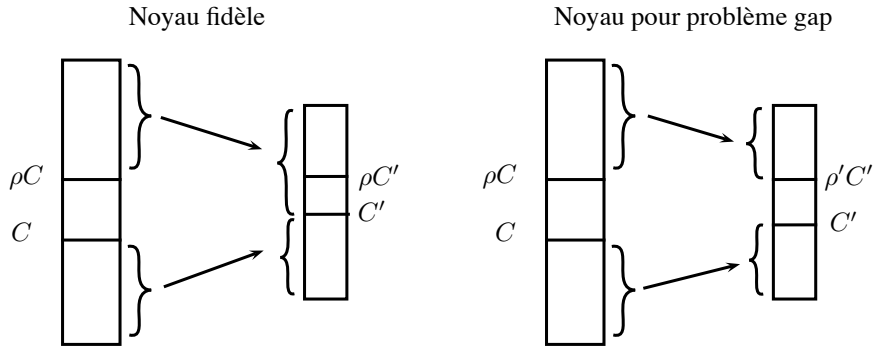


FIGURE 9 – Schéma représentant la différence entre un noyau fidèle et un noyau pour un problème gap. L'axe vertical représente la valeur d'une solution optimale, les flèches représentent comment sont transformées les instances, en fonction de la valeur de leur solution optimale.

pour un problème gap est plus difficile à obtenir, puisque, informellement, un noyau fidèle va « écraser » le gap existant, alors qu'un noyau pour un problème gap va le conserver, en réduisant simplement la taille de l'instance. Ainsi, on peut montrer qu'un noyau pour un problème gap implique un noyau fidèle, mais pas l'inverse. Cette différence est schématisée par la Figure 9.

2.5.3 Bornes inférieures de noyaux ρ-fidèles

Dans cette section, nous adaptons aux noyaux ρ-fidèles la méthode pour établir des bornes inférieures de noyaux, présentée Section 2.3.4. Pour cela, nous introduisons la notion de *gap-preserving cross-composition*. Nous aurons également besoin de la notion de « gap-introducing reduction » présentée Section 2.2.2.
Ici aussi nous ne présentons les compositions que dans leur version conjonctive (c'est à dire utilisant la version « OR »), mais des résultats similaires peuvent être obtenus en version disjonctive, depuis la preuve de la « AND-conjecture » en 2012 par A. Drucker [52].
Enfin, nous présentons la définition suivante pour deux problèmes de minimisation, mais une définition similaire peut être facilement obtenue lorsque l'un (ou les deux) problèmes sont des problèmes de maximisation.

Définition 30. *Soient* $\Pi_1 = (sol_1, cost_1, goal), \Pi_2 = (sol_2, cost_2, goal)$ *deux problèmes de minimisation définis sur l'alphabet* Σ, $\kappa : \Sigma^* \times \mathbb{N} \longrightarrow \mathbb{N}$ *une paramétrisation de* $\Sigma^* \times \mathbb{N}$, *et* $\mathcal{R}$ *une relation d'équivalence polynomiale sur* $\Sigma^* \times \mathbb{N}$.
Une gap-preserving cross-composition de Π_1 *vers* Π_2 *avec rapports* (ρ_1, ρ_2) *pour* $\rho_1, \rho_2 : \Sigma^* \longrightarrow [1, +\infty]$ *est un algorithme qui, étant donnée une suite* $(x_1, c_1), \cdots, (x_t, c_t)$ *de* t *éléments de* $\Sigma^* \times \mathbb{N}$ *appartenant à la même classe d'équivalence de* $\mathcal{R}$, *construit*

une instance $(x^*, c^*) \in \Sigma^* \times \mathbb{N}$ *en un temps polynomial en la taille de la suite d'instances telle que :*

- *s'il existe* $i \in \{1, \cdots, t\}$ *tel que* $(x_i, c_i) \in {\Pi_1}^{dec}$*, alors* $(x^*, c^*) \in {\Pi_2}^{dec}$*,*
- *si pour tout* $i \in \{1, \cdots, t\}$ *on a* $(x_i, \rho_1(x_i) \cdot c_i) \notin {\Pi_1}^{dec}$*, alors* $(x^*, \rho(x^*) \cdot c^*) \notin {\Pi_2}^{dec}$*,*
- $\kappa(x^*, c^*)$ *est polynomial en* $(\max_{i=1}^{t} |x_i| + c_i) + \log_{|\Sigma|} t$*.*

On rappelle que pour un problème de minimisation $\Pi = (sol, cost, goal)$, $(x, c) \in \Pi^{dec}$ signifie qu'il existe $y \in sol(x)$ tel que $cost(y) \leq c$.

Comme on peut le voir, une gap-preserving cross-composition n'est finalement qu'une cross-composition entre les versions décisions de deux problèmes d'optimisation préservant le « gap » entre solutions. Le résultat suivant montre qu'en la combinant avec une gap-introducing reduction, il est possible d'établir des bornes inférieures de noyaux, d'une manière similaire au Théorème 12.

Théorème 14. *Soient* $L \subseteq \Sigma^*$ *un problème,* $\Pi_1 = (sol_1, cost_1, goal_1)$*,* $\Pi_2 = (sol_2, cost_2, goal_2)$ *deux problèmes de minimisation sur l'alphabet* Σ*,* $\kappa : \Sigma^* \times \mathbb{N} \longrightarrow \mathbb{N}$ *une paramétrisation de* $\Sigma^* \times \mathbb{N}$ *et* $\rho_1, \rho_2 : \mathbb{N} \longrightarrow [1, +\infty]$*. Supposons que :*

- *Il existe une gap-introducing reduction de* L *vers* $\textsc{Gap}_{\rho_1}$-Π_1*.*
- *Il existe une gap-preserving cross-composition de* Π_1 *vers* Π_2 *avec rapports* (ρ_1, ρ_2)*.*
- L *est* $\mathcal{NP}$*-difficile.*

Si (Π_2, κ) *admet un noyau* ρ_2*-fidèle de taille polynomiale, alors* $\mathcal{NP} \subseteq co\mathcal{NP}/poly$*.*

Preuve. La preuve consiste à construire un algorithme de distillation de SAT vers $OR(\tilde{\Pi}_2^{dec})$.

Construction. Soit $(\phi_1, \cdots, \phi_t)$ une suite d'instances de SAT.
Comme L est $\mathcal{NP}$-difficile, nous transformons chacune d'entre elle en une instance de L, obtenant ainsi une suite $(x_1, \cdots, x_t) \in {\Sigma^*}^t$.
Utilisant la gap-introducing reduction de L vers $\textsc{Gap}_{\rho_1}$-Π_1, nous obtenons une suite d'instances de ${\Pi_1}^{dec}$ $(y_1, c_1), \cdots, (y_t, c_t)$.
Utilisant la relation d'équivalence polynomiale de la gap-preserving cross-composition, nous partitionnons cette suite en p classes d'équivalences

$$(y_1^1, c_1^1), \cdots, (y_{t_1}^1, c_{t_1}^1),$$

$$\cdots$$

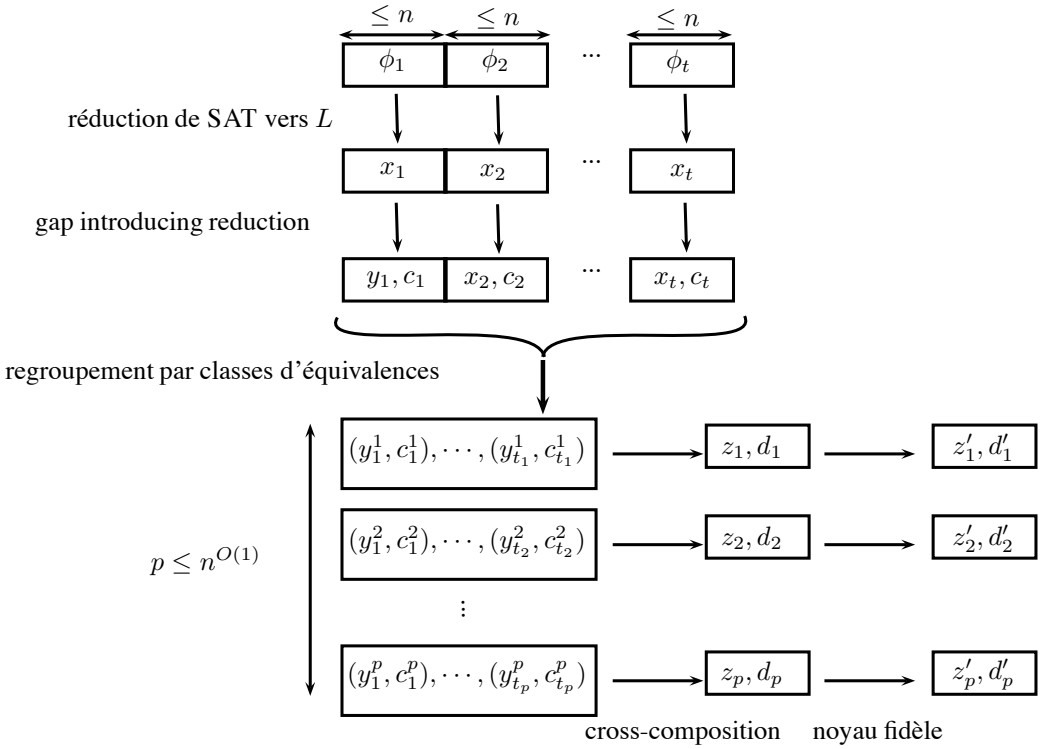


FIGURE 10 – Schéma de la construction pour la preuve du Théorème 14.

$$(y_1^j, c_1^j), \cdots, (y_{t_j}^j, c_{t_j}^j)$$
$$\cdots$$
$$(y_1^p, c_1^p), \cdots, (y_{t_p}^p, c_{t_p}^p)$$

Pour tout $j \in \{1, \cdots, p\}$, on applique, en temps polynomial en $\sum_{i=1}^{t_j} |y_i^j| + c_i^j$ la gap-preserving cross-composition Π_1 vers Π_2 avec rapports (ρ_1, ρ_2), obtenant ainsi une instance (z_j, d_j).
Puis, nous appliquons le noyau ρ_2-fidèle de (Π_2, κ) sur (z_j, d_j), permettant d'obtenir en temps polynomial en $|z_j|+d_j$ pour tout $j \in \{1, \cdots, p\}$, une instance (z'_j, d'_j). Nous obtenons en somme une suite $(z'_1, d'_j), \cdots, (z'_p, d'_p)$, instance du problème $OR(\tilde{\Pi}_2^{dec})$.
Un schéma de la construction est disponible Figure 10.

Temps d'exécution et taille de l'instance. Sachant que tous les algorithmes utilisés sont polynomiaux en la taille de leur entrée, il est clair que la construction peut être réalisée en temps polynomial en $\sum_{i=1}^{t} |x_i|$.
En outre, pour tout $j \in \{1, \cdots, p\}$, la taille de (z'_j, d'_j) est bornée par un polynôme en $\kappa(z_j, d_j)$ (par définition du noyau). Puisque (z_j, d_j) est le résultat d'une gap-preserving cross-composition, $\kappa(z_j, d_j)$ est borné par un polynôme en $(\max_{i=1}^{t_j} |y_i^j +$

$c_i^j) + \log(t_j)$, qui est lui même borné par un polynôme en $\max_{i=1}^{t} |x_i|$, et donc en $\max_{i=1}^{t} |\phi_i|$ puisque les réductions utilisées sont polynomiales. Enfin, on rappelle que par définition d'une relation d'équivalence polynomiale, p est polynomial en $\max_{i=1}^{t} |y_i| + c_i$.

Équivalence des solutions, préservation du gap. Supposons qu'il existe $i^* \in \{1, \cdots, t\}$ tel que ϕ_{i^*} soit satisfiable. Alors $x_{i^*} \in L$, et, par définition d'une gap-introducing reduction, cela implique que $(y_{i^*}, c_{i^*}) \in {\Pi_1}^{dec}$, et donc il existe $j \in \{1, \cdots, p\}$ et $i \in \{1, \cdots, t_j\}$ tels que $(y_i^j, c_i^j) \in {\Pi_1}^{dec}$. Par définition d'une gap-preserving cross-composition, cela implique que $(z_j, d_j) \in {\Pi_2}^{dec}$, qui implique $(z'_j, d'_j) \in {\Pi_2}^{dec}$ par définition d'un noyau fidèle, et donc la suite d'instance $(z'_1, d'_j), \cdots, (z'_p, d'_p)$ est bien positive.
Inversement, supposons que pour tout $i \in \{1, \cdots, t\}$, la formule Φ_i n'est pas satisfiable. On a donc $x_i \notin L$ pour tout $i \in \{1, \cdots, t\}$. Par définition d'une gap-introducing reduction, ceci implique que $(y_i, \rho_1(y_i) \cdot c_i) \notin {\Pi_1}^{dec}$. Ainsi, pour tout $(j, i) \in \{1, \cdots, p\} \times \{1, \cdots, t_j\}$, on a $(y_i^j, \rho_1(y_i^j) \cdot c_i^j) \notin {\Pi_1}^{dec}$, et par définition d'une gap-preserving cross-composition, on a $(z_j, \rho_2(z_j) \cdot c_j) \notin {\Pi_2}^{dec}$ pour tout $j \in \{1, \cdots, p\}$. Puis par le noyau ρ_2-fidèle, on a $(z'_j, \rho_2(z'_j) \cdot c'_j) \notin {\Pi_2}^{dec}$ pour tout $j \in \{1, \cdots, p\}$, et la suite d'instances obtenue est négative comme désiré. □

2.5.4 Quelques premiers résultats

Une application simple des principes présentés ci-dessus concerne le problème CLIQUE, dont nous rappelons la définition :

CLIQUE
Entrée : un graphe $G = (V, E)$
Sortie : un ensemble de sommets $C \subseteq V$ tel que $G[C]$ est une clique
But : maximiser $|C|$

La version de décision prenant donc un entier $c \in \mathbb{N}$ en paramètre, la question étant de savoir s'il existe une clique de taille au moins c dans le graphe. Concernant la paramétrisation standard $\kappa(G, c) = c$, le problème est $\mathcal{W}[1]$-complet [51], ce qui implique qu'il est impossible, sauf si $\mathcal{FPT} = \mathcal{W}[1]$, d'espérer trouver un noyau pour celui-ci, même de taille exponentielle. Concernant les noyaux fidèles, le Théorème 13 montre qu'un noyau (exponentiel) impliquerait un algorithme d'approximation paramétré. Or, les récents travaux sur ce sujet [19, 72] portent à croire qu'un tel algorithme est peu problable d'exister. Il est alors pertinent de se pencher sur des paramétrisation structurelles du problème.
Dans le résultat intial sur les bornes inférieures de noyaux [14], H. Bodlaender et al. montrent que CLIQUE paramétré par la tree-width du graphe ne peut pas admettre de noyau polynomial, sauf si $\mathcal{NP} \subseteq co\mathcal{NP}/poly$(ce dernier étant en revanche $\mathcal{FPT}$, par les travaux de B. Courcelle par exemple [42]). Ce résultat fut ensuite

renforcé par l'introduction des cross-compositions [17], en établissant, toujours sauf si $\mathcal{NP} \subseteq co\mathcal{NP}/poly$, que CLIQUE ne peut admettre de noyau polynomial s'il est paramétré par la taille du vertex cover minimum du graphe, un paramètre plus grand que la tree-width. Ci-dessous nous montrons en revanche que muni de cette paramétrisation, il admet un noyau $(1-\epsilon)$-fidèle linéaire pour tout $\epsilon > 0$. Nous généralisons finalement le résultat pour toute la hiérarchie de paramètres « distance à un graphe de tree-width w », pour tout w fixé. D'un point de vue négatif, nous montrons finalement que CLIQUE ne peut admettre de noyau ρ-fidèle polynomial lorsque paramétré par la tree-width du graphe d'entrée, pour tout $\rho \in]0, 1]$ constant, sauf si $\mathcal{NP} \subseteq co\mathcal{NP}/poly$.

Résultat positif

Définissons tout d'abord formellement la paramétrisation utilisée par la suite. Étant donné un graphe $G = (V, E)$ et un entier $w \in \mathbb{N}$, on définit :

$$d_{tw\leq w}(G) = \min_{S\subseteq V}\{|S| : tw(G[V \setminus S]) \leq w\}$$

Notons que pour $w = 0$ et $w = 1$ respectivement, ces paramètres correspondent à la taille minimum d'un vertex cover et d'un feedback vertex set. En effet, $d_{tw\leq 0}G$ par exemple est le nombre minimum de sommets à retirer afin d'obtenir un graphe de tree-width 0, *i.e.* un graphe sans arêtes, c'est donc bien la taille minimum d'un vertex cover.
Afin que cette paramétrisation soit pertinente en pratique, nous rajouterons dans l'entrée du problème un ensemble $S \subseteq V$ de sommets tel que $tw(G[V \setminus S]) \leq w$. Pour plus de détails concernant l'ajout de structures pour les paramétrisations structurelles, nous renvoyons le lecteur vers la Section 2.2 de [79].

Théorème 15. *Pour tout $w \in \mathbb{N}$ et tout $\epsilon > 0$,* CLIQUE *paramétré par $d_{tw\leq w}$ admet un noyau $(1-\epsilon)$-fidèle linéaire.*

Preuve. Soient $G = (V, E)$, $c \in \mathbb{N}$, et $S \subseteq V$ tel que $tw(G[V \setminus S]) \leq w$.
Tout d'abord, si $c \leq \frac{w+1}{\epsilon}$, alors il est possible de décider en temps polynomial si le graphe contient une clique de taille c ou non, et ainsi de retourner une instance triviale en fonction. Dans ce cas, il est clair que nous obtenons un noyau 1-fidèle de taille constante.
Supposons donc que $c \geq \frac{w+1}{\epsilon}$, ce qui implique $1 - \epsilon \leq \frac{c-w-1}{c}$. Le noyau consiste simplement à retourner $G' = G[S]$, avec $c' = c - w - 1$ et $S' = S$. La taille de l'instance ainsi retournée est trivialement linéaire en $|S|$.
Supposons que G contienne une clique C de taille c. Alors, puisque $tw(G[V\setminus S]) \leq w$, on a $|C \cap (V \setminus S)| \leq w+1$. Ainsi, G' contient forcément une clique de taille au moins $c - (w + 1) = c'$.
Inversement, si G' a une clique de taille c', alors cette clique existe également dans G, et elle est de taille $c' = \frac{c-w-1}{c}c \cdot c \geq (1 - \epsilon) \cdot c$, ce qui termine la preuve, puisque

le noyau se construit clairement en temps polynomial. □

Corollaire 1. *Pour tout $\epsilon > 0$,* Clique *admet un noyau $(1-\epsilon)$-fidèle de taille linéaire si paramétré par la taille d'un vertex cover ou d'un feedback vertex set du graphe.*

Résultat négatif

Comme annoncé, nous montrons qu'à l'autre bout de la hiérarchie de paramètres, l'existence d'un noyau fidèle est improbable.

Théorème 16. *S'il existe $\rho \in]0,1]$, tel que* Clique *admet un noyau ρ-fidèle polynomial lorsque paramétré par la tree-width du graphe d'entrée, alors $\mathcal{NP} \subseteq co\mathcal{NP}/poly$.*

Preuve. Il est connu [74] que quel que soit $\rho \in]0,1]$, le problème Gap$_\rho$-Clique est $\mathcal{NP}$-difficile. Ainsi, il existe une gap-introducing reduction de ce dernier vers Clique. Puis, remarquons que l'union disjointe d'une suite de graphes est une gap-preserving cross-composition de Clique vers lui-même paramétré par la tree-width. En effet, définissons d'abord la relation d'équivalence : deux instances (G_1, c_1) et (G_2, c_2) sont équivalentes si $c_1 = c_2$. Considérons maintenant une suite d'instances équivalentes $(G_1, c), \cdots, (G_t, c)$, notons G' l'union disjointe de ces graphes, et posons $c' = c$. Il est clair que s'il existe $i \in \{1, \cdots, t\}$ tel que G_i contienne une clique de taille c, alors cette clique existe également dans G'. Inversement, si pour tout $i \in \{1, \cdots, t\}$, le graphe G_i ne contient pas de clique de taille $\rho \cdot c$, alors G' ne peut pas non plus avoir une telle clique.
Enfin, remarquons que $tw(G') = \max_{i=1}^{t} tw(G_i)$, et le paramètre est de la taille désirée, ce qui termine la preuve. □

2.5.5 Perspectives

La recherche de noyaux fidèles pour un problème peut avoir plusieurs intérêts : d'un point de vue théorique, cela permet tout d'abord de mieux comprendre la complexité intrinsèque du problème considéré vis-à-vis d'une paramétrisation donnée, et de connaître ainsi les limites des algorithmes de pré-traitement. Cela permet aussi, dans le cadre de la résolution approchée de problèmes en temps paramétré, de savoir intuitivement ce qu'il est possible de réaliser sans effectuer de branchement. D'un point de vue pratique ensuite, ils donnent un cadre théorique pour l'analyse de règles de pré-traitement qui ne peuvent être capturées par les noyaux exacts, en raison du fait par exemple qu'ils ne préservent pas tout à fait l'exactitude des solutions. Les noyaux fidèles permettraient enfin de pouvoir drastiquement réduire la taille d'une instance d'un problème difficile tout en ayant une certaine garantie de performance sur la qualité des solutions conservées. Pour utiliser ce qui précède, on pourrait par

exemple diminuer la taille d'une instance de Clique en cherchant d'abord un vertex cover de grande taille (à l'aide d'un algorithme approché par exemple) avant de lancer un algorithme exact exponentiel.

Les problèmes candidats pour la recherche de noyaux ρ-fidèles doivent finalement remplir deux conditions :

- Comme nous l'avons vu précédemment, un noyau ρ-fidèle implique immédiatement (sous réserve que le problème soit solution-constructible) un algorithme ρ-approché en temps $\mathcal{FPT}$. Ainsi, les bornes inférieures d'approximation en temps $\mathcal{FPT}$ (et d'autant plus en temps polynomial) valent aussi pour les noyaux fidèles. De même, un algorithme ρ-approché s'exécutant en temps $\mathcal{FPT}$ ou polynomial donne immédiatement un noyau ρ-fidèle de taille constante.

- De la même manière, un noyau d'une certaine taille donne immédiatement un noyau 1-fidèle de la même taille, il est donc préférable de se concentrer sur des problèmes paramétrés n'admettant pas de noyau polynomial (à moins d'améliorer drastiquement la taille).

Dans cette optique, nous proposons quelques problèmes ouverts concernant l'obtention de noyaux fidèles, avec une brève explication pour chacun d'eux.

Problème ouvert 1. *Est-ce que* Dominating Set *admet un noyau ρ-fidèle polynomial lorsque paramétré par la taille d'un vertex cover du graphe, pour un certain $\rho \in [1, +\infty]$?*

Étant donné un vertex cover $S \subseteq V$, s'il existe $x \neq y$ avec $N(x) = N(y)$, alors supprimer x ou y est une règle de pré-traitement préservant l'exactitude des solutions, qui permet d'obtenir une instance avec $|V \setminus S| \leq 2^{|S|}$, et donc un noyau exponentiel. Peut-on alors définir d'autres règles préservant de manière approchée les solutions afin d'obtenir un noyau polynomial ? Concernant les noyaux exacts, Dominating Set ne peut admettre de noyau polynomial (toujours paramétré par la taille d'un vertex cover), sauf si $\mathcal{NP} \subseteq co\mathcal{NP}/poly$[48]. Avant d'attaquer ce problème ouvert, une étape possible est de s'intéresser au problème Hitting Set paramétré par la taille de l'univers :

Hitting Set
Entrée : Un ensemble U de n éléments (univers), un ensemble $\mathcal{F}$ d'ensembles de U, $k \in \mathbb{N}$
Sortie : Existe-t-il un ensemble $S \subseteq U$ de taille k tel que $\forall F \in \mathcal{F}$ on a $F \cap S \neq \emptyset$?

Problème ouvert 2. *Est-ce que* Hitting Set *paramétré par la taille de l'univers admet un noyau ρ-fidèle polynomial pour un certain $\rho \in [1, +\infty]$?*

Ce problème est en fait un cas particulier du problème précédent, en ce sens qu'un noyau ρ-fidèle polynomial pour Dominating Set paramétré par la taille d'un vertex cover impliquerait un noyau ρ-fidèle polynomial pour Hitting Set. Un cas particulier de ce dernier serait de traiter le problème d-Hitting Set, pour lequel un noyau de taille $(2d-1)k^{d-1}+k$ existe [2], alors qu'il a été prouvé [44] qu'il est inutile d'espérer un noyau de taille $\mathcal{O}(k^{d-\epsilon})$ pour tout $\epsilon > 0$ (sauf si $\mathcal{NP} \subseteq co\mathcal{NP}/poly$). On pourrait alors imaginer, grâce au noyaux fidèles, obtenir une taille de noyau dont le polynôme ne dépend pas de d.

Une autre question ouverte concerne le problème Independent Set :

Problème ouvert 3. *Est-ce que* Independent Set *admet un noyau ρ-fidèle polynomial s'il est paramétré par la tree-width du graphe d'entrée, pour un certain $\rho \in]0,1]$?*

Le problème, muni de cette paramétrisation, est clairement $\mathcal{FPT}$, mais il n'admet pas de noyau polynomial [14]. Lorsqu'il est paramétré par la taille d'un vertex cover ou d'un feedback vertex set, le problème admet des noyaux (exacts) polynomiaux puisqu'il est équivalent à Vertex Cover.

Problème ouvert 4. *Est-ce que* TreeWidth *muni de sa paramétrisation standard admet un noyau ρ-fidèle polynomial, pour un certain $\rho \in [1,+\infty]$?*

En utilisant la AND-composition triviale [14], le problème n'admet pas de noyau polynomial exact (sauf si $\mathcal{NP} \subseteq co\mathcal{NP}/poly$). Cependant, il n'y a aucune borne inférieure sur une quelconque approximation polynomiale (sauf avec un terme additif [18]), et le meilleur rapport d'approximation (en temps polynomial) est $\mathcal{O}(\log(n))$ [18]. D'un autre côté, le problème est $\mathcal{FPT}$ [15], et de nombreux algorithmes plus simples fournissent une solution approchée en temps $\mathcal{FPT}$ [102, 103].

Problème ouvert 5. *Est-ce que* TreeWidth *paramétré par la taille d'un vertex cover admet un noyau ρ-fidèle avec un nombre linéaire de sommets, ou, mieux, de taille (en termes de bit nécessaires à l'encodage) $\mathcal{O}(|X|^{2-\epsilon})$, où X est un vertex cover du graphe d'entrée ?*

Dans [79] est proposé un noyau avec $\mathcal{O}(|X|^2)$ sommets pouvant être encodé avec $\mathcal{O}(|X|^3)$ bits. De plus, l'auteur prouve qu'il est impossible, sauf si $\mathcal{NP} \subseteq co\mathcal{NP}/poly$, d'obtenir un noyau avec $\mathcal{O}(|X|^{2-\epsilon})$ en nombre de bits. Ainsi un noyau fidèle avec un nombre linéaire de sommets améliorerait la borne supérieure connue, tandis qu'un noyau fidèle de taille $\mathcal{O}(|X|^{2-\epsilon})$ (en nombre de bits) serait en quelque sorte optimal.

Chapitre 3

Problèmes de Compactions de Graphes

Plan du Chapitre :

Dans ce chapitre, nous étudions des problèmes d'optimisation liés aux compactions de graphes. Ces problèmes, bien que peu étudiés dans la littérature, ont un intérêt à la fois pratique et théorique. En effet, d'un point de vue pratique, nous verrons comment ces problèmes peuvent avoir des applications en génie logiciel. D'un point de vue théorique enfin, nous verrons qu'ils peuvent être vus comme des problèmes de partitions, de modifications de graphes ou bien d'homomorphismes. Après avoir défini formellement les problèmes et présenté un bref état de l'art, nous exhiberons ces liens. Ensuite, nous nous concentrerons sur le problème particulier Sparsest k-Compaction, en donnant des résultats algorithmiques positifs et négatifs pour la recherche de solutions exactes ou approchées. Nous étudions ensuite la complexité du problème lorsque des contraintes sont imposées soit sur la solution, soit sur l'instance d'entrée. Nous terminerons ensuite ce chapitre par une liste de problèmes ouverts.

Ces travaux ont donné lieu à la publication suivante :

- [116] *On the Sum-Max Graph Partitioning Problem*, avec Marin Bougeret, Rodolphe Giroudeau et Jean-Claude König. Elsevier Theoretical Computer Science, vol. 540-541, pp. 143-155, 2014.
  (Une version préliminaire de cet article fut publié dans les actes de la conférence *ISCO 2012*, Springer LNCS 7422, pp. 297-208 [115])

3.1 Définitions et problèmes liés

3.1.1 Définitions

La définition la plus simple d'une compaction est de l'exprimer sous forme d'une partition de ses sommets. Dans l'intégralité de ce chapitre, tous les graphes, en plus d'être simples, non orientés et sans boucle, comme mentionné dans le Chapitre 1, sont supposés *connexes*. Autrement dit, pour toute paire de sommets du graphe il existe un chemin dont ils sont les extrémités.
Une notion importante pour la suite est celle de graphe quotient.

Définition 31. *Soient $G = (V, E)$ un graphe, et $\mathcal{P} = \{V_1, \cdots, V_k\}$ une k-partition de V. Le graphe quotient de G par $\mathcal{P}$, que l'on note $G_{\mathcal{P}}$, est le graphe dont l'ensemble des sommets est $\mathcal{P}$, et où $\{V_i, V_j\}$ est une arête s'il existe $u \in V_i$ et $v \in V_j$ tels que $\{u, v\} \in E$.*

La Figure 11 illustre ce qu'est un graphe quotient par rapport à une partition donnée.

Les problèmes de compactions peuvent, d'une manière générique, être définis des deux manières suivantes, selon si l'on s'intéresse à des problèmes de décision ou des problèmes d'optimisation. Dans ce qui suit, Π et f désignent respectivement

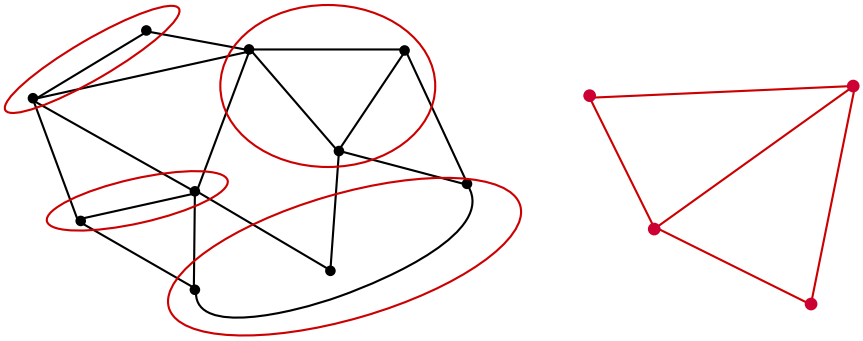


FIGURE 11 – Exemple de graphe et d'une partition de ses sommets (à gauche), et le graphe quotient résultant (à droite).

une propriété et un paramètre de graphes. Des exemples de propriétés Π peuvent être « ne comporte pas de cycle » ou « être planaire », alors que des exemples de paramètres de graphes peuvent être le diamètre ou la tree-width.

Π-COMPACTION
Entrée : Un graphe $G = (V, E)$, un entier $k \in \mathbb{N}$
Sortie : Existe-t-il une k-partition $\mathcal{P}$ de V telle que $\Pi(G_{\mathcal{P}})$?

f-COMPACTION
Entrée : Un graphe $G = (V, E)$, un entier $k \in \mathbb{N}$
Sortie : Une k-partition P de G.
But : Optimiser $f(G_P)$

Comme on peut le voir, une compaction n'est finalement qu'une partition, et les problèmes que nous étudierons concerneront des propriétés que devra respecter le graphe quotient.
Dans la suite du document, les éléments des partitions (qui sont des sous-ensembles de V) pourront être appelés *ensembles*, *sous-ensembles*, *parties* ou *clusters*, ce dernier terme étant plutôt privilégié.

3.1.2 Problèmes liés

Les problèmes de partitionnement de graphes sont des problèmes classiques en optimisation combinatoire, et sont au cœur de nombreuses problématiques appliquées. De manière similaire aux définitions précédentes, ils consistent généralement à trouver une partition des sommets d'un graphe sous diverses contraintes et optimisant une certaine fonction objectif, dépendant de l'application. L'une des motivations principales est de partitionner les sommets, représentant des entités d'un certain domaine, en parties cohésives, et plutôt isolées des autres entités. Parmi les appli-

cations de ce principe, on peut citer le problème de classification de données, la recherche de communautés, ou bien les problèmes liés au VLSI [13].
Le problème théorique se rapprochant le plus de ces problématiques est le problème Min-k-Cut :

Min-k-Cut
Entrée : Un graphe $G = (V, E)$, une pondération de ses arêtes $w : E \longrightarrow \mathbb{N}$, un entier $k \in \mathbb{N}$
Sortie : Une k-partition de V : $\{V_1, \cdots, V_k\}$ avec $V_i \neq \emptyset$ pour tout $i \in \{1, \cdots, k\}$
But : Minimiser $\sum_{i,j \in \{1,\cdots,k\}, i \neq j} cut(V_i, V_j)$

Avec $cut(V_i, V_j) = |\{\{u, v\} : u \in V_i, v \in V_j\}|$ le nombre d'arêtes ayant une extrémité dans V_i et une extrémité dans V_j.
Min-k-Cut a été très largement étudié ces trente dernières années, tant du point de vue de la complexité classique [61], résolution exacte [65], paramétrée [50] ou approchée [105].

D'autres fonctions objectif ont déjà été étudiées dans la littérature. Au lieu de minimiser le poids des arêtes qui séparent les clusters, on peut, dans la même idée, optimiser le poids des arêtes qui sont à l'intérieur de chaque cluster [73]. On peut également combiner les deux idées, et minimiser le « cut ratio » de la partition [100]. Certaines études généralisent ces problèmes à l'aide de formulations génériques : [68] donne des bornes inférieures de complexité lorsque l'objectif est de maximiser une certaine fonction des arêtes se trouvant à l'intérieur des clusters, [86] propose un algorithme exponentiel avec un temps d'exécution de $\mathcal{O}^*(2^n)$ pour toute une classe de problèmes de partition tels que Max-k-Cut, k-Domatic-Partition ou Chromatic Number. Enfin, [54] définit le problème M-Partitioning, où le but est de trouver une partition des sommets respectant certaines contraintes définies par une matrice M, et donne des résultats de complexité selon la forme de la matrice.

Il est également possible de voir les problèmes de compaction de graphes comme des problèmes de modification de graphes. Ces problèmes prennent, de manière générique, un graphe G et un entier t, et la question et de savoir s'il est possible d'obtenir un graphe ayant certaines propriétés en effectuant, à partir de G, au plus t modifications. Ces modifications peuvent être diverses et variées :

- suppression/ajout d'un sommet,
- suppression/ajout d'une arête,
- contraction d'une arête,
- subdivision d'une arête,

Par exemple, voir [96] pour un état de l'art sur les problèmes de modifications d'arêtes.

Étant donné un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$, on peut voir qu'obtenir une compaction de G vers un graphe H à k sommet est en fait équivalent à obtenir H à partir de G en utilisant au plus $n - k$ modifications, où l'unique modification disponible est la contraction de paires de sommets (pas forcément adjacents). Par extension, on appellera cette modification une *compaction* de paires de sommets. Soient $u, v \in V$, avec $u \neq v$, la compaction de u et v consiste à supprimer u et v du graphe, créer un nouveau sommet uv relié aux anciens voisins de u et de v.
De cette caractérisation nous pouvons définir une nouvelle paramétrisation structurelle correspondant au nombre de compactions de paires de sommets nécessaires pour arriver au graphe souhaité. Étant donnée une instance de Sparsest k-Compaction (ou Bounded Degree k-Compaction) composée d'un graphe $G = (V, E)$ et d'un entier k, on définira

$$\hat{k} = |V| - k$$

On utilisera notamment ce paramètre dans le cas de l'étude de la complexité de Sparsest k-Compaction dans les split graphes, en Section 3.5.3.

Une autre façon de regarder les problèmes de compaction est d'utiliser les homomorphismes de graphes. Un homomorphisme d'un graphe $G = (V_G, E_G)$ vers un graphe $H = (V_H, E_H)$ est une fonction $h : V_G \longrightarrow V_H$ telle que $\{x, y\} \in E_G$ implique $\{h(x), h(y)\} \in E_H$ pour tout $x, y \in V_G$ (voir [75] pour plus d'informations sur les homomorphismes). Il est facile de voir que le problème générique de compaction peut en fait s'exprimer d'une manière équivalente par la recherche d'un homomorphisme particulier du graphe d'entrée G vers un graphe H respectant la propriété Π. Les particularités de l'homomorphisme sont les suivantes :

- Le graphe d'arrivée H doit être réflexif (autrement dit, avoir une boucle sur chaque sommet), car dans le cas contraire nous serions contraints de chercher une partition où chaque cluster induit un ensemble indépendant.

- L'homomorphisme doit être arête-surjectif sur toutes les arêtes (pour qu'une arête dans H signifie l'existence d'une arête entre deux clusters dans G) sauf sur les boucles (car nous n'interdisons pas le fait que les clusters soient indépendants).

Plusieurs articles de N. Vikas [109, 110, 111, 112] se focalisent sur la complexité du problème H-Compaction, défini pour tout graphe H fixé :

H-**Compaction**
Entrée : Un graphe $G = (V, E)$
Sortie : Existe-t-il une compaction de G vers H ?

Ces articles étudient la complexité de ce problème selon les classes de graphes auxquelles G et H appartiennent, obtenant tantôt la polynomialité tantôt la $\mathcal{NP}$-difficulté du problème. Ils remarquent notamment que demander à compacter un graphe G vers le chemin à k sommets P_k est en fait équivalent à demander si le diamètre de G est au moins $k-1$.

3.1.3 Problèmes étudiés et applications en remodularisation de logiciels

Alors que les problèmes de compactions que nous avons vu précédemment sont des problèmes de décision, nous nous sommes plutôt concentrés sur des problèmes d'optimisation. Plus précisément, nous nous sommes concentrés sur les problèmes suivants :

SPARSEST k-COMPACTION
Entrée : Un graphe $G = (V, E)$, un entier $k \in \mathbb{N}$
Sortie : Une k-partition P de G.
But : Minimiser $|E(G_P)|$

BOUNDED DEGREE k-COMPACTION
Entrée : Un graphe $G = (V, E)$, un entier $k \in \mathbb{N}$
Sortie : Une k-partition P de G.
But : Minimiser $\Delta(G_P)$

On rappelle qu'étant donné un graphe G, $E(G)$ et $\Delta(H)$ désignent respectivement son ensemble d'arêtes et son degré maximum. Pour une partition $P = \{V_1, \cdots, V_k\}$ de V (et donc une solution réalisable pour SPARSEST k-COMPACTION ou BOUNDED DEGREE k-COMPACTION), on notera $cost(P)$ la valeur de cette solution selon le problème considéré : le nombre d'arêtes du graphe quotient G_P pour SPARSEST k-COMPACTION, ou son degré maximum pour BOUNDED DEGREE k-COMPACTION. Dans la version de décision de chacun de ces problèmes, nous noterons C la valeur de la solution souhaitée.

Nous serons également amenés à considérer des versions pondérées de ces problèmes. Dans celles-ci, le graphe d'entrée comportera en plus une fonction de poids $w : E \longrightarrow \mathbb{N}$. Puis, différentes manières de définir un graphe quotient dans le cas pondéré peuvent être envisagées, en particulier pour la définition des poids sur les arêtes de celui-ci. Le poids que nous définirons pour nos problème sera le poids maximal d'une arête entre les deux clusters considérés. Plus formellement, étant donnés $G = (V, E)$, $k \in \mathbb{N}$, $w : E \longrightarrow \mathbb{N}$, et une k partition de V $P = \{V_1, \cdots, V_k\}$, on définit le graphe quotient $G_P = (P, E')$, et $w' : E' \longrightarrow \mathbb{N}$ de la manière suivante : $w(V_i, V_j) = \max_{u \in V_i, v \in V_j} w(u, v)$ pour tout $i, j \in \{1, \cdots, k\}$, $i \neq j$. On aura alors l'équivalent suivant du problème SPARSEST k-COMPACTION :

Weighted Sparsest k-Compaction
Entrée : Un graphe $G = (V, E)$, $k \in \mathbb{N}$, $w : E \longrightarrow \mathbb{N}$
Sortie : Une compaction de G vers un graphe à k sommets $H = (V_H, E_H)$ pondéré par $w : E_H \longrightarrow \mathbb{N}$
But : Minimiser $\sum_{e \in E_H} w(e)$

Une application de ces problèmes concerne la remodularisation de logiciels en langage orienté objets. En effet, un programme écrit dans un langage orienté objets (tel que *Java*, *C++*) peut être représenté par un graphe dont les sommets sont les classes du programme, et où deux sommets sont reliés si les classes correspondantes sont en relation. Ces relations peuvent être diverses, et peuvent par exemple correspondre aux relations d'héritage des classes, ou aux relations d'usage d'une classe par une autre. Dans un soucis d'organisation, les logiciels à objets sont, la plupart du temps, partitionnés (ou modularisés) en parties plus petites (à l'aide de « paquetages » par exemple en *Java*), et la manière de partitionner ces systèmes est un élément déterminant concernant l'évolution et la maintenance futures de ceux-ci. Ainsi cette représentation en graphes permet, grâce à la résolution de problèmes de partition, de pouvoir trouver une modularisation de bonne qualité, en utilisant des métriques jouant le rôle de fonctions objectif [117]. Ces métriques, guidées par des principes de génie logiciel, reposent principalement sur l'idée de créer des clusters ayant une forte cohésion, tout en diminuant le couplage entre les clusters. L'une d'entre elles repose sur le principe d'*impact de changement* [1] : étant donné qu'une modification d'un paquetage implique la modification de tous les paquetages reliés à celui-ci, une bonne modularisation organisera les classes de telles façons que le nombre de futurs changements soit minimum. Ainsi, nous rechercherons une partition des sommets du graphe telle que :

- le nombre d'arêtes du graphe quotient soit minimum, ou
- chaque cluster soit relié à peu d'autres clusters.

Chacun de ces critères sont respectivement modélisés par Sparsest k-Compaction et Bounded Degree k-Compaction.

3.2 Difficulté des problèmes

Cette section présente la difficulté des deux problèmes mentionnés ci-avant : Sparsest k-Compaction et Bounded Degree k-Compaction.

3.2.1 Difficulté de Sparsest k-Compaction

Nous nous intéressons d'abord à la difficulté de Sparsest k-Compaction, tant en résolution exacte polynomiale, paramétrée ou approchée. En effet, nous présentons

une réduction depuis le problème INDEPENDENT SET, préservant à la fois le paramètre k et l'écart d'approximation.
L'idée de la réduction est très simple : étant donné un graphe connexe $G = (V, E)$, nous ajoutons un sommet universel ω afin de former le graphe G' (un sommet universel est un sommet relié à tous les autres sommets du graphe). On remarque alors que si G a un ensemble indépendant de taille k, alors la partition de G' qui consiste à mettre chacun de ses sommets dans un singleton, et tous les autres sommets dans un dernier ensemble, forme une $(k+1)$-partition de coût k, puisque les seules arêtes sont entre l'ensemble contenant ω et les autres. Il est également facile de montrer que la réciproque est vraie, puisque toute $(k+1)$-partition doit forcément avoir un coût d'au moins k à cause du sommet universel. Le théorème suivant montre qu'il est même possible de préserver le gap d'approximation du problème INDEPENDENT SET.

Théorème 17. *Pour tout $\epsilon > 0$, pour toute fonction $r = \mathcal{O}(n^{1-\epsilon})$, il existe une réduction gap preserving depuis* GAP$_{r'}$-INDEPENDENT SET *vers* GAP$_r$-SPARSEST k-COMPACTION *préservant le paramètre, avec $r'(n) = 2r(n+1)$.*

Preuve. On rappelle la définition du problème GAP$_\rho$-INDEPENDENT SET, pour toute fonction $\rho : \mathbb{N} \longrightarrow [1, +\infty]$:

GAP$_\rho$ INDEPENDENT SET
Entrée : Un graphe $G = (V, E)$, $k \leq |V|$
Sortie : Décider si $\alpha(G) \geq k$ ou $\alpha(G) < \frac{1}{\rho(|V|)}k$.

Soit une fonction $r : \mathbb{N} \longrightarrow [1, +\infty]$. Étant donnés un graphe $G = (V, E)$ avec $|V| = n$, et $k \leq |V|$, on construit un graphe $G' = (V', E')$ avec $|V'| = n'$, et $k' \leq |V'|$ tels que :

- G' et k' sont constructibles en temps polynomial
- $k' = f(k)$ pour une certaine fonction f
- s'il existe un ensemble indépendant de taille k dans G, alors il existe une solution de coût $k' - 1$ dans G'
- s'il n'existe pas d'ensemble indépendant de taille $\frac{1}{2r(n+1)}k$ dans G, alors il n'existe pas de solution de coût $r(n')k'$ dans G'.

Soient $G = (V, E)$, et $k \leq |V|$ une instance de INDEPENDENT SET. La réduction consiste simplement à ajouter un sommet universel ω. Autrement dit, nous construisons $G' = (V', E')$ avec $V' = V \cup \{\omega\}$ et $E' = E \cup \{\{\omega, v\}, v \in V\}$. Nous définissons également $k' = k + 1$.
Supposons qu'il existe un ensemble indépendant de taille k $S = \{s_1, \cdots, s_k\}$ dans G. Nous construisons la partition $\{V_1, \cdots, V_{k+1}\}$ suivante : $V_i = \{s_i\}$ pour tout $i \in \{1, \cdots, k\}$, et $V_{k+1} = V' \setminus S$. Par construction, il est facile de vérifier que

$cost(V_1, \cdots, V_{k'}) = k$.
Supposons maintenant qu'il n'existe pas d'ensemble indépendant de taille $\frac{2}{r(n)}k$ dans G, et soit $\{V_1, \cdots, V_{k'}\}$ une k'-partition des sommets de G'. Puisque ω est un sommet universel, il est clair que le coût de cette solution sera d'au moins k, plus le nombre d'arêtes x entre les k clusters ne contenant pas ω. Le reste de la preuve consiste à borner inférieurement x, en utilisant le fait que G ne contient pas d'ensemble indépendant de taille rk. En effet, par le théorème de Turan [107], tout graphe H à n_H sommets et m_H arêtes contient un ensemble indépendant de taille au moins $\frac{n_H^2}{2m_H+n_H}$. Ainsi, on a l'inégalité suivante :

$$\frac{1}{2r(n+1)}k \geq \alpha(G) \geq \frac{k^2}{2x+k}$$

qui implique $x \geq kr(n+1) - \frac{k}{2}$. Et donc $x+k \geq kr(n+1) + \frac{k}{2} \geq kr(n+1) = kr(n')$. La réduction s'applique facilement en temps polynomial, et le paramètre $k' = k+1$ est préservé, ce qui termine la démonstration. □

On a alors les corollaires suivants :

Corollaire 2. SPARSEST k-COMPACTION *est $\mathcal{NP}$-complet.*

Corollaire 3. SPARSEST k-COMPACTION *est $\mathcal{W}[1]$-difficile.*

Corollaire 4. SPARSEST k-COMPACTION *est non approximable à un facteur $\mathcal{O}(n^{1-\epsilon})$ pour tout $\epsilon > 0$, sauf si $\mathcal{P} = \mathcal{NP}$.*

3.2.2 Difficulté de BOUNDED DEGREE k-COMPACTION

Nous montrons maintenant que BOUNDED DEGREE k-COMPACTION est $\mathcal{NP}$-difficile, même lorsque le graphe d'entrée est un arbre. Ce travail a été réalisé en collaboration avec A. Benard, M. Bougeret et R. Giroudeau.

Théorème 18. BOUNDED DEGREE k-COMPACTION *est $\mathcal{NP}$-difficile, même lorsque le graphe d'entrée est un arbre.*

Preuve. Nous réduisons depuis le problème $\mathcal{NP}$-complet 3-PARTITION [61] :

3-PARTITION
Entrée : Un entier $B \in \mathbb{N}$, un multi-ensemble S de $3m$ variables, muni d'une fonction de poids $w : S \longrightarrow \mathbb{N}$, avec pour tout $s \in S$, $\frac{B}{4} < w(s) < \frac{B}{2}$
Sortie : Existe-t-il une m-partition de S : $\{S_1, \cdots, S_m\}$ telle que $\sum_{s \in S_i} w(s) \leq B$ pour tout $i \in \{1, \cdots, m\}$?

Rappelons qu'un multi-ensemble est un ensemble dans lequel les éléments peuvent se répeter.

Soit une instance de 3-PARTITION comme définie ci-dessus, telle que B divise $m-1$. Il est facile de voir que cette contrainte sur l'instance d'entrée conserve la $\mathcal{NP}$-difficulté du problème. Nous commençons tout d'abord par multiplier tous les poids $w(s) \in S$ par $\frac{m-1}{B}$, de telle sorte que nous cherchons maintenant une partition $\{S_1, \cdots, S_m\}$ telle que $\sum_{s \in S_i} w(s) \leq m-1$ pour tout $i \in \{1, \cdots, m\}$. Par ce qui précède, on a toujours $w(s) \in \mathbb{N}$ pour tout $s \in S$.
Étant donnée une telle instance, nous construisons le graphe $G = (V, E)$ suivant : pour tout $s \in S$, on définit un sommet v_s, ainsi que $w(s)$ sommets pendants $a_s^1, \cdots, a_s^{w(s)}$ connectés à v_s. Notons $A = \{a_s^x : s \in S, x \in \{1, \cdots, w(s)\}\}$ et $V_s = \{v_s : s \in S\}$. On ajoute enfin un sommet ω connecté à v_s pour tout $s \in S$.
Il est facile de voir que le graphe ainsi construit est un arbre (de profondeur trois si l'on considère ω comme la racine). On définit enfin $k = \sum_{s \in S} w(s) + m + 1$, et $C = m$. Cette construction peut clairement s'exécuter en temps polynomial.
Montrons maintenant l'équivalence entre les solutions.

- Supposons qu'il existe une partition $S_1, \cdots, S_m$ de S telle que $\sum_{s \in S} w(S) \leq m-1$ pour tout $i \in \{1, \cdots, m\}$. On construit alors la k-partition des sommets de G suivante : pour tout $i \in \{1, \cdots, m\}$, si $S_i = \{s, s', s''\}$, alors on construit la partie $V_i = \{v_s, v_{s'}, v_{s''}\}$. Enfin, nous mettons chacun des sommets restants dans un singleton. On peut vérifier que la partition ainsi construite est une k-partition, et que le graphe quotient associé est de degré maximum $m = C$. En effet, le cluster contenant ω est adjacent aux m parties de V_s, et pour chaque $i \in \{1, \cdots, m\}$, la partie V_i est adjacent à $m-1$ partitions parmi A (par définition d'une solution au problème 3-PARTITION), et au cluster contenant ω.

- Inversement, supposons que nous avons une k-partition $P = \{P_1, \cdots, P_k\}$ des sommets de G dont le graphe quotient a un degré maximum au plus C. Sans perdre de généralité supposons que $P_1, \cdots, P_{m'} \in P$ sont les clusters qui contiennent au moins un élément de V_s, pour un certain $m' \leq k$. Ainsi, $P \backslash \{P_1, \cdots, P_{m'}\}$ sont des clusters composés de sommets de $A \cup \{\omega\}$ seulement. Montrons que $m' = m$:

  - Si $m' < m$, alors on peut voir que même si tous les autres clusters sont des singletons, P ne peut pas contenir k clusters ou plus.

  - Si $m' \geq m+1$, alors deux cas peuvent apparaître. Tout d'abord, s'il existe $i \in \{1, \cdots, m'\}$ tel que $w \in P_i$ (autrement dit, ω n'est pas dans un cluster contenant des sommets de V_s), alors puisque ω est adjacent à tous les sommets de V_s, la partition est forcément de coût au moins $m+1 = C+1$. Ainsi $\omega \notin P_i$ pour tout $i \in \{1, \cdots, m'\}$. Mais puisque ω est adjacent à tous les sommets de V_s, le degré du cluster contenant ω doit être au moins $m' > C$, ce qui est impossible.

Ainsi nous avons $m' = m$, et par un argument de comptage, ceci implique que pour tout $i \in \{(m+1), \cdots, k\}$, P_i est un singleton. Il est alors facile de voir que V_s est partitionné en m clusters, correspondant à une m-partition $S_1, \cdots, S_m$ de S telle que pour tout $i \in \{1, \cdots, m\}$ on a $\sum_{s \in S_i} w(s) \leq m - 1$, autrement dit, nous avons une solution pour 3-Partition comme désiré.

□

Puisque les arbres sont de tree-width 1, nous avons le corollaire suivant :

Corollaire 5. Bounded Degree k-Compaction *n'est pas* $\mathcal{XP}$ *paramétré par la tree-width du graphe d'entrée, sauf si* $\mathcal{P} = \mathcal{NP}$.

3.3 Algorithmes d'approximation

3.3.1 Analyse de l'algorithme glouton pour la version pondérée

Présentation de l'algorithme glouton

Un algorithme glouton pour résoudre le problème Weighted Sparsest k-Compaction est de supprimer des arêtes jusqu'à ce que le nombre de composantes connexes (qui représenteront les différentes parties de la partition) atteigne k. Étant donné que le coût d'une solution dépend du poids de ces arêtes retirées, il est naturel de les traiter par ordre de poids croissant. Il est facile de voir qu'un tel algorithme, formellement présenté par l'Algorithme 2, s'exécute en temps polynomial, plus précisément en temps $\mathcal{O}(|E| \log |E|)$.

Algorithme 2 Un algorithme glouton pour Weighted Sparsest k-Compaction

```
trier E dans l'ordre croissant des poids.
j ← 1
pour i = 1 à k − 1 faire
  tant que G a i composantes connexes faire
    G ← G\{e_j}
    j ← j + 1
  fin tant que //soit w_i le poids de la dernière arête retirée.
fin pour
retourner les composantes connexes de G.
```

Analyse de l'algorithme

Notations. Soient $G = (V, E)$ et $k \leq |V|$ une instance de notre problème. Posons $\theta = \max\{\frac{w(e)}{w(e')} : e, e' \in E, e \neq e', w(e') \geq w(e)\}$, qui est défini tel que pour tout

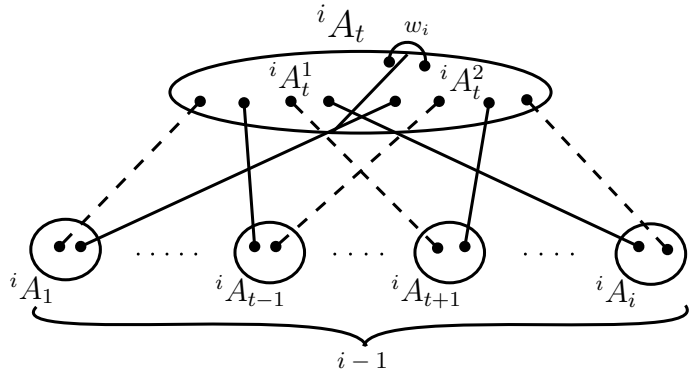


FIGURE 12 – Les pointillés représentent les arêtes de poids maximum entre ${}^iA_t$ et les autres clusters, déjà présents dans C_A. Les traits pleins représentent les au plus $(i-1)$ nouvelles arêtes (arêtes ajoutées) C_A après la séparation de ${}^iA_t$

$e, e' \in E$ tels que $w(e) \leq w(e')$, on a en fait $w(e) \leq \theta w(e')$. À une solution $S = \{S_1, \cdots, S_k\}$ du problème, on associe l'ensemble $C_S = \{c_1, \cdots, c_{p_s}\}$ des arêtes de poids maximum entre chaque paire de clusters, avec $p_s \leq \binom{k}{2}$. Ainsi, le coût de la solution S est défini par $cost(S) = \sum_{i=1}^{p_s} w(c_i)$.

Soit $A = \{A_1, \cdots, A_k\}$ la solution retournée par l'algorithme, et $\{{}^iA_a, \cdots, {}^iA_i\}$ la solution partielle au début de l'étape i de l'algorithme. La boucle « tant que » consiste à séparer un cluster ${}^iA_t$ pour un certain $t \in \{1, \cdots, i\}$ entre deux clusters ${}^iA_t^1$ et ${}^iA_t^2$. Ainsi, en séparant ce cluster, on ajoute à C_A plusieurs arêtes : tout d'abord l'arête que l'on vient de retirer, celle de poids maximum entre ${}^iA_t^1$ et ${}^iA_t^2$, mais aussi au plus $(i-1)$ arêtes (que l'on appellera *arêtes supplémentaires*) entre ${}^iA_t^1$ ou ${}^iA_t^2$ et les autres clusters (*c.f.* Figure 12. Toutes ces arêtes ajoutent donc à la valeur de la solution le terme w_i (coût de l'arête de poids maximum entre ${}^iA_t^1$ et ${}^iA_t^2$), et $(i-1)$ termes $(\alpha_i^j)_{j=1,\cdots,(i-1)}$: pour $j \in \{1, \cdots, (i-1)\}$, si l'arête de poids maximum entre ${}^iA_t$ et ${}^iA_j$ avait son extrémité dans ${}^iA_t^1$ (resp. dans ${}^iA_t^2$), alors α_i^j est égal à l'arête de poids maximum entre ${}^iA_t$ et ${}^iA_t^1$ (resp. dans ${}^iA_t^2$), ou à 0 si les deux clusters n'étaient en fait pas adjacents. Par définition, on a :

$$cost(A) = \sum_{i=1}^{k-1} (w_i + \sum_{j=1}^{i-1} \alpha_i^j) \tag{3.1}$$

Préliminaires. Afin de prouver le résultat d'approximation (Théorème 19), certaines propriétés sont d'abord nécessaires. Tout d'abord, étant donné que les arêtes sont retirées dans un ordre croissant des poids, il est clair que $w_1 \leq w_2 \leq \cdots \leq w_{k-1}$. En outre, on a le lemme suivant :

Lemme 1. *Considérons le début de l'étape i, ainsi que la partition correspondante $\{{}^iA_1, \cdots, {}^iA_i\}$. Pour tout $t \in \{1, \cdots, i\}$, il est possible de majorer supérieurement la somme des poids des arêtes les plus lourdes sortant de ${}^iA_t$ de la manière suivante :*

$$\sum_{\substack{j=1 \\ j \neq t}}^{i} w(e_{t,j}) \leq \sum_{j=1}^{i-1} w_j,$$

où e_t,j représente l'arête de poids maximum entre ${}^iA_t$ et ${}^iA_j$.

Preuve. La preuve est par récurrence sur i. On peut facilement vérifier que le théorème est vrai pour les premières étapes : pour $i = 1$ le résultat est trivial car il n'y a qu'un seul cluster, et pour $i = 2$ il n'y a qu'une arête de poids maximum entre les deux clusters. Considérons donc le début de l'étape $i+1$: lors de l'étape d'avant, ${}^iA_t$ a été séparé en ${}^iA_t^1$ et ${}^iA_t^2$, ajoutant ainsi w_i au coût de la solution. Pour $j_0 \neq t$, remarquons que l'arête $e_{j_0,t}$ (l'arête entre ${}^iA_{j_0}$ et ${}^iA_t$ avant la séparation) est maintenant remplacée par deux arêtes e_{j_0,t_1} et e_{j_0,t_2}, avec $\max(w(e_{j_0,t_1}), w(e_{j_0,t_2})) = w(e_{j_0,t})$. Sans perdre de généralité, supposons que $\max(w(e_{j_0,t_1}), w(e_{j_0,t_2})) = w(e_{j_0,t}) = w(e_{j_0,t_1})$ et notons ${}^iS_{j_0}$ la somme de toutes les arêtes de poids maximum entre ${}^iA_{j_0}$ et tous les autres clusters (y compris ${}^iA_t^1$ et ${}^iA_t^2$). Nous obtenons :

$$\begin{aligned} {}^iS_{j_0} &= \sum_{\substack{j=1 \\ j \neq j_0, j \neq t}}^{i} w(e_{j_0,j}) + \underbrace{w(e_{j_0,t_1})}_{w(e_{j_0,t})} + \underbrace{w(e_{j_0,t_2})}_{\leq w_i} \\ &\leq \sum_{j=1}^{i-1} w_j + w_i \quad \text{en utilisant l'hypothèse de récurrence.} \end{aligned}$$

Les mêmes arguments s'appliquent pour ${}^iA_t^1$ et ${}^iA_t^2$, ce qui termine la preuve. □

Corollaire 6. *Considérons le début de l'étape i, et la partition correspondante $\{{}^iA_j, \cdots, {}^iA_i\}$. Lors de la séparation de ${}^iA_t$, le poids total des arêtes supplémentaires est majoré de la manière suivante : $\sum_{j=1}^{i-1} \alpha_i^j \leq \theta \sum_{j=1}^{i-1} w_j$.*

Preuve. En ré-utilisant la notation $e_{j,t}$ du Lemme 1, soit $\tilde{e}_{j,t}$ (avec $j \neq t$) l'arête supplémentaire entre ${}^iA_j$ et ${}^iA_t$. Par exemple, si $e_{j,t}$ était en fait une arête entre ${}^iA_j$ et ${}^iA_t^1$, alors $e_{j,t}$ désigne l'arête entre ${}^iA_j$ et ${}^iA_t^2$. Par définition de θ, on a $w(\tilde{e}_{j,t}) \leq \theta w(e_{j,t})$ et donc $\sum_{j=1}^{i-1} \alpha_i^j = \sum_{j=1, j\neq t}^{i} w(\tilde{e}_{j,t}) \leq \theta \sum_{j=1}^{i-1} w_j$ (*c.f.* Lemme 1). □

Le dernier argument nécessaire à la preuve du Théorème 19 est une borne inférieure de la valeur de toute solution optimale :

Lemme 2. *Soit $i < |V|$, et S une $(i+1)$-partition de V, avec $C_S = \{c_1, \cdots, c_{p_s}\}$. On a toujours $\sum_{j=1}^{p_s} w(c_j) \geq \sum_{j=1}^{i} w_j$.*

Preuve. La preuve est par récurrence sur i. Le résultat est vrai pour $i = 1$, car l'algorithme donne clairement une 2-partition optimale. Supposons maintenant une $(i+1)$-partition S pour un $i \geq 2$, avec $C_S = \{c_1, \cdots, c_{p_s}\}$. Posons $w_M = \max_{j=1,\cdots,p_s} w(c_j)$, et soit (S_{i_1}, S_{i_2}) deux ensembles de S contenant chacun une extrémité d'une arête de poids w_M. En considérant la i-partition obtenue en fusionnant S_{i_1} et S_{i_2}, et en utilisant l'hypothèse de récurrence, on a $\sum_{j=1}^{p_s} w(c_j) - w_M \geq \sum_{j=1}^{i-1} w_j$. Enfin, par construction, toute $(i+1)$-partition génère forcément une arête de poids au moins w_i, puisque par définition de l'algorithme après la suppression de toutes les arêtes de poids strictement inférieurs à w_i, nous n'obtenons pas de $(i+1)$-partition. Nous obtenons ainsi $w_M \geq w_i$, ce qui termine la preuve. □

Preuve du ratio d'approximation et discussion. Nous sommes maintenant prêts à montrer le résultat suivant :

Théorème 19. *L'algorithme 2 est une* $(1 + (\frac{k}{2} - 1)\theta)$*-approximation pour* Weighted Sparsest k-Compaction.

Preuve. En utilisant le Lemme 2 avec une solution optimale, il est suffisant de montrer l'inégalité suivante :

$$cost(A) \leq (1 + (\frac{k}{2} - 1)\theta) \sum_{i=1}^{k-1} w_i \tag{3.2}$$

Montrons-la par récurrence sur k. Ici également l'inégalité est clairement vraie pour $k = 2$. Supposons qu'elle soit vraie pour tout $k = 1, \cdots, t$, et montrons la pour $k = t + 1$. Par l'équation 3.1 et l'hypothèse de récurrence, on a :

$$\begin{aligned}
cost(A) &\leq (1 + (\frac{t}{2} - 1)\theta) \sum_{i=1}^{t-1} w_i + w_t + \sum_{j=1}^{t-1} \alpha_t^j \\
&= (1 + (\frac{t}{2} - 1)\theta) \sum_{i=1}^{t-1} w_i + w_t + \frac{1}{2} \sum_{j=1}^{t-1} \alpha_t^j + \frac{1}{2} \sum_{j=1}^{t-1} \alpha_t^j \\
&\leq (1 + (\frac{t}{2} - 1)\theta) \sum_{i=1}^{t-1} w_i + w_t + \frac{1}{2}\theta \sum_{j=1}^{t-1} w_j + \frac{1}{2} \sum_{j=1}^{t-1} \alpha_t^j \quad \text{en utilisant le Corollaire 6} \\
&\leq (1 + (\frac{t}{2} - 1)\theta) \sum_{i=1}^{t-1} w_i + w_t + \frac{1}{2}\theta \sum_{j=1}^{t-1} w_j + \frac{1}{2}(t-1)\theta w_t \quad \text{car } \alpha_t^j \leq \theta w_t \\
&\leq (1 + (\frac{t+1}{2} - 1)\theta) \sum_{i=1}^{t-1} w_i + w_t + (\frac{t+1}{2} - 1)\theta w_t \\
&\leq (1 + (\frac{t+1}{2} - 1)\theta) \sum_{i=1}^{t} w_i
\end{aligned}$$

Ce qui termine la preuve. □

En observant le ratio d'approximation ainsi obtenu, on peut remarquer que les solutions deviennent meilleures lorsque θ tend vers 0, ce qui arrive lorsque l'écart de poids entre toute paire d'arêtes devient grand. Ceci n'est pas très surprenant, étant donné que l'algorithme ne se base que sur les poids des arêtes, et non pas sur la structure du graphe. Remarquons tout de même qu'il est possible d'adapter la réduction du Théorème 17 afin de montrer que Weighted Sparsest k-Compaction reste $\mathcal{NP}$-difficile dans le cas où tous les poids sont différents (en donnant par exemple à l'arête i un poids de $1 + \epsilon i$ pour un certain $\epsilon > 0$).
En fait, on peut remarquer que le facteur $\frac{k}{2}$ du ratio d'approximation provient principalement du nombre d'arêtes de la solution. En effet, dans le pire des cas (où chaque arête a un poids de 1), la solution de l'algorithme peut former une clique de taille k entre tous les clusters, alors qu'une solution optimale est un arbre, donnant un ratio de $\binom{k}{2}/(k-1) = \frac{k}{2}$. Nous nous servons de cette idée pour montrer que le ratio montré précédemment est atteint :

Proposition 1. *Le ratio d'approximation de l'Algorithme 2 est atteint.*

Preuve. L'instance consiste en un split graphe $G = (C \cup S, E)$, où $C = \{c_1, \cdots, c_k\}$ induit une clique et où $S = \{s_1, \cdots, s_k\}$ induit un ensemble indépendant. La construction est résumée Figure 13. Nous définissons trois catégories d'arêtes :

- $X = \{\{c_i, s_j\}$ tels que $i \neq j$ ou $j = 1\}$
- $Y = \{\{c_i, c_j\}$ tels que $i \neq j\}$
- $Z = \{\{c_i, s_j\}$ tels que $i = j$ et $j \neq 1\}$

Les poids sont définis comme suit : pour tout $e \in Z$, on pose $w(e) = 1$; pour tout $e \in Y$, on pose $w(e) = 1 + \epsilon$, et pour tout $e \in X$, on pose $w(e) = 1 + 2\epsilon$, avec $\epsilon > 0$. Il est clair que l'algorithme va commencer par supprimer les arêtes de X, puis celles de Y. À ce moment, une $(k+1)$-partition est créée, et son coût est d'au moins $\frac{k(k+1)}{2}$.
Puis, considérons la $(k+1)$-partition $\{V_1, \cdots, V_k, V_{k+1}\}$ suivante : pour tout $j \in \{1, \cdots, k\}$, $V_j = \{s_j\}$, et $V_{k+1} = C$. Le coût de cette partition est de $k + 2\epsilon(k-1)$. Soient $\mathcal{A}$ et $\mathcal{OPT}$ les coûts respectifs de la solution donnée par l'algorithme et d'une solution optimale pour cette instance. Par ce qui précède, on a $\mathcal{A} \geq \frac{k(k+1)}{2}$ et $\mathcal{OPT} \leq k + 2\epsilon(k-1)$. Lorsque $\epsilon \longrightarrow 0$, nous avons d'une part $\theta \longrightarrow 1$, et

$$\frac{\mathcal{A}}{\mathcal{OPT}} \geq \frac{k(k+1)}{2k + 4\epsilon(k-1)} \longrightarrow \frac{k+1}{2}$$

ce qui prouve le résultat (rappelons que nous sommes dans le cas d'une $(k+1)$-partition).

□

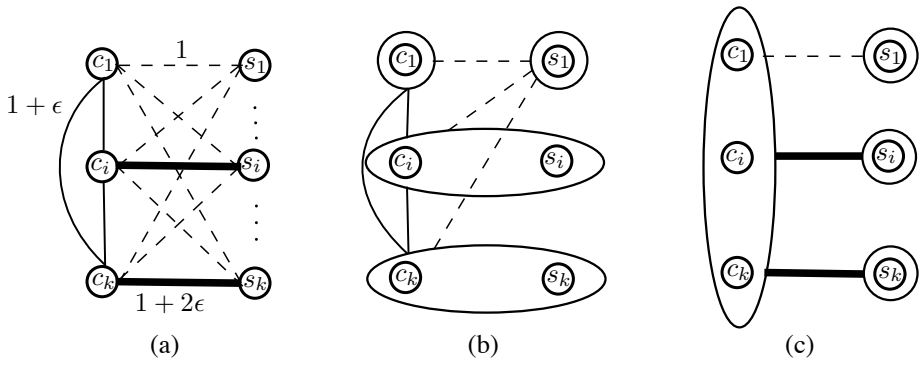


FIGURE 13 – (a) : Exemple d'une instance où le ratio est atteint. Les arêtes de X sont représentées par des tirets, celles de Y par des traits pleins, et celles de Z par des traits en gras. (b) : solution donnée par l'Algorithme 2. (c) : une solution de coût $k + 2\epsilon(k-1)$.

3.3.2 Utilisation du diamètre du graphe

Dans cette partie, nous nous intéressons à l'influence du diamètre du graphe dans la résolution de SPARSEST k-COMPACTION et sa version pondérée. Nous montrons que le problème est $\mathcal{NP}$-difficile pour les graphes de diamètre deux et trois, polynomial pour les graphes de diamètre $d \geq k-1$, et que la version pondérée est $\mathcal{NP}$-difficile pour les graphes de diamètre d fixé pour tout $d \in \{4, \cdots, k\}$. Enfin, nous présentons un algorithme polynomial pour la version non pondérée donnant des solutions $(\frac{k}{2} - d + 2 + \frac{d^2-3d}{2(k-1)})$-approchées où d est le diamètre du graphe.

Complexité pour les valeurs extrémales du diamètre

Le premier théorème résume les différents résultats négatifs :

Théorème 20. SPARSEST k-COMPACTION *est $\mathcal{NP}$-difficile pour les graphes de diamètre deux et trois.* WEIGHTED SPARSEST k-COMPACTION *est $\mathcal{NP}$-difficile pour les graphes de diamètre d pour tout $d \in \{4, \cdots, k\}$ fixé.*

Preuve. Notons tout d'abord que dans la réduction du Théorème 17, le graphe obtenu est de diamètre deux (car contenant un sommet universel), donc le premier point est immédiat.
Nous adaptons maintenant cette même réduction afin d'obtenir le résultat pour la version pondérée pour les graphes de diamètre $d \in \{4, \cdots, k\}$ fixé, et montrons que ce résultat tient lorsqu'il n'y a pas de poids pour $d = 3$. Nous réduisons donc depuis INDEPENDENT SET : Soient $G = (V, E)$, $k \leq |V|$ et $k \geq \delta \geq 3$. Nous construisons G', composé de l'union disjointe de δ copies de $G : G_1, \cdots, G_\delta$, où pour tout $i = 1, \cdots, \delta$ on a $G_i = (V_i, E_i)$, avec $V_i = \{v_1^i, \cdots, v_n^i\}$, et on pose $w(e) = 1$ pour tout $e \in \bigcup_{i=1}^{\delta} E_i$. On ajoute également pour tout $i = 1, \cdots, \delta$ un sommet α_i

connecté à tous les sommets de V_i. Enfin, pour tout $i = 1, \cdots, (\delta-1)$, avec des arêtes de poids $M > k$, on connecte α_i à α_{i+1}, et pour tout $j \in \{1, \cdots, k\}$, on connecte v_j^i à v_j^i. Il est clair que cette réduction peut se réaliser en temps polynomial, et que le diamètre du graphe obtenu est $d = \delta + 1$. Nous montrons maintenant que G contient un ensemble indépendant de taille k si et seulement s'il existe une $(k+1)$-partition de coût k dans G'.

Supposons qu'il existe un ensemble indépendant S de taille k dans G, et sans perdre de généralité supposons que $S = \{v_1, \cdots, v_k\}$. Alors nous construisons la $(k+1)$-partition $(P_1, \cdots, P_{k+1})$ suivante :

- $P_i = \{v_i^j : j \in \{1, \cdots, \delta\}\}$ pour tout $i \in \{1, \cdots, k\}$
- $P_{k+1} = V' \setminus \left(\bigcup_{i=1}^{k} P_i\right)$

Par construction, et comme S est un ensemble indépendant, il est facile de vérifier que cette partition est de coût k. En effet, les seules arêtes seront entre P_{k+1} et les autres clusters.

Réciproquement, étant données les arêtes de poids M, il est clair que pour tout $j \in \{1, \cdots, n\}$, les sommets $\{v_j^i : i \in \{1, \cdots, \delta\}$ doivent être dans le même cluster, de même que tous les sommets $\{\alpha_i : i \in \{1, \cdots, \delta\}$. Ainsi, si G' contient une $(k+1)$-partition de coût k, alors G a un ensemble indépendant de taille k.

Lorsque $\delta = 2$, nous obtenons un graphe de diamètre trois. On peut vérifier dans ce cas que la réduction fonctionne même si $M = 1$. On a ainsi que le problème non pondéré est $\mathcal{NP}$-difficile pour les graphes de diamètre trois. En revanche, les limites de cette construction sont les suivantes : l'équivalence ne fonctionne plus pour $\delta = 3$ si $M = 1$, car, étant donné qu'il n'y a pas d'arête entre G_1 et G_3, la structure de la solution ne peut plus être contrôlée. De plus, ajouter un graphe complet entre chaque graphe G_i ferait décroître le diamètre du graphe. □

Nous montrons maintenant que si le diamètre du graphe est trop grand, alors le problème devient polynomial :

Théorème 21. SPARSEST k-COMPACTION *est polynomial si* $d \geq k-1$*, où* d *est le diamètre du graphe.*

Preuve. Soit $G = (V, E)$ un graphe de diamètre $d \geq k-1$, et soit v_0 un sommet d'excentricité maximale. On définit alors la $(d+1)$-partition suivante :

$$\Gamma^i(v_0) = \begin{cases} \{v_0\} & \text{si} \quad i = 0 \\ N(v_0) & \text{si} \quad i = 1 \\ \{N(v) : v \in \Gamma^{i-1}(v_0)\} \backslash \Gamma^{i-2}(v_0) & \text{si} \quad 1 < i \leq d \end{cases}$$

On retourne alors la partition $P = \{\Gamma^0(v_0), \cdots, \Gamma^{k-2}, \bigcup_{i=k-1}^{d} \Gamma^i(v_0)\}$. Par construction on a bien une k-partition puisque $\gamma^i(v_0) \cap \Gamma^j(v_0) = \emptyset$ pour tout $i \neq j$. Elle est de coût minimum car son graphe quotient est un chemin (rappelons que nous ne nous intéressons qu'à des graphes connexes). □

Approximation dépendant du diamètre

Dans la sous-section précédente, nous nous sommes intéressés à la résolution du problème lorsque le diamètre du graphe d'entrée prenait des valeurs extrémales. Pour les valeurs intermédiaires maintenant, nous utilisons ce diamètre afin de développer un algorithme d'approximation dont le rapport d'approximation dépend de celui-ci.

Soient $G = (V, E)$ un graphe de diamètre $d < k - 1$ et $k \leq |V|$. L'algorithme consiste à construire la même partition que dans le cas où $d \geq k - 1$, en utilisant un sommet d'excentricité maximum, et puis à modifier cette partition afin d'obtenir exactement k clusters. Soit v_0 un sommet d'excentricité maximum, et considérons la $(d+1)$-partition $\{\Gamma^0(v_0), \cdots, \Gamma^d(v_0)\}$ comme définie dans le Théorème 21. Pour tout $i \in \{0, \cdots, d\}$, on pose $y_i = |\Gamma^i(v_0)|$. Puisque le graphe est de diamètre $d < k-1$, l'algorithme EXPAND-AND-SPLIT va casser chaque cluster $\Gamma^i(v_0)$ en x_i clusters différents, les x_i étant choisis arbitrairement de telle façon que $\sum_{i=0}^{d} x_i = k$. Un exemple d'une telle partition est décrite Figure 14.

Théorème 22. *L'algorithme* EXPAND-AND-SPLIT *donne une solution* $(\frac{k}{2} - d + 2 + \frac{d^2-3d}{2(k-1)})$*-approchée en temps polynomial.*

Preuve. Tout d'abord, il est clair que l'algorithme proposé s'exécute en temps polynomial. Notons $\mathcal{A}$ la valeur de la solution retournée par l'algorithme. On a :

$$
\begin{aligned}
2\mathcal{A} \;\leq\; & \overbrace{\sum_{i=0}^{d} x_i(x_i-1)}^{\text{arêtes entre les ensembles dans } \Gamma^i(v_0)} + \overbrace{\sum_{i=0}^{d-1} 2x_i x_{i+1}}^{\text{arêtes entre les ensembles de } \Gamma^i(v_0) \text{ et } \Gamma^{i+1}(v_0)} \\
=\; & \sum_{i=0}^{d} x_i^2 - \sum_{i=0}^{d} x_i + \sum_{i=0}^{d-1} 2x_i x_{i+1} \\
=\; & \sum_{i=0}^{d} x_i^2 - k + \sum_{i=0}^{d} 2x_i x_{i+1} - 2x_0 x_d \\
=\; & \left(\sum_{i=0}^{d} x_i\right)^2 - \sum_{\substack{0\leq i,j\leq d \\ |i-j|\geq 2}} x_i x_j - k - 2x_d \\
\leq\; & \left(\sum_{i=0}^{d} x_i\right)^2 - \sum_{\substack{0\leq i,j\leq d \\ |i-j|\geq 2}} x_i x_j - k \\
=\; & k^2 - k - \Delta_d \qquad \text{avec } \Delta_d = \sum_{\substack{0\leq i,j\leq d \\ |i-j|\geq 2}} x_i x_j \qquad (3.3)
\end{aligned}
$$

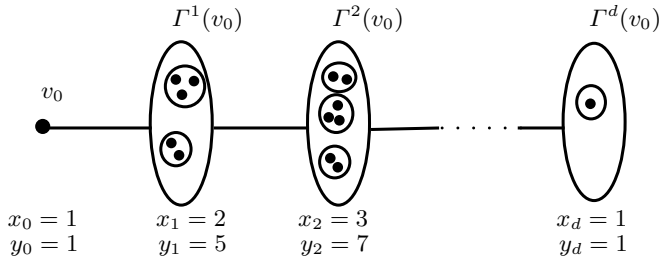


FIGURE 14 – Illustration de l'algorithme EXPAND-AND-SPLIT

Lemme 3. *Soit* $(x_0, \cdots, x_d) \in \mathbb{N}^{d+1}$ *tels que* $\sum_{i=0}^{d} x_i = k$ *et* $x_i > 0$ *pour tout* $i \in \{0, \cdots, d\}$. *Alors* Δ_d *atteint une valeur minimum s'il existe* $i_0 \in \{1, \cdots, d-1\}$ *tel que* $x_{i_0} = k - d$, *et* $x_i = 1$ *pour tout* $i \neq i_0$.

Preuve. Étant donné $d > 0$, le but est de trouver $(x_0, \cdots, x_d)$ avec $x_i \in \mathbb{N}^*$ pour tout $i \in \{1, \cdots, d\}$ tel que

$$z = \sum_{\substack{0 \leq i,j \leq d \\ i-j \geq 2}} x_i x_j$$

est minimum, sous les contraintes :

- $\sum_{i=0}^{d} x_i = k$
- $x_i > 0 \ \forall i \in \{0, ..., d\}$

Remarquons que $2z = \Delta_d$ (à cause de la somme sur i et j tels que $j - i \geq 2$, au lieu de $|j - i| \geq 2$). Notons z^* la valeur d'une solution optimale. Si $d = 1$, alors le résultat est trivial puisque $z^* = 0$. Si $d = 2$, alors $z^* = 1$ pour $x_0 = 1$, $x_1 = k - 2$ et $x_2 = 1$. Nous considérons donc dans la suite le cas $d > 2$.
Dans un soucis de clarté, on définit $x_{-1} = x_{d+1} = 0$. Remarquons que

$$\begin{aligned} z = \sum_{j-i \geq 2} x_i x_j &= \left(\sum_{i,j=0}^{d} x_i x_j \right) - \left(\sum_{i=0}^{d} x_i (x_{i-1} + x_i + x_{i+1}) \right) \\ &= k^2 - \sum_{i=0}^{d} x_i (x_{i-1} + x_i + x_{i+1}) \end{aligned}$$

Et donc minimiser z revient finalement à maximiser

$$t = \sum_{i=0}^{d} x_i (x_{i-1} + x_i + x_{i+1})$$

sous les mêmes contraintes. Ici aussi notons t^* la valeur d'une solution optimale. Pour tout $i \in \{1, \cdots, d\}$, on définit $\alpha_i = x_{i-1} + x_i + x_{i+1}$. On a ainsi $t = \sum_{i=0}^{d} x_i \alpha_i$. Le reste de la preuve est divisée en trois étapes :

- Étape 1 : il existe une solution optimale telle que $x_0 = x_d = 1$.
- Étape 2 : il existe une solution optimale telle que :
  - $\exists j \in \{1, \cdots, d-1\}$ tel que $\alpha_j = k - d + 2$.
  - $\forall i \in \{1, \cdots, d\}$ avec $|j - i| \geq 2$, on a $x_i = 1$.
- Étape 3 : il existe une solution optimale telle que :
  - $\exists j \in \{1, \cdots, d-1\}$ tel que $x_j = k - d$.
  - $\forall i \neq j$ on a $x_i = 1$.
- Preuve de l'étape 1 : Soit $(x_0, \cdots, x_d)$ une solution optimale. Sans perdre de généralité, supposons par contradiction que $x_0 > 1$, et définissons la solution $(x'_0, \cdots, x'_d)$ suivante :
  - $x'_0 = x_0 - 1$,
  - $x'_1 = x_1 + 1$,
  - $x'_i = x_i$ pour tout $i \in \{1, \cdots, d-1\}$.

  Avec $\alpha'_i = x'_{i-1} + x'_i + x'_{i+1}$ pour tout $i \in \{0, \cdots, d\}$. On a :

$$\begin{aligned} \sum_{i=0}^{d} x'_i \alpha'_i &= (x_0 - 1)\alpha_0 + (x_1 + 1)\alpha_1 + x_2\alpha_2 + \sum_{i=3}^{d} x_i\alpha_i \\ &= \sum_{i=0}^{d} x_i\alpha_i + x_2 \end{aligned}$$

  Puisque $x_2 > 0$, $(x_0, \cdots, x_d)$ n'est pas une solution optimale, une contradiction.

- Preuve de l'étape 2 : soit $(x_0, \cdots, x_d)$ une solution optimale avec $x_0 = x_d = 1$, et soit $j \in \{0, \cdots, d\}$ tel que $\alpha_j = \max_{k=2}^{d} \alpha_k$ (remarquons que $j \neq 0$ et $j \neq d$). Par contradiction, supposons qu'il existe $i \notin \{j-1, j, j+1\}$ tel que $x_i > 1$. Définissons alors la solution suivante $(x'_0, \cdots, x'_d)$:
  - $x'_i = x_i - 1$
  - $x'_j = x_j + 1$
  - $\forall k \in \{i, j\}$ $x'_k = x_k$

Avec $\alpha'_i = x'_{i-1} + x'_i + x'_{i+1}$ pour tout $i \in \{0, ..., d\}$. On distingue alors deux cas :
Premier cas : $i+1 = j-1$ (le cas $i-1 = j+1$ étant symétrique). On a alors :

$$\sum_{k=0}^{d} x'_k \alpha'_k = \sum_{k=0}^{d} x_k \alpha_k + 2$$

Second cas : $i \notin \{j-2, j-1, j, j+1, j+2\}$. On a alors :

$$\sum_{k=0}^{d} x'_k \alpha'_k = \sum_{k=0}^{d} x_k \alpha_k + 2\alpha_j - 2\alpha_i + 2$$

Avec $2\alpha_j - 2\alpha_i + 2 > 0$. Ainsi, dans les deux cas, $(x_0, \cdots, x_d)$ n'est pas une solution optimale : une contradiction.

- Preuve de l'étape 3 : soit $(x_0, \cdots, x_d)$ une solution optimale avec :
  - $x_{j-1} + x_j + x_{j+1} = k - d + 2$ pour un certain $j \in \{1, \cdots, d-1\}$,
  - $x_i = 1$ pour tout $i \notin \{j-1, j, j+1\}$.

  Et supposons que $x_{j-1} > 1$. Définissons alors la solution $(x'_0, \cdots, x'_d)$ suivante :
  - $x'_{j-1} = x_{j-1} - 1$,
  - $x_j = x_j + 1$,
  - pour tout $k \notin \{j-1, j\}$, $x'_k = x_k$.

  Avec $\alpha'_i = x'_{i-1} + x'_i + x'_{i+1}$ pour tout $i \in \{0, \cdots, d\}$. On a alors :

$$\sum_{k=0}^{d} x'_k \alpha'_k = \sum_{k=0}^{d} x_k \alpha_k - \alpha_{j-1} + \alpha_j + x_{j+1} - x_{j-2}$$

  Et puisque $-\alpha_{j-1} + \alpha_j + x_{j+1} - x_{j-2} \geq 0$, $(x'_0, \cdots, x'_d)$ est également une solution optimale, ce qui termine la preuve du Lemme 3.

□

Par le lemme qui précède, on a :

$$\Delta_d \geq 2\left(k(d-2) - \frac{d(d-1)}{2} + 2\right) \tag{3.4}$$

Et en utilisant les équations (3.3) et (3.4), on obtient :

$$\begin{aligned} 2\mathcal{A} &\leq k^2 - k - 2k(d-2) + d(d-1) - 4 \\ &\leq k^2 - 2kd + 3k + d^2 - d - 4 \end{aligned}$$

Soit $\mathcal{OPT}$ la valeur d'une solution optimale au problème Sparsest k-Compaction. Étant donné que $\mathcal{OPT} \geq k-1$ (puisque le graphe d'entrée est connexe), on a :

$$\begin{aligned} \rho = \frac{2\mathcal{A}}{2\mathcal{OPT}} &\leq \frac{k^2 - 2kd + 3k + d^2 - d - 4}{2(k-1)} \\ &\leq \frac{k}{2} - d + 2 + \frac{d^2 - 3d}{2(k-1)} \end{aligned}$$

Ce qui prouve le résultat. □

3.4 Partitions déséquilibrées, lien avec Sparsest k-Subgraph

Dans cette section, nous montrons comment des propriétés structurelles de solutions simples, que nous appelons *partitions déséquilibrées*, peuvent permettre de développer des algorithmes d'approximation pour Sparsest k-Compaction, tout en exhibant un lien avec le problème Sparsest k-Subgraph. Plus précisément, nous montrons tout d'abord que toute solution P à Sparsest k-Compaction peut être re-structurée en une solution P' telle que $cost(P') \leq (2 - \frac{2}{k})cost(P)$. On relie ensuite la structure d'une telle solution vers le problème Sparsest k-Subgraph. En combinant ces deux remarques, on montre qu'un algorithme ρ-approché pour Sparsest k-Subgraph implique un algorithme $(\rho + 1 - \frac{2\rho}{k})$-approché pour Sparsest k-Compaction. Enfin, nous analysons l'espérance de la valeur de l'algorithme construisant une solution déséquilibrée randomisée. Nous dé-randomisons ensuite ce dernier afin d'obtenir un algorithme déterministe polynomial $(1 + \frac{m(k-2)}{n(n-1)})$-approché. Comme on peut le soupçonner, cet algorithme donne de bons résultats lorsque le graphe d'entrée a une faible densité.

3.4.1 Définition et lien avec Sparsest k-Subgraph

Définition 32. *Étant donné $G = (V, E)$ et $k \in \mathbb{N}$, une k-partition de V est dite déséquilibrée si $(k-1)$ clusters sont des singletons.*

Lemme 4. *Soient $G = (V, E)$ et $k \in \mathbb{N}$. Soit P une solution à* Sparsest k-Compaction *pour (G, k). Alors on peut construire en temps polynomial une solution déséquilibrée P' telle que $cost(P') \leq (2 - \frac{2}{k})cost(P)$.*

Preuve. Soient G, k et P tels que définis dans l'énoncé, et notons $P = \{V_1, \cdots, V_k\}$. Sans perdre de généralité, supposons que V_k soit un sommet de degré maximum dans G_P. Pour tout $i \in \{1, \cdots, (k-1)\}$, choisissons x_i un sommet arbitraire de V_i, et construisons la partition déséquilibrée $P' = \{V'_1, \cdots, V'_k\}$ suivante :

- $V'_i = \{x_i\}$ pour tout $i \in \{1, \cdots, (k-1)\}$,

- $V'_k = V \setminus \left(\bigcup_{i=1}^{k-1} V_i\right)$.

Soit q le nombre d'arêtes dans G_P entre V_k et les autres sommets, et q' le nombre d'arêtes dans G_P entre V'_k et les autres sommets. On a alors :

$$cost(P') \leq cost(P) + q' - q$$

Et par définition $q' \leq k-1 \leq cost(P)$ et $kq \geq 2cost(P)$ (puisque q est égal au degré maximum de G_P, qui est un graphe à k sommets et $cost(P)$ arêtes. On a donc :

$$\begin{aligned} cost(P') &\leq cost(P) + cost(P) - \frac{2 \cdot cost(P)}{k} \\ &= (2 - \frac{2}{k})cost(P) \end{aligned}$$

Ce qui termine la preuve. □

Ainsi, d'après le lemme précédent, une stratégie immédiate pour obtenir une solution $(2 - \frac{2}{k})$-approchée est d'énumérer tous les $(k-1)$-uplets de sommets. En fait, nous pouvons facilement voir qu'il suffit de trouver $(k-1)$ sommets induisant le nombre minimum d'arêtes dans G. Ce problème porte le nom de Sparsest k-Subgraph, et sera plus longuement étudié dans le Chapitre 4 suivant. Dans ce qui suit, nous généralisons le lemme précédent aux solutions approchées se transférant de Sparsest k-Subgraph à Sparsest k-Compaction.

Lemme 5. *S'il existe une solution ρ-approchée pour* Sparsest $(k-1)$-Subgraph, *alors il existe une solution $(\rho + 1 - \frac{2\rho}{k})$-approchée pour* Sparsest k-Compaction.

Preuve. Soient $G = (V, E)$ et $k \in \mathbb{N}$. Soit $S = \{x_1, \cdots, x_{k-1}\}$ une solution ρ-approchée pour Sparsest $(k-1)$-Subgraph, et soit $P^* = \{V_1, \cdots, V_k^*\}$ une solution optimale pour Sparsest k-Compaction à (G, k). Comme dans le lemme précédent on suppose sans perdre de généralité que V_k^* est un sommet de degré maximum dans G_{P^*}. On construit la partition $P = \{V_1, \cdots, V_k\}$ suivante :

- $V_i = \{x_i\}$ pour tout $i \in \{1, \cdots, (k-1)\}$
- $V_k = V \setminus \left(\bigcup_{i=1}^{k-1} V_i\right)$

Soit q (resp. q^*) le nombre d'arêtes dans G_P (resp. G_{P^*}) entre V_k (resp. V_k^*) et les autres sommets, et soit α (resp. α^*) le nombre d'arêtes dans G_P (resp. G_{P^*}) du sous-graphe induit par $\{V_1, \cdots, V_{k-1}\}$ (resp. $V_1^*, \cdots, V_{k-1}^*$). Par définition, on a $cost(P) = q + \alpha$ et $cost(P^*) = q^* + \alpha^*$. Comme S est une solution ρ-approchée, on a $\alpha \leq \rho\alpha^*$. Ainsi :

$$\begin{aligned} cost(P) = q + \alpha + q^* - q^* &\leq q + \rho\alpha^* + q^* - q \\ &= q + \rho(\alpha^* + q^*) - \rho q^* \\ &= q + \rho cost(P^*) - \rho q^* \end{aligned}$$

Et, puisque $q \leq k-1 \leq cost(P^*)$ et $kq^* \geq 2cost(P^*)$ (comme précédemment), on a :

$$cost(P) \leq cost(P^*) \cdot (\rho + 1 - \frac{2\rho}{k})$$

Ce qui termine la preuve. □

3.4.2 Sur les graphes de faible densité

Dans cette partie, nous analysons l'algorithme construisant une solution déséquilibrée aléatoire. Plus précisément, l'algorithme choisit $(k-1)$ sommets de manière aléatoire[1] $\{X_1, \cdots, X_{k-1}\}$, les place dans des singletons, et construit un dernier cluster avec le reste des sommets du graphe. On note $\mathcal{A}_{rand}$ la variable aléatoire correspondant à la valeur de la solution correspondante. Comme on peut s'y attendre, l'espérance de $\mathcal{A}_{rand}$ décroît plus la densité du graphe d'entrée est faible.

Lemme 6. $\mathbb{E}(\mathcal{A}_{rand}) \leq (k-1)(1 + \frac{m(k-2)}{n(n-1)})$.

Preuve. En effet, il est possible de borner inférieurement l'espérance de $\mathcal{A}_{rand}$ de la manière suivante :

$$\mathbb{E}(\mathcal{A}_{rand}) \leq k - 1 + \frac{1}{2} \sum_{i,j \in \{X_1,\ldots,X_{k-1}\}, i \neq j} \mathbb{E}(\mathbb{1}_{i,j})$$

Où $\mathbb{1}_{i,j}$ est égal à 1 s'il existe une arête entre le sommet i et le sommet j, et est égal à 0 autrement.
De plus, pour tout i et j on a $\mathbb{E}(\mathbb{1}_{i,j}) = m\frac{2}{n(n-1)}$. Ainsi, on a :

$$\mathbb{E}(\mathcal{A}_{rand}) \leq k - 1 + (k-1)(k-2)\frac{m}{n(n-1)}$$

comme désiré. □

On montre maintenant que ce rapport d'approximation peut en fait être obtenu de manière déterministique, en utilisant des arguments de dé-randomisation classiques.

Théorème 23. *Il existe un algorithme polynomial déterministe pour* Sparsest k-Compaction *avec un rapport d'approximation de* $\left(1 + \frac{m(k-2)}{n(n-1)}\right)$.

Preuve. Nous dé-randomisons l'algorithme précédent en utilisant la méthode des probabilités conditionnelles. Informellement, à chacune des $(k-1)$ étapes nous trouvons en temps polynomial le prochain sommet « le plus intéressant », autrement dit, le

[1] Prendre uniformément x sommets aléatoirement correspond à prendre de manière uniforme un élément parmi l'ensemble des ensembles de taille x.

sommet qui minimise notre borne inférieure sur la probabilité conditionnelle.
Considérons tout d'abord une instance $G = (V, E)$, $k \in \mathbb{N}$. Pour tout ensemble de $(k-1)$ sommets $\{x_1, \cdots, x_{k-1}\}$, on définit

$$f(\{x_1, \cdots, x_{k-1}\}) = k - 1 + \frac{1}{2} \sum_{i,j \in \{x_1, \cdots, x_{k-1}\}, i \neq j} \mathbb{1}_{i,j}$$

En fait, $f(\{x_1, \cdots, x_{k-1}\})$ correspond à la borne inférieure utilisée dans le Lemme 6 pour le coût d'une solution déséquilibrée où $\{x_1, \cdots, x_{k-1}\}$ sont des singletons. Ainsi, le Lemme 6 nous informe que lorsque l'on choisit uniformément les $\{X_1, \cdots, X_{k-1}\}$, on a

$$\mathbb{E}(f(\{x_1, \cdots, x_{k-1}\})) \leq b(n, m, k)$$

avec $b(n, m, k) = (k-1)\left(1 + \frac{m(k-2)}{n(n-1)}\right)$.
Revenons maintenant à la dé-randomisation. Pour tout $i \in \{1, \cdots, (k-2)\}$ et tout ensemble de i différents sommets $\{x_1, \cdots, x_i\}$, soit $X(\{x_1, \cdots, x_i\})$ la variable aléatoire suivante :

- On choisit aléatoirement (parmi $V \setminus \{x_1, \cdots, x_i\}$) $(k-1-i)$ différents sommets $(X_t)_{1 \leq t \leq k-1-i}$.
- On retourne $f(\{x_1, \cdots, x_i, X_1, \cdots, X_{k-1-i}\})$.

Ainsi, le principe suivant est suffisant pour prouver par induction sur i (pour $i \in \{1, \cdots, k-2\}$) qu'il est possible de trouver en temps polynomial $\{x_1, \cdots, x_i\}$ tel que $\mathbb{E}(X(x_1, \cdots, x_i)) \leq b(n, m, k)$. En effet ceci est clairement vrai pour $i = 0$ puisque $\mathbb{E}(X(\emptyset)) = \mathbb{E}(f(\{X_1, \cdots, X_{k-1}\})) \leq b(n, m, k)$. Puis considérons $i \geq 1$, et montrons comment choisir le $(i+1)^{ème}$ sommet. On a :

$$\mathbb{E}(X(x_1, \ldots, x_i)) = \frac{1}{n-i} \sum_{x_t \notin \{x_1, \ldots, x_i\}} \mathbb{E}(X(x_1, \ldots, x_i, x_t))$$

Et puisque $\mathbb{E}(X(x_1, \cdots, x_i)) \leq b(n, m, k)$, il doit forcément exister $x_t \notin \{x_1, \cdots, x_i\}$ tel que $\mathbb{E}(X(x_1, \cdots, x_i, x_t)) \leq b(n, m, k)$, on choisit donc x_{i+1} tel que :

$$x_{i+1} = \underset{x_t \notin \{x_1, \ldots, x_i\}}{\operatorname{argmin}} \left(\mathbb{E}(X(x_1, \ldots, x_i, x_t))\right)$$

Remarquons que pour tout x_t, $\mathbb{E}(X(x_1, \cdots, x_i, x_t))$ peut être calculé en temps polynomial en utilisant la définition de f et la linéarité de l'espérance (même si les X_t ne sont pas indépendants). □

3.5 Variantes et contraintes sur les instances

Dans cette section, nous étudions la complexité du problème Sparsest k-Compaction lorsque des contraintes sont imposées soit sur la solution (minoration de la taille des clusters par exemple) soit sur les instances (petites valeurs de k, ou classes de graphes particulières).
En particulier, dans l'optique de la résolution du problème pour tout k fixé (*c.f.* Problème ouvert 6 de la Section 3.6), nous montrons que le Sparsest k-Compaction est polynomial pour $k = 3$ et $k = 4$. Nous montrons également que le problème n'est pas plus facile lorsqu'une borne inférieure sur la taille des clusters est imposée. Enfin, nous étudions la complexité du problème dans les split graphes, en montrant qu'il est $\mathcal{NP}$-difficile et en obtenant des résultats positifs et négatifs de complexité paramétrée selon les paramètres considérés.

3.5.1 Petites valeurs de k

Tout d'abord, il est facile de voir que Sparsest k-Compaction est polynomial si $k = 3$. En effet, dans ce cas le graphe quotient de la solution est soit P_3, induisant deux arêtes, soit C_3, induisant trois arêtes. Or, un graphe admet une compaction vers P_3 si et seulement s'il est de diamètre deux, ce qui est détectable en temps polynomial. Si tel n'est pas le cas alors n'importe quelle autre partition engendrera C_3 en graphe quotient. Le résultat suivant montre que même dans le cas pondéré (*c.f.* Section 3.1.3), le problème reste polynomial :

Théorème 24. Weighted Sparsest k-Compaction *est polynomial pour* $k = 3$.

Preuve. Soient $G = (V, E)$ le graphe d'entrée, et $w : E \longrightarrow \mathbb{N}$ une pondération des arêtes. Le principe de l'algorithme est d'énumérer toutes les paires (ou triplets, selon si le graphe quotient a deux ou trois arêtes) d'arêtes afin de « deviner » les arêtes les plus lourdes entre les clusters d'une solution optimale (*i.e.* les arêtes qui seront prises en compte dans le graphe quotient). Ainsi, pour toute paire (ou triplets) d'arêtes, l'algorithme essaie de ranger les sommets restants dans les clusters, sans changer la valeur de la solution.
Nous considérons alors deux cas : celui où une solution optimale contient seulement deux arêtes entre les clusters (le graphe quotient étant alors un chemin), et celui où toute solution optimale contient trois arêtes (le graphe quotient étant alors un cycle). Soit (V_1, V_2, V_3) la partition que l'algorithme construit, et (V_1^*, V_2^*, V_3^*) une solution optimale.

- Premier cas : le graphe quotient de (V_1^*, V_2^*, V_3^*) contient deux arêtes. Supposons donc que nous disposons des deux arêtes e_a^* et e_b^* les plus lourdes entre les trois clusters (*c.f.* Figure 15a). Soit $a = w(e_a^*)$, $b = w(e_b^*)$, et notons $e_a^* = \{a_1, a_2\}$ et $e_b^* = \{b_1, b_2\}$. Sans perdre de généralité nous supposons que $a_i \in V_i^*$, $b_i \in V_{i+1}^*$ pour $i \in \{1, 2\}$, et $a \leq b$. La première étape est de

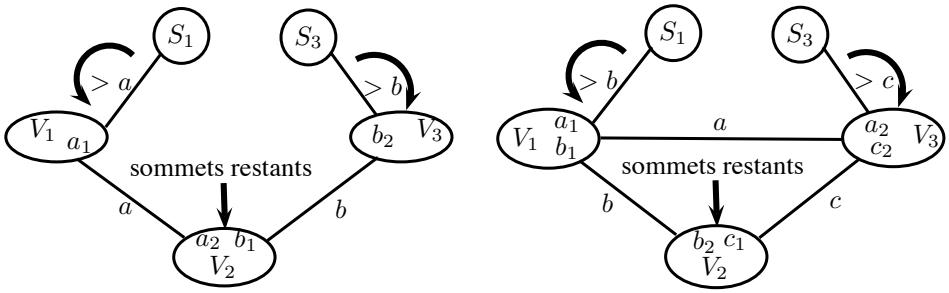


FIGURE 15 – Illustration de l'algorithme polynomial pour $k = 3$. Les flèches en gras représentent l'affectation des sommets dans les clusters. (15a) : Cas où une solution optimale contient deux arêtes. (15b) : Cas où toute solution optimale contient trois arêtes.

recopier la solution optimale, et donc d'ajouter a_1 dans V_1, a_2 et b_1 dans V_2, et b_2 dans V_3.
Soit S_1 (resp. S_3) l'ensemble des sommets accessibles depuis V_1 (resp. V_3) en utilisant des arêtes de poids strictement plus grand que a (resp. b). Étant donné que le coût de la solution optimale considérée est $a + b$, nous savons que :

- $S_1 \subset V_1^*$ et $S_3 \subset V_3^*$,
- $S_1 \cap S_3 = \emptyset$,
- il n'y a pas d'arête entre S_1 et S_3.

Dans un second temps l'algorithme ajoute S_1 dans V_1 et S_3 dans V_3.
Enfin, l'algorithme affecte tous les autres sommets à V_2. Il est facile de voir que cette stratégie ne fera pas apparaître d'arête entre V_1 et V_3, et n'augmentera pas la valeur de la solution par rapport à l'optimale, puisque les sommets restants ne sont adjacents à aucun sommet de V_1 (resp. V_3) par une arête de poids strictement plus grand que a (resp. b).

- Second cas : le graphe quotient de (V_1^*, V_2^*, V_3^*) contient trois arêtes. Ici également nous supposons que nous disposons des arêtes e_a^*, e_b^* et e_c^* de plus grand poids entre les clusters (*c.f.* Figure 15b). On note $a = w(e_a^*)$, avec $e_a^* = \{a_1, a_2\}$ (où $a_1 \in V_1^*$, $a_2 \in V_3^*$), $b = w(e_b^*)$, avec $e_b^* = \{b_1, b_2\}$ (où $b_1 \in V_1^*$, $b_2 \in V_2^*$), et $c = w(e_c^*)$, avec $e_c^* = \{c_1, c_2\}$ (où $c_1 \in V_2^*$, $c_2 \in V_3^*$). Sans perdre de généralité on suppose $a \leq b \leq c$. Ici aussi l'algorithme recopie la solution optimale en plaçant a_1, b_1 dans V_1, b_2, c_1 dans V_2 et a_2, c_2 dans V_3.
Soit S_1 (resp. S_3) l'ensemble des sommets accessibles depuis V_1 (resp. V_3) en

utilisant des arêtes de poids strictement supérieur à b (resp. c). En utilisant les mêmes arguments que précédemment, on sait que :

- $S_i \subset V_I^*$ pour $i \in \{1, 3\}$,
- $S_1 \cap S_3 = \emptyset$,
- il n'y a pas d'arête entre S_1 et S_3 de poids strictement supérieur à a.

On ajoute alors S_i à V_i pour $i \in \{1, 2, 3\}$, et l'algorithme termine en affectant tous les sommets restants à V_2. Comme précédemment, il est clair que cette solution sera de coût $a + b + c$, tout comme la solution optimale.

Concernant la complexité de l'algorithme, celle-ci est bornée par l'énumération de toutes les paires et triplets d'arêtes, et est donc en $\mathcal{O}(m^3)$. □

Le raisonnement précédent, concernant la résolution de Sparsest k-Compaction pour $k = 3$, n'est plus applicable pour $k = 4$. En effet, il nécessite finalement de résoudre le problème H-Compaction introduit par N. Vikas qui prouva, entre autres, que celui-ci est $\mathcal{NP}$-difficile lorsque H est non-chordal [109], et donc en particulier lorsque $H = C_4$, le cycle à quatre sommets. Il est cependant possible de contourner cette difficulté en évitant de résoudre ce cas particulier pour pouvoir résoudre Sparsest k-Compaction en temps polynomial lorsque $k = 4$.

Théorème 25. Sparsest k-Compaction *est polynomial pour* $k = 4$.

Preuve. Soit $G = (V, E)$ le graphe d'entrée, avec $|V| \geq 4$. L'algorithme consiste à essayer de construire une solution de coût $C = 3, 4, 5, 6$, successivement.
Il est clair que si $D(G) \geq 3$, alors il est facile de construire une 4-partition de coût 3, le graphe quotient étant un chemin (on rappelle que $D(G)$ est le diamètre du graphe G).
Autrement, montrons que G admet une partition de coût 3 si et seulement s'il contient un ensemble indépendant de taille 3. En effet, si un tel ensemble $\{x, y, z\}$ existe, alors $\{x\}, \{y\}, \{z\}, V \setminus \{x, y, z\}$ est clairement une partition de coût 3. Inversement, si $D(G) < 3$, et que G admet une 4-partition de coût 3, alors le graphe quotient doit être homomorphe à l'étoile $K_{1,3}$, et contient donc un ensemble indépendant de taille 3.
Supposons maintenant que G ne contient pas de 4-partition de coût 3. S'il existe $x, y, z \in V$ tels que $\{x, y\}, \{x, z\} \notin E$, alors $\{x\}, \{y\}, \{z\}, V \setminus \{x, y, z\}$ est une 4-partition de coût au plus quatre. Autrement, chaque sommet de G est adjacent à au moins $|V| - 2$ autres sommets. Dans ce cas, il est facile de voir que sauf si $G = C_4$, il ne peut admettre de 4-partition de coût 4.
Ainsi, si G n'est pas complet, il existe $x, y \in V$ tel que $\{x, y\} \notin E$. Dans ce cas on peut clairement construire une 4-partition de coût 5. Sinon, le graphe est complet et la seule solution est de coût 6.

Le temps d'exécution d'un tel algorithme est $\mathcal{O}(n^3)$ puisqu'il est borné par l'énumération de paires et de triplets de sommets. □

La stratégie utilisée dans le résultat précédent est de contourner la $\mathcal{NP}$-difficulté de compacter vers certains graphes quotients. Plus précisément, s'il est impossible de tester en temps polynomial s'il existe une compaction vers un certain graphe quotient, alors nous essayons de compacter vers un graphe de même coût (même nombre d'arêtes) mais plus facile. Dans l'exemple précédent, au lieu d'essayer de compacter vers C_4, nous essayons avec le graphe 3-pan[2]. Si une telle compaction n'existe pas, alors nous essayons de dégager des contraintes sur le graphe G nous permettant de continuer plus facilement. Malheureusement, cette stratégie se généralise difficilement à tout k constant, et la question de l'appartenance à $\mathcal{XP}$ de Sparsest k-Compaction est toujours ouverte.

3.5.2 Contrainte sur la taille des clusters

Nous nous intéressons à deux variantes du problème Sparsest k-Compaction, où il est demandé une solution avec un nombre minimum (ou exact) de sommets dans chaque cluster. Nous montrons que même avec ces contraintes, les deux problèmes restent $\mathcal{NP}$-difficiles. Les deux preuves sont similaires et suivent la même idée que celle du Théorème 17 sur la difficulté de Sparsest k-Compaction.

$\geq$Sparsest k-Compaction
Entrée : Un graphe $G = (V, E)$, deux entiers $k, l \in \mathbb{N}$
Sortie : Une k-compaction $\{V_1, \cdots, V_k\}$ de G vers un graphe $H = (V_H, E_H)$ telle que $|V_i| \geq l$ pour tout $i \in \{1, \cdots, k\}$
But : Minimiser $|E_H|$

Théorème 26. *$\geq$Sparsest k-Compaction est $\mathcal{NP}$-difficile et $\mathcal{W}[1]$-difficile (paramétré par k) pour tout $l \in \mathbb{N}$ fixé.*

Preuve. Soit $l \in \mathbb{N}$. Nous réduisons depuis le problème Independent Set. Soit $G = (V, E)$ un graphe et $k \leq \mathbb{N}$. Nous supposons $l \leq |V|$, le problème étant trivial autrement. On construit le graphe G' comme l'union disjointe de l copies de G : $G_1, \cdots, G_l$, avec $G_i = (V_i, E_i)$ pour tout $i \in \{1, \cdots, l\}$. Pour chaque sommet v de G, on relie deux à deux ses copies dans G' (on obtient donc $|V|$ cliques de taille l chacune). Puis, pour tout $i \in \{1, \cdots, l\}$, on ajoute un sommet α_i adjacent à tous les sommets de G_i. On relie enfin α_i à α_{i+1} pour tout $i \in \{1, \cdots, (l-1)\}$. Il est clair que cette construction peut s'effectuer en temps polynomial. Montrons maintenant que G contient un ensemble indépendant de taille k si et seulement si G' admet une $(k+1)$-compaction de coût k avec au moins l sommets dans chaque cluster.
Soit $S = \{s_1, \cdots, s_k\}$ un ensemble indépendant de taille k dans G ($s_i \in V$ pour

[2]le 3-pan est un cycle de taille 3 avec un sommet pendant.

tout $i \in \{1, \cdots, k\}$). Pour tout $i \in \{1, \cdots, l\}$, on définit l'ensemble de sommets correspondants $S^i = \{s_1^i, \cdots, s_k^i\}$ de G_i. On construit alors la $(k+1)$-partition $P = \{V_1, \cdots, V_{k+1}\}$ de G' comme suit :

- $V_i = \bigcup_{j=1}^{l} \{s_i^j\}$,
- $V_{k+1} = V \setminus \left(\bigcup_{i=1}^{l} V_i\right)$.

Par construction, et puisque S^i est un ensemble indépendant et α_i relié à chaque sommet dans G_i, on a bien une $(k+1)$-compaction de coût k dans G', et chaque cluster contient au moins l sommets.
Inversement, du fait des adjacences des sommets α_i et des cliques pour chaque copie de chaque sommet de G, il existe un cluster contenant tous les α_i pour $i \in \{1, \cdots, l\}$. En effet, dans le cas contraire chaque α_i serait dans le même cluster que G_i, ce qui serait impossible à cause des cliques. Ainsi, à cause des adjacences de ce cluster, les k autres ne peuvent être adjacents deux à deux, ce qui implique que toutes les copies d'un même sommet doivent être dans le même cluster. Il est facile de voir que ces contraintes impliquent que G contient un ensemble indépendant de taille k.
Enfin, la réduction préservant le paramètre standard d'INDEPENDENT SET, la $\mathcal{W}[1]$-difficulté suit. □

$_{=}$**SPARSEST k-COMPACTION**
Entrée : Un graphe $G = (V, E)$, deux entiers $k, l \in \mathbb{N}$
Sortie : Une k-compaction $\{V_1, \cdots, V_k\}$ de G vers un graphe $H = (V_H, E_H)$ telle que $\lvert V_i \rvert = l$ pour tout $i \in \{1, \cdots, k\}$
But : Minimiser $\lvert E_H \rvert$

Théorème 27. *$_{=}$SPARSEST k-COMPACTION est $\mathcal{NP}$-difficile, $\mathcal{W}[1]$-difficile (paramétré par k), et non-approximable à facteur $\mathcal{O}(n^{1-\epsilon})$ sauf si $\mathcal{P} = \mathcal{NP}$.*

Preuve. Nous exhibons une réduction gap-preserving depuis INDEPENDENT SET, préservant le paramètre. Soient $G = (V, E)$, et $k \in \mathbb{N}$. On construit G' à partir de G en ajoutant k cliques de $n - k$ sommets $A_1, \cdots, A_k$. Puis, on ajoute à G' un sommet universel ω. L'objectif est de trouver une $(k+1)$-partition de coût k où chaque cluster est de taille exactement $(n - k + 1)$.
Si G a un ensemble indépendant $S = \{s_1, \cdots, s_k\}$, alors on construit la $(k+1)$-compaction suivante : pour tout $i \in \{1, \cdots k\}$, on forme le cluster $A_i \cup \{s_i\}$, puis on forme une $(k+1)^{ème}$ partition avec tous les sommets restants. Par construction, il est facile de voir que cette $(k+1)$-partition est de coût k, et chaque cluster contient $(n - k + 1)$ sommets.
Inversement, étant donné $r \leq 1$, s'il n'existe pas d'ensemble indépendant de taille $r \cdot k$ dans G, alors à cause des adjacences du sommet universel ω, toute $(k+1)$-partition de G' est de coût au moins $k + x$, où x est le nombre minimum d'arêtes

dans un graphe de taille k ne contenant pas d'ensemble indépendant de taille $r \cdot k$ (puisqu'un ensemble indépendant de taille s dans le graphe quotient implique un ensemble indépendant de la même taille dans le graphe d'entrée). De la même façon que dans la preuve du Théorème 17, nous avons :

$$r \cdot k \geq \frac{k^2}{n + 2x}$$

Ce qui implique que le coût de toute solution dans G' doit être au moins $k \cdot \frac{1}{2r}$, ce qui termine la réduction gap-preserving.
En outre, le paramètre standard k d'INDEPENDENT SET est préservé, d'où la $\mathcal{W}[1]$-difficulté. □

3.5.3 Dans les split graphes

Nous nous intéressons maintenant à la complexité du problème SPARSEST k-COMPACTION (sans contrainte sur la taille des clusters) dans la classe des split graphes. Nous présenterons tout d'abord quelques résultats sur la structure des solutions optimales dans ces graphes, et montrerons que le problème est $\mathcal{FPT}$ lorsqu'il est paramétré par k. Nous proposons ensuite deux paramétrisations plus fines, mais montrons que malheureusement le problème est $\mathcal{W}[1]$-difficile pour ces deux paramétrisations, en exhibant des liens intéressants avec d'autres problèmes d'optimisation. Ces réductions s'exécutant en temps polynomial, elles impliquent que le problème est $\mathcal{NP}$-difficile dans les split graphes, malgré leur apparente simplicité.

Structures des solutions

On rappelle qu'un split graphe est un graphe $G = (V, E)$ dont l'ensemble de sommets peut être partitionné en deux ensembles Q et S, induisant respectivement une clique et un ensemble indépendant.
Le résultat suivant s'intéresse à la structure des solutions optimales à SPARSEST k-COMPACTION dans les split graphes. Informellement, on montre que le problème se ramène simplement à partitionner la clique, chaque sommet de l'ensemble indépendant pouvant toujours être mis dans un singleton.

Théorème 28. *Soit $G = (V, E)$ un split graphe, avec $V = Q \cup S$ où Q induit une clique et S induit un ensemble indépendant, et soit $k \in \mathbb{N}$. Il existe toujours une solution optimale à* SPARSEST k-COMPACTION *$P^* = \{P_1, \cdots, P_k\}$ telle que G_{P^*} est un split graphe. De plus, si l'on note Q^* sa clique et S^* son ensemble indépendant, on a pour tout $i \in \{1, \cdots, k\}$:*

- *soit $P_i \subseteq S$ ou $P_i \subseteq Q$, et*
- *$P_i \subseteq S \Rightarrow |P_i| = 1$.*

Preuve.

Tout d'abord, il est facile de voir que lors de la compaction de deux sommets u, v, on a :

- si u et v appartiennent à la clique, alors la taille de la clique diminue de 1, et la taille de l'ensemble indépendant ne change pas.
- si u et v appartiennent à l'ensemble indépendant, ou bien si l'un appartient à l'ensemble indépendant et l'autre à la clique, alors la taille de l'ensemble indépendant diminue de 1, et la taille de la clique ne change pas.

Ceci prouve bien que G_{P^*} est un split graphe. Soit $n_{Q^*} = |Q^*|$. Il est facile de voir, par ce qui précède, que pour tout $X \in S^*$ on a $X \subseteq S$ (et pour tout $X \in Q^*$ on a $X \subseteq Q \cup S$). Dans ce qui suit, nous montrons qu'il est possible de modifier P^* sans augmenter son coût de telle sorte que pour tout $X \in Q^*$ on a $X \subseteq Q$, et pour tout $X \in S^*$ on a $|X| = 1$. Considérons $X \in Q^*$, avec $T = X \cap Q$ tel que $|T| > 0$ et $|X \setminus T| > 0$. On modifie alors P^* en séparant X en T et $X \setminus T$, et fusionnant T avec n'importe quel autre cluster de Q^* (*c.f.* Figure 16). La fusion supprimera alors $n_{Q^*} - 1$ arêtes du graphe. D'un autre côté, $X \setminus T$ est maintenant adjacent à au plus $n_{Q^*} - 1$ clusters parmi Q^*, donc aucune arête n'est finalement ajoutée.
On suppose pour la suite que P^* a été modifiée telle que pour tout $X \in Q^*$ on a $X \subseteq Q$ (et pour tout $X \in S^*$ on a $X \subseteq S$), sans augmenter le coût. Soit $X \in S^*$ avec $|X| = t > 1$. On distingue deux cas de figure :

- si $n_{Q^*} \geq t$, alors on modifie P^* en séparant X en t singletons, et fusionnant t clusters parmi Q^* (*c.f.* Figure 17). Après cette opération, le nombre d'arêtes supplémentaires est au plus $(t-1)(n_{Q^*}-t+1)$ arêtes (créées par la séparation), mais la fusion supprime $\binom{t}{2}+(t-1)(n_{Q^*}-t)$ arêtes. Puisqu'on a supposé $t \geq 2$, cette opération ne peut pas faire augmenter le coût.
- si $n_{Q^*} < t$, alors on modifie P^* en choisissant arbitrairement n_{Q^*} sommets de X que l'on place dans des singletons. On fusionne ensuite tous les clusters de Q^* ensemble, avec également les $t - n_{Q^*}$ sommets restants de X. Ici également il est facile de voir que le fait de séparer les sommets de X peut créer au plus n_{Q^*} nouvelles arêtes, mais la fusion de Q^* en supprime $\binom{n_{Q^*}}{2}$, ce qui termine la preuve.

□

Algorithme $\mathcal{FPT}$

Dans cette partie, nous montrons comment résoudre Sparsest k-Compaction en temps $\mathcal{FPT}$ dans les split graphes, paramétré par k. Notons que le problème reste $\mathcal{NP}$-difficile dans les split graphes, comme le montrerons les résultats de $\mathcal{W}[1]$-difficulté de la section suivante qui sont en fait des réduction polynomiales.

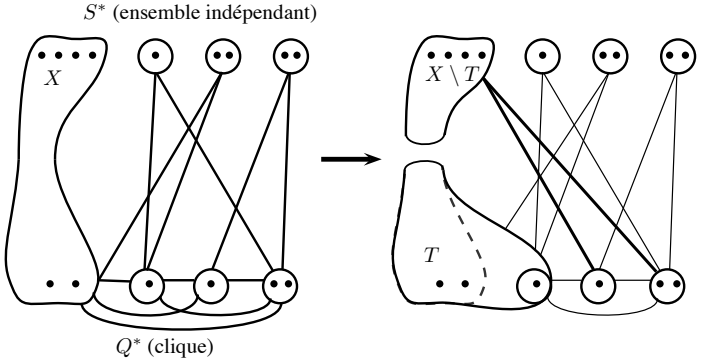


FIGURE 16 – Première restructuration d'une solution (preuve du Théorème 28).

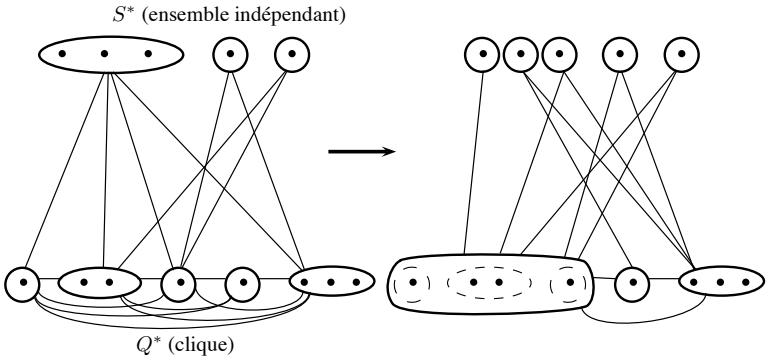


FIGURE 17 – Seconde restructuration du solution (preuve du Théorème 28).

Théorème 29. SPARSEST k-COMPACTION *est $\mathcal{FPT}$ dans les split graphes, paramétré par k.*

Preuve. Nous montrons en fait un résultat plus fort : SPARSEST k-COMPACTION est $\mathcal{FPT}$ (paramétré par k) dans toutes les classes de graphes dans lesquelles le nombre maximum de cliques maximales peut être borné par une fonction de k (et cet ensemble peut être obtenu en temps $\mathcal{FPT}$). Cette propriété vaut pour les split graphes, puisque dans ce cas le nombre de cliques maximales est borné par la taille de l'ensemble indépendant (plus un), qui est lui-même borné par $(k-2)$ (car si l'ensemble indépendant vaut au moins $k-1$, alors il est facile de construire une solution optimale, en mettant chaque sommet de l'ensemble indépendant dans un singleton, et le reste dans un dernier cluster).
Soit $G = (V, E)$ le split graphe d'entrée respectant la propriété précédente, $k, C \in \mathbb{N}$, et soit $Q = \{Q_1, \cdots, Q_t\}$ l'ensemble des cliques maximales de G. Soit $P^* = \{P_1^*, \cdots, P_k^*\}$ une solution optimale de cette instance (*i.e.* telle que son coût soit au plus C). Pour tout $i \in \{1, \cdots, t\}$, soit $S_i = \{j \in \{1, \cdots, k\} : Q_i \cap P_j^* \neq \emptyset\}$. Autrement dit, S_i contient les indices des éléments de P^* contenant des sommets de la clique Q_i. Il est clair que S_i peut être « deviné » en temps $\mathcal{FPT}$ pour tout

$i \in \{1, \cdots, t\}$ (puisque $t = f(k)$ pour une certaine fonction f), puisque S_i est en fait un sous-ensemble de $\{1, \cdots, k\}$.
Puis, étant donnés les ensembles S_i pour tout $i \in \{1, \cdots, t\}$, on construit une solution de la manière suivante : pour tout $v \in V$, on définit I_v comme l'instersection des ensembles S_j tels que $v \in Q_j$. On remarque que $I_v \neq \emptyset$ pour tout $v \in V$, puisque il contient forcément l'entier $s \in \{1, \cdots, k\}$ tel que $v \in P^*_s$. Enfin, on construit la partition $P = \{P_1, \cdots, P_k\}$ en choisissant un élément $s \in I_v$ pour tout $v \in V$, et en mettant le sommet v dans le cluster P_s. Par ce qui précède, cette partition peut clairement se construire en temps $\mathcal{FPT}$. Il suffit de montrer que son coût est au plus C.
Pour cela, nous montrons que le graphe quotient G_P a le même ensemble d'arêtes que G_{P^*}. En effet, soient $i, j \in \{1, \cdots, k\}$ tels que $\{P_i, P_j\}$ est une arête de G_P, et montrons que c'est également une arête de G_{P^*}. Soient $u \in P_i$ et $v \in P_j$ tels que $\{u, v\} \in E$, et soit $h \in \{1, \cdots, t\}$ tel que $u, v \in Q_h$ (autrement dit, Q_h est une clique maximale contenant u et v). Par définition de P, on a que $i \in I_u$ et $j \in I_v$, ce qui implique que i,$j \in S_h$. Ainsi, il existe $u', v' \in Q_h$ tel que $u' \in P^*_i$ et $v' \in P^*_j$, et puisque Q_h est une clique, les clusters P^*_i et P^*_j sont forcément adjacents dans G_{P^*}, ce qui termine la preuve. □

Autres paramétrisations : $\mathcal{W}[1]$-difficulté

Comme nous avons pu le voir précédemment, les solutions optimales à Sparsest k-Compaction dans les split graphes consistent finalement à partitionner les sommets de la clique. Ainsi, une paramètrisation plus fine[3] consiste à considérer le nombre de clusters qui partitionnent la clique. Étant donnée une instance $G = (V, E)$, $k \in \mathbb{N}$ avec $V = C \cup S$ où C induit une clique et S induit un ensemble indépendant, on définit :

$$\bar{k} = k - |S|$$

Malheureusement, le problème devient $\mathcal{W}[1]$-difficile lorsqu'il est muni de cette paramétrisation, comme le montre le résultat suivant :

Théorème 30. Sparsest k-Compaction *paramétré par* $\bar{k}$ *est* $\mathcal{W}[1]$*-difficile dans les split graphes.*

Preuve. On donne pour cela une réduction préservant le paramètre depuis le problème Min-k-Cut, prouvé $\mathcal{W}[1]$-difficile par [50]. On rappelle le problème :

Min-k-Cut
Entrée : Un graphe $G = (V, E)$, $k, C \in \mathbb{N}$
Sortie : Existe-t-il une partition $V_1, \cdots, V_k$ de V telle que $\sum_{i,j \in \{1,\cdots,k\}, i \neq j} cut(V_i, V_j) \leq C$?

[3] Dans le sens où ce nouveau paramètre est toujours plus petit que k.

Avec $cut(V_i, V_j) = |\{\{u, v\} : u \in V_i, v \in V_j\}|$ le nombre d'arêtes ayant une extrémité dans V_i et une extrémité dans V_j.

Étant donnée une instance de Min-k-Cut, on note $V = \{v_1, \cdots, v_n\}$ et $E = \{e_1, \cdots, e_m\}$. On construit alors le split graphe $G' = (V', E')$ avec $V' = Q \cup S$, où :

- $Q = \{q_1, \cdots, q_n\}$ induisant une clique de taille n (représentant les sommets de G).
- $S = \{s_1, \cdots, s_m\}$ induisant un ensemble indépendant de taille m (représentant les arêtes de G).

Puis, pour tout $i \in \{1, \cdots, m\}$ et tout $a, b \in \{1, \cdots, n\}$, on connecte s_i avec q_a et q_b si $e_i = \{v_a, v_b\}$. On peut facilement voir que G' est un split graphe avec $(n + m)$ sommets et $(\binom{n}{2} + 2m)$ arêtes. On définit en outre $k' = m + k$ et $C' = \binom{k}{2} + m + C$. On remarque que la construction peut se réaliser en temps polynomial, et que le paramètre $\bar{k} = k' - |S| = k$ est préservé. Nous montrons maintenant l'équivalence entre les solutions :

- Supposons que $V_1, \cdots, V_k$ est une solution pour l'instance de Min-k-Cut, et construisons une solution $P = \{P_1, \cdots, P_k, P_{k+1}, \cdots, P_{k+m}\}$ pour Sparsest k-Compaction dans G'. On partitionne tout d'abord Q à partir de $V_1, \cdots, V_k$ (*i.e.* le sommet $q_i \in Q$ est placé dans le cluster P_j si $v_i \in V_j$). Enfin, nous formons un singleton à partir de chaque sommet de S. Considérons une arête $e_j \in E$ pour $j \in \{1, \cdots, m\}$. Si les extrémités de e_j appartiennent à deux clusters différents, alors l'élément de P contenant s_j (le sommet de S correspondant à e_j) est adjacent à deux clusters parmi $P_1, \cdots, P_k$. Inversement, si les extrémités de l'arête e_j appartiennent au même cluster V_i, alors s_j n'est adjacent qu'à P_i. Ainsi, le nombre d'arêtes du graphe quotient G_P est $(\binom{k}{2} + m + C)$ (on rappelle que $P_1, \cdots P_k$ induit une clique).
- Inversement, soit $P = \{P_1, \cdots, P_k\}$ une solution à Sparsest k-Compaction dont le graphe quotient a $C' = \binom{k}{2} + m + C$ arêtes au plus. Par le Théorème 28, on peut supposer sans perdre de généralité ni d'optimalité de P que cette dernière est telle que pour tout $i \in \{1, \cdots k\}$, on a $P_i \subseteq Q$, et pour tout $i \in \{(k+1), \cdots, (m+k)\}$, on a $P_i = \{s_{i-k}\}$. Ainsi $P_1, \cdots, P_k$ correspond à une partition $V_1, \cdots V_k$ des sommets de G en k clusters. De plus, $P_1, \cdots, P_k$ sont deux à deux adjacents par construction, et pour tout $i \in \{(k+1), \cdots, (m+k)\}$, le cluster P_i est adjacent à soit un soit deux clusters parmi $P_1, \cdots, P_k$. Puisque le coût de la partition P est au plus $C' = \binom{k}{2} + m + C$, ceci implique qu'il existe au plus C clusters parmi $P_{k+1}, \cdots, P_{k+m}$ qui sont adjacents à deux clusters parmi $P_1, \cdots, P_k$. Il est alors facile de voir que cela implique qu'il existe au plus C arêtes dans G ayant leurs deux extrémités dans des clusters différents dans $V_1, \cdots, V_k$, et on a bien une solution de coût au plus C pour Min-k-Cut dans G comme désiré.

□

Un autre paramètre potentiellement intéressant pour Sparsest k-Compaction est le paramètre dual du paramètre standard, comme défini en Section 3.1 :

$$\hat{k} = n - k$$

Correspondant au nombre de compactions de paires de sommets à réaliser pour obtenir le graphe quotient. Malheureusement, nous montrons que Sparsest k-Compaction est $\mathcal{W}[1]$-difficile muni de cette paramétrisation, même dans le cas des split graphes.

Théorème 31. Sparsest k-Compaction *paramétré par* $\hat{k}$ *est* $\mathcal{W}[1]$*-difficile, même dans les split graphes.*

Preuve. Nous exhibons pour cela une réduction préservant le paramètre depuis le problème Clique. Soient $G = (V, E)$ un graphe avec n sommets et m arêtes, et $k \leq |V|$. La construction de $G' = (V', E')$ est la même que celle du théorème précédent (Théorème 30), et on a donc $V' = Q' \cup S'$ avec $Q' = \{q_1, \cdots, q_n\}$ qui induit une clique représentant les sommets de G et $S' = \{s_1, \cdots, s_m\}$ qui induit un ensemble indépendant représentant les arêtes de G, connectant Q' et S' selon les adjacences de G. On pose enfin

$$k' = m + n + k - 1$$

et

$$C' = 2m + \binom{n}{2} - k(k-1) - (n-k)(k-1)$$

On a ainsi $\hat{k} = |V'| - k' = k - 1$ et le paramètre est bien préservé. Montrons que G contient une clique de taille k si et seulement si G' contient une k'-partition de coût C'.

- Soit $K \subseteq V$ une clique de taille k dans G. Sans perdre de généralité on suppose $K = \{v_1, \cdots, v_k\}$. L'idée est de placer les sommets correspondants dans Q' : $\{q_1, \cdots, q_k\} = K'$ dans un cluster, et de placer tous les autres sommets de G' dans un singleton. Il est facile de voir que le nombre de clusters ainsi formés est bien k'. En outre, le nombre d'arêtes du graphe quotient est $|E'| - x$, avec

$$x = \underbrace{\frac{k(k-1)}{2}}_{\text{arêtes entre } K' \text{ et } S} + \overbrace{\frac{k(k-1)}{2}}^{\text{arêtes de } K'} + \underbrace{(n-k)(k-1)}_{\text{arêtes entre } K' \text{ et } Q \setminus K'}$$

  Et donc $m' - x = C'$ comme désiré.

- Supposons maintenant qu'il existe une k'-partition P telle que G_P contienne C' arêtes ou moins. De même que pour le théorème précédent, nous supposons que P est telle que décrite dans le Théorème 28. Dans ce qui suit nous noterons de la même manière les sommets de G_P et les clusters de P (par exemple, pour tout $X \in Q^* \cup S^*$, on a $X \subseteq V$). Pour résumer, tous les clusters de P^* sont des singletons, sauf pour quelques uns (parmi Q^*) qui ne contiennent que des sommets de Q. Puisque $k' = m+n-k+1$, Q (et donc V) est en fait partitionné en $(n-k+1)$ clusters, ou, autrement dit, qu'il faut compacter k sommets entre eux. Il est clair que n'importe quelle $(n-k+1)$-partition de Q impliquera un graphe quotient avec au plus $2m + \binom{n}{2} - \frac{k(k-1)}{2} - (n-k)(k-1)$ arêtes, mais afin d'obtenir au plus C' arêtes, on peut facilement voir que ces compactions doivent « supprimer » $\binom{k}{2}$ arêtes du graphe. Ceci peut se faire uniquement si G contient une clique de taille k, ce qui termine la preuve.

□

3.6 Conclusion, problèmes ouverts

Dans ce chapitre, nous nous sommes intéressés à la complexité algorithmique de problèmes de partition de graphes appelés problèmes de compaction. Ces problèmes consistent à chercher dans un graphe donné une partition de ses sommets en k clusters, telle que le graphe quotient, obtenu en « compactant » chacun des clusters, respecte une certaine propriété ou optimise une certaine fonction objectif. Les deux principales fonctions objectifs auxquelles nous nous sommes intéressés étaient de minimiser le nombre d'arêtes du graphe quotient : Sparsest k-Compaction, et minimiser le degré maximum de celui-ci : Bounded Degree k-Compaction. Une application possible de ces problèmes est leur lien avec la remodularisation de logiciels écrits en langage orienté objets, où une partition correspond à un « découpage » du logiciel. Concernant Bounded Degree k-Compaction, nous avons montré qu'il était $\mathcal{NP}$-difficile, même dans les arbres. Concernant Sparsest k-Compaction, nous avons montré qu'il était aussi difficile que le problème Independent Set du point de vue de la complexité classique, approchée ou paramétrée. Puis nous avons développé plusieurs algorithmes d'approximation à facteur non constant pour le problème et sa version pondérée. Nous avons également étudié sa complexité en ajoutant des contraintes soit sur la solution, soit sur l'instance d'entrée, en fixant par exemple le nombre de clusters demandés ou en montrant qu'il restait difficile dans les split graphes.

Malgré ces résultats, de nombreuses questions restent ouvertes concernant ces problèmes. Dans ce qui suit nous en présentons quelques-uns, accompagnés d'une brève description de pistes.

Problème ouvert 6. *Est-ce que* Sparsest k-Compaction *est dans* $\mathcal{XP}$ *paramétré par* k *?*

C'est probablement la question la plus intéressante concernant ce problème : savoir s'il existe un algorithme polynomial pour chaque valeur de k fixée. Comme nous l'avons vu à la fin de ce chapitre, la réponse est positive pour $k = 3$ et $k = 4$, cependant la méthode utilisée ne semble pas s'étendre de manière triviale à une valeur de k générique. En effet, l'idée consiste à énumérer tous les graphes quotient à k sommets dans un premier temps, et à chercher une compaction vers chacun d'entre eux. Cette méthode trouve rapidement ses limites, puisque le problème H-Compaction (compacter vers un graphe H fixé) est $\mathcal{NP}$-difficile pour tous les graphes non chordaux. Pour $k = 4$ et lorsque nous essayons de compacter vers C_4, nous outrepassons ce problème en essayant d'abord de compacter vers un graphe chordal ayant le même nombre d'arêtes.

Problème ouvert 7. *Est-ce que* Sparsest k-Compaction *dans les split graphes est dans* $\mathcal{XP}$ *paramétré par* $\bar{k}$ *?*

On rappelle que $\bar{k}$ correspond au nombre de clusters partitionnant la clique du split graphe. Un tel algorithme généraliserait en fait l'algorithme $\mathcal{XP}$ connu pour Min-k-Cut dans les graphes généraux [66] à une version dans les hypergraphes. En effet la réduction présentée dans le Théorème 30 fonctionne dans l'autre sens, et donne un hypergraphe. Le problème revient alors à partitionner les sommets d'un hypergraphe en k clusters en minimisant le nombre de paires de sommets appartenant à une même hyper-arête et se retrouvant dans des clusters différents.

Problème ouvert 8. *Est-ce que* Sparsest k-Compaction *est* $\mathcal{FPT}$ *paramétré par la tree-width du graphe d'entrée ?*

Le problème est clairement $\mathcal{FPT}$ s'il est paramétré par la tree-width et k, puisqu'expressible par une formule MSO dont la taille dépend de k. Dans les arbres, le problème est polynomial puisqu'il suffit de compacter successivement les feuilles avec leur voisin pour obtenir une partition de coût $(k-1)$, optimale pour les graphes connexes.

Problème ouvert 9. *Est-ce que* Bounded Degree k-Compaction *est* $\mathcal{XP}$ *paramétré par* (k, C) *?*

On rappelle que C est le paramètre standard : le degré maximum désiré pour le graphe quotient. Pour une réponse positive à cette question, la même idée que pour le problème ouvert 6 peut s'appliquer. Pour un résultat négatif, une possibilité est de montrer la $\mathcal{NP}$-difficulté pour une valeur de C fixée. Pour $C = 2$ par exemple, les graphes quotients possibles sont soit P_k soit C_k. Comme vu précédemment, compacter vers C_k est $\mathcal{NP}$-difficile, mais compacter vers P_k est trivial puisqu'il correspond à tester si le diamètre du graphe est au moins $k - 1$. Ainsi, il suffirait de montrer que compacter vers C_k est $\mathcal{NP}$-difficile pour les graphes de diamètre $k - 2$. Malheureusement, le graphe obtenu dans la réduction de [109] ne respecte pas cette propriété.

Chapitre 4

Recherche d'un Sous-Graphe à Cardinalité Fixée

Plan du Chapitre :

Dans ce chapitre, nous nous intéressons à la famille des problèmes de recherche d'un sous-graphe à cardinalité fixée. D'une manière générale, ces problèmes prennent en entrée un graphe $G = (V, E)$ et un entier k, et demandent un ensemble de k sommets optimisant une certaine fonction objectif donnée. La fonction objectif sur laquelle nous nous sommes focalisés durant cette thèse est d'induire le minimum d'arêtes possible. Comme nous allons le voir, ce problème généralise le problème classique Independent Set et est donc difficile dans les graphes généraux, que ce soit du point de vue de l'approximation ou de la complexité paramétrée. Nous étudions alors la complexité de ce problème dans des classes de graphes restreintes, notamment dans celles où Independent Set est résoluble en temps polynomial, telles que les sous-classes des graphes parfaits. Nous nous intéressons aux graphes chordaux, et montrons que le problème y est $\mathcal{NP}$-difficile, mais admet un algorithme polynomial 2-approché ainsi qu'un algorithme $\mathcal{FPT}$, mais pas de noyau polynomial. Nous nous intéressons ensuite aux graphes d'intervalles, et donnons quelques algorithmes d'approximation (avec un meilleur rapport que dans le cas des graphes chordaux) et paramétrés (avec une paramétrisation plus fine) malgré la question toujours ouverte de sa complexité (polynomial ou $\mathcal{NP}$-difficile). De même que pour les chapitres précédents, nous terminerons par une liste de problèmes ouverts.

Ces travaux ont donné lieu aux publications suivantes :

- [114] *Approximating the Sparsest k-Subgraph in Chordal Graphs*, avec Marin Bougeret et Rodolphe Giroudeau. Theory of Computing Systems, à paraître. (Une version préliminaire de cet article fut publié dans les actes de la conférence *WAOA 2013*, Springer LNCS 8447, pp. 73-84 [113])

- [21] *Parameterized Complexity of the Sparsest k-Subgraph Problem in Chordal Graphs*, avec Marin Bougeret, Nicolas Bousquet et Rodolphe Giroudeau. Proceedings of SOFSEM 2014, Springer LNCS 8327, pp. 150-161.

4.1 Introduction et travaux existants

4.1.1 Présentation, définitions des problèmes

Parmi les problèmes de graphes, les problèmes de recherche d'un sous-graphe particulier sont un des types de problèmes les plus largement étudiés. Ceux-ci prennent le plus souvent en entrée un graphe $G = (V, E)$, et demandent en sortie un ensemble de sommets de taille maximum (ou minimum pour des problèmes de minimisation) tel qu'une certaine propriété soit vérifiée par cet ensemble. Parmi ces problèmes, les plus connus sont Independent Set, où la propriété est de n'induire aucune arête, Clique, où la propriété est d'induire un graphe complet, ou Vertex Cover, où la propriété est de couvrir toutes les arêtes du graphe à l'aide des sommets

(problème de minimisation pour ce dernier). D'un point de vue de la complexité classique, approchée ou paramétrée, de nombreux résultats furent obtenus pour ces trois problèmes dans le cas général, comme en témoigne la très riche littérature sur le sujet. En effet, ils furent tout d'abord parmi les premiers problèmes de graphes à être prouvés $\mathcal{NP}$-complets, et figurent notamment dans la liste des vingt et un problèmes de R. Karp [81]. Ensuite, ils ont été largement étudiés d'un point de vue de l'approximation dans un premier temps, et de la complexité paramétrée dans un second temps : alors qu'il existe plusieurs algorithmes 2-approchés [99] et $\mathcal{FPT}$ [51] pour Vertex Cover (et même plusieurs noyaux polynomiaux), on sait qu'il est peu probable que Clique et Independent Set admettent un algorithme approché avec un ratio $\mathcal{O}(n^{1-\epsilon})$ quel que soit $\epsilon > 0$ [118] ou un algorithme $\mathcal{FPT}$ [51], le premier sous l'hypothèse que $\mathcal{P} \neq \mathcal{NP}$, le second sous l'hypothèse que $\mathcal{FPT} \neq \mathcal{W}[1]$. Concernant Vertex Cover, on soupçonne même fortement que le ratio d'approximation de deux ainsi que les noyaux polynomiaux connus sont les meilleurs possibles (sous l'hypothèse de la *Unique Game Conjecture* pour le premier [84], et sous l'hypothèse $coNP \not\subseteq NP/poly$ pour le second [44]). On dit d'ailleurs souvent que ces problèmes sont les *drosophiles* de l'approximation et de la complexité paramétrée, tant leur étude a permis de développer des méthodes devenues classiques pour obtenir des résultats positifs et négatifs dans chacun de ces domaines.

D'un point de vue théorique et pratique, il est naturel de s'intéresser à des généralisations de ces problèmes. L'une des généralisations à laquelle nous nous sommes intéressés sont les problèmes d'*optimisation à cardinalité fixée*. Pour Vertex Cover par exemple, au lieu de chercher le sous-ensemble de taille minimum nécessaire pour couvrir toutes les arêtes du graphe, on cherchera un sous-ensemble de taille k qui couvrira le nombre maximum d'arêtes. De manière similaire, la version cardinalité fixée de Independent Set (resp. Clique) consiste à chercher le sous-ensemble de k sommets qui va induire le nombre minimum (resp. maximum) d'arêtes. Il est facile de voir que ces versions, dîtes « à cardinalité fixée » ne sont pas plus faciles à résoudre que les versions de base : en effet, avoir un algorithme permettant de résoudre la version cardinalité fixée de Independent Set nous permettrait de résoudre Independent Set, en l'exécutant avec des valeurs de k décroissantes à partir de n (ou même à l'aide d'une recherche par dichotomie). Les versions à cardinalité fixée de Vertex Cover, Independent Set et Clique s'appellent respectivement Maximum k-Coverage, Sparsest k-Subgraph et Densest k-Subgraph, et leurs définitions formelles sont les suivantes (en version « décision ») :

Sparsest k-Subgraph
Entrée : un graphe $G = (V, E)$, $k \leq |V|$, $C \leq \binom{k}{2}$.
Sortie : Existe-t-il un ensemble $S \subseteq V$ de taille k tel que $G[S]$ induise au plus C arêtes ?

DENSEST k-SUBGRAPH
Entrée : un graphe $G = (V, E)$, $k \leq |V|$, $C \leq \binom{k}{2}$.
Sortie : Existe-t-il un ensemble $S \subseteq V$ de taille k tel que $G[S]$ induise au moins C arêtes ?

MAXIMUM k-COVERAGE
Entrée : un graphe $G = (V, E)$, $k \leq |V|$, $C \leq |E|$.
Sortie : Existe-t-il un ensemble $S \subseteq V$ de taille k tel que S couvre au moins C arêtes ?

4.1.2 Travaux existants

Ces problèmes ont été étudiés indépendamment depuis de nombreuses années. Puis, un premier état de l'art, avec une bibliographie annotée, a été proposé par [25], et dans un article plus récent, L. Cai [28] s'est intéressé à la résolution exacte paramétrée de ceux-ci, en donnant par exemple des algorithmes $\mathcal{XP}$ avec des meilleurs temps d'exécution théoriques que les algorithmes naïfs. Notons que dans leur version « décision », SPARSEST k-SUBGRAPH et MAXIMUM k-COVERAGE sont équivalents (car trouver k sommets qui induisent le minimum d'arêtes est équivalent à trouver $n - k$ sommets qui couvrent le maximum d'arêtes), et SPARSEST k-SUBGRAPH dans un graphe G est équivalent à DENSEST k-SUBGRAPH dans le complémentaire de G. En fait, on peut observer la même analogie entre ces trois problèmes que celle que l'on retrouve entre INDEPENDENT SET, VERTEX COVER et CLIQUE. Dans le paragraphe suivant, nous détaillons les résultats connus concernant MAXIMUM k-COVERAGE, SPARSEST k-SUBGRAPH et DENSEST k-SUBGRAPH concernant les complexités classique, approchée et paramétrée dans les graphes généraux ou des classes de graphes restreintes. La plupart de ces résultats sont résumés dans le Tableau 2.

Comme dit précédemment, ces trois problèmes étant des généralisations de problèmes $\mathcal{NP}$-complets, ils le sont donc également. Plus précisément, la difficulté de INDEPENDENT SET pour l'approximation et la complexité paramétrée se transmet également : SPARSEST k-SUBGRAPH est $\mathcal{W}[1]$-difficile pour le paramètre k, et le 0 dans la fonction objectif correspondant au cas INDEPENDENT SET implique qu'il est inapproximable (pour tout ratio) sauf si $\mathcal{P} = \mathcal{NP}$, et para-$\mathcal{NP}$-complet pour le paramètre standard (la valeur de la solution, *i.e.* le nombre d'arêtes induites).
Concernant DENSEST k-SUBGRAPH, il est également $\mathcal{W}[1]$-difficile en tant que généralisation de CLIQUE, mais son approximabilité a été pendant longtemps (et continue d'être) un grand challenge : en effet, le meilleur ratio d'approximation connu a longtemps été $n^{1/3-\epsilon}$ pour un certain $\epsilon \approx 1/60$ [56], avant d'être légèrement battu plus récemment par [12] avec un ratio de $n^{1/4}$. D'un autre côté, le seul résultat négatif est la non-existence de $\mathcal{PTAS}$ (sous l'hypothèse que certains problèmes de $\mathcal{NP}$ n'ont

pas d'algorithme sous-exponentiel randomisé) de S. Khot [83], laissant un écart considérable pour des avancées soit positives soit négatives. Cependant, de nombreuses personnes dans la communauté conjecturent que Densest k-Subgraph n'admet pas d'algorithme d'approximation avec un ratio constant.
Concernant Maximum k-Coverage dans les graphes généraux, les choses sont plus positives : tout d'abord, l'algorithme glouton qui choisit k sommets de plus grands degrés atteint un ratio de 1,582 [76]. Ensuite, ce problème nous permet de voir la puissance de l'approximation paramétrée : alors qu'il est $\mathcal{W}[1]$-difficile pour le paramètre k [70] et $\mathcal{APX}$-complet [101], il admet un schéma d'approximation en temps paramétré [91]. Autrement dit, en temps $\mathcal{FPT}$ en k, il est impossible de trouver une solution exacte, et en temps polynomial, il est impossible de trouver une solution arbitrairement proche de l'optimal, mais en temps $\mathcal{FPT}$ en k et ϵ, une solution $(1+\epsilon)$-approchée peut être trouvée pour tout $\epsilon > 0$. Enfin, lorsque paramétré par la valeur de la solution : le nombre d'arêtes à couvrir (qui peut être arbitrairement plus grand que k), la technique de séparation aléatoire donne un algorithme $\mathcal{FPT}$ [29].

Intéressons nous maintenant à la complexité de ces problèmes dans des classes de graphes restreintes. Il est par exemple pertinent de s'intéresser aux graphes parfaits, car Clique, Independent Set et Vertex Cover y sont polynomiaux. À notre connaissance, ce sont D. G. Corneil et Y. Perl qui, en 1984 [41], sont les premiers à s'intéresser à cette question. Ils établissent notamment la polynomialité de Densest k-Subgraph dans les split graphes, et sa $\mathcal{NP}$-difficulté dans les graphes chordaux et les graphes bipartis de degré maximum trois (entre autres). Ces résultats ne se transmettent malheureusement pas en totalité à Sparsest k-Subgraph, étant donné que le complémentaire d'un chordal (resp. biparti) n'est pas nécessairement un chordal (resp. biparti). Il implique ainsi seulement que Sparsest k-Subgraph (et donc Maximum k-Coverage) est $\mathcal{NP}$-difficile pour les graphes parfaits (car les graphes parfaits sont, eux, stables par passage au complémentaire, et les chordaux sont des graphes parfaits). Concernant les graphes bipartis, deux articles récents, de G. Joret et A. Vetta [80] d'une part, et de N. Apollonio et B. Simeone [6] d'autre part, montrent indépendamment que Maximum k-Coverage (et donc Sparsest k-Subgraph) est $\mathcal{NP}$-difficile dans cette classe de graphes. Cependant la manière la plus simple d'arriver à ce résultat est de suivre la réduction de [33]. Dans la Section 4.2, nous complétons tous ces résultats en montrant que Sparsest k-Subgraph et Maximum k-Coverage sont bien $\mathcal{NP}$-difficiles dans les graphes chordaux. D'un point de vue de l'approximation, beaucoup de travaux ont été consacrés à Densest k-Subgraph dans les graphes chordaux et leurs sous-classes : dans [89], les auteurs donnent un algorithme 3-approché dans les graphes chordaux, et dans [35], les auteurs donnent un algorithme $\mathcal{O}(\sigma)$-approché pour les graphes possédant un σ-ordre d'élimination, une généralisation des ordres d'élimination simpliciaux qui permettent de généraliser les graphes chordaux (on a en fait $\sigma = 1$ pour les graphes chordaux) et d'autres classes de graphes d'intersections. Concernant un grand nombre de sous-

Classes de graphes	Densest k-Subgraph	Sparsest k-Subgraph	Maximum k-Coverage
graphes généraux	$\mathcal{NP}$-h (*c.f.* max clique) $n^{\frac{1}{4}+\epsilon}$-approx. [12]	$\mathcal{NP}$-h, not approx. (*c.f.* indep. set)	$\mathcal{NP}$-h, $\mathcal{W}$[1]-h [71] 2-approx.[26] exact $O^*(1{,}4^C)$ [85]
chordaux	$\mathcal{NP}$-h [41] 3-approx [89]	$\mathcal{NP}$-h **[Thm. 33]** 2-approx **[Thm. 34]**	$\mathcal{NP}$-h (*c.f.* SkS)
intervalles	OPEN $\mathcal{PTAS}$ [98]	OPEN, $\mathcal{FPT}(C)$ **[Thm. 40]**	OPEN $\mathcal{FPT}(n\text{-}k)$ (*c.f.* SkS)
int. propres	OPEN $\mathcal{PTAS}$ [98]	OPEN, $\mathcal{PTAS}$ **[Thm. 39]**	OPEN
bipartis	$\mathcal{NP}$-h [41]	$\mathcal{NP}$-h (*c.f.* MkC)	$\mathcal{NP}$-h [80]
line graphes	OPEN	$\mathcal{P}$ (*c.f.* MkC)	$\mathcal{P}$ [5]
planaires	OPEN	$\mathcal{NP}$-h (*c.f.* indep. set)	$\mathcal{NP}$-h (*c.f.* SkS)
cographes, split, tree-width bornée	$\mathcal{P}$ [41]	$\mathcal{P}$ [24]	$\mathcal{P}$ (*c.f.* SkS)
degré max. deux	$\mathcal{P}$ [41]	$\mathcal{P}$ [24]	$\mathcal{P}$ (*c.f.* SkS)
degré max. trois	$\mathcal{NP}$-h [41]	$\mathcal{NP}$-h (*c.f.* indep. set)	$\mathcal{NP}$-h (*c.f.* SkS)

Table 2 – Principaux résultats pour Sparsest k-Subgraph, Densest k-Subgraph et Maximum k-Coverage dans différentes classes de graphes.

classes non triviales des graphes chordaux, il s'avère que la question de la complexité ($\mathcal{NP}$-difficile *versus Polynomial*) de Densest k-Subgraph est toujours ouverte. Malgré cela, des travaux proposent des algorithmes d'approximation dans ces cas particuliers : dans [88], les auteurs donnent des algorithmes approchés pour des graphes chordaux ayant des décompositions en cliques particulières (étoiles, ou degré de l'arbre constant), et dans [98], une $\mathcal{PTAS}$ est proposée pour les graphes d'intervalles (nous reviendrons plus en détails sur le cas des graphes d'intervalles en début de Sous-Section 4.6). Enfin, un dernier cas intéressant est le cas des graphes planaires : alors que la complexité de Sparsest k-Subgraph est immédiate, étant donné que Independent Set y est $\mathcal{NP}$-difficile, la complexité de Densest k-Subgraph est toujours inconnue, contrairement à ce qu'affirment certains articles dans la littérature. En effet [82] est souvent cité pour affirmer que Densest k-Subgraph est $\mathcal{NP}$-difficile dans les graphes planaires, mais les auteurs traitent en fait seulement de la version du problème où l'on recherche une solution connexe. Les auteurs posent d'ailleurs en conclusion la question de la complexité dans le cas sans cette contrainte de connexité, et celle-ci est, à notre connaissance, toujours ouverte.

4.1.3 Résultats obtenus

Dans les sous-sections suivantes nous complétons tous ces résultats, principalement concernant Sparsest k-Subgraph dans les graphes chordaux et les graphes d'intervalles. Tout d'abord, dans la Sous-Section 4.2 nous montrons que Sparsest k-Subgraph est bien $\mathcal{NP}$-difficile dans les graphes chordaux. Dans cette même

classe de graphes, nous donnons dans les Sous-Sections 4.3 et 4.4 respectivement un algorithme 2-approché et un algorithme $\mathcal{FPT}$ paramétré par k. Nous adaptons également le résultat de $\mathcal{NP}$-difficulté en montrant en Sous-Section 4.5 qu'il est peu probable d'obtenir un noyau polynomial, toujours paramétré par k. Enfin, dans la Sous-Section 4.6 nous étudions le cas de Sparsest k-Subgraph dans les graphes intervalles : nous donnons un algorithme $\mathcal{FPT}$ pour sa paramétrisation standard (plus fine que celle par k), et, enfin, pour les graphes d'intervalles propres, nous adaptons la $\mathcal{PTAS}$ de [98], et montrons que l'algorithme glouton donne un rapport d'approximation de deux, alors qu'il peut être arbitrairement mauvais dans les graphes d'intervalles (non propres).

4.2 Dans les graphes chordaux : difficulté

4.2.1 Prélude : le cas de Densest k-Subgraph

Dans cette section, nous montrons que Sparsest k-Subgraph est $\mathcal{NP}$-difficile dans les graphes chordaux. Malgré la similarité apparente entre Densest k-Subgraph et Sparsest k-Subgraph, la nécessité du passage au complémentaire du graphe d'entrée les différencie dans des classes de graphes particulières. Dans les graphes chordaux par exemple, la structure des solutions n'est pas la même d'un problème à l'autre. En effet intuitivement, dans le cas de Densest k-Subgraph, la solution aura tendance à choisir des cliques entières de la décomposition arborescente, et à certains endroits du graphe, des cliques qui sont adjacentes entre elles. D'un autre côté, les solutions optimales de Sparsest k-Subgraph auront tendance à s'éparpiller dans chaque clique maximale du graphe. Cette différence empêche d'utiliser les mêmes principes pour les deux problèmes, précisément lors des réductions. En effet, comme nous allons le voir, la réduction de $\mathcal{NP}$-difficulté de Densest k-Subgraph dans les chordaux de [41] utilise massivement le fait de devoir choisir des régions du graphe adjacentes entre elles en encodant une clique dans le graphe d'entrée.

Théorème 32 ([41])**.** Densest k-Subgraph *est $\mathcal{NP}$-difficile dans les graphes chordaux.*

Preuve. Comme dit précédemment, les auteurs réduisent depuis le problème classique Clique. Étant donné un graphe $G = (V, E)$ et un entier k, on construit G' à partir de G en ajoutant d'abord une arête pour chaque paire de sommets de G (on forme ainsi une clique de taille n). Puis, pour toute arête $\{u, v\} \in E$, on crée une clique F_e de n sommets, tous reliés à u et v. Les sommets de G' sont ainsi partitionnés en deux ensembles de sommets A et F représentant respectivement les sommets et les arêtes de G (A est de taille n, et F est de taille $n \cdot |E|$). Il est facile de vérifier que G' ainsi construit est chordal, est qu'il est constructible en temps polynomial. On prouve enfin que G a une clique de taille k si et seulement si G' contient $k' = k + \binom{k}{2} n$ sommets qui induisent $C' = \binom{k}{2}\binom{n+2}{2}$ arêtes.

Si G a une clique Q de taille k, alors il est facile de voir que prendre les sommets correspondant à Q dans G' ainsi que les cliques F_e telles que e est une arête de la clique donne un ensemble de k' sommets induisant C' arêtes exactement.
Inversement, soit L un ensemble de k' sommets de G' induisant C' arêtes. On note $a = |L \cap A|$, et $f = |L \cap F|$. Montrons que forcément $a = k$.
Supposons d'abord que $a < k$. Le nombre d'arêtes induites par L est égal à la somme entre :

- les arêtes induites par les a sommets choisis parmi A : il y en a $\binom{a}{2}$ étant donné que A est une clique.
- les arêtes induites par les f sommets choisis parmi F : il y en a au plus $\frac{f}{n} \cdot \binom{n}{2}$
- les arêtes entre ces deux derniers ensembles. On remarque qu'il y a au plus $\binom{a}{2}n$ sommets de F ayant deux voisins parmi $L \cap A$, les autres en ayant au plus un.

On a donc dans ce cas :

$$\begin{aligned} cost(L) &\leq \binom{a}{2} + \frac{f}{n}\binom{n}{2} + \binom{a}{2}2n + f - \binom{a}{2}n \\ &< \binom{k}{2}\binom{n+2}{2} \end{aligned}$$

Ce qui contredit la définition de L.
Si maintenant $a > k$, alors soit $d = a - k$ (on a alors $f = \binom{k}{2}n - d$). Ici aussi comptons les arêtes induites par L :

- parmi les sommets de A : $\binom{a}{2}$
- parmi les sommets de F : il est facile de voir que le nombre d'arêtes est maximum lorsque $\binom{k}{2} - 1$ cliques sont prises entièrement dans la solution, plus $n - d$ sommets d'un autre gadget. On a donc au plus $(\binom{k}{2} - 1)\binom{n}{2} + \binom{n-d}{2}$ arêtes.
- il y a f sommets de F qui ont au plus deux voisins parmi les sommets de A. Donc au plus $2(\binom{k}{2}n - d)$ arêtes.

On a donc en tout :

$$\begin{aligned} cost(L) &\leq \binom{a}{2} + (\binom{k}{2} - 1)\binom{n}{2} + \binom{n-d}{2} + 2(\binom{k}{2}n - d) \\ &< \binom{k}{2}\binom{n+2}{2} \end{aligned}$$

Ce qui contredit ici aussi la définition de L.
Ainsi on a $a = k$, et l'unique possibilité pour obtenir C' arêtes est que les sommets

choisis parmi F forment $\binom{k}{2}$ cliques, toutes adjacentes aux k sommets choisis parmi A. Autrement dit, G doit contenir une clique de taille k. □

Comme dit précédemment, une solution optimale pour SPARSEST k-SUBGRAPH aura plutôt tendance à s'éparpiller partout dans chaque clique du graphe. Ainsi, intuitivement, on ne peut plus, comme pour DENSEST k-SUBGRAPH, encoder si facilement à l'aide d'une clique le fait qu'un sommet ou une arête est dans la solution. Il faut donc élaborer des gadgets à la place, qui d'une part conservent la chordalité du graphe, et en gardant à l'esprit d'autre part que la solution choisira des sommets parmi chacun des gadgets. Il faudra alors utiliser la structure de la solution à l'intérieur de chaque gadget pour encoder le fait que l'entité correspondante (sommet ou arête) est active dans le graphe de départ. C'est l'intuition de la preuve, qui est présentée plus en détail dans les parties suivantes.

4.2.2 Idée générale

On rappelle que le but est de montrer que SPARSEST k-SUBGRAPH reste $\mathcal{NP}$-difficile dans les graphes chordaux. La preuve est une réduction depuis le problème classique CLIQUE dans les graphes généraux. Nous donnons tout d'abord une idée générale informelle avant de se lancer précisément dans la construction plus technique. Étant donné un graphe quelconque $G = (V, E)$ et $k \in \mathbb{N}$, nous construisons d'abord (similairement à DENSEST k-SUBGRAPH) le *split graphe d'adjacence* de G : nous créons une clique A de n sommets représentant les sommets de G (en fait, pour des raisons techniques, A sera dupliqué n fois et comportera donc n^2 sommets), et un ensemble indépendant F de taille m représentant les arêtes de G, en connectant A et F en fonction des adjacences du graphe : un sommet de F, représentant une arête $\{u, v\}$ du graphe sera connecté aux sommets de A représentant u et v. Ensuite, nous remplaçons chaque sommet de l'ensemble indépendant (représentant une arête $e \in E$) par une copie F_e du gadget représenté par la Figure 18 (à gauche). Nous allons par la suite montrer qu'une solution optimale à SPARSEST k-SUBGRAPH dans le graphe construit doit capturer le même nombre de sommets dans chaque gadget F_e. Cependant, deux manières se présenteront pour choisir ces sommets dans chaque gadget. La première façon encodera le fait que l'arête e correspondante appartient à la solution (clique) de G, et la seconde façon le fait que l'arête n'y appartient pas. La première solution aura un coût moins important que la seconde (au niveau du nombre d'arêtes induites), mais comportera des sommets adjacents à la clique A, contrairement à la seconde solution. Ainsi, puisque nous voulons minimiser le nombre d'arêtes du graphe induit, il sera interdit pour un gadget de choisir des sommets adjacents à des sommets que l'on aura choisi dans A. Enfin, une clique de taille k dans G sera encodée en *ne* prenant *pas* les sommets de A correspondants, permettant d'avoir $\binom{k}{2}$ gadgets où l'on peut prendre une solution du premier type, et $m - \binom{k}{2}$ gadgets où l'on peut prendre une solution du second type, afin de n'avoir aucune arête entre les sommets de la solution parmi A et les sommets de la solution

parmi F.

4.2.3 Le gadget

Nous décrivons ici le gadget F_e annoncé ci-dessus, et représenté par la Figure 18 (à gauche). Le gadget est composé de trois ensembles de T sommets X, Y et Z (la valeur de T sera fixée ultérieurement). On définit $X = \{x_1, \cdots, x_T\}$, $Y = \{y_1, \cdots, y_T\}$ et $Z = \{z_1, \cdots, z_T\}$. L'ensemble X induit une clique, l'ensemble Z induit un ensemble indépendant, et l'ensemble Y induit une clique de taille $(T-1)$ sur les sommets $\{y_2, \cdots, y_T\}$. Enfin, pour tout $i \in \{1, \cdots, T\}$, nous connectons x_i à y_i, et y_i à tous les sommets de Z. Ainsi, nous obtenons un biparti complet entre Y et Z, et un couplage entre X et Y.

L'idée de la réduction est d'imposer la solution à prendre exactement $2T$ sommets parmi chaque gadget. Il est facile de voir que les $2T$ sommets induisant le nombre minimum d'arêtes dans le gadget sont les sommets $X \cup Z$, induisant exactement $\binom{T}{2}$ arêtes (à cause de la clique induite par Z). Cependant, comme nous le verrons par la suite, le gadget sera justement connecté au reste du graphe par cet ensemble Z. Ainsi, si nous voulons interdire la solution de prendre des sommets parmi Z, les $2T$ sommets restants seront $X \cup Y$, qui induisent $(\binom{T}{2} + 1)$ arêtes, soit exactement une arête de plus que pour la première solution $X \cup Z$.

4.2.4 La construction

Soient $G = (V, E)$ et $k \leq |V|$ l'instance de départ de Clique, où l'on cherche dans G une clique de taille k. On note $V = \{v_1, \cdots, v_n\}$, $E = \{e_1, \cdots, e_m\}$ et on pose

$$T = n(n-k).$$

Dans ce qui suit nous allons construire en temps polynomial $G' = (V', E')$, et $k', C' \in \mathbb{N}$ tels que G' est un graphe chordal qui contient k' sommets induisant au plus C' arêtes si et seulement si G contient une clique de taille k. La construction est résumée à la Figure 18.

Les sommets V' de G' sont répartis en deux ensembles A et F :

- A est une clique de taille n^2, et on note $A = \{a_i^j : i,j \in \{1, \cdots, n\}\}$. Pour tout $i \in \{1, \cdots, n\}$, la *« colonne »* $A_i = \{a_i^j : j \in \{1, \cdots, n\}\}$ représentera le sommet v_i de G.

- pour tout $i \in \{1, \cdots, m\}$, F contient une copie F_i du gadget présenté précédemment, composé des ensembles de sommets X_i, Y_i et Z_i, avec $X_i = \{x_1^i, \cdots, x_T^i\}$, $Y_i = \{y_1^i, \cdots, y_T^i\}$ et $Z_i = \{z_1^i, \cdots, z_T^i\}$. De plus, si $e_i = \{v_p, v_q\}$, tous les sommets de Z_i sont connectés à tous les sommets de A_p et A_q.

Enfin on définit

$$k' = 2mT + T$$

et

$$C' = m\binom{T}{2} + \binom{T}{2} + (m - \binom{k}{2}).$$

Il est facile de voir que cette réduction s'exécute en temps polynomial. Montrons également que le graphe G' obtenu est bien un graphe chordal. En effet, nous avons le schéma d'élimination simplicial suivant :

- Pour tout $i \in \{1, \cdots, m\}$ quel que soit l'ordre, nous pouvons éliminer tous les sommets de X_i dans un ordre quelconque, car ce sont tous des sommets de degré un et qu'ils forment un ensemble indépendant.
- Pour tout $i \in \{1, \cdots, m\}$, quel que soit l'ordre comme précédemment, nous pouvons éliminer tous les sommets de Y_i, car il est facile de voir que leur voisinage est une clique quel que soit l'ordre dans lequel on les considère.
- Pour tout $i \in \{1, \cdots, m\}$ toujours quel que soit l'ordre, nous pouvons éliminer tous les sommets de Z_i, car ici également quel que soit l'ordre considéré leur voisinage est une clique (car Z_i est une clique, A est une clique, et Z_i est relié à un sous-ensemble de A par un biparti complet).
- Il ne reste finalement que A, qui est une clique.

On remarque que le graphe ainsi construit possède une décomposition arborescente de hauteur 4.
Nous prouvons maintenant que G a une clique de taille k si et seulement si G' contient k' sommets induisant C' arêtes ou moins.

4.2.5 Équivalence des solutions

Nécessité

Supposons que $K \subseteq V$ est une clique de taille k dans G. Sans perdre de généralité, on suppose que $K = \{v_1, \cdots, v_k\}$. On note $E_0 = \{1, \cdots \binom{k}{2}\}$ les indices des arêtes de la clique, et $E_1 = \{(\binom{k}{2}+1), \cdots, m\}$ les autres indices. Nous construisons l'ensemble K' suivant (la construction est également illustrée dans la Figure 18 à droite, par les rectangles grisés) :

- Pour tout $i \in \{(k+1), \cdots, n\}$, nous ajoutons A_i à K'. Cela ajoute $n(n-k) = T$ sommets.
- Pour tout $i \in \{1, \cdots, m\}$, nous ajoutons X_i à K'. Cela ajoute mT sommets.
- Pour tout $i \in \{1, \cdots, m\}$, si $i \in E_0$ nous ajoutons Z_e à K', et si $i \in E_1$, nous ajoutons Y_e à K'. Nous ajoutons en tout mT sommets.

Il est facile de vérifier que K' contient $k' = 2mT+T$ sommets. Les arêtes induites sont les suivantes :

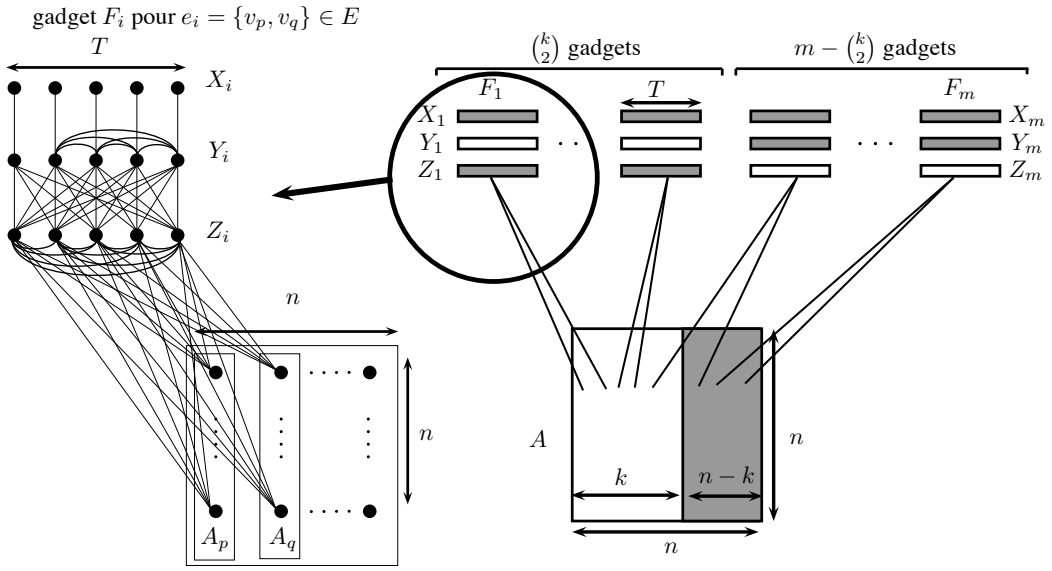


FIGURE 18 – Schéma de la réduction, avec un exemple de gadget F_i sur la gauche et ses adjacences envers A. Les rectanges gris représentent les sommets de la solution.

- $\binom{T}{2}$ arêtes induites par les sommets choisis parmi A.
- Tous les gadgets induisent au moins $\binom{T}{2}$ arêtes, sauf les gadgets de E_1 (il y en a $m - \binom{k}{2}$) qui en induisent une de plus (*c.f.* la construction du gadget en Sous-Section 4.2.3).
- Par construction, la solution ne comporte aucune arête entre les sommets choisis parmi A et les sommets choisis parmi les gadgets.

D'où l'ensemble induit bien C' arêtes.

Suffisance

Supposons maintenant que K' est un ensemble de k' sommets de G' induisant C' arêtes ou moins. Nous re-définissons les ensembles E_0 et E_1 de la manière suivante : $E_0 = \{i \in \{1, \cdots, m\}$ où, si $e_i = \{v_p, v_q\}$, alors $\forall j \in \{1, \cdots, n\}$ on a $a_p^j \notin K'$ et $a_q^j \notin K'\}$, et $E_1 = \{1, \cdots m\} \setminus E_0$.
En fait, E_0 représente les indices des gadgets qui *ne* sont *pas* adjacents à des sommets de la solution parmi A (gadgets de gauche dans la Figure 18), et E_1 représente les indices des gadgets qui sont adjacents à au moins un sommet de la solution parmi A (gadgets de droite dans la Figure 18).

La preuve consiste à modifier K' afin d'obtenir une solution de la même « forme » que précédemment. Pour cela, nous allons remplacer certains sommets de K' par d'autres sommets de $V' \setminus K'$ tels que le coût (en terme de nombre d'arêtes induites)

de la nouvelle solution n'augmente pas. Pour cela, nous allons utiliser la notion de *remplacement sûr*.
Pour tout $R \subseteq V'$, on note $tr(R) = K' \cap R$ la trace de R dans K', et pour tout $v \in V'$, on note $\mu(v) = |tr(N(v))|$ le nombre de voisins de v appartenant à la solution K'.

Définition 33. *Soient $u \in K'$ et $v \in V' \setminus K'$. On dit que (u,v) est un remplacement sûr si on a :*

- $\mu(v) \leq \mu(u)$ *si u et v ne sont pas adjacents dans G'.*
- $\mu(v) - 1 \leq \mu(u)$ *si u et v sont adjacents dans G'.*

Il est facile de voir que si (u,v) est un remplacement sûr, alors $(K' \setminus \{u\}) \cup \{v\}$, la solution obtenue en retirant u de K' et en ajoutant v, a un coût au moins aussi petit que K'. Comme dit précédemment, le reste de la preuve consiste à transformer K' via des remplacements sûrs afin d'obtenir une solution encodant un ensemble de sommets et d'arêtes dans G, qui formeront une clique si K' est bien un ensemble de k' sommets qui induit moins de C' arêtes dans G' comme supposé. Pour des raisons de lisibilité, nous maintiendrons et mettrons à jour les définitions de E_0 et E_1 au fil des remplacements de sommets (sommets de A plus particulièrement) : par exemple, si nous retirons de K' un sommet $u \in A$, tel que ce sommet était le seul voisin d'un gadget F_i, alors i passe de E_1 à E_0. De même, après un remplacement sûr, K' désigne la nouvelle solution obtenue.

Ces différents remplacements sont effectués dans les Lemmes 7, 8 et 9, nous permettant d'obtenir quatre cas différents concernant la forme de la solution K'. Enfin, pour chacun de ces quatre cas nous montrons que les conditions du Lemme 10 sont vérifiées, lemme qui prouve l'existence d'une clique de taille k dans G.

Lemme 7. *On peut supposer que pour tout $i \in \{1, \cdots, m\}$, on a $X_i \subseteq K'$.*

Preuve. Soit $S = \bigcup_{i=1}^{m} X_i$. Comme $k' > |S|$, il existe toujours $u \in K' \setminus S$. Supposons qu'il existe $i \in \{1, \cdots, m\}$ et $j \in \{1, \cdots, T\}$ tel que $x_j^i \notin K'$.

- Dans ce cas, si son voisin y_j^i n'appartient pas non plus à K', alors on a $\mu(x_j^i) = 0$ et (z, u) est un remplacement sûr quel que soit $z \in K' \setminus S$.
- Si en revanche $y_j^i \in K'$, alors dans ce cas $\mu(x_j^i) = 1$. Mais comme x_j^i et y_j^i sont connectés, (y_j^i, x_j^i) est un remplacement sûr.

Après ces remplacements, on a bien $X_i \subseteq K'$ pour tout $i \in \{1, \cdots, m\}$.

□

Lemme 8. *On peut supposer que nous nous trouvons dans l'un des cas suivants :*

- *Cas A1 : pour tout $i \in E_0$, on a $tr(Z_i) = Z_i$.*
- *Cas A2 : pour tout $i \in E_0$, on a $tr(Y_i) = \emptyset$.*

Preuve. Nous allons d'abord restructurer chaque gadget de E_0 indépendamment, puis les gadgets entre eux.
Soit $i \in E_0$ tel que $tr(Y_i) \neq \emptyset$ et $tr(Z_i) \neq Z_i$, et soit $j_0 = \max\{j \in \{1, \cdots, T\} : y_j^i \in tr(Y_i)\}$ et j_1 tel que $z_{j_1}^i \notin tr(Z_i)$.
Rappelons d'abord que d'après le Lemme 7, on a $x_{j_0}^i \in K'$. Si $j_0 \neq 1$, alors $\mu(y_{j_0}^i) = y + z + 1$, avec $y = |tr(Y_i) \cap N(y_{j_0}^i)|$ et $z = |tr(Z_i) \cap N(y_{j_0}^i)| = |tr(Z_i)|$.
En outre, on a $\mu(z_{j_1}^i) \leq y + z + 1$ (plus précisément : $\mu(z_{j_1}^i) = y + z + 1$ si $y_1^i \in K'$, et $\mu(z_{j_1}^i) = y + z$ sinon).
Informellement, ce remplacement va consister à « perdre » l'arête reliée au sommet de X_i, et à « récupérer » au plus une arête, due à y_1^i. On a donc $\mu(z_{j_1}^i) \leq \mu(y_{j_0}^i)$, et $(z_{j_1}^i, y_{j_0}^i)$ est un remplacement sûr.
Si $j_0 = 1$, alors on a $tr(Y_i) = \{y_1^i\}$. Supposons qu'il existe j_1 tel que $z_{j_1}^i \notin tr(Z_i)$. On a alors $\mu(y_1^i) = z + 1$ où $z = |N(y_1^i) \cap tr(Z_i)|$, et $\mu(z_{j_1}^i) = z + 1$. Ici aussi $(z_{j_1}^i, y_{j_0}^i)$ est un remplacement sûr.
Après tous ces remplacements, pour tout $i \in E_0$ on a $tr(Y_i) \neq \emptyset \implies tr(Z_i) = Z_i$.
On procède maintenant à des remplacements entre deux gadgets de E_0 : s'il existe $a, b \in E_0$ tels que $tr(Y_a) \neq \emptyset$ et $tr(Z_b) \neq Z_b$, alors soit j_0 tel que $y_{j_0} \in tr(Y_a)$, et soit j_1 tel que $z_{j_1}^b \notin tr(Z_b)$. On a alors $\mu(y_{j_0}^a) \geq T + 1$ et $\mu(z_{j_1}) \leq T - 1$, et $(z_{j_1}^i, y_{j_0}^i)$ est un remplacement sûr.
Ainsi, ces remplacements se terminent soit lorsque $tr(Y_i) = \emptyset$ pour tout $i \in E_0$, ou lorsque $tr(Z_i) = Z_i$ pour tout $i \in E_0$, ce qui termine la preuve du lemme. □

Lemme 9. *On peut supposer que nous nous trouvons dans l'un des cas suivants :*

- *Cas B1 : pour tout $i \in E_1$, on a $tr(Y_i) = Y_i$.*
- *Cas B2 : pour tout $i \in E_1$, on a $tr(Z_i) = \emptyset$.*

Preuve. Informellement, la preuve repose sur le fait que remplacer un sommet de Z_i par un sommet de Y_i nous permet de « supprimer » au moins une arête avec les sommets de A, et de « récupérer » une arête avec un sommet de X_i.
De la même manière que pour le Lemme 8, on restructure dans un premier temps chaque gadget indépendamment : soit $i \in E_1$ tel que $tr(Z_i) \neq \emptyset$ et $tr(Y_i) \neq Y_i$. Soit $j_0 = \max\{j \in \{1, \cdots, T\} : y_j^i \notin K'\}$ et soit j_1 tel que $z_{j_1}^i \in tr(Z_i)$.
Rappelons que d'après la définition de E_1, il existe $u, r \in \{1, \cdots, n\}$ tel que $z_{j_1}^i$ est adjacent à a_u^r. On a donc :

$$\mu(z_{j_1}^i) \geq y + z + 1,$$

avec $y = |N(z_{j_1}^i) \cap Y_i|$ et $z = |N(z_{j_1}^i) \cap Z_i|$. En outre, on a :

$$\mu(y_{j_0}^i) \leq z + y + 2,$$

en effet, $|N(y^i_{j_0}) \cap Z_i| = z+1$, $|N(y^i_{j_0}) \cap Y_i| \leq y$ et $|N(y^i_{j_0}) \cap X_i| = 1$. Comme $y^i_{j_0}$ et $z^i_{j_1}$ sont adjacents, $(z^i_{j_1}, y^i_{j_0})$ est un remplacement sûr.
Après tous ces remplacements, étant donné $i \in E_1$, on a $tr(Z_i) \neq \emptyset \implies tr(Y_i) = Y_i$. On restructure maintenant les gadgets entre eux. Supposons qu'il existe $a, b \in E_1$ tels que $tr(Z_a) \neq \emptyset$ et $tr(Y_b) \neq Y_b$. Soit j_0 tel que $y^b_{j_0} \notin tr(Y_b)$ et j_1 tel que $z^a_{j_1} \in tr(Z_a)$. On a alors $\mu(z^a_{j_1}) \geq T+1$ et $\mu(y^b_{j_0}) \leq T-1$, et $(y^b_{j_0}, z^a_{j_1})$ est un remplacement sûr, et ces remplacements se terminent soit lorsque $tr(Y_i) = Y_i$ pour tout $i \in E_1$, soit lorsque $tr(Z_i) = \emptyset$ pour tout $i \in E_1$, comme demandé. □

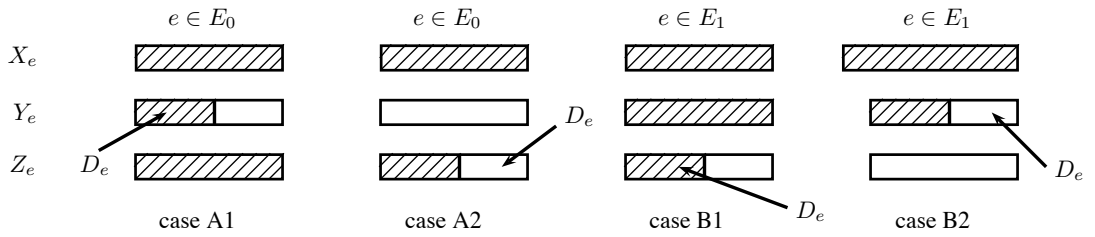


FIGURE 19 – Schéma des différents cas. Les rectangles hachurés représentent les sommets de K'.

Définissons maintenant pour chaque cas et chaque gadget $i \in \{1, \cdots, m\}$ l'ensemble de sommets $D_i \subseteq Y_i \cup Z_i$ qui doivent être remplacés (*c.f.* Figure 19) :

- Cas $A1$: pour tout $i \in E_0$, $D_i = tr(Y_i)$.
- Cas $A2$: pour tout $i \in E_0$, $D_i = Z_i \setminus K'$.
- Cas $B1$: pour tout $i \in E_1$, $D_i = tr(Z_i)$.
- Cas $B2$: pour tout $i \in E_1$, $D_i = Y_i \setminus K'$.

On remarque que si $D_i = \emptyset$ pour tout $i \in E_0$ (resp. $i \in E_1$), alors les cas $A1$ et $A2$ (resp. $B1$ et $B2$) se confondent. Si une telle éventualité se produit pour tout $i \in \{1, \cdots, m\}$, alors nous pouvons directement conclure, comme le montre le lemme suivant.

Lemme 10. *Si $D_i = \emptyset$ pour tout $i \in \{1, \cdots, m\}$, alors G contient une clique de taille k.*

Preuve. Par construction on a $|tr(A)| = T$ et $|tr(F_i)| = 2T$ pour tout $i \in \{1, \cdots, m\}$. Ainsi, $cost(tr(A)) = \binom{T}{2}$ et $cost(tr(F_i)) = \binom{T}{2} + 1$ si $Y_i \subseteq K'$, et $cost(tr(F_i)) = \binom{T}{2}$ si $Z_i \subseteq K'$. Par construction, $Y_i \subseteq K'$ si et seulement si $i \in E_1$. Donc, puisque $cost(K') \leq \binom{T}{2} + m\binom{T}{2} + m - \binom{k}{2}$, on doit forcément avoir $|E_1| \leq m - \binom{k}{2}$, qui est équivalence à $|E_0| \geq \binom{k}{2}$. Ainsi, il existe au plus $\lceil \frac{|A|-T}{n} \rceil = k$ sommets dans G qui

induisent au moins $\binom{k}{2}$ arêtes. Autrement dit, G contient une clique de taille k. □

Le reste de la preuve consiste maintenant à analyser les quatre cas induits par les Lemmes 8 et 9.

Cas A1 et B1. Résumons la situation : la solution K' peut être partitionnée en $K'_A = tr(A)$, les sommets sélectionnés parmi A, et $K'_F = K' \setminus K'_A$, les sommets sélectionnés parmi les gadgets. Soit $\Delta_0 = \sum_{i \in E_0} |D_i|$ le nombre de sommets « en trop » dans les gadgets F_i, pour $i \in E_0$, et $\Delta_1 = \sum_{i \in E_1} |D_i|$ le nombre de sommets « en trop » dans les gadgets F_i, pour $i \in E_1$, et soit $\Delta = \Delta_0 + \Delta_1$. Remarquons que $|K'_A| = T - \Delta$, étant donné qu'une solution « normale » qui ne prend aucun sommet supplémentaire dans les gadgets doit prendre T sommets dans A. De plus :

- Les sommets de K' des gadgets indicés par E_0 ne sont pas adjacents à K'_A (par définition de E_0).
- Chaque gadget de E_0 induit au moins $\binom{T}{2}$ arêtes (étant donné que nous sommes dans le cas $A1$).
- Chaque gadget de E_1 induit au moins $\binom{T}{2} + 1$ arêtes (étant donné que nous sommes dans les cas $B1$).
- Chacun des Δ_0 sommets supplémentaires dans les gadgets $i \in E_0$ est adjacent à au moins $(T+1)$ sommets dans K' (un tel sommet est forcément dans Y_i, et est donc connecté à tous les sommets de Z_i qui sont dans la solution).
- Chacun des Δ_1 sommets supplémentaires dans les gadgets $i \in E_1$ est adjacent à au moins $(T+1)$ sommets dans K' (un tel sommet est dans Z_i, et est donc connecté aux T sommets de Y_i et à au moins un sommet de K'_A).

Ces observations nous permettent de minorer le coût total de K' de la manière suivante :

$$\begin{aligned} cost(K') &\geq |E_0|\binom{T}{2} + |E_1|(\binom{T}{2} + 1) + \Delta_0(T+1) + \Delta_1(T+1) + \binom{T-\Delta}{2} \\ &= m\binom{T}{2} + |E_1| + \Delta T + \Delta + \binom{T}{2} - \binom{\Delta}{2} - \Delta(T-\Delta) \\ &\geq m\binom{T}{2} + (m - |E_0|) + \binom{T}{2} + \frac{\Delta^2}{2} \end{aligned}$$

Informellement, dans une « mauvaise » solution (c'est-à-dire si les D_i ne sont pas vides), plus Δ est grand, moins de sommets nous pouvons prendre dans A (nous pourrons en fait en prendre $T - \Delta$, au lieu de T), et nous obtiendrons donc plus de gadgets dans E_0 (au moins plus que $\binom{k}{2}$), ce qui nous permet à

priori de faire diminuer le coût (les gadgets de E_0 coûtent une arête de moins que ceux de E_1). Cependant, comme nous allons le voir, le coût engendré par les sommets supplémentaires dans les gadgets est supérieur à celui gagné par le fait d'avoir plus de gadgets dans E_0.
Supposons que G ne contienne pas de clique de taille k, et montrons que K' induit forcément strictement plus que C' arêtes.
Effectuons d'abord la division euclidienne de Δ par n : soient q et r tels que $\Delta = qn + r$, avec $r < n$. Majorons maintenant $|E_0|$: comme il y a $T - \Delta$ sommets dans K'_A, le nombre de « colonnes » vides (une colonne A_u est vide si aucun des sommets de A_u n'est pris dans la solution) est au plus $\lceil n - \frac{T-\Delta}{n} \rceil \leq k + q$. Comme G ne contient pas de clique de taille k, les au plus $(k + q)$ sommets correspondant à ces $(k + q)$ colonnes ne peuvent pas induire une clique de taille $k + q$, et donc $|E_0| < \binom{k+q}{2}$. Nous obtenons ainsi :

$$\begin{aligned} cost(K') &> m\binom{T}{2} + (m - \binom{k+q}{2}) + \binom{T}{2} + \frac{\Delta^2}{2} \\ &= C' - (\binom{q}{2} + kq) + \frac{\Delta^2}{2} \end{aligned}$$

Et, comme $\frac{\Delta^2}{2} \geq \binom{q}{2} + kq$, on a $cost(K') > C'$ comme désiré (sauf si $\Delta = 0$ mais alors le Lemme 10 s'applique).

Cas A2 et B2. Comme précédemment, soient $\Delta_0 = \sum_{i \in E_0} |D_i|$, $\Delta_1 = \sum_{i \in E_1} |D_i|$, et $\Delta = \Delta_0 + \Delta_1$ (remarquons que cette fois-ci $D_i \not\subseteq K'$ pour tout $i \in \{1, \cdots, m\}$), et supposons que $\Delta > 0$. Pour tout $u \in tr(A)$, on a clairement $\mu(u) \geq T$. D'un autre côté, pour tout $i \in \{1, \cdots, m\}$ tel qu'il existe $v \in D_i$, on a $\mu(v) \leq T$ (car si $i \in E_1$, alors $D_i \subseteq Y_i$, et si $i \in E_0$, alors v n'est pas adjacent à $tr(A)$ par définition de E_0). Ainsi, (v, u) est un remplacement sûr. Étant donné qu'avant ce remplacement on avait $|tr(A)| = T + \Delta$, il est clair que l'on peut répéter ces remplacements Δ fois (c'est-à-dire (v, u), avec $u \in tr(A)$ et $v \in D_i$ pour un certain $i \in \{1, \cdots, m\}$). Après ceux-ci, la nouvelle valeur de Δ est 0, *i.e.* $D_i = \emptyset$ pour tout $i \in \{1, \cdots, m\}$. On peut ainsi appliquer le Lemme 10 et G a une clique de taille k.

Cas A2 et B1. S'il existe un couple $i \in E_0$ et $u \in D_i$, alors $\mu(u) < T$. Si un tel sommet existe, alors soit $|tr(A)| > T$, ou il existe $i' \in E_1$ tel qu'il existe $v \in D_{e'}$. Dans le premier cas, pour tout $x \in tr(A)$ on a $\mu(x) \geq T$ et donc (x, u) est un remplacement sûr. Dans le second cas on a $\mu(v) > T$ et ici aussi (v, u) est un remplacement sûr.
Après ces remplacements, on a clairement $D_i = \emptyset$ pour tout $i \in E_0$ et on peut appliquer ici aussi le Lemme 10.

Cas A1 et B2. Ce cas est similaire au précédent. S'il existe $i \in E_1$ tel qu'il existe $u \in D_i$, alors $\mu(u) < T$. Si un tel sommet existe, alors soit $|tr(A)| > T$, ou il

existe $i' \in E_0$ tel qu'il existe $v \in D_{i'}$. Dans le premier cas, pour tout $x \in tr(A)$, on a $\mu(x) \geq T$, et (x, u) est un remplacement sûr. Dans le second cas, on a $\mu(v) > T$ et ici aussi (v, u) est un remplacement sûr.
Après ces remplacements, on a clairement $D_i = \emptyset$ pour tout $i \in E_1$ et on peut appliquer ici aussi le Lemme 10.

Nous obtenons enfin le théorème suivant :

Théorème 33. SPARSEST k-SUBGRAPH *est $\mathcal{NP}$-difficile dans les graphes chordaux.*

Et le corollaire suivant pour MAXIMUM k-COVERAGE, étant donné que les deux problèmes sont équivalents :

Corollaire 7. MAXIMUM k-COVERAGE *est $\mathcal{NP}$-difficile dans les graphes chordaux.*

4.3 Dans les graphes chordaux : approximation

4.3.1 Présentation de l'algorithme

Cette section s'intéresse à la résolution approchée avec garantie de performance de SPARSEST k-SUBGRAPH dans les graphes chordaux. Tout d'abord, il est clair que tout algorithme doit retourner un ensemble indépendant de taille k si celui-ci existe, la fonction objectif valant alors 0 dans ce cas. Ainsi, une idée naturelle d'algorithme est de chercher tout d'abord un ensemble indépendant S de taille maximum (ce qui se fait en temps polynomial dans les graphes chordaux). Si celui-ci est de taille k ou plus, alors l'algorithme est terminé. Sinon, plusieurs stratégies sont possibles.

Une première est de supprimer cet ensemble indépendant du graphe, chercher à nouveau un ensemble indépendant maximum, et de répéter ces étapes jusqu'à obtenir k sommets. En fait, cette approche est exactement l'algorithme donnant une solution 3-approchée pour DENSEST k-SUBGRAPH dans les graphes chordaux [89] (en cherchant des cliques au lieu d'ensembles indépendants). Malheureusement, cet algorithme peut être arbitrairement mauvais concernant SPARSEST k-SUBGRAPH, comme le montre la Figure 20, même dans le cas des graphes d'intervalles. Cependant, nous verrons par la suite (Section 4.6.3) qu'il donne une solution 2-approchées dans les graphes d'intervalles propres.
Une autre idée naturelle pour contrer le défaut de l'algorithme précédent est la suivante : après avoir choisi un premier ensemble indépendant maximum, nous affectons des poids aux sommets restants, correspondant à la taille de leur voisinage dans la solution partielle que nous sommes en train de construire. Ainsi, à chaque étape, l'algorithme consiste à choisir un ensemble indépendant (appelé *couche* par la suite) parmi les sommets de poids minimum. Il met ensuite à jour les poids des sommets restants comme indiqué précédemment (pour des raisons techniques que nous détaillerons plus bas, le poids n'est en fait pas exactement égal à la taille du

voisinage dans la solution). La définition formelle de l'algorithme est donnée dans la prochaine sous-section. Avant cela, nous décrivons d'abord l'idée de l'analyse.

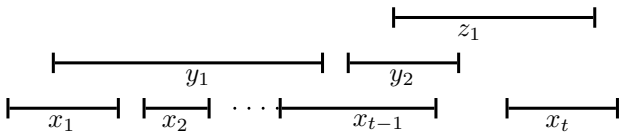


FIGURE 20 – Dans cet exemple, prendre des ensembles indépendants maximum successifs donne un ratio d'approximation non borné. Pour $k = t + 2$, l'algorithme choisit la solution $\{x_1, ..., x_t, y_1, y_2\}$ de coût t alors que la solution $\{x_1, ..., x_t, y_2, z_1\}$ est de coût quatre.

L'idée de la preuve est de restructurer une solution optimale S^* afin d'obtenir la solution retournée par l'algorithme S, en majorant l'augmentation du coût à chacune des restructurations. Pour cela, nous transformons S^* petit à petit pour introduire chaque couche de S les unes après les autres, en gardant toujours au fil des modifications une solution de taille k. Observons maintenant l'intuition de la preuve pour la première couche : soit $L = \{n_1, \cdots, n_{|L|}\}$ le premier ensemble indépendant choisi par l'algorithme. Pour chaque n_j qui n'appartient pas à S^*, nous supprimons de S^* le premier voisin de n_j à sa droite dans l'ordre d'élimination (et qui appartient à S^*), et ajoutons n_j à la place. Comme on peut le voir dans la Figure 21, le degré de $x \in S^*$ ($x \notin L$) ne va augmenter que de 1 dans la solution. Pour les autres couches, l'idée générale restera la même, mais les poids viendront en revanche compliquer la tâche.

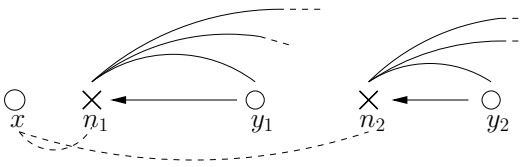


FIGURE 21 – Idée de la restructuration d'une solution optimale S^*. Les cercles représentent les sommets de S^*, alors que les croix représentent les sommets de L choisis par l'algorithme. En remplaçant y_1 par n_1 et y_2 par n_2, le degré d'un sommet x qui n'est pas concerné par cette restructuration ne peut augmenter que d'un. En effet, x ne peut pas être connecté à n_1 et n_2, étant donné que L est un ensemble indépendant. Notons que les sommets sont représentés de gauche à droite par rapport à l'ordre d'élimination choisi du graphe.

4.3.2 L'algorithme et son analyse

Présentation

Nous proposons l'Algorithme 3. Comme décrit informellement ci-dessus, il prend successivement un ensemble indépendant parmi les sommets de poids minimum, et met à jour les poids en fonction de la solution partielle. En revanche, pour des raisons techniques, les poids ne sont pas exactement égaux à la taille du voisinage dans l'ensemble de sommets auparavant choisis. En effet, alors de la restructuration d'une solution optimale afin d'insérer une couche L_i de la solution de l'algorithme, nous verrons que le degré de presque tous les sommets non concernés par cette restructuration sera augmenté d'au plus un. Pour cette raison, nous ajoutons un « bonus » de -1 au poids de ces sommets (*c.f.* Ligne 13). Cependant, le degré de certains autres sommets n'augmentant pas, nous ne leur donnons pas de bonus (*c.f.* Ligne 11). Ces bonus nous permettent de montrer qu'à la fin de l'algorithme, la valeur W retournée (correspondant à la somme des poids des sommets choisis) est une borne inférieure du coût optimal (*c.f.* Lemme 13). Il nous suffira enfin de montrer que la véritable valeur de la solution retournée par l'algorithme est inférieure à $2W$ (Lemme 14), ce qui prouvera que l'Algorithme 3 est une 2-approximation.

Algorithme 3 Une 2-approximation pour SPARSEST k-SUBGRAPH dans les graphes chordaux.

1: $S \leftarrow \emptyset$, $W \leftarrow 0$, $i \leftarrow 0$, $w_0(x) = 0\ \forall x \in V$
2: **tant que** $|S| \leq k$ **faire**
3: $L_i \leftarrow$ un ens. indép. max. de $G[\{x \in V \backslash (L_0 \cup ... \cup L_{i-1}) : w_i(x) = i\}]$
4: $S \leftarrow S \cup L_i$ // ou les $(k - |S \cup L_i|)$ premiers sommets de L_i si $|S \cup L_i| > k$
5: $W \leftarrow W + i|L_i|$ // mise à jour des poids
6: **pour** $x \in V$ **faire**
7: **si** $x \in (L_0 \cup ... \cup L_i)$ **alors**
8: $w_{i+1}(x) = w_i(x)$
9: **sinon**
10: **si** $d(x, L_i) = 0$ OU $(d(x, L_i) = 1$ ET $w_i(x) = i)$ **alors**
11: $w_{i+1}(x) = w_i(x) + d(x, L_i)$
12: **sinon**
13: $w_{i+1}(x) = w_i(x) + d(x, L_i) - 1$
14: **fin si**
15: **fin si**
16: **fin pour**
17: $i \leftarrow i + 1$
18: **fin tant que**
19: $t \leftarrow i - 1$ // L_t est la dernière couche de l'algorithme
20: **retourner** (S,W)

Remarque 1. *L'ensemble indépendant de taille maximale construit à la ligne 3 est construit de manière gloutonne de la manière suivante : on prend le premier sommet de l'ordre d'élimination simplicial, on supprime son voisinage, et on répète l'opération jusqu'à ce que le graphe devienne vide. Il est facile de vérifier que l'ensemble ainsi obtenu est un ensemble indépendant, et qu'il est de taille maximale (c.f. Section 1.3.2).*

Montrons d'abord que l'ajout des bonus ne font pas trop décroître les poids des sommets :

Lemme 11. $\forall i \in \{0, \cdots, t\}$, $\forall x \in V \setminus \{L_0, \cdots, L_i\}$, *on a* $w_{i+1} \geq i + 1$.

Preuve. Soient i et x tels que décrits dans le lemme, et supposons par induction que $w_i(x) \geq i$ (par définition $w_0(x) = 0$). Si $w_i(x) \geq i + 1$, alors l'inégalité est vraie. Autrement, $w_i(x) = i$, et par construction de l'algorithme (Ligne 11), si $d(x, L_i) \geq 1$, alors $w_{i+1}(x) \geq i + 1$. Enfin, si par contre $d(x, L_i) = 0$, alors x doit forcément appartenir à L_i, ce qui contredit la définition de x. □

Restructuration de solutions optimales et preuve du rapport

Avant de se lancer dans le cœur de la preuve, plusieurs notations sont nécessaires. Soit S^* une solution optimale pour notre problème. La preuve consiste à modifier S^* pour obtenir la solution de l'algorithme, en bornant l'augmentation du coût lors de ces modifications. On définit ainsi par récurrence une série de solutions $(S_i^*)_{i=-1,\cdots,t}$, avec $S_{-1}^* = S^*$ la solution optimale et $S_t^* = S$ la solution obtenue par l'algorithme. Pour tout $i \in \{-1, \cdots, t\}$, on a $S_i^* \subseteq V$ et $|S_i^*| = k$. Comme dit précédemment, les restructurations s'opère en insérant successivement chaque couche de S dans S^*, ainsi, pour chaque $i \in \{-1, \cdots, t\}$, on a $(L_0, \cdots, L_i) \subseteq S_i^*$. Montrons maintenant comment pour chaque étape $i \in \{0, \cdots, t\}$, on restructure S_{i-1}^* en S_i^*.
Nous partitionnons d'abord la couche L_i en deux ensembles de sommets M_i et N_i, selon s'ils appartiennent à S_{i-1}^* ou non : $L_i = M_i \cup N_i$, avec $M_i = L_i \cap S_{i-1}^*$ (et donc $N_i = L_i \setminus S_{i-1}^*$).
La restructuration consiste ainsi à ajouter les sommets de N_i à S_{i-1}^*, en supprimant un ensemble de sommets bien choisi (*c.f.* Définition 34) $D_i \subseteq S_{i-1}^*$ (avec $|D_i| = |N_i|$, afin d'avoir toujours un ensemble de taille k). On aura alors ensuite $S_i^* = (S_{i-1}^* \setminus D_i) \cup N_i$.
En outre, nous posons les définitions suivantes : $R_i = S_{i-1}^* \setminus (D_i \cup L_0 \cup \cdots \cup L_i)$, et $T_i = M_i \cup D_i$. En fait, R_i représente les sommets de la solution optimale qui n'ont pas encore été restructurés, et qui n'ont pas été concernés par la $i^{ème}$ restructuration, et T_i représente l'ensemble des sommets de la solution optimale concernés par la $i^{ème}$ restructuration. La situation est résumée par la Figure 22.
Afin de borner l'augmentation du coût lors des restructurations, nous montrons dans le Lemme 12 que le degré des sommets de R_i n'augmente que d'au plus un. La définition suivante montre comment choisir l'ensemble D_i :

Définition 34. *Soit $i \in \{0, \cdots, t\}$, posons $N_i = \{n_1, \cdots, n_{p_i}\}$, et supposons que les sommets sont ordonnés selon l'ordre d'élimination simplicial du graphe : $n_1 < \cdots < n_{p_i}$. Pour tout $j = 1, \cdots, p_i$ successivement, on choisit un sommet $y_j \in S^*_{i-1} \setminus (L_0 \cup \cdots \cup L_i)$ comme suit :*

$$y_j = \begin{cases} \min(Q_j) & si \quad Q_j \neq \emptyset \\ \max(S^*_{i-1} \setminus (L_0 \cup ... \cup L_i \cup \{y_1, ..., y_{j-1}\})) & si \quad Q_j = \emptyset \end{cases} \tag{4.1}$$

avec $Q_j = \{y \in S^*_{i-1} \setminus (L_0 \cup \cdots L_i \cup \{y_1, \cdots, y_{j-1}\})$ tel que $n_j < y$, et $\{n_j, y\} \in E\}$ (c.f. Figure 23). Enfin, on définit $D_i = \{y_j : i \in \{1, \cdots, p_i\}\}$.

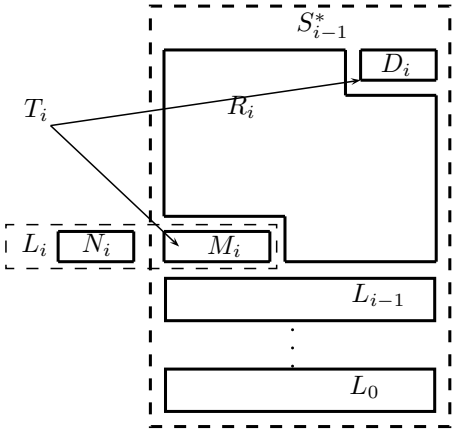


FIGURE 22 – Résumé des notations concernant l'ensemble S^*_{i-1}. Nous obtenons S^*_i à partir de S^*_{i-1} en supprimant D_i et en ajoutant N_i. Remarquons que $R_{i-1} = R_i \cup T_i$, et $R_i \cap T_i = \emptyset$.

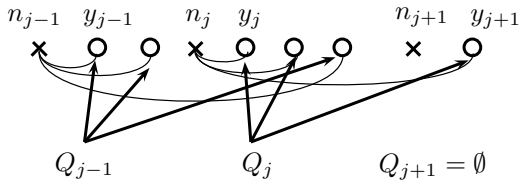


FIGURE 23 – Exemple d'ensembles Q_j, avec les sommets y_j respectifs. Les sommets sont représentés de gauche à droite en fonction de l'ordre d'élimination simplicial. Les cercles représentent les sommets de la solution optimale considérée, et les croix représentent les sommets choisis par l'algorithme qui ne sont pas dans la solution optimale. Les arêtes entre les sommets de la solution optimale ont été omises par soucis de clarté.

Il est facile de vérifier que $|D_i| = |N_i|$, puisque tous les y_j sont distincts. Maintenant que D_i est défini, nous pouvons borner l'augmentation du degré des sommets de R_i. On rappelle que $T_i = M_i \cup D_i$, et par définition $R_i = R_{i-1} \setminus T_i$.

Lemme 12. *Soit $R_i = A_i \cup B_i$, avec $A_i = \{x \in R_i : d(x, L_i) = 0$ ou $(d(x, L_i) = 1$ et $w_i(x) = i)\}$, et $B_i = R_i \setminus A_i$. On a :*

- *si $x \in A_i$, alors $d(x, L_i) \leq d(x, T_i)$,*
- *si $x \in B_i$, alors $d(x, L_i) \leq d(x, T_i) + 1$.*

Ce qui implique immédiatement que pour tout $x \in R_i$, on a $w_{i+1}(x) \leq d(x, T_i) + w_i(x)$.

Preuve. Montrons que si $x \in A_i$, alors $d(x, N_i) \leq d(x, D_i)$, et si $x \in B_i$, alors $d(x, N_i) \leq d(x, D_i) + 1$. Comme $L_i = M_i \cup N_i$, et $T_i = M_i \cup D_i$, et que ces unions sont disjointes, cela prouvera les inégalités demandées.

- Si $x \in A_i$, alors doit $d(x, L_i) = 0$, ce qui implique évidemment le résultat, ou $d(x, L_i) = 1$ et $w_i(x) = i$. Dans ce cas également, si $d(x, N_i) = 0$ le résultat est immédiat, donc supposons que $d(x, N_i) = 1$, *i.e.* qu'il existe un sommet n_{j_0} de N_i tel que x et n_{j_0} sont adjacents. Deux cas sont alors possibles :
  - Premier cas : $x < n_{j_0}$. Rappelons que par définition, n_{j_0} est l'unique voisin de x dans L_i. Ainsi, x n'est pas adjacent aux sommets de L_i qui sont avant n_{j_0} dans l'ordre. De plus, rappelons que $w_i(x) = i$. Ainsi, ce cas ne peut pas arriver puisque par définition de l'algorithme, x aurait dû être pris dans L_i à la place de n_{j_0}.
  - Second cas : $n_{j_0} < x$. Il est clair que $Q_j \neq \emptyset$ (puisque au moins $x \in Q_{j_0}$). De plus, sachant que $x \notin D_i$ par définition, on a $y_{j_0} < x$. Or, comme l'ordre choisi est un ordre d'élimination simplicial, puisque $\{n_{j_0}, y_{j_0}\} \in E$ et $\{n_{j_0}, x\} \in E$, on doit forcément avoir $\{x, y_{j_0}\} \in E$. D'où $d(x, D_i) = 1$ et le résultat suit.
- Si $x \in B_i$, alors soient $N_i^- = \{y \in N_i : y < x\}$, et $N_i^+ = N_i \setminus N_i^-$. Pour tout $n_j \in N_i^-$ tel que $\{n_j, x\} \in E$, alors comme précédemment $Q_j \neq \emptyset$, et par définition de la chordalité du graphe, on a $\{y_j, x\} \in E$ avec $y_j \in D_i$. Ainsi, supposons qu'il existe $n_{j_1}, n_{j_2} \in N_i^+$ tel que $\{x, n_{j_1}\}, \{x, n_{j_2}\} \in E$. Par définition de l'ordre d'élimination simplicial, on doit avoir $\{n_{j_1}, n_{j_2}\} \in E$, ce qui est impossible car L_i est un ensemble indépendant. Ceci prouve que $d(x, N_i) \leq d(x, D_i) + 1$.

□

On introduit maintenant la fonction ζ qui calcule le coût d'une solution intermédiaire S_i^* en fonction des poids des sommets. Pour tout $i \in \{0, \cdots, t\}$ on pose

$$\zeta(S_i^*) = cost(R_i) + \sum_{x \in R_i} w_{i+1}(x) + \sum_{x \in L_0 \cup \cdots \cup L_i} w_{i+1}(x).$$

Par extension, on a $\zeta(S_{-1}^*) = cost(S^*)$ et $\zeta(S_t^*) = \sum_{x \in S} w_t(x) = W$.

Lemme 13. *Pour tout $i \in \{0, \cdots, t\}$, D_i est tel que $\zeta(S_i^*) \leq \zeta(S_{i-1}^*)$.*

Preuve. En effet, on a :

$$\begin{array}{rcll} \zeta(S_i^*) & = & cost(R_i) + \sum_{x \in R_i} w_{i+1}(x) + \sum_{x \in L_0 \cup \cdots \cup L_i} w_{i+1}(x) & \\ & = & cost(R_i) + \sum_{x \in R_i} w_{i+1}(x) + i|L_i| & \\ & & \quad + \sum_{x \in L_0 \cup \cdots \cup L_{i-1}} w_{i+1}(x) & \\ & \leq & cost(R_i) + \sum_{x \in R_i} (w_i(x) + d(x,T_i)) + i|L_i| & \\ & & \quad + \sum_{x \in L_0 \cup \cdots \cup L_{i-1}} w_i(x) & \text{par le Lemme 12} \end{array}$$

De plus, sachant que $R_{i-1} = R_i \cup T_i$ et $|T_i| = |L_i|$, on a :

$$\begin{array}{rcl} \zeta(S_{i-1}^*) & = & cost(R_{i-1}) + \sum_{x \in R_{i-1}} w_i(x) + \sum_{x \in L_0 \cup \cdots \cup L_{i-1}} w_i(x) \\ & = & cost(R_i) + cost(R_i, T_i) + \sum_{x \in R_i} w_i(x) + \sum_{x \in T_i} w_i(x) \\ & & \quad + \sum_{x \in L_0 \cup \cdots \cup L_{i-1}} w_i(x) \\ & \geq & cost(R_i) + cost(R_i, T_i) + \sum_{x \in R_i} w_i(x) + i|L_i| \\ & & \quad + \sum_{x \in L_0 \cup \cdots \cup L_{i-1}} w_i(x) \end{array}$$

qui rejoint la borne supérieure précédente. □

Le lemme précédent nous permet d'avoir $W = \zeta(S_t^*) \leq \zeta(S_{-1}^*) = cost(S^*)$. Ainsi, la dernière brique nécessaire pour montrer que l'Algorithme 3 est 2-approché est le résultat suivant :

Lemme 14. *$cost(S) \leq 2W$.*

Preuve. Informellement, lorsqu'une couche L_i est créée, le coût W est mis à jour en y ajoutant $i|L_i|$. Ainsi, pour tout $x \in L_i$, le coût W est incrémenté de i au lieu de $d(x, L_0 \cup \cdots \cup L_{i-1})$. Il suffit donc de montrer que pour tout $x \in L_i$ on a $d(x, L_0 \cup \cdots \cup L_{i-1}) \leq 2i$.
Soit $x \in L_i$. Pour tout $l \in \{0, \cdots, i\}$, posons $w_l = w_l(x)$ le poids de x avant la création de la couche L_l. Ainsi, les poids successifs de x est la suite $(x_0, \cdots, x_i)$, avec $x_0 = 0$ et $x_i = i$. Une fois que x est ajouté dans L_i, son poids n'est plus changé. Montrons par récurrence que pour tout $l \in \{1, \cdots, i\}$ on a $d(x, L_0 \cup \cdots \cup L_{l-1}) \leq x_l + l$.

- Pour le cas de base $l = 1$, remarquons par définition de l'algorithme que $x_1 = 1$ si $d(x, L_0) = 1$, et $x_1 = d(x, L_0) - 1$ sinon, on a donc dans tous les cas $x_1 \geq d(x, L_0) - 1$.
- Supposons maintenant que l'inégalité est vraie pour un certain l, et montrons la pour $l + 1$. Posons $z = d(x, L_l)$. On a $d(x, L_0 \cup \cdots \cup L_l) = d(x, L_0 \cup \cdots \cup L_{l-1}) + z \leq x_l + l + z$. Comme $x_{l+1} \geq x_l + z - 1$, l'inégalité suit.

Pour tout $x \in L_i$, on a donc $d(x, L_0 \cup \cdots \cup L_{i-1}) \leq x_i + i = 2i$, et donc

$$cost(S) = \sum_{i=1}^{t} \sum_{x \in L_i} d(x, L_0 \cup \cdots \cup L_{i-1}) \leq 2 \sum_{i=1}^{t} i|L_i| = 2W$$

□

Comme expliqué précédemment, tout ceci nous permet de conclure que l'algorithme est une 2-approximation. Un exemple d'instance qui atteint le ratio est décrit dans le prochain paragraphe.

Théorème 34. *L'algorithme 3 admet un rapport de garantie de performance de deux. De plus, il existe des instances pour lesquelles le rapport est atteint.*

Ratio atteint et discussions

Un exemple d'instance ou le ratio est atteint est le suivant : supposons un graphe à cinq sommets $\{x_1, x_2, x_3, x_4, x_5\}$ avec comme arêtes $\{x_1, x_2\}$, $\{x_2, x_3\}$ et $\{x_4, x_5\}$ (remarquons que $(x_1, x_2, x_3, x_4, x_5)$ est un ordre d'élimination simplicial). Avec $k = 4$, l'algorithme va d'abord choisir x_1, x_3 et x_4. Puis, par définition des poids, on aura $w_1(x_2) = w_1(x_5) = 1$, et l'algorithme peut prendre x_2 au lieu de x_5, formant une solution de coût deux à la place d'une solution de coût un. Il est facile de voir qu'il est possible d'obtenir le même résultat pour un graphe connexe.
On peut remarquer que la raison pour laquelle l'algorithme effectue des mauvais choix dans cette instance est l'ajout des bonus : en effet si le poids était bien égal à la taille du voisinage dans la solution, l'algorithme trouverait facilement une solution optimale sur ce petit exemple. Malheureusement, ces bonus ont du être ajoutés pour que notre analyse fonctionne, et il est donc en fait fortement possible que l'algorithme sans ces bonus donne également une solution 2-approchée voire mieux, étant donné que nous n'avons pas non plus pu trouver d'instance où celui-ci donne également une solution 2-approchée. Une question intéressante serait de savoir si cet algorithme, sans les bonus, donne une solution approchée, et si oui, quel est le rapport d'approximation. Nous sommes en revanche convaincus qu'une telle preuve, si elle existe, nécessiterait une approche différente de la nôtre (*i.e.* majorer l'augmentation du coût indépendamment de chaque sommet après chacune des restructurations).

4.4 Dans les graphes chordaux : algorithme $\mathcal{FPT}$

Dans les sous-sections à venir, nous nous intéressons à la résolution exacte paramétrée de Densest k-Subgraph et Sparsest k-Subgraph dans les graphes chordaux. Nous montrons en fait que les deux problèmes se comportent de la même manière lorsque paramétré par k : ils sont tous les deux $\mathcal{FPT}$ et n'admettent pas de noyau polynomial (sous certaines hypothèses classiques). Cependant, comme nous allons le voir, les résultats concernant Densest k-Subgraph sont immédiats, alors que ceux de Sparsest k-Subgraph sont plus délicats à obtenir. Ainsi, les deux résultats concernant le premier sont traités dans en Sous-Section 4.4.1, alors que l'algorithme $\mathcal{FPT}$ et la non-existence de noyau polynomial sont respectivement traités en Sous-Section 4.4.2 et Section 4.5.

4.4.1 Prélude : le cas de Densest k-Subgraph

Montrons tout d'abord que Densest k-Subgraph est immédiatement $\mathcal{FPT}$ dans les graphes chordaux : l'algorithme consiste d'abord à chercher dans le graphe une clique de taille maximum (en temps polynomial). Si celle-ci est de taille k ou plus, alors c'est forcément une solution optimale au problème. Dans le cas contraire, nous utilisons une propriété connue des graphes chordaux, qui est que leur tree-width est égale à la taille d'une clique maximum moins un. Ainsi dans ce cas, la tree-width du graphe est bornée par $(k-1)$, et une programmation dynamique classique sur une décomposition arborescente permet de trouver une solution optimale en temps $\mathcal{FPT}$ en k. Un tel exemple de programmation dynamique dans ce cas précis (Densest k-Subgraph dans les graphes de tree-width bornée) est même donné par [22].
Concernant l'existence d'un noyau polynomial, il est facile d'exhiber une cross-composition depuis Densest k-Subgraph dans les graphes chordaux (qui est bien un problème $\mathcal{NP}$-difficile). En effet, soit une suite de t graphes chordaux $G_1, \cdots, G_t$, chacun ayant n sommets, et $k, C \in \mathbb{N}$ (dans ce cas, la relation d'équivalence polynomiale est d'avoir le même nombre de sommets, ainsi que la même valeur de k et C). Ajoutons d'abord pour tout $i \in \{1, \cdots, t\}$ une clique K_i de n^2 sommets connectée à tous les sommets de G_i. Il est clair que le graphe résultant est un graphe chordal ayant $t \cdot (n + n^2)$ sommets (en effet, étant donné que chaque G_i est chordal, il est impossible de créer un cycle de longueur quatre ou plus en ajoutant les cliques K_i). Supposons maintenant qu'il existe $i \in \{1, \cdots, t\}$ tel que G_i contienne un ensemble X de k sommets qui induisent au moins C arêtes. Alors il est facile de vérifier que $X \cup K_i$ est un ensemble de $(k + n^2)$ induisant $(\binom{n^2}{2} + kn^2 + C)$ arêtes. Inversement, si l'on cherche dans le graphe résultant un ensemble de $(k + n^2)$ sommets induisant $(\binom{n^2}{2} + kn^2 + C)$ arêtes, il est clair qu'une clique entière K_i doit être dans la solution, pour un certain $i \in \{1, \cdots, t\}$ (puisqu'aucun autre sous-ensemble ne peut induire $\binom{n^2}{2}$ arêtes). Puis, on peut voir que les autres sommets de la solution doivent avoir un degré moyen d'au moins n^2. Ainsi, les autres k sommets doivent être pris dans G_i, et doivent forcément induire au moins C arêtes.
On a ainsi le résultat suivant :

Théorème 35. Densest k-Subgraph *paramétré par k dans les graphes chordaux est $\mathcal{FPT}$, et n'admet pas de noyau polynomial, sauf si $\mathcal{NP} \subseteq co\mathcal{NP}/poly$.*

Une remarque intéressante à propos de la cross-composition précédente est qu'elle fonctionne aussi pour les graphes d'intervalles, à condition que les graphes d'entrée soient aussi des graphes d'intervalles (en effet, on ne fait que prendre l'union disjointe, et rajouter à chaque graphe une clique universelle). Malheureusement, étant donné que la complexité ($\mathcal{NP}$-difficile *versus Polynomial*) de Densest k-Subgraph dans les graphes d'intervalles n'est toujours pas connue, on ne peut déduire aucun résultat négatif sous les hypothèses classiques. Cela montre en revanche que la $\mathcal{NP}$-difficulté de Densest k-Subgraph dans les graphes d'intervalles impliquerait qu'il n'existe pas non plus de noyau polynomial (sauf si $\mathcal{NP} \subseteq co\mathcal{NP}/poly$).

En prenant la contraposée de cette assertion, et toujours sous les même hypothèses de complexité, on peut dire qu'il suffit[1] de trouver un noyau polynomial pour DENSEST k-SUBGRAPH dans les graphes d'intervalles pour montrer que le problème n'est pas $\mathcal{NP}$-difficile. Malheureusement, la question de l'existence d'un tel noyau est toujours ouverte.

4.4.2 Le cas de SPARSEST k-SUBGRAPH

Nous proposons maintenant un algorithme $\mathcal{FPT}$ pour SPARSEST k-SUBGRAPH dans les graphes chordaux, paramétré par k. Cette partie est découpée en trois sous-parties : dans l'introduction, nous donnons les définitions, la terminologie et les notations nécessaires, les premières observations ainsi que l'idée de l'algorithme. Nous décrivons ensuite celui-ci en donnant les règles de pré-traitement et de branchement, et concluons enfin en discutant de la terminaison et du temps d'exécution.

Introduction

Définitions et notations. Soient $G = (V, E)$ le graphe chordal d'entrée, ainsi que $T = (\mathcal{X}, A)$ sa décomposition arborescente correspondante, comme définie dans la Section 1.3.3 (page 30). Rappelons que pour tout $X \in \mathcal{X}$, X induit une clique dans G. Pour éviter toute ambiguïté, les éléments de V seront toujours appelés « sommets », alors que les éléments de $\mathcal{X}$ seront appelés « nœuds ». Pour $v \in V$, on note $\mathcal{X}_v$ l'ensemble des nœuds de T contenant le sommet v.
On note respectivement par $\mathcal{L}$ et $\mathcal{I}$ les ensembles de feuilles et de nœuds internes de T (on a donc $\mathcal{X} = \mathcal{L} \cup \mathcal{I}$). Dans la suite, nous supposerons que T est enraciné en un nœud arbitraire X_r. Pour $X \in \mathcal{X}$, on note $pred(X) \in \mathcal{X}$ l'unique prédécesseur de X dans T (par convention, on a $pred(X_r) = \emptyset$), et par $succ(X) \subseteq \mathcal{X}$ l'ensemble des successeurs de X dans T. Pour un sommet $v \in V$ (resp. un nœud $X \in \mathcal{X}$), on note $d(v)$ (resp. $d(X)$) son degré dans G (resp. dans T). Pour un ensemble de sommets $U \subseteq V$ (resp. un ensemble de nœuds $A \subseteq \mathcal{X}$), on note $G[U]$ (resp. $T[A]$) le sous-graphe induit par U (resp. la sous-forêt induite par A). On dit qu'un sommet $v \in V$ est « esseulé[2] » (resp. « presque esseulé ») si $|\mathcal{X}_v| = 1$ (resp. $|\mathcal{X}_v| = 2$), autrement dit, s'il n'apparaît que dans exactement un nœud (resp. exactement deux nœuds) de T.

[1] En effet, un noyau polynomial est théoriquement plus facile à obtenir qu'un algorithme polynomial, étant donné que l'existence d'un algorithme polynomial donne immédiatement un noyau de taille constante.

[2] Notons qu'un sommet esseulé est toujours ce que l'on appelle plus communément un sommet « simplicial » (un sommet dont le voisinage est une clique). Cependant, si un nœud de l'arbre est inclus dans un autre nœud de l'arbre, un sommet simplicial peut ne pas être un sommet esseulé. Étant donné que nous ne faisons aucune supposition sur la décomposition arborescente, et que l'on dupliquera notamment des nœuds durant l'algorithme, nous préfèrerons le terme « esseulé ».

Premières observations. Dans les graphes chordaux, un ensemble indépendant maximum peut être construit en temps polynomial. Ainsi, si un tel ensemble est de taille k ou plus, nous le retournons naturellement. Ainsi, nous supposerons dans la suite de l'algorithme que le graphe d'entrée ne comporte pas d'ensemble indépendant de taille k ou plus.
Nous pouvons également assumer que pour toute feuille $L \in \mathcal{L}$, on a $L \not\subseteq pred(L)$ (autrement, nous contractons les deux nœuds). Ainsi, pour chaque feuille $L \in \mathcal{L}$, il existe $x \in L \setminus pred(L)$, autrement dit, un sommet esseulé dans L. De plus, il est facile de remarquer que l'ensemble de ces sommets esseulés issus des feuilles forment un ensemble indépendant. Tout ceci implique l'observation suivante :

Observation 1. *On peut supposer que* $|\mathcal{L}| < k$.

Observons maintenant deux autres propriétés simples vérifiées par les solutions optimales.

- Soit $S \subseteq V$ un ensemble de k sommets, et supposons qu'il existe $x \in S$ et $y \notin S$ tels que $\mathcal{X}_y \subsetneq \mathcal{X}_x$. Alors, cela signifie par définition que $N(y) \subsetneq N(x)$. Ainsi, si l'on remplace x par y dans la solution, le nombre d'arêtes ne peut que décroître. On dira qu'un ensemble S est « clos par inclusion » s'il n'existe pas de tels x et y. Ainsi, par ce qui précède, il existe toujours une solution optimale close par inclusion.

- L'autre propriété est que toute solution optimale close par inclusion contient forcément un sommet esseulé par feuille de T. Ainsi, comme dit précédemment, puisque chaque feuille $L \in \mathcal{L}$ n'est pas contenue dans $pred(L)$, il existe un sommet esseulé $x \in L$. Étant donnée une solution optimale close par inclusion S, soit celle-ci intersecte L, et puisqu'elle est close par inclusion elle contient x, soit celle-ci n'intersecte pas L, et dans ce cas là il est possible d'échanger n'importe quel autre sommet de S par x (en effet cette opération ne peut pas créer de nouvelles arêtes, puisque $N(x) = L$).

Idée de l'algorithme. Notre méthode peut être résumée comme suit : nous sélectionnons d'abord dans chaque feuille L un sommet esseulé (*c.f.* l'observation précédente), et on devine par un branchement (à l'aide une fonction binaire $w(L) \in \{0, 1\}$) si d'autres sommets de L doivent être pris dans la solution (valeur un) ou non (valeur 0). La largeur d'un tel branchement sera borné par k, en raison de l'Observation 1. Puis, étant donnée une feuille L avec $w(L) = 1$, on ajoute d'abord à la solution les sommets « les plus intéressants » de la feuille (en choisissant par exemple un sommet esseulé). Puis, lorsque ce n'est plus possible (le voisinage des sommets restants de la feuille peuvent ne pas être comparables si ces sommets apparaissent sur des sous-arbres incomparables en terme d'inclusion), nous appliquons des règles de branchement afin de créer de nouveaux sommets intéressants.

Terminologie de l'algorithme. Comme annoncé précédemment, l'algorithme est un algorithme de branchement, composé de règles de pré-traitement (qui ne nécessitent pas de branchement) et de règles de branchement. Ces règles sont ordonnées, et à chaque étape de l'algorithme, celui-ci essaie d'appliquer séquentiellement l'une d'elle. Ainsi, lors de l'utilisation d'une règle, nous supposerons que toutes les règles précédentes ne peuvent pas s'appliquer.

Durant l'algorithme, une solution partielle S (initialisée à $\emptyset$) sera construite, et l'instance d'entrée composée de G, k ainsi que de la décomposition T sera modifiée. Dans le but d'éviter de trop lourdes notations, ces variables dénoteront toujours l'instance courante que nous sommes en train de traiter et de modifier. Le graphe de départ, en revanche, sera noté G_0, et N_0 désignera la fonction de voisinage dans G_0.

Dans ce qui suit, « prendre » un sommet $v \in V$ dans la solution signifiera que v est ajouté à S, et supprimé de G et de T (en supprimant chacune de ses occurences). À l'inverse, « supprimer » un sommet signifiera le supprimer de G et de T. Si, à un moment donné, une feuille de T devient vide, alors on supprimera automatiquement cette feuille.

Soit $F \in \mathcal{I}$ une feuille de $T[\mathcal{I}]$ (*i.e.* un nœud de T dont tous les successeurs sont des feuilles). Le nœud F est appelé « mauvais nœud » s'il existe un sommet u apparaissant dans au moins deux feuilles de $succ(F)$ (autrement dit, si ses feuilles ne sont pas disjointes). On note $\#MN$ le nombre de mauvais nœuds de T. Enfin, on note $\#PF$ (pour « presque-feuille ») le nombre de nœuds internes de T dont au moins un successeur est une feuille. On a ainsi $\#MN, \#PF \leq |\mathcal{L}| < k$.

De plus, comme dit précédemment, à certaines feuilles de l'arbre $\mathcal{L}^* \subseteq \mathcal{L}$ sera associée une étiquette binaire, qui indiquera si une feuille donnée doit intersecter la solution (valeur un) ou non (valeur 0). Plus formellement, on définit une fonction $w : \mathcal{L}^* \longrightarrow \{0, 1\}$. Au début de l'algorithme, l'ensemble des feuilles étiquetées est vide : $\mathcal{L}^* = \emptyset$. Pour une solution $S \subseteq V_0$ (de k sommets), on dira que S « respecte » les étiquettes w si pour tout $L \in \mathcal{L}^*$, on a $w(L) = 0$ si et seulement si $S \cap L = \emptyset$. Pendant le déroulement de l'algorithme, nous utiliserons le terme de « deviner » une étiquette $w(L)$ pour une feuille $L \in \mathcal{L}$. Cela signifie en fait d'essayer les deux choix possibles (cohérents avec les choix précédents), en créant au plus deux exécutions distinctes de l'algorithme. Afin d'éviter des notations trop lourdes, l'ensemble $\mathcal{L}^*$ sera mis à jour automatiquement (*i.e.* une feuille L appartiendra à $\mathcal{L}^*$ si nous devinons $w(L)$).

On ajoute également une fonction $g : \mathcal{L}^* \longrightarrow 2^S$. Informellement, nous modifierons la fonction g durant l'algorithme afin que celle-ci représente le voisinage des sommets restants V dans la solution partielle S en construction. Il est à noter que

cette fonction n'intervient que dans l'analyse de l'algorithme, et plus précisément dans le but du maintien des invariants (voir plus bas).

Exactitude et temps d'exécution. Comme souvent dans les algorithmes de branchement, nous bornerons le temps d'exécution en bornant à la fois la profondeur ainsi que le degré maximum de l'arbre de recherche. Plus précisément, nous montrerons que :

- Chaque règle s'applique en temps $\mathcal{FPT}$.
- Le degré de branchement de chaque règle est une fonction de k seulement.
- Les règles de branchement font toujours strictement décroître le vecteur (k, $\#MN$, $\#PF$), dans l'ordre lexicographique, dont la valeur initiale dépend de k uniquement.
- Les règles de pré-traitement ne font pas croître (k, $\#MN$, $\#PF$), et font strictement décroître $|V| + |\mathcal{X}|$.

De ce fait, le nombre total de branchements de l'arbre de recherche sera une fonction de k seulement, et le nombre d'étapes entre deux branchements sera $\mathcal{FPT}$ en k (rappelons que $|\mathcal{X}|$ est polynomial en n), impliquant ainsi un temps d'exécution $\mathcal{FPT}$.
Nous décrivons maintenant les invariants maintenus tout au long de l'algorithme. Rappelons que S désigne la solution partielle en construction.

1. L'arbre T est toujours une décomposition arborescente de G (comme définie Section 1.3.3), et G est un sous-graphe induit de G_0.
2. Si un sommet de S est adjacent à un sommet $v \in V$, alors v doit apparaître dans une feuille étiquetée. Autrement dit : $N_0(S) \cap V \subseteq \bigcup_{L \in \mathcal{L}^*} L$.
3. Le voisinage d'un sommet $u \in V$ dans la solution partielle est défini par l'union des ensembles $g(L)$, pour toutes les feuilles L où u apparaît. Autrement dit : $g : \mathcal{L}^* \longrightarrow 2^S$ est telle que $\forall u \in V$ on a $N_0(u) \cap S = \bigcup_{L \in \mathcal{X}_u \cap \mathcal{L}^*} g(L)$.
   En particulier, cet invariant implique que s'il existe $u, v \in V$ tels que $\mathcal{X}_u \cap \mathcal{L}^* \subseteq \mathcal{X}_v \cap \mathcal{L}^*$ (*i.e.* v apparaît dans au moins autant de feuilles étiquetées que u), alors on a $N_0(u) \cap S \subseteq N_0(v) \cap S$ (*i.e.*, v est adjacent à au moins autant de sommets de S que u).
4. S'il existe une solution optimale close par inclusion $S^* \subseteq V$ telle que $S \subseteq S^*$, et telle que S^* respecte les étiquettes w, alors l'un des branchements retourne une solution optimale du problème.

Les règles de pré-traitement et de branchement

Il est à noter que dans chaque règle, la valeur de certaines variables parmi G, T, k, w et g seront modifiées. Cependant, par soucis de clarté, nous ne mentionnerons pas les variables qui restent inchangées.

Pré-traitement 1 : nœuds dupliqués inutiles.
S'il existe un nœud $X \in \mathcal{X}$ tel que $X \notin \mathcal{L}^*$ et $X \subseteq pred(X)$, alors contracter X et $pred(X)$ (*i.e.* supprimer X et connecter tous les successeurs $Y \in succ(X)$ à $pred(X)$).

Lemme 15. *La règle de pré-traitement 1 est sûre.*

Preuve. Clairement T est toujours une décomposition arborescente de G et donc l'invariant 1 est respecté. Comme $X \notin \mathcal{L}^*$, les feuilles étiquetées $\bigcup_{L\in\mathcal{L}^*} L$ restent inchangées, au même titre que la solution S. Ainsi, les invariants 2 et 3 sont respectés. Enfin, w n'est pas modifié non plus, et donc l'invariant 4 est juste.
Observons maintenant les paramètres qui décroissent lors de l'application de cette règle.Tout d'abord k reste inchangé. Ensuite, il est possible que $\#MN$ augmente, puisque $pred(X)$ peut devenir un mauvais nœud. Cependant dans ce cas on peut remarquer que $\#PF$ décroît strictement, et donc finalement (k, $\#PF$, $\#MN$) décroît strictement. Autrement, si $\#MN$ n'augmente pas, alors $\#PF$ n'augmente pas non plus. Enfin, $|\mathcal{X}|$ décroît dans tous les cas. □

Pré-traitement 2 : suppression d'un sommet (presque) esseulé.
S'il existe une feuille non étiquetée $L \in \mathcal{L}^*$ telle que $w(L) = 0$, alors si L contient un sommet esseulé v, le supprimer. Autrement, si L contient un sommet presque esseulé v, le supprimer.

Lemme 16. *La règle de pré-traitement 2 est sûre.*

Preuve. La règle consiste simplement à retirer un sommet du graphe et de l'arbre correspondant, donc l'invariant 1 est clairement vérifié. Nous ne modifions ainsi pas la solution S (ni la fonction g), les invariants 2 et 3 sont donc justes. Enfin, par définition des étiquettes de w, une solution optimale S^* les respectant ne contiendra pas v (quel que soit le cas considéré), et donc l'invariant 4 est vérifié.
Puis, clairement $|V| + |\mathcal{X}|$ décroît, alors que k reste inchangé. Cependant $\#MN$ peut augmenter lorsque $L = \{v\}$ et que $succ(pred(L)) = L$ (autrement dit, L était l'unique feuille de $pred(L)$, et $pred(pred(L))$ est une presque-feuille). Dans ce cas L est supprimée, $pred(L)$ devient une feuille et $pred(pred(L))$ peut devenir un mauvais nœud dans le cas où il a au moins un autre successeur qui est une feuille. Mais alors cela signifie qu'en fait $pred(L)$ et $pred(pred(L))$ étaient deux presque-feuilles, et la suppression de v (et donc de L) fait strictement décroître $\#PF$, ce qui prouve que $(k, \#MN, \#PF)$ ne peut pas croître. □

Branchement 1 : sélection d'un sommet esseulé.
S'il existe une feuille $L \in \mathcal{L}^*$ telle que $w(L) = 1$, et L contient un sommet esseulé v, alors prendre v dans la solution et décroître la valeur de k de 1. De plus, ajouter v dans $g(L)$, et si L ne devient pas vide, deviner une étiquette $w(L)$.

Lemme 17. *La règle de Branchement 1 est sûre.*

Preuve. Ici également un sommet est retiré du graphe et de l'arbre, donc l'invariant 1 reste vrai. Puis, les voisins de V dans le graphe restant doivent forcément apparaître dans la feuille L seulement (puisque v est esseulé), qui reçoit une étiquette $w(L)$ et appartient donc à $\mathcal{L}^*$. Ceci prouve que l'invariant 2 est respecté. On remarque également que v est ajouté dans $g(L)$, afin de maintenir l'invariant 3. Pour le dernier invariant, considérons une solution optimale S^* close par inclusion, qui respecte les étiquettes, et telle que $S \subseteq S^*$. Puisque $w(L) = 1$ et que S^* respecte les étiquettes, on a $S^* \cap L \neq \emptyset$. Et, puisque S^* est close par inclusion, S^* doit contenir un sommet esseulé de L s'il en exite un. On peut donc supposer sans perdre de généralité que $v \in S^*$, puisque tous les sommets esseulés de L ont exactement le même voisinage dans S et dans V.
Enfin, il est clair que k décroît par définition de la règle. □

Remarque 2. *À ce point, puisque la règle de Pré-traitement 1 ne s'applique pas, chaque feuille non étiquetée restante $L \in \mathcal{L} \setminus \mathcal{L}^*$ contient un sommet esseulé. La règle de branchement suivante permet de traiter ces feuilles.*

Branchement 2 : nœuds sans etiquette.
S'il existe $L \in \mathcal{L} \setminus \mathcal{L}^*$, alors prendre un sommet esseulé $v \in L$ dans la solution et décrémenter k de 1. Ajouter v dans $g(L)$, et si L ne devient pas vide, alors deviner une étiquette $w(L)$.

Lemme 18. *La règle de Branchement 2 est sûre.*

Preuve. En utilisant les mêmes arguments que pour la règle de Branchement 1, les invariants 1, 2 et 3 sont respectés. Considérons une solution optimale S^* close par inclusion, qui respecte les étiquettes et telle que $S \subseteq S^*$. Supposons que $v \notin S^*$. Puisque L n'a pas d'étiquette, deux cas peuvent apparaître : soit $S^* \cap L = \emptyset$, soit $S^* \cap L \neq \emptyset$.
Dans le premier cas, puisque l'invariant 2 implique $N_0(v) \cap S = \emptyset$, et puisque v est un sommet esseulé, on a $N_0(v) \cap S^* = \emptyset$. Ainsi, remplacer n'importe quel autre sommet de S^* par v ne peut que faire décroître son coût, une contradiction.
Dans le cas où $S^* \cap L \neq \emptyset$, le même raisonnement que pour la règle de Branchement 4 permet de prouver que l'invariant 4 est respecté.
Enfin, ici également il est clair que k décroît par définition de la règle. □

Remarque 3. *À ce stade, on remarquera que $\mathcal{L}^* = \mathcal{L}$. Autrement dit, toutes les feuilles de T comportent une étiquette. En effet, supposons qu'il existe $L \in \mathcal{L} \setminus \mathcal{L}^*$. Si L contient un sommet esseulé, alors la règle de Branchement 1 doit s'appliquer. Autrement, la règle de Pré-traitement 1 s'applique.*
De plus, aucune feuille ne contient de sommet esseulé, puisque dans le cas contraire soit la règle de Branchement 1 soit la règle de Pré-traitement 2 s'appliquerait.

Branchement 3 : traitement des mauvais nœuds.
S'il existe un mauvais nœud $F \in \mathcal{X}$, soit $C = \bigcup_{L \in succ(F)} L$ l'ensemble des sommets contenus dans les feuilles de F. Partitionner C en classe d'équivalences $C_1, \cdots, C_t$ par rapport à la relation d'équivalence suivante : deux sommets $u, v \in C$ sont équivalents si $\mathcal{X}_u \cap succ(F) = \mathcal{X}_v \cap succ(F)$ (autrement dit, u et v apparaissent dans le même sous-ensemble de feuilles de F). Pour tout $i \in \{1, \cdots, t\}$, soit $\mathcal{L}^i \subseteq succ(F)$ l'ensemble de feuilles dans lequel les sommets de C_i apparaissaient avant le partitionnement. Remplacer les feuilles de F par $C_1, \cdots, C_t$, et, pour tout $i \in \{1, \cdots, t\}$, deviner une étiquette $w(C_i)$ et définir $g(C_i) = \bigcup_{L \in \mathcal{L}^i} g(L)$.

L'intuition de la règle est d'assurer que pour les nœuds dont tous les successeurs sont des feuilles, ces dernières sont deux à deux disjointes, et forment une partition telle que deux sommets d'une même feuille après l'application de la règle apparaissaient dans le même sous-ensemble de feuilles avant la règle. Les règles de branchement suivantes peuvent créer de nouveaux mauvais nœuds, mais font décroître k.

Lemme 19. *La règle de Branchement 3 est sûre.*

Preuve. Remarquons tout d'abord que $\#MN$ décroît, alors que k et $\#PF$ restent identiques. Vérifions maintenant les invariants.
Puisque les sommets qui apparaissaient dans au moins une feuille avant la règle apparaissent toujours dans une feuille, l'invariant 2 est respecté. Par la Remarque 3, aucune feuille ne contient un sommet esseulé. Ainsi, tous les sommets de C sont contenus dans F et induisent donc une clique. Puisque nous ne modifions pas F, aucun sommet ni arête n'est supprimé du graphe, ce qui maintient l'invariant 1. Afin de prouver l'invariant 3, considérons $i \in \{1, \cdots, t\}$, et $u \in C_i$. Avant le partitionnement, nous avions :

$$\begin{aligned} N_0(u) \cap S &= \bigcup_{L \in \mathcal{X}_u \cap \mathcal{L}} g(L) \\ &= \left(\bigcup_{L \in \mathcal{X}_u \cap succ(F)} g(L) \right) \cup \left(\bigcup_{L \in \mathcal{X}_u \cap (\mathcal{L} \setminus succ(F))} g(L) \right) \end{aligned}$$

Et, par définition, on a maintenant $\bigcup_{L \in \mathcal{X}_u \cap succ(F)} g(L) = g(C_i)$. Ainsi, l'invariant est respecté.

Prouvons enfin que l'invariant 4 reste vrai : soit S^* une solution optimale close par inclusion, satisfaisant les étiquettes, et telle que $S \subseteq S^*$. Si l'on considère le branchement où chaque nouvelle feuille C_i reçoit l'étiquette correspondant à $S^* \cap C_i$, la solution satisfera ces étiquettes, ce qui prouve l'invariant 4. □

Branchement 4 : sélection dans un nœud père.

S'il existe $L \in \mathcal{L}$ telle que $pred(L)$ contient un sommet esseulé v, alors prendre v dans la solution, décrémenter k de 1, et créer une nouvelle feuille N adjacente à $pred(L)$ et contenant les sommets $L \setminus \{v\}$. Enfin, deviner une étiquette $w(N)$ et poser $g(N) = \{v\}$.

Lemme 20. *La règle de Branchement 4 est sûre.*

Preuve. Tout d'abord, il est clair que l'invariant 1 est vérifié, puisque la règle consiste simplement en la suppression d'un sommet du graphe et de l'arbre, et en la duplication d'un nœud de l'arbre en feuille. Puis, puisque la feuille N a été créée et contient tous les voisins de v, et puisque une étiquette $w(N)$ a été devinée, l'invariant 2 est préservé. Concernant l'invariant 3, remarquons que pour tout $u \in pred(L)$, son voisinage dans la solution partielle $(N_0(u) \cap S)$ après la règle est exactement l'union entre ses voisinages dans la solution partielle précédente, et $\{v\}$. Par définition de $g(N)$, et puisque u appartient maintenant à N, cela prouve que l'invariant est respecté.
Prouvons maintenant que l'invariant 4 est maintenu. Soit S^* une solution optimale close par inclusion, satisfaisant les étiquettes, et telle que $S \subseteq S^*$. Si $S^* \cap pred(L) \neq \emptyset$, alors le résultat est immédiat puisque v est esseulé. Autrement, deux cas peuvent apparaître :

- premier cas : il existe $L \in pred(L)$ tel que $S^* \cap L \neq \emptyset$. Dans ce cas, soit $u \in S^* \cap L$. Puisque L ne contient aucune sommet esseulé (*c.f.* Remarque 3), S^* n'est en fait pas close par inclusion, une contradiction.
- second cas : pour tout $L \in pred(L)$ on a $S^* \cap L = \emptyset$. Dans ce cas, cela signifie que $N_0(v) \cap S^* = \emptyset$, et donc on peut remplacer n'importe quel autre sommet de S^* par v sans diminuer le coût.

Enfin, clairement k est décrémenté de 1. □

Branchement 5 : sélection dans une feuille.

S'il existe $L \in \mathcal{L}$ tel que $w(L) = 1$ et L contient un sommet presque esseulé v (contenu dans L et dans $pred(L)$), alors prendre v dans la solution, décrémenter k de 1, et créer une nouvelle feuille N adjacente à $pred(L)$ contenant les sommets $pred(L) \setminus \{v\}$. Si L ne devient pas vide, alors deviner une nouvelle étiquette $w(L)$. Enfin, deviner une étiquette $w(N)$, ajouter v dans $g(L)$, et poser $g(N) = \{v\}$.

Lemme 21. *La règle de Branchement 5 est sûre.*

Preuve. Par la même justification que pour la règle précédente, l'invariant 1 est toujours vérifié. De plus, les voisins de v dans le graphe restant doivent apparaître dans la nouvelle feuille N, qui reçoit une étiquette $w(N)$. Ainsi, les invariants 2 et 3 sont maintenus (remarquons qu'on ajoute v dans $g(L)$, et que $g(N)$ a été fixé à $\{v\}$).
Concernant l'invariant 4, soit S^* une solution optimale close par inclusion, satisfaisant les étiquettes, et telle que $S \subseteq S^*$. Soit $x \in S^* \cap L$ (un tel sommet existe forcément, d'après l'étiquette $w(L)$), et supposons que $v \notin S^*$. Par l'invariant 2, on a $N_0(v) \cap S \subseteq N_0(x) \cap S$. Puisqu'il n'y a pas de sommet esseulé dans L (*c.f.* Remarque 3), on a $N_0(v) \cap S^* \cap V \subseteq N_0(v) \cap S^* \cap V$. Puisque $S^* = S \cup (S^* \cap V)$, le résultat suit.
Enfin, ici aussi k est décrémenté de 1. □

Fin de l'algorithme

Lemme 22. *Si aucune règle ne peut s'appliquer, alors soit G est vide, ou $k = 0$.*

Preuve. Montrons d'abord que la profondeur de T est au plus un (*i.e.* au plus une étoile). Supposons par contradiction qu'il existe un nœud interne F de profondeur au moins un, *i.e.*, au moins une feuille est adjacente à F, et $F \neq X_r$ (et donc $pred(F)$ existe). Par la règle de Pré-traitement 2 et la règle de Branchement 5, aucune feuille de F n'a de sommet presque esseulé. Donc, chaque sommet qui apparaît dans F et une feuille de F apparaît aussi dans $pred(F)$. De plus, la règle de Pré-traitement 1 assure que $F \not\subseteq pred(F)$. Ainsi, il existe un sommet $v \in F$ tel que $v \notin pred(F)$. Ainsi, v doit être un sommet esseulé de F et la règle de Branchement 4 peut s'appliquer, une contradiction.
L'arbre T est donc de profondeur au plus 1. Puisque la règle de Branchement 3 ne peut pas s'appliquer, les feuilles de X_r sont disjointes. Ainsi, chaque sommet présent dans une feuille est soit un sommet esseulé soit un sommet presque esseulé. Soit L une feuille. Si $w(L) = 0$, alors la règle de Pré-traitement 2 peut s'appliquer. Autrement, la règle de Branchement 1 ou 5 peut s'appliquer, tant que X_r contient une feuille.
Ainsi, T est maintenant réduit à une clique. Si $k = 0$ alors le résultat suit. Si $k > 0$, alors puisque chaque sommet est esseulé, la règle de Branchement 1 peut s'appliquer et on peut ainsi choisir un sommet arbitraire de la clique dans la solution. □

D'après le lemme précédent, l'algorithme s'arrête lorsque soit le graphe est vide ou que $k = 0$. Si le graphe est vide et que $k > 0$, alors cela signifie que la branche considérée (de l'arbre de recherche) ne conduira pas à une solution optimale au problème. Autrement, nous comparons les coûts de toutes les solutions ainsi construites dans chaque branche. Puisque l'invariant 4 est préservé tout au long de l'algorithme, l'une des solutions aura un coût optimal.

D'après l'introduction et tous les lemmes correspondant aux règles, la taille de l'arbre de recherche est bien $\mathcal{FPT}$ en k. De plus, remarquons que chaque règle s'exécute en temps $\mathcal{FPT}$. C'est clairement le cas pour les règles de Pré-traitement 1 et 2, et pour les règles de Branchement 1, 2, 4 et 5. Concernant la règle de Branchement 3, qui consiste à partitionner les feuilles d'un nœud, elle s'exécute en temps $\mathcal{FPT}$ à partir du moment où $|\mathcal{L}|$ est une fontion de k. C'est clairement le cas au début de l'algorithme, puisque $|\mathcal{L}| < k$, et on peut remarquer qu'au fil de l'algorithme, le nombre de feuilles augmente de 1 dans les règles de Branchement 4 et 5, et par une fonction du nombre de feuilles précédent dans la règle de Branchement 3. Puisque les règles de branchement ne sont appliquées que $f(k)$ fois pour une certaine fonction f, la règle de Branchement 3 s'exécute bien en temps $\mathcal{FPT}$.
Nous obtenons ainsi le théorème suivant :

Théorème 36. Sparsest k-Subgraph *est $\mathcal{FPT}$ dans les graphes chordaux, paramétré par k.*

En analysant le temps d'exécution de l'algorithme, tel qu'il est écrit, on peut observer que celui-ci est en fait une tour d'exponentielle de hauteur k, étant donné que la règle de Branchement 3 crée 2^t nouvelles feuilles, où t est le nombre de feuilles avant la règle. Cependant, on peut modifier légèrement l'algorithme afin d'obtenir un temps d'exécution en $\mathcal{O}^*(2^{k^2})$. En effet, après l'application de cette règle, toutes les feuilles avec $w(L) = 0$ peuvent en fait être regroupées en une unique feuille, étant donné que ces sommets ne sont pas dans la solution. Puisque les feuilles sont disjointes, le nombre de feuilles L telles que $w(L) = 1$ est au plus k (puisqu'au moins un sommet de ces feuilles sera dans la solution). Ainsi, le nombre de feuilles après l'application de cette règle peut être finalement majoré par $k + 1$. Puis, comme dit précédemment, les seules règles de branchement qui font croître le nombre de feuilles sont les règles 4 et 5, qui ajoutent toutes les deux au plus une feuille. Cependant, puisque ces branchements font décroître k, le nombre maximum de feuilles d'un nœud F avant l'application de la règle de Branchement 3 est $2k$. En résumé, cette règle (qui borne le temps d'exécution de tout l'algorithme) s'exécute en temps $\mathcal{O}^*(2^{\mathcal{O}(k^2)})$ (on a au plus 2^{2k} feuilles, et on choisit au plus k feuilles L telles que $w(L) = 1$). Pour des raisons de clarté, nous avons omis ces modifications dans l'algorithme.

4.5 Dans les graphes chordaux : absence de noyau polynomial

4.5.1 Intuition

Dans cette partie, nous montrons que Sparsest k-Subgraph paramétré par k dans les graphes chordaux n'admet pas de noyau polynomial. La prevue consiste en une cross-composition depuis le problème Clique dans les graphes généraux, et est en fait une extension de la preuve de $\mathcal{NP}$-complétude présentée en Section 4.2. Nous

rappelons d'abord l'idée de la réduction de $\mathcal{NP}$-complétude, avant de rappeler le principe d'une cross-composition et d'expliquer informellement la modification de la réduction qui permet de l'obtenir. Nous montrons ensuite formellement le résultat dans la preuve du Théorème 37.
Étant donné un graphe $G = (V, E)$ et $k \in \mathbb{N}$, on construit d'abord une clique A représentant les sommets de G. On représente aussi chaque arête $e_j = \{u, v\} \in E$ par un gadget F_j, et on connecte les sommets représentant u et v dans A à certains sommets de F_j. La réduction force ensuite la solution à prendre parmi A les représentant de $(n-k)$ sommets de G (qui correspondront au complémentaire d'un ensemble de sommet S de taille k dans G), et à prendre le même nombre de sommets parmi chaque gadget. L'idée principale est que le coût de la solution à l'intérieur de chaque gadget augmente de 1 si les sommets choisis sont adjacents à des sommets de A qui ont été sélectionnés. Ainsi, étant donné que le but est de minimiser le coût, on essaiera de maximiser le nombre de gadgets adjacents aux sommets représentant les sommets de S (*i.e.* les sommets qui *n'ont pas* été pris dans la solution), le maximum étant lorsque S induit une clique dans G.

La différence majeure entre une réduction classique et une cross-composition est que l'on reçoit maintenant en entrée une suite de t instances de Clique (la définition formelle d'une cross-composition est disponible Section 4.5). Les contraintes, outre l'équivalence de solutions entre l'une des instances et l'instance créée, concernent la taille de cette dernière ainsi que la valeur du paramètre obtenu. En effet, alors que l'instance peut dépendre polynomialement de la suite d'instances (et donc de t), le paramètre choisi de l'instance obtenue (le nombre de sommets k de la solution pour notre problème) ne peut dépendre (polynomialement) que de la taille maximale de chacune des instances, et du logarithme de t. Cette interdiction du paramètre de dépendre linéairement ou plus de t est la principale difficulté lors de l'élaboration d'une cross-composition. Cette difficulté est d'autant plus marquée pour Sparsest k-Subgraph dans les graphes chordaux que, par exemple, il est obligatoire que le paramètre k de l'instance que l'on construit soit supérieur à la taille d'un ensemble indépendant maximum. Ainsi, le graphe que l'on construit ne peut pas avoir d'ensemble indépendant maximum de taille linéaire en t. Ceci nous force plus ou moins à construire de grosses cliques, en particulier pour le choix des sommets des instances d'entrée. Pour sélectionner l'instance positive parmi les t instances d'entrée, nous utiliserons la possibilité du paramètre de dépendre polynomialement en le logarithme de t en utilisant, comme c'est maintenant devenu classique pour les cross-compositions, l'encodage binaire des indices $1, \cdots, t$.

Ainsi, afin d'adapter la réduction de $\mathcal{NP}$-difficulté en une cross-composition, nous ajoutons un *sélecteur d'instances* en plus de la clique A (qui joue le rôle de *sélecteur de sommets*) et des gadgets (qui jouent le rôle de *sélecteur d'arêtes*). Ce sélecteur d'instances sera composé de $2 \log t$ gadgets (où t est le nombre d'instances de la cross-composition), permettant, comme dit précédemment, d'encoder la re-

présentation binaire de l'index de chaque instance (à l'aide de paires de gadgets représentant les 0 et les un). Ces gadgets seront les mêmes que pour le sélecteur d'arêtes. Pour des raisons techniques, le sélecteur d'instances devra être dupliqué un certain nombre de fois, au même titre que la clique A, qui sera, elle, dupliquée t fois afin de modéliser les ensembles de sommets de chaque instance. Un schéma de la construction obtenue est présentée Figure 24. Étant donné que les gadgets sont les mêmes que pour la preuve de $\mathcal{NP}$-complétude, nous ne rappellerons pas leur définition, et renvoyons le lecteur à la Section 4.2.3.

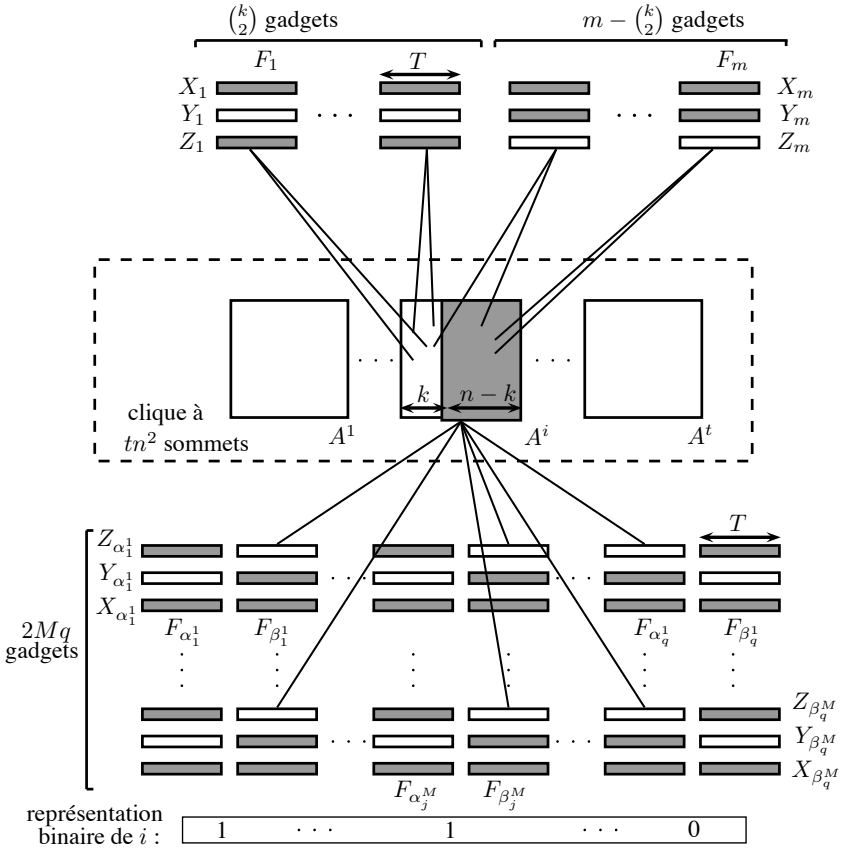


FIGURE 24 – Schéma de la cross-composition. Les rectangles gris représentent les sommets de la solution.

4.5.2 Démonstration

Théorème 37. SPARSEST k-SUBGRAPH *paramétré par k n'admet pas de noyau polynomial dans les graphes chordaux, sauf si co$\mathcal{NP} \subseteq \mathcal{NP}/poly$.*

Preuve. Soit $(G_1, k_1), \cdots, (G_t, k_t)$ une suite de t instances de CLIQUE, avec $G_i = (V_i, E_i)$ pour tout $i \in \{1, \cdots, t\}$. Sans perdre de généralité nous supposons que

$t = 2^q$ pour un certain $q \in \mathbb{N}$ (si ce n'est pas le cas, il est facile de voir que l'on peut ajouter au plus t instances triviales négatives). Pour des raisons de clarté nous posons :

$$T = n(n-k),$$

et

$$M = n^6.$$

Nous supposons que les t instances sont équivalentes selon la relation polynomiale d'équivalente suivante : pour tout $i, j \in \{1, \cdots, t\}$, on a $|V_i| = |V_j| = n$, $|E_i| = |E_j| = m$, et $k_i = k_j = k$. Nous construisons l'instance $G' = (V', E')$, k', C' suivante :

Sélecteur de sommets. On construit tout d'abord une clique A de $t \cdot n^2$ sommets, décomposée en t sous-clique A^i représentant chacune l'ensemble de sommets de l'instance G_i. Ainsi chaque clique A^i est elle-même décomposée en n sous-cliques $A^i_1, \cdots, A^i_n$, chacune représentant un sommet de G_i.

Sélecteur d'arêtes. Étant donné que toutes les instances ont le même nombre d'arêtes, on construit m gadgets $(F_j)_{j=1,\cdots,m}$, où chaque F_j est composé des ensembles de sommets X_j, Y_j et Z_j comme décrit précédemment. Pour chaque $i \in \{1, \cdots, t\}$, si la $j^{\text{ème}}$ arête de G_i relie les sommets u et v, alors on connecte tous les sommets de Z_j à tous les sommets de A^i_u et A^i_v.

Sélecteur d'instances. On ajoute $2qM$ gadgets $(F_{\alpha^h_j})^{h=1..M}_{j=1..q}$ et $(F_{\beta^h_j})^{h=1..M}_{j=1..q}$, chacun composés de $X_{\alpha^h_j}, Y_{\alpha^h_j}$ et $Z_{\alpha^h_j}$ (resp. $X_{\beta^h_j}, Y_{\beta^h_j}$ et $Z_{\beta^h_j}$). L'idée est d'encoder la représentation binaire de chaque entier $i \in \{1, \cdots, t\}$. Chaque paire $(F_{\alpha^h_j}, F_{\beta^h_j})$ représentera le $j^{\text{ème}}$ bit de l'entier i, où les gadgets indexés par α représenteront un 0, et les gadgets indexés par β représenteront un 1. Ces gadgets sont dupliqués M fois pour des raisons techniques seulement. Plus précisément, pour tout $i \in \{1, \cdots, t\}$, on considère la représentation binaire de i encodée par un vecteur $b_i \in \{0,1\}^q$. Pour tout $j \in \{1, \cdots, q\}$, si le $j^{\text{ème}}$ bit de b est un 0, alors on connecte tous les sommets de A^i à tous les sommets de $Z_{\alpha^h_j}$, pour tout $h \in \{1, \cdots, M\}$. Autrement, on connecte tous les sommets de A^i à tous les sommets de $Z_{\beta^h_j}$, pour tout $h \in \{1, \cdots, M\}$.

Valeurs de k' et C'

On pose :

$$k' = T + 2Tm + 4TqM,$$

et

$$C' = \binom{T}{2} + \binom{T}{2}(m + 2Mq) + (m - \binom{k}{2}) + Mq.$$

Il est clair que cette instance peut être construite en temps polynomial en $\sum_{i=1}^{t} |G_i| + k_i$. De plus, remarquons que le paramètre k' obtenu est polynomial en n, k et $\log t$, comme requis pour une cross-composition. Le lemme suivant prouve que le graphe obtenu est chordal.

Lemme 23. *Le graphe G' est chordal.*

Preuve. On construit un schéma d'élimination simplicial. Remarquons d'abord que tous les gadgets sont indépendants les uns des autres. Pour cette raison nous les traitons d'abord un par un, dans un ordre arbitraire. On considère ainsi un gadget F composé des ensembles de sommets X, Y et Z. On commence par éliminer les sommets de X dans un ordre arbitraire, chacun étant un sommet simplicial. Puis, on élimine chaque sommet de Y dans un ordre arbitraire, chacun étant un sommet simplicial à cause notamment du biparti complet entre Y et Z. Enfin, ici aussi quel que soit le gadget F considéré, son ensemble Z correspondant est une clique, et est adjacent à certains sommets de A, qui sont forment clique également. On peut donc éliminer les sommets de Z dans un ordre arbitraire. À ce point nous avons tout éliminer sauf la clique A, et il est ainsi trivial de terminer l'ordre d'élimination. □

Il ne reste plus qu'à montrer qu'il existe $i \in \{1, \cdots, t\}$ tel que G_i contienne une clique de taille k si et seulement si G' contient un ensemble de k' sommets induisant C' arêtes ou moins.

Lemme 24. *Soit $i \in \{1, \cdots, t\}$ tel que G_i contient une clique de taille k. Alors G' contient k' sommets qui induisent C' arêtes ou moins.*

Preuve. Soit $K \subseteq V_i$ une clique de taille k dans G_i. Sans perdre de généralité on suppose $K = \{v_1, \cdots, v_k\}$, et que les arêtes induites par K sont $\{e_1, \cdots, e_{\binom{k}{2}}\}$. Soit $b \in \{0,1\}^q$ la représentation binaire de i. On construit l'ensemble $K' \subseteq V'$ comme suit :

- pour tout $j \in \{1, \cdots, \binom{k}{2}\}$, K' contient X_j et Z_j. On a alors $2T$ sommets qui induisent $\binom{T}{2}$ arêtes pour chaque gadget F_j.
- pour tout $j \in \{\binom{k}{2}+1, \cdots, m\}$, K' contient X_j et Y_j. On a alors $2T$ sommets qui induisent $(\binom{T}{2}+1)$ arêtes pour chaque gadget F_j.
- pour tout $u \in \{(k+1), \cdots, n\}$, K' contient A^i_u. Cela rajoute T sommets induisant $\binom{T}{2}$ arêtes.
- pour tout $h \in \{1, \cdots, M\}$ et tout $j \in \{1, \cdots, q\}$, K' contient $X_{\alpha^h_j}$ et $X_{\beta^h_j}$. De plus, si le $j^{\text{ème}}$ bit de b est un 1, alors on ajoute $Y_{\beta^h_j}$ et $Z_{\alpha^h_j}$ à K'. Si au contraire c'est un 0, alors on ajoute $Y_{\alpha^h_j}$ et $Z_{\beta^h_j}$ à K'. Il est facile alors de voir que quelle que soit la paire de gadgets $F_{\beta^h_j}$ et $F_{\alpha^h_j}$ considérée, on ajoute $4T$ sommets qui induisent $(2\binom{k}{2}+1)$ arêtes.

Enfin, on peut vérifier que ces différents ensembles de sommets sélectionnés sont indépendants par construction. On a ainsi k' sommets induisant C' arêtes. □

Lemme 25. *Si G' contient k' sommets qui induisent au plus C' arêtes, alors il existe $i \in \{1, \cdots, t\}$ tel que G_i contient une clique de taille k.*

Preuve. Soit K' un ensemble de k' sommets de G' induisant C' arêtes. Nous rappelons les notations utilisées pour la preuve du Théorème 33. Pour un ensemble de sommets $S \subseteq V'$, on note $tr(S) = S \cap K'$ sa trace dans la solution K'. Pour un sommet $v \in V'$, on note $\mu(v) = |tr(N(v))|$ la taille de son voisinage dans K'.
On note $\mathcal{I} = \{1, \cdots, m\} \cup \{\alpha_j^h\}_{j=1..q}^{h=1..M} \cup \{\beta_j^h\}_{j=1..q}^{h=1..M}$ l'ensemble des différents indices des gadgets. De la même manière que précédemment, pour tout $\gamma \in \mathcal{I}$, le gadget F_γ contient les ensembles de sommets $X_\gamma = \{x_1^\gamma, \cdots, x_T^\gamma\}$, $Y_\gamma = \{y_1^\gamma, \cdots, y_T^\gamma\}$ et $Z_\gamma = \{z_1^\gamma, \cdots, z_T^\gamma\}$.
On définit $E_0 = \{\gamma \in \mathcal{I}$ tel que $\forall x \in tr(A)$, aucun sommet de Z_γ n'est adjacent à $x\}$. Autrement dit, E_0 représente les indices de tous les gadgets F_γ qui ne sont pas adjacents à des sommets choisis par la solution parmi la clique A. On note son complémentaire $E_1 = \mathcal{I} \setminus E_0$, qui est donc l'ensemble des gadgets qui sont adjacents à au moins un sommet de $tr(A)$.

Comme pour la preuve de $\mathcal{NP}$-complétude, la preuve consiste à modifier la solution à base de substitutions de sommets, afin que K' encode une solution au problème Clique pour l'un des graphes d'entrée. Plus précisément, nous rappelons la définition d'un *remplacement sûr*, déjà utilisé pour la preuve du Théorème 33. Soient $u \in K'$ et $v \in V' \setminus K'$. On dit que (u, v) est un remplacement sûr si :

- $\mu(v) \leq \mu(u)$ si u et v ne sont pas adjacents dans G'.
- $\mu(v) - 1 \leq \mu(u)$ si u et v sont adjacents dans G'.

Si (u, v) est un remplacement sûr, alors il est facile de voir que $(K' \setminus \{u\}) \cup \{v\}$ est un ensemble de taille k' n'induisant pas plus d'arêtes que K'.
Sans un soucis de clarté, nous garderons les mêmes notations et mettrons à jour les définitions de K', E_0 et E_1 lors de tous les remplacements. Par exemple, s'il existe $\gamma \in E_1$ tel que F_γ n'est adjacent qu'à un seul sommet u parmi $tr(A)$, et si un remplacement supprime u de K', alors γ passe de E_1 à E_0.

Le reste de la preuve est découpé comme suit : les Affirmations 1, 2 et 3 permettent de supposer que K' a une forme particulière, avec au final quatre cas possibles. Nous traitons enfin ces quatre cas séparément, et utilisons les Affirmations 4 et 5 afin de conclure.
Comme dit précédemment, la preuve est une extension de la réduction prouvant la $\mathcal{NP}$-difficulté du problème. Aussi, les preuves les Affirmations 1, 2 et 3 sont respectivement identiques aux preuves des Lemmes 7, 8 et 9.

Affirmation 1. *On peut supposer que pour tout $\gamma \in \mathcal{I}$, on a $X_\gamma \subseteq K'$.*

Preuve. Soit $S = \bigcup_{\gamma \in \mathcal{I}} X_\gamma$. Comme $k' > |S|$, il existe toujours $u \in K' \setminus S$. Supposons qu'il existe $\gamma \in \mathcal{I}$ et $j \in \{1, \cdots, T\}$ tel que $x_j^\gamma \notin K'$. Dans ce cas, si son voisin y_j^γ

n'appartient pas non plus à K', alors on a $\mu(x_j^\gamma) = 0$ et (z, u) est un remplacement sûr quel que soit $z \in K' \setminus S$. Si en revanche $y_j^\gamma \in K'$, alors dans ce cas $\mu(x_j^\gamma) = 1$. Mais comme x_j^γ et y_j^γ sont connectés, (y_j^γ, x_j^γ) est un remplacement sûr. Après ces remplacements, on a bien $X_\gamma \subseteq K'$ pour tout $i \in \mathcal{I}$. □

Affirmation 2. *On peut supposer que nous nous trouvons dans l'un des cas suivants :*

- *Cas A1 : pour tout $\gamma \in E_0$, on a $tr(Z_\gamma) = Z_\gamma$.*
- *Cas A2 : pour tout $\gamma \in E_0$, on a $tr(Y_\gamma) = \emptyset$.*

Preuve. Nous allons d'abord restructurer chaque gadget de E_0 indépendamment, puis les gadgets entre eux. Soit $\gamma \in E_0$ tel que $tr(Y_\gamma) \neq \emptyset$ et $tr(Z_\gamma) \neq Z_\gamma$, et soit $j_0 = \max\{j \in \{1, \cdots, T\} : y_j^\gamma \in tr(Y_\gamma)\}$ et j_1 tel que $z_{j_1}^\gamma \notin tr(Z_\gamma)$.
Rappelons d'abord que d'après le Lemme 1, on a $x_{j_0}^\gamma \in K'$. Si $j_0 \neq 1$, alors $\mu(y_{j_0}^\gamma) = y + z + 1$, avec $y = |tr(Y_\gamma) \cap N(y_{j_0}^\gamma)|$ et $z = |tr(Z_\gamma) \cap N(y_{j_0}^\gamma)| = |tr(Z_\gamma)|$. En outre, on a $\mu(z_{j_1}^\gamma) \leq y + z + 1$ (plus précisément : $\mu(z_{j_1}^\gamma) = y + z + 1$ si $y_1^\gamma \in K'$, et $\mu(z_{j_1}^\gamma) = y + z$ sinon).
Informellement, ce remplacement va consister à « perdre » l'arête reliée au sommet de X_γ, et à « récupérer » au plus une arête, due à y_1^γ. On a donc $\mu(z_{j_1}^\gamma) \leq \mu(y_{j_0}^\gamma)$, et $(z_{j_1}^\gamma, y_{j_0}^\gamma)$ est un remplacement sûr.
Si $j_0 = 1$, alors on a $tr(Y_\gamma) = \{y_1^\gamma\}$. Supposons qu'il existe j_1 tel que $z_{j_1}^\gamma \notin tr(Z_\gamma)$. On a alors $\mu(y_1^\gamma) = z+1$ où $z = |N(y_1^\gamma) \cap tr(Z_\gamma)|$, et $\mu(z_{j_1}^\gamma) = z+1$. Ici aussi $(z_{j_1}^\gamma, y_{j_0}^\gamma)$ est un remplacement sûr.
Après tous ces remplacements, pour tout $\gamma \in E_0$ on a $tr(Y_\gamma) \neq \emptyset \implies tr(Z_\gamma)$. On procède maintenant à des remplacements entre deux gadgets de E_0 : s'il existe $a, b \in E_0$ tels que $tr(Y_a) \neq \emptyset$ et $tr(Z_b) \neq Z_b$, alors soit j_0 tel que $y_{j_0} \in tr(Y_a)$, et soit j_1 tel que $z_{j_1}^b \notin tr(Z_b)$. On a alors $\mu(y_{j_0}^a) \geq T + 1$ et $\mu(z_{j_1}) \leq T - 1$, et $(z_{j_1}^i, y_{j_0}^i)$ est un remplacement sûr.
Ainsi, ces remplacements se terminent soit lorsque $tr(Y_\gamma) = \emptyset$ pour tout $\gamma \in E_0$, ou lorsque $tr(Z_\gamma) = Z_\gamma$ pour tout $\gamma \in E_0$, ce qui démontre l'Affirmation 2. □

Affirmation 3. *On peut supposer que nous nous trouvons dans l'un des cas suivants :*

- *Cas B1 : pour tout $\gamma \in E_1$, on a $tr(Y_\gamma) = Y_\gamma$.*
- *Cas B2 : pour tout $\gamma \in E_1$, on a $tr(Z_\gamma) = \emptyset$.*

Preuve. Informellement, la preuve repose sur le fait que remplacer un sommet de Z_γ par un sommet de Y_γ nous permet de « supprimer » au moins une arête avec les sommets de A, et de « récupérer » une arête avec un sommet de X_γ. De la même manière que pour l'Affirmation 2, on restructure dans un premier temps chaque

gadget indépendamment : soit $\gamma \in E_1$ tel que $tr(Z_\gamma) \neq \emptyset$ et $tr(Y_\gamma) \neq Y_\gamma$. Soit $j_0 = \max\{j \in \{1, \cdots, T\} : y_j^\gamma \notin K'\}$ et soit j_1 tel que $z_{j_1}^\gamma \in tr(Z_\gamma)$.
Rappelons que d'après la définition de E_1, il existe $u, b \in \{1, \cdots, n\}$ tel que $z_{j_1}^\gamma$ est adjacent à a_u^b. On a donc

$$\mu(z_{j_1}^\gamma) \geq y + z + 1,$$

avec $y = |N(z_{j_1}^\gamma) \cap Y_\gamma|$ et $z = |N(z_{j_1}^\gamma) \cap Z_\gamma|$. En outre, on a

$$\mu(y_{j_0}^\gamma) \leq z + y + 2,$$

en effet, $|N(y_{j_0}^\gamma) \cap Z_\gamma| = z + 1$, $|N(y_{j_0}^\gamma) \cap Y_\gamma| \leq y$ et $|N(y_{j_0}^\gamma) \cap X_\gamma| = 1$). Comme $y_{j_0}^\gamma$ et $z_{j_1}^\gamma$ sont adjacents, $(z_{j_1}^\gamma, y_{j_0}^\gamma)$ est un remplacement sûr. De plus, après tous ces remplacements, étant donné $\gamma \in E_1$, on a $tr(Z_\gamma) \neq \emptyset \implies tr(Y_\gamma) = Y_\gamma$.
On restructure maintenant les gadgets entre eux. Supposons qu'il existe $a, b \in E_1$ tels que $tr(Z_a) \neq \emptyset$ et $tr(Y_b) \neq Y_b$. Soit j_0 tel que $y_{j_0}^b \notin tr(Y_b)$ et j_1 tel que $z_{j_1}^a \in tr(Z_a)$. On a alors $\mu(z_{j_1}^a) \geq T+1$ et $\mu(y_{j_0}^b) \leq T_1$, et $(y_{j_0}^b, z_{j_1}^a)$ est un remplacement sûr, et ces remplacements se terminent soit lorsque $tr(Y_\gamma) = Y_\gamma$ pour tout $\gamma \in E_1$, soit lorsque $tr(Z_\gamma) = \emptyset$ pour tout $\gamma \in E_1$, comme demandé. □

On définit maintenant, pour chaque $\gamma \in \mathcal{I}$, l'ensemble de sommets $D_\gamma \subseteq Y_\gamma \cup Z_\gamma$ qui doivent être remplacés. Ces cas sont résumés Figure 25.

- Cas A1 : pour tout $\gamma \in E_0$, $D_\gamma = Y_\gamma \cap K'$.
- Cas A2 : pour tout $\gamma \in E_0$, $D_\gamma = Z_\gamma \setminus K'$.
- Cas B1 : pour tout $\gamma \in E_1$, $D_\gamma = Z_\gamma \cap K'$.
- Cas B2 : pour tout $\gamma \in E_1$, $D_\gamma = Y_\gamma \setminus K'$.

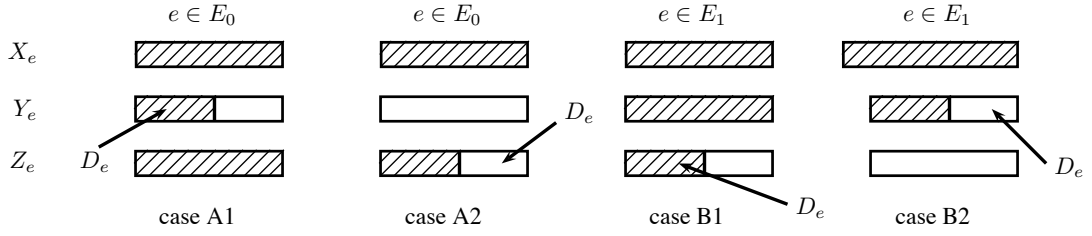


FIGURE 25 – Schéma des différents cas. Les rectangles hachurés représentent les sommets de K'.

On remarque que si $D_\gamma = \emptyset$ pour tout $\gamma \in E_0$ (resp. $\gamma \in E_1$), alors les cas A1 et A2 (resp. B1 et B2) se confondent. Si une telle chose se produit pour tout $i \in \{1, \cdots, m\}$, alors nous pouvons directement conclure, comme l'Affirmation 5. Avant cela, nous montrons que si nous sommes dans le cas A1 et B1 (ou si $D_\gamma = \emptyset$ pour tout $\gamma \in \mathcal{I}$), alors la solution K' parmi la clique A est en fait restreinte à une seule sous-clique A^i pour un certain $i \in \{1, \cdots, t\}$.

Affirmation 4. *Si nous sommes dans les cas A1 et B1 (ou si $D_\gamma = \emptyset$ pour tout $\gamma \in \mathcal{I}$), alors il existe $i \in \{1, \cdots, t\}$ tel que $tr(A) \subseteq A^i$.*

Preuve. Soit $\Delta = \sum_{\gamma \in \mathcal{I}} |D_\gamma|$, et supposons par contradiction qu'il existe $i, j \in \{1, \cdots, t\}$, avec $i \neq j$, tels que $tr(A^i) \neq \emptyset$ et $tr(A^j) \neq \emptyset$. Tout d'abord, puisque nous sommes dans les cas A1 et B1, le nombre d'arêtes induites par chaque gadget est au moins $\binom{T}{2}$. Ensuite, soit S (resp. $\bar{S}$) le nombre de paires de gadgets correspondant aux bits où les représentations binaires de i et j sont les mêmes (resp. diffèrent). On a alors :

$$S + \bar{S} = Mq. \tag{4.2}$$

Et puisque $i \neq j$, les représentations binaires de i et j diffèrent sur au moins un bit, et on a donc $\bar{S} \geq M$.
Comptons alors le nombre d'arêtes induites par chaque paire de gadgets, selon si elles correspondent à une valeur de bit identique pour les représentations de i et j ou non.
Soit $p \in \{1, \cdots, q\}$ tel que les représentations binaires de i et j sont les mêmes. Alors, pour tout $h \in \{1, \cdots, M\}$, trois cas peuvent se produire :

1. $Y_{\alpha_p^h} \subseteq K'$ et $Y_{\beta_p^h} \subseteq K'$. Dans ce cas, la paire de gadgets induit au moins $(2\binom{T}{2} + 2)$ arêtes.

2. $Y_{\alpha_p^h} \subseteq K'$ et $Z_{\beta_p^h} \subseteq K'$ (ou le contraire). Dans ce cas, la paire de gadgets induit au moins $(2\binom{T}{2} + 1)$ arêtes.

3. $Z_{\alpha_p^h} \subseteq K'$ et $Z_{\beta_p^h} \subseteq K'$. Dans ce cas, la paire de gadgets induit au moins $(2\binom{T}{2} + 2T)$ arêtes, puisque soit $Z_{\alpha_p^h}$ ou $Z_{\beta_p^h}$ est adjacent à au moins un sommet de $tr(A^i)$, et au moins un sommet de $tr(A^j)$.

Ainsi, dans tous les cas chaque paire de gadgets induit au moins $(2\binom{T}{2} + 1)$ arêtes. Intéressons nous maintenant à un bit $p \in \{1, \cdots, q\}$ où les représentations binaires de i et j diffèrent. Alors, pour tout $h \in \{1, \cdots, M\}$, remarquons que $Z_{\alpha_p^h}$ et $Z_{\beta_p^h}$ sont tous les deux adjacents à au moins un sommet parmi $A^i \cup A^j$. Ici aussi trois cas peuvent apparaître :

1. $Y_{\alpha_p^h} \subseteq K'$ et $Y_{\beta_p^h} \subseteq K'$. Dans ce cas, la paire de gadgets induit au moins $(2\binom{T}{2} + 2)$ arêtes.

2. $Y_{\alpha_p^h} \subseteq K'$ et $Z_{\beta_p^h} \subseteq K'$ (ou le contraire). Dans ce cas, la paire de gadgets induit au moins $(2\binom{T}{2} + T + 1)$ arêtes, puisque dans ce cas $Z_{\alpha_p^h}$ et $Z_{\beta_p^h}$ sont chacun adjacents à au moins un sommet de $tr(A^i) \cup tr(A^j)$.

3. $Z_{\alpha_p^h} \subseteq K'$ et $Z_{\beta_p^h} \subseteq K'$. Dans ce cas, la paire de gadgets induit au moins $(2\binom{T}{2} + 2T)$ arêtes, puisque soit $Z_{\alpha_p^h}$ ou $Z_{\beta_p^h}$ est adjacent à au moins un sommet de $tr(A^i)$, et au moins un sommet de $tr(A^j)$.

Ainsi, dans tous les cas chaque paire de gadgets induit au moins $(2\binom{T}{2}+2)$ arêtes. En outre, le nombre d'arêtes induites par $tr(A)$ est au moins $(\binom{T}{2}-\binom{\Delta}{2}-\Delta(T-\Delta))$, puisque c'est une clique de taille $(T-\Delta)$. Pour résumer :

- $tr(A)$ induit $(\binom{T}{2}-\binom{\Delta}{2}-\Delta(T-\Delta))$ arêtes.
- chaque gadget (du sélecteur d'instances ou d'arêtes) induit au moins $\binom{T}{2}$ arêtes (il y a en tout $(m+2Mq)$ gadgets), et plus précisément :
  - chaque paire de gadgets correspondant à une valeur de bit identique dans la représentation binaire de i et j induit $(2\binom{T}{2}+1)$ arêtes (*i.e.* une de plus que les gadgets « normaux »). Il y a S telles paires de gadgets.
  - chaque paire de gadgets correspondant à une valeur de bit différente dans la représentation binaire de i et j induit $(2\binom{T}{2}+2)$ arêtes. Il y a $\bar{S}$ telles paires de gadgets.

  On a donc, en utilisant l'équation (4.2) et le fait que $\bar{S} \geq M$, on a :

$$\begin{aligned} cost(K') &\geq \binom{T}{2}-\binom{\Delta}{2}-\Delta(T-\Delta)+\binom{T}{2}(m+2Mq)+S+2\bar{S} \\ &= \binom{T}{2}-\binom{\Delta}{2}-\Delta(T-\Delta)+\binom{T}{2}(m+2Mq)+Mq+\bar{S} \\ &\geq \binom{T}{2}-\binom{\Delta}{2}-\Delta(T-\Delta)+\binom{T}{2}(m+2Mq)+Mq+M \end{aligned}$$

  et donc, en rappelant que $C' = \binom{T}{2}+\binom{T}{2}(m+2Mq)+(m-\binom{k}{2})+Mq$, on a :

$$cost(K') - C' \geq M - m + \binom{k}{2} - \binom{\Delta}{2} - \Delta(T-\Delta)$$

  Puisque $M = n^6$, on a $cost(K') > C'$, ce qui est impossible. Ainsi, il est impossible d'avoir de tels i et j, et l'affirmation suit.

□

L'affirmation suivante nous permettra ensuite de conclure chaque cas.

Affirmation 5. *Si $D_\gamma = \emptyset$ pour tout $\gamma \in \mathcal{I}$, alors il existe $i \in \{1, \cdots, t\}$ tel que G_i contient une clique de taille k.*

Preuve. Par construction on a $|tr(A)| = T$ et $|tr(F_\gamma)| = 2T$ pour tout $\gamma \in \mathcal{I}$. Donc, $cost(tr(A)) = \binom{T}{2}$ et $cost(tr(F_\gamma)) = \binom{T}{2}+1$ si $\gamma \in E_1$, et $cost(tr(F_\gamma)) = \binom{T}{2}$ si $\gamma \in E_0$. On a donc $cost(K') \geq \binom{T}{2}+\binom{T}{2}(m+2Mq)+|E_1|$.
Par l'Affirmation 4, il existe un unique $i \in \{1, \cdots, t\}$ tel que $tr(A) \subseteq A^i$. Ainsi, il

existe au plus Mq gadgets parmi le sélecteur d'instances qui ne sont pas adjacents à $tr(A)$, et qui appartiennent donc à E_0. Ceci implique qu'il existe au moins Mq gadgets parmi le sélecteur d'instances qui appartiennent à E_1. Soit $E_0^e = E_0 \cap \{1, \cdots, m\}$ la restriction de E_0 au sélecteur d'arêtes, et $E_1^e = \{1, \cdots, m\} \setminus E_0^e$. Les arguments précédents permettent de montrer que $|E_1^e| \leq m - \binom{k}{2}$, ce qui implique $|E_0^e| \geq \binom{k}{2}$. De plus, chaque gadget $j \in E_0^e$ correspondant à l'arête $e_j = \{u, v\}$ de G_i est adjacent aux cliques A_u^i et A_v^i, qui doivent être telles que $tr(A_u^i) = tr(A_v^i) = \emptyset$ par définition de E_0. Cependant, puisque $|tr(A)| = |tr(A^i)| = T$, le nombre de telles cliques est au plus $n - \lceil \frac{T}{n} \rceil = k$. Ceci prouve que les $|E_0^e|$ arêtes de G correspondantes peuvent être induites par au plus k sommets, *i.e.* G contient une clique de taille k. □

Il ne nous reste plus qu'à combiner les quatre cas des Affirmations 2 et 3, et de montrer qu'à chaque fois, l'Affirmation 5, permettant de conclure, s'applique.

Cas A1 et B1. Soit $\Delta = \sum_{\gamma \in \mathcal{I}} |D_\gamma|$, et supposons que $\Delta > 0$ (autrement l'Affirmation 5 s'applique). Comptons alors simplement le nombre d'arêtes induites par une telle solution. Pour cela, nous comptons séparément le nombre d'arêtes induites par les sommets de la solution parmi A, et le nombre d'arêtes induites par les sommets de la solution parmi chaque gadget.
Tout d'abord, il est clair que $|tr(A)| = T - \Delta$, et donc le nombre d'arêtes induites par $tr(A)$ est $(\binom{T}{2} - \binom{\Delta}{2} - \Delta(T - \Delta))$, puisque A est une clique. De plus, puisque nous sommes dans les cas A1 et B1, la trace de la solution dans chaque gadget (du sélecteur d'instances ou d'arêtes) induit au moins $\binom{T}{2}$ arêtes. Plus précisément, pour chaque gadget $\gamma \in \mathcal{I}$, trois cas peuvent apparaître :

- si $D_\gamma = \emptyset$, alors $tr(F\gamma)$ induit exactement $\binom{T}{2}$ arêtes si $\gamma \in E_0$, et exactement $(\binom{T}{2} + 1$ arêtes si $\gamma \in E_1$.
- si $D_\gamma \neq \emptyset$, alors :
  - si $\gamma \in E_0$, alors comme chaque sommet de Y_γ est adjacent à tous les sommets de Z_γ, et à un sommet de X_γ, $tr(F_\gamma)$ induit exactement $(\binom{T}{2} + |D_\gamma|(T+1))$ arêtes.
  - si $\gamma \in E_1$, alors puisque chaque sommet de Z_γ est adjacent à tous les sommets de Y_γ, et à au moins un sommet de $tr(A)$, $tr(F_\gamma)$ induit au moins $(\binom{T}{2} + 1 + |D_\gamma|(T+1))$ arêtes (on rappelle que si $\Delta_\gamma = \emptyset$, le gadget induit exactement $\binom{T}{2}$ arêtes).

Ainsi, en sommant sur tous les gadgets, la solution parmi les gadgets induit $(\binom{T}{2}(m + 2Mq) + |E_1| + \Delta(T+1))$ arêtes.
On définit $E_0^e = E_0 \cap \{1, \cdots, m\}$ la restriction de E_0 aux indices des gadgets du sélecteur d'arêtes, et $E_0^b = E_0 \setminus E_0^b$ la restriction de E_0 aux indices des gadgets du sélecteur d'instances. De la même manière on définit $E_1^e = E_1 \cap \{1, \cdots, m\}$ et $E_1^b = E_1 \setminus E_1^e$. Par l'Affirmation 4, il existe un unique $i \in \{1, \cdots, t\}$ tel que $tr(A) \subseteq A^i$. Ceci implique que $|E_0^b| = |E_1^b| = Mq$. Informellement, cela

signifie que pour chaque paire de gadgets du sélecteur d'instances, seulement un des deux est connecté à A^i et donc à $tr(A)$, en fonction de la valeur du bit correspondant. On a donc $|E_1| = Mq + |E_1^e| = Mq + m - |E_0^e|$. En combinant ces égalités, on a :

$$\begin{aligned} cost(K') - C' &\geq \binom{k}{2} + \Delta(T+1) - |E_0^e| - \binom{\Delta}{2} - \Delta(T-\Delta) \\ &= \frac{\Delta(\Delta+3)}{2} + \binom{k}{2} - |E_0^e| \end{aligned}$$

Et donc, si l'on suppose que $cost(K') - C' \leq 0$, on a :

$$|E_0^e| \geq \frac{\Delta(\Delta+3)}{2} + \binom{k}{2} \tag{4.3}$$

De l'autre côté, le nombre de sommets de G_i induisant les arêtes de E_0^e est au plus $k + \lfloor \frac{\Delta}{n} \rfloor$. On a donc $|E_0^e| \leq \binom{(k+\lfloor \frac{\Delta}{n} \rfloor)}{2}$, et donc :

$$\binom{k + \lfloor \frac{\Delta}{n} \rfloor}{2} \geq \frac{\Delta(\Delta+3)}{2} + \binom{k}{2}$$

Si $\Delta < n$, alors $\lfloor \frac{\Delta}{n} \rfloor = 0$, ce qui contredit l'inégalité précédente. Si $\Delta > n$, alors cela contredit l'inégalité (4.3) puisque par définition $|E_0^e| \leq m$. On doit ainsi avoir $\Delta = 0$, et l'affirmation 5 permet de conclure.

Cas A2 et B2. On pose $\Delta_0 = \sum_{\gamma \in E_0} |D_\gamma|$, $\Delta_1 = \sum_{\gamma \in E_1} |D_\gamma|$, et $\Delta = \Delta_0 + \Delta_1$. On remarque que pour tout $u \in tr(A)$, on a $\mu(u) \geq T$. D'un autre côté, pour tout $\gamma \in \mathcal{I}$ tel qu'il existe $v \in D_\gamma$, on a $\mu(v) \leq T$ (en effet, si $\gamma \in E_1$, alors $D_\gamma \subseteq Y_\gamma$, et si $\gamma \in E_0$, alors v n'est pas adjacent à $tr(A)$ par définition de E_0). Ainsi, (u, v) est un remplacement sûr. On peut facilement voir qu'il est possible de répéter ce remplacement Δ fois, et qu'après ceux-ci, on a $D_\gamma = \emptyset$ pour tout $\gamma \in \mathcal{I}$. On conclut enfin en utilisant l'Affirmation 5.

Cas A2 et B1. S'il existe $\gamma \in E_0$ tel qu'il existe $u \in D_\gamma$, alors $\mu(u) < T$. Si un tel sommet existe, alors soit $|tr(A)| > T$, ou alors il existe $\gamma' \in E_1$ tel qu'il existe $v \in D_{\gamma'}$. Dans le premier cas, pour tout $x \in tr(A)$ on a $\mu(x) \geq T$, et (u, x) est un remplacement sûr. Dans le second cas, on a $\mu(v) > T$ et ici aussi (u, v) est un remplacement sûr. À la fin de ces remplacements on a également que $D_\gamma = \emptyset$ pour tout $\gamma \in E_0$, et l'Affirmation 5 permet de conclure.

Cas A1 et B2. S'il existe $\gamma \in E_1$ tel qu'il existe $u \in D_\gamma$, alors $\mu(u) < T$. Si un tel sommet existe, alors soit $|tr(A)| > T$, ou il existe $\gamma' \in E_0$ tel qu'il existe $v \in D_{\gamma'}$. Dans le premier cas, pour tout $x \in tr(A)$ on a $\mu(x) \geq T$, et (u, x) est un remplacement sûr. Dans le second cas on a $\mu(v) > T$ et ici aussi (u, v) est un remplacement sûr. À la fin de ces remplacements, on a $D_\gamma = \emptyset$ pour tout $\gamma \in E_1$, et l'Affirmation 5 permet de conclure.

□

Ce dernier lemme termine la preuve du Théorème 37. □

4.6 Dans les graphes d'intervalles

4.6.1 Introduction et travaux connexes

Rappelons tout d'abord que la complexité ($\mathcal{NP}$-complet *versus* $\mathcal{P}$) de Sparsest k-Subgraph, au même titre que Densest k-Subgraph, n'est toujours pas connue à ce jour dans les graphes d'intervalles ou même dans les graphes d'intervalles propres. Cependant, nous sommes persuadés qu'il s'agit d'une question difficile, pour les raisons suivantes.
Tout d'abord, comme indiqué dans l'introduction, les premiers travaux concernant la complexité de Densest k-Subgraph (et donc indirectement de Sparsest k-Subgraph) ont été menés par D. G. Corneil et Y. Perl [41] en 1984. Dans leur article, ils prouvent entre autres que Densest k-Subgraph est $\mathcal{NP}$-difficile dans les graphes bipartis et les graphes chordaux, et laissent la question de la complexité dans les graphes d'intervalles et les graphes d'intervalles propres non résolue. Depuis, et malgré de nombreuses tentatives de la part de la communauté, aucun avancement majeur n'a été réalisé sur cette question, et un article [93] avait même donné un algorithme polynomial pour Densest k-Subgraph dans les graphes d'intervalles. Cependant, il s'est avéré qu'une des preuves contenait une erreur [94], rendant faux les résultats obtenus. Enfin, on remarque que mis à part des cas triviaux (comme les graphes planaires), les deux problèmes se sont toujours comportés de la même manière sur les classes de graphes jusqu'à présent étudiées. De ce fait, et même si à priori aucun des deux problèmes ne se réduit à l'autre dans les graphes d'intervalles (propres), il est clair que les idées pourraient en revanche se transférer, et résoudre la question pour Sparsest k-Subgraph serait une grande avancée pour la même question concernant Densest k-Subgraph.

Ainsi, et de la même manière que pour d'autres travaux [9, 88, 98], où des algorithmes d'approximation furent développés pour Densest k-Subgraph dans des classes de graphes où la complexité ($\mathcal{NP}$-complet *versus* $\mathcal{P}$) était toujours inconnue, nous présentons dans les sections suivantes des algorithmes d'approximation et paramétrés pour Sparsest k-Subgraph dans les graphes d'intervalles et les graphes d'intervalles propres. Plus précisément, dans la Sous-Section 4.6.3, nous montrons que l'algorithme glouton présenté précédemment, qui donne des solutions arbitrairement mauvaises dans le cas des graphes chordaux et même des graphes d'intervalles, donne des solutions 2-approchées dans le cas des graphes d'intervalles propres. Puis, en adaptant l'algorithme de [98] pour Densest k-Subgraph, nous présentons dans la Sous-Section 4.6.4 une $\mathcal{PTAS}$ pour Sparsest k-Subgraph, toujours dans les

graphes d'intervalles propres. Enfin, nous présentons en Sous-Section 4.6.5 un algorithme $\mathcal{FPT}$ dans les graphes d'intervalles pour SPARSEST k-SUBGRAPH et sa paramétrisation standard (le nombre d'arêtes de la solution). Alors que l'analyse du premier algorithme ressemble à celle de la 2-approximation dans les graphes chordaux (décomposition de la solution en couches, et transformation d'une solution optimale en la solution donnée par l'algorithme), les deux autres algorithmes reposent chacun sur une décomposition du graphe d'entrée en séparateurs, une programmation dynamique venant ensuite parcourir cette décomposition. Avant tout cela, nous donnons les notations nécessaires relatives aux graphes d'intervalles.

4.6.2 Notations

Nous rappelons qu'on graphe d'intervalles est un graphe d'intersection d'un ensemble d'intervalles (connexes) de l'ensemble des réels. Plus précisément, pour un ensemble d'intervalles donnés, le graphe d'intersection correspondant comporte un sommet par intervalle, et on connecte deux sommets si les deux intervalles correspondant s'intersectent. Si aucun intervalle ne contient proprement un autre intervalle, alors on dit que le graphe correspondant est un graphe d'intervalles propres. Dans les deux cas, l'ensemble d'intervalles qui correspond à un certain graphe d'intervalles (propres) est appelé *modèle d'intersection* du graphe. Les graphes d'intervalles peuvent être reconnus, et un modèle d'intervalle peut être construit en temps linéaire [20]. Pour cette raison, nous supposerons toujours par la suite que notre graphe est muni d'un modèle d'intersection, et nous ne ferons aucune distinction entre les sommets du graphe et les intervalles correspondants, ainsi qu'entre les arêtes du graphe et les intersections correspondantes.

Pour les trois sous-sections qui suivent, nous supposerons que $G = (V, E)$ est le graphe d'entrée de notre problème (avec, comme précédemment, $|V| = n$ et $|E| = m$). Le modèle d'intersection associé à ce graphe est l'ensemble d'intervalles $\mathcal{I} = \{I_1, \cdots, I_n\}$. Sans perte de généralité, il est possible de supposer que les intervalles sont en fait des intervalles de $\mathbb{N}$, et que, sans changer la topologie du graphe, que toutes les extrémités des intervalles sont distinctes. Soit un intervalle $I \in \mathcal{I}$, on note $r(I) \in \mathbb{N}$ (resp. $l(i) \in \mathbb{N}$) son extrémité droite (resp. gauche). Par extension, pour un ensemble d'intervalles $S \subseteq \mathcal{I}$, on note $r(S) = \arg\max_{I \in S} r(S)$, et $l(S) = \arg\max_{I \in S} l(S)$. Sauf indication contraire, lorsqu'un ensemble $S \subseteq \mathcal{I}$ d'intervalles sera considéré, on supposera toujours qu'ils sont triés par extrémité droite croissante (on peut vérifier qu'un tel ordre est un ordre d'élimination simplicial, puisque tous les voisins d'un intervalle I « à droite » de l'ordre intersecteront forcément $r(I)$, et formeront ainsi une clique). Ainsi, pour $S \subseteq \mathcal{I}$, et $p \leq |S|$, on appellera p *premiers* (resp. *derniers*) *intervalles de* S l'ensemble des p premiers (resp. derniers) intervalles de l'ensemble S selon l'ordre donné précédemment. Enfin, comme précédemment dans le chapitre, pour tout $S \subseteq \mathcal{I}$, on note $cost(S)$ le nombre d'intersections parmi les intervalles de S.

4.6.3 Analyse de l'algorithme glouton dans les graphes d'intervalles propres

Introduction et définition de l'algorithme

Dans cette sous-section, nous analysons l'algorithme glouton présenté en introduction de la section 4.3, où un algorithme approché a été présenté dans les graphes chordaux. Nous montrons que cet algorithme glouton donne, dans les graphes d'intervalles propres, une solution 2-approchée. Ainsi, l'intérêt du résultat qui suit n'est pas d'obtenir un algorithme d'approximation pour le cas des graphes d'intervalles propres, puisque l'algorithme précédent donne des solutions 2-approchées dans une classe de graphes plus générale. Son intérêt est simplement d'étudier ses performances, en remarquant que pour les graphes d'intervalles propres, il peut être arbitrairement mauvais (*c.f.* Figure 20), alors qu'il donne une solution 2-approchée pour les graphes d'intervalles propres. Il est également intéressant de noter que cet algorithme est plus simple que celui présenté dans le cas des graphes chordaux.

L'algorithme consiste à prendre successivement des ensembles indépendants maximaux du graphe, jusqu'à atteindre k sommets. Étant donné que la recherche d'un ensemble indépendant dans un graphe d'intervalles propres se fait en temps polynomial, l'algorithme ainsi obtenu est bien polynomial. La description formelle ainsi que les notations qui y sont liées sont décrites dans l'Algorithme 4.

Algorithme 4 Algorithme glouton pour Sparsest k-Subgraph dans les graphes d'intervalles propres.

$S \leftarrow \emptyset$, $i \leftarrow 0$
tant que $|S| \leq k$ **faire**
  $i \leftarrow i + 1$, $L_i \leftarrow \emptyset$
  tant que $(|S| + |L_i| \leq k)$ et (L_i n'est pas un ensemble indépendant maximum) **faire**
    Ajouter à L_i le premier intervalle de $\mathcal{I}$ qui n'intersecte aucun intervalle de L_i
  fin tant que
  $S \leftarrow S \cup L_i$, $\mathcal{I} \leftarrow \mathcal{I} \setminus L_i$
fin tant que
retourner S

Pour la suite de l'analyse, nous introduisons la définition suivante :

Définition 35. *Soit $S \subseteq \mathcal{I}$ une solution de* Sparsest k-Subgraph *dans les graphes d'intervalles propres. Une décomposition couche par couche de S est une partition $(L_1, \cdots, L_c)$ de S telle que :*

- *L_1 est un ensemble indépendant maximum du graphe induit par S (mais pas nécessairement un ensemble indépendant maximum du graphe d'entrée).*

- *Pour tout* $l = 2, \cdots, c$, L_l *est défini récursivement comme un ensemble indépendant maximum (non vide) du sous-graphe induit par* $S \setminus \bigcup_{i=1}^{l-1} L_i$.

De plus, pour tout $i \in \{1, \cdots, c\}$, *on note* $x_i = |L_i|$.

Il est facile de voir que les ensembles L_i créés dans l'Algorithme 4 est une décomposition couche par couche de la solution créée S. En effet à chaque étape L_i est même un ensemble indépendant maximum de $\mathcal{I} \setminus \bigcup_{l=1}^{i-1} L_l$. Cette propriété nous assure que lorsque le coût optimal du graphe d'entrée est 0, l'algorithme retourne bien une solution optimale.

L'analyse

La preuve repose principalement sur le lemme suivant :

Lemme 26. *Soit* S *une solution quelconque à* Sparsest k-Subgraph *dans les graphes d'intervalles propres, et soit* $(L_1, \cdots, L_c)$ *une décomposition de* S *couche par couche. Le coût de la solution vérifie :*

$$\sum_{l=2}^{c}(l-1)x_l \leq cost(S) \leq 2\sum_{l=2}^{c}(l-1)x_l$$

Preuve. Soit $l \in \{2, \cdots, c\}$ et $I_j \in L_l$. Comptons le nombre d'intersections entre I_j et les intervalles des couches précédentes : soit $l' < l$. L'argument principal est que le nombre d'intersections entre I_j et les intervalles de $L_{l'}$ est soit un soit deux. En effet, il y a au moins un intervalle $I_{j'} \in L_{l'}$ tel que $I_{j'}$ et I_j s'intersectent, car autrement $L_{l'}$ ne serait pas un ensemble indépendant maximum. De plus, si I_j intersecte trois (ou plus) intervalles de $L_{l'}$, alors l'un de ces intervalles serait inclu proprement dans I_j, ce qui est impossible puisque nous sommes dans le cas des graphes d'intervalles propres. Pour tout $l, l' \in \{1, \cdots c\}$, on définit $cost(l', l)$ comme le nombre d'intersections entre les intervalles de $L_{l'}$ et les intervalles de L_l. Ainsi, on a $x_l \leq cost(l', l) \leq 2x_l$. En sommant sur toutes les couches, on a

$$cost(S) = \sum_{l=2}^{c}\sum_{l'=1}^{l-1} cost(l', l) \leq 2\sum_{l=2}^{c}(l-1)x_l$$

La borne inférieure est obtenue de la même manière, en utilisant la même sommation. □

Pour la suite de l'analyse, on considère S^* une solution optimale au problème, et $(L_1^*, \cdots, L_{c^*}^*)$ une décomposition couche par couche de S^*. De plus, pour tout $i \in \{1, \cdots, c^*\}$, on note $|L_i^*| = x_i^*$.

Étant donné le lemme précédent, si S^* désigne une solution optimale au problème, et S la solution de l'algorithme, on a

$$cost(S^*) \geq \sum_{l=2}^{c^*}(l-1)x_l^*$$

et

$$cost(S) \leq 2\sum_{l=2}^{c}(l-1)x_l$$

Ainsi, afin d'obtenir un rapport d'approximation de deux, il nous suffit de montrer que

$$\sum_{l=2}^{c}(l-1)x_l \leq \sum_{l=2}^{c^*}(l-1)x_l^*$$

Ceci est prouvé en combinant les deux lemmes suivants 27 et 28. Informellement, cette inégalité est vraie car l'Algorithme 4 maximise les premiers x_l (qui a un petit cœfficient), étant donné que chaque couche est un ensemble indépendant maximum dans le graphe restant.

Lemme 27. *Pour tout $l \in \{1, \cdots, \min\{c, c^*\}\}$, on a $\sum_{i=1}^{l} x_i \geq \sum_{i=1}^{c^*} x_i^*$.*

Preuve. Soit $\underline{c} = \min\{c, c^*\}$. Soit $l \in \{1, \cdots, \underline{c}\}$, on définit $F_l(S) = \sum_{i=1}^{l} x_i$, et $F(S) = (F_1(S), \cdots, F_{\underline{c}}(S))$. Le but est de re-structurer S^* en plusieurs étapes (et couche par couche), définissant ainsi une suite de solutions intermédiaires $S_1^*, \cdots, S_t^*$ pour un certain $t \in \mathbb{N}$, avec $S_1^* = S^*$, $S_t^* = S$, et pour tout $j \in \{2, \cdots, t\}$, $|S_j^*| = k$, et $F(S_{j-1}^*) \leq F(S_j^*)$ (où $\leq$ est l'ordre produit usuel sur $\mathbb{N}^{\underline{c}}$).
On décrit maintenant la manière de transformer S_i^* en S_{i+1}^*. Soit $(L_1^*, \cdots, L_{c'}^*)$ la décomposition couche par couche de S_i^*, et supposons que les $(x-1)$ premières couches sont égales aux $(x-1)$ premières couches de S^*. Autrement dit, $x \geq 1$ est la plus petite valeur telle que $L_x^* \neq L_x$, et $L_l^* = L_l \ \forall l \in \{1, \cdots, x-1\}$. On pose $L_x = \{i_1, \cdots, i_a\}$, et $L_x^* = \{i_1^*, \cdots, i_b^*\}$. Notons tout d'abord que $b \leq a$, puisque par construction L_x est un ensemble indépendant maximum du graphe induit par $\mathcal{I} \setminus \bigcup_{i=1}^{x-1} L_i$.
On distingue maintenant deux cas :

- Supposons d'abord que $L_x^* \subseteq L_x$, impliquant que L_x^* n'est pas la dernière couche. Dans ce cas là on définit S_{i+1}^* en ajoutant un intervalle $I \in L_x \setminus L_x^*$ à L_x^* (on rappelle que I n'appartient pas à L_l^* quel que soit $l > x$, étant donné que L_x^* est maximum), et en supprimant un intervalle quelconque de L_l^*, pour un $l > x$. Cette transformation nous assure que $F(S_{i+1}^*) \geq F(S_i^*)$.

- Dans le second cas, soit $j \leq b$ la valeur minimum telle que $i_j^* \neq i_j$, et $i_l^* = i_l$ pour tout $l \in \{1, \cdots, j\}$. Autrement dit, j est le plus petit indice tel que les intervalles diffèrent entre L_x^* et L_x. On a alors $i_j < i_j^*$ (ce qui signifie que

l'intervalle i_j est à gauche de i^*_j), étant donné que l'algorithme crée L_x en choisissant i_j comme le premier intervalle non adjacent à ceux précédemment choisis. Deux sous-cas sont alors possibles :

- Si i_j n'est pas utilisé dans S^*_i (c'est-à-dire $i_j \notin \bigcup_{l=x+1}^{c^*} L^*_l$), alors on ajoute i_j à L^*_x, et on supprime i^*_j de L^*_x. Avec cette re-structuration, L^*_x est toujours un ensemble indépendant, puisque $i_j < i^*_j$.
- Enfin, on considère le cas où $i_j \in S^*_i$. Plus précisément il existe $y > x$ tel que $i_j \in L^*_y$ (*c.f.* Figure 26). Posons alors $L^*_y = \{i'_1, \cdots, i'_{a'}\}$, et soit p tel que $i'_p = i_j$. On effectue alors les re-structurations suivantes entre L^*_x et L^*_y. Tout d'abord, on ajoute i_j $(= i'_p)$ à L^*_x. Comme L^*_x est en particulier maximal, il y a forcément une arête entre i_j et i^*_j. On peut ainsi supprimer i^*_j de L^*_x, et l'ajouter à L^*_y. S'il n'y a pas d'arête entre i^*_j et i'_{p+1}, alors la re-structuration est finie (et la construction de S^*_{i+1} est terminée). Autrement, on continue les échanges tant que le dernier intervalle échangé n'intersecte pas un autre intervalle, ou tant que l'une des couches est vide. Plus précisément, soit $a \geq 0$ le plus grand indice tel que $(i'_p, i^*_j), (i^*_j, i'_{p+1}), (i'_{p+1}, i^*_{j+1}), \cdots, (i^*_{j+a}, i'_{p+a+1})$ sont dans E. On supprime d'abord $\{i'_p, \cdots, i'_{p+a+1}\}$ de L^*_y, et on les ajoute dans L^*_x. Puis, on supprime $\{i^*_j, \cdots, i^*_{j+a}\}$ de L^*_x, et on les ajoute à L^*_y. Enfin, si $(i'_{p+a+1}, i^*_{j+a+1}) \in E$, on supprime également i^*_{j+a+1} de L^*_x, et on l'ajoute à L^*_y. Remarquons qu'après tous ces échanges, soit les cardinalités de L^*_x et L^*_y n'ont pas bougé, soit L^*_x a un intervalle en plus et L^*_y un intervalle en moins. Dans les deux cas, on a $F(S^*_{i+1}) \geq F(S^*_i)$, ce qui termine la démonstration.

□

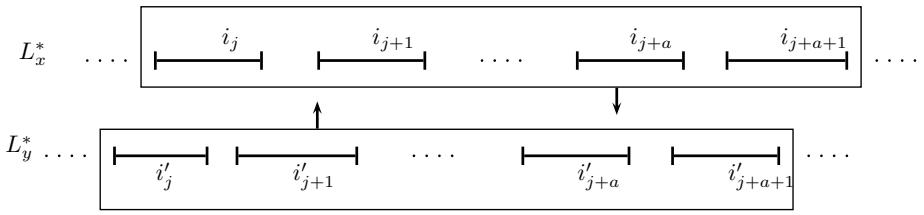


FIGURE 26 – Exemple de re-structurations de S^*_i à S^*_{i+1}.

Lemme 28. *Étant donnés* $c \leq c^*$, $(x_1, \cdots, x_c)$, $(x^*_1, \cdots, x^*_{c^*})$, *et* $(a_1, \cdots, a_{c^*})$ *tels que :*

- $\sum_{i=1}^{c} x_i = \sum_{i=1}^{c^*} x^*_i$
- $\forall l \in \{1, \ldots, c\}, \sum_{i=1}^{l} x_l \geq \sum_{i=1}^{l} x^*_l$

- $a_i \leq a_{i+1}$

On a $\sum_{l=1}^{c} a_l x_l \leq \sum_{l=1}^{c^*} a_l x_l^*$.

Preuve. La preuve est par induction sur c.
Le cas $c = 1$ est trivial. Supposons maintenant que le lemme est vrai pour tout $c \leq c^* \leq n-1$, et considérons le cas $c \leq c^* \leq n$. Il est clair d'après les hypothèses que $x_1 \geq x_1^*$. Soit $\Delta \geq 0$ tel que $x_1 = x_1^* + \Delta$. On ré-équilibre les cœfficients en définissant $x_1'^* = x_1^* + \Delta$, $x_2'^* = x_2^* - \Delta$ (on peut alors avoir $x_2'^* \leq 0$), et $x_l'^* = x_l^*$ pour $l \geq 3$.
On obtient alors $\sum_{l=1}^{c} a_l x_l = a_1 x_1 + \sum_{l=2}^{c} a_l x_l = a_1 x_1'^* + \sum_{l=2}^{c} a_l x_l$ (puisque par définition $x_1'^* = x_1$). Par récurrence, on a $\sum_{l=2}^{c} a_l x_l \leq \sum_{l=2}^{c^*} a_l x_l'^*$. Enfin, puisque $\sum_{l=1}^{c^*} a_l x_l'^* \leq \sum_{l=1}^{c^*} a_l x_l^*$, le résultat suit. □

Nous avons maintenant toutes les briques nécessaires pour prouver le résultat :

Théorème 38. *L'Algorithme 4 est un algorithme* 2*-approché pour* Sparsest k-Subgraph *dans les graphes d'intervalles propres. De plus, cette borne est atteinte.*

Preuve. Soit S la solution retournée par l'algorithme, et S^* une solution optimale au problème. Comme précédemment $(x_1, \cdots, x_c)$ et $(x_1^*, \cdots, x_{c^*}^*)$ désignent les cardinalités de chaque couche de S et S^* respectivement. D'après les Lemmes 26, 27 et 28, on a :

$$cost(S) \leq 2\sum_{l=2}^{c}(l-1)x_l \leq 2\sum_{l=2}^{c^*}(l-1)x_l^* \leq 2cost(S^*)$$

Ce qui prouve le rapport d'approximation.
Enfin, la Figure 27 montre un ensemble de sept intervalles propres, pour laquelle, avec $k = 5$, l'Algorithme 4 donne une solution de coût quatre, alors qu'une solution de coût deux existe. □

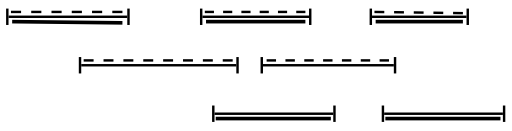


Figure 27 – Instance qui atteint le rapport d'approximation de deux. Les intervalles de l'instance sont dessinés avec des traits continus. La solution de l'algorithme est représentée par des pointillés, et une solution optimale est représentée par des traits gras.

4.6.4 $\mathcal{PTAS}$ par programmation dynamique dans les intervalles propres

Dans cette sous-section, nous montrons comment obtenir une $\mathcal{PTAS}$ pour SPARSEST k-SUBGRAPH dans les graphes d'intervalles propres. L'algorithme peut être vu en fait comme une adaptation de l'algorithme présenté dans [98] donnant une $\mathcal{PTAS}$ pour DENSEST k-SUBGRAPH dans les graphes d'intervalles. L'algorithme décompose d'abord notre instance de manière gloutonne en un chemin de séparateurs. Puis, on montre que dans chaque séparateur, toute solution peut être modifiée en une autre solution dont le coût est peu degradé, et qui est assez simple pour que toutes les solutions de ce type puissent être énumérées en temps polynomial. Enfin, une programmation dynamique parcourt la décomposition, en énumérant à chaque bloc toutes les solutions simples possibles. Nous supposerons d'abord que l'instance est connexe, et montrerons à la fin comment adapter à un graphe non connexe.

Décomposition du graphe

On décrit maintenant la décomposition de l'instance. On rappelle que cette dernière est un ensemble d'intervalles propres $\mathcal{I} = \{I_1, \cdots, I_n\}$, où les intervalles sont triés par extrémité droite croissante (on remarquera que pour les graphes d'intervalles propres, trier par extrémité droite ou gauche revient en fait au même). La décomposition consiste en une suite de blocs $(B_1, \cdots, B_a)$, chaque bloc B_i étant lui même décomposé en un sous-bloc gauche L_i et droit R_i.
Plus formellement, on pose $I_{m_1} = I_1$, $L_1 = \{I_{m_1}\}$, $R_1 = \{I_j : j > m_1,\ I_j$ intersecte $I_{m_1}\}$. Puis, étant donné $i > 1$, on définit (tant qu'il reste des instances à droite de R_i :

- $I_{m_{i+1}}$ est le dernier intervalle qui intersecte un intervalle de R_i.
- $L_{i+1} = \{I_j : j \leq m_{i+1},\ I_j$ intersecte $I_{m_{i+1}}$, et $I_j \notin R_i\}$.
- $R_{i+1} = \{I_j : j > m_{i+1},\ I_j$ intersecte $I_{m_{i+1}}\}$.

Un schéma de la décomposition est donné à la Figure 28.

On note a le plus grand entier pour lequel I_{m_i} est défini, et pour tout $i \in \{1, \cdots, a\}$, on note $B_i = L_i \cup R_i$. Ainsi, par définition, pour tout $i \in \{1, \cdots, a\}$, (L_i, R_i) forme une partition de B_i. Il est également facile de voir que L_i et R_i forment chacun une clique et un séparateur du graphe. Enfin, on remarque que $I_{m_i} \in L_i$ pour tout $i \in \{1, \cdots, a\}$, et que R_a peut être vide.
Pour tout $i \in \{1, \cdots, a\}$, et toute solution $S \subseteq \mathcal{I}$, on note $L_i^S = L_i \cap S$, $R_i^S = R_i \cap S$, et $B_i^S = B_i \cap S$.
Avec ces définitions, il est possible de décomposer le coût d'une solution S comme la somme des coûts à l'intérieur de chaque bloc, et les coûts entre deux blocs successifs.

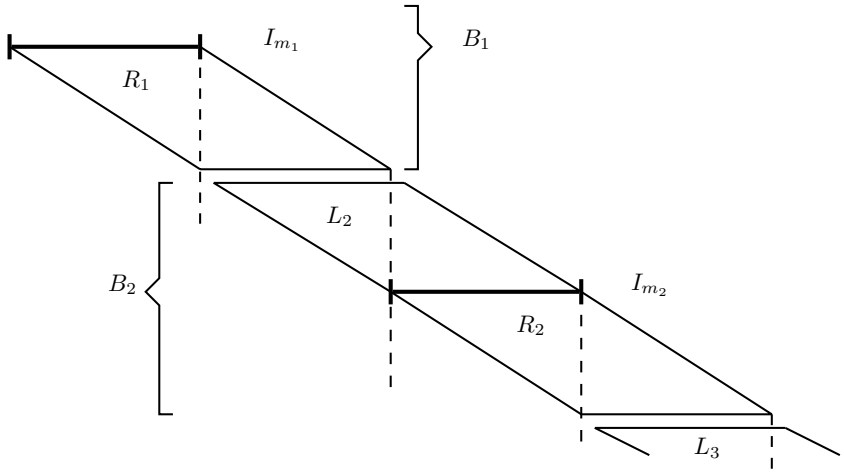


FIGURE 28 – Schéma de la décomposition utilisée.

Autrement dit, pour $S \subseteq \mathcal{I}$, on a :

$$cost(S) = \sum_{i=1}^{a} cost(B_i^S) + \sum_{i=1}^{a-1} cost(R_i^S, L_{i+1}^S)$$

où $cost(P, Q)$ est égal au nombre d'arêtes entre les intervalles de P et les intervalles de Q, pour $P, Q \subseteq \mathcal{I}$.
Enfin, il est clair que cette décomposition peut être réalisée de manière gloutonne en temps polynomial. Dans le prochain paragraphe, on montre comment modifier une solution en une solution possédant une structure simple. Pour cela, nous introduisons la notion de *compactage*, à ne pas confondre avec les problèmes de compaction du Chapitre 3.

Compactage de blocs

Soit $Comp$ une fonction injective de $\mathcal{I}$ dans $\mathcal{I}$. Pour tout $S \subseteq \mathcal{I}$, on note $Comp(S) = \bigcup_{I \in S} Comp(I)$. La fonction $Comp$ est un compactage si pour tout $S \subseteq \mathcal{I}$, et tout $i \in \{1, \cdots, a\}$, les conditions suivantes sont vérifiées :

- pour tout $I \in R_i^S$, on a $Comp(I) \in R_i$ et $r(Comp(I)) \leq r(I)$
- pour tout $I \in L_i^S$, on a $Comp(I) \in L_i$ et $r(I) \leq r(Comp(I))$

Informellement, un compactage permet de « pousser » les intervalles de B_i^S vers le centre I_{m_i}. L'idée est qu'après une telle modification, le coût d'une solution peut augmenter à l'intérieur de chaque bloc, mais pas entre les blocs.
On dira que $Comp$ est un ρ-compactage si le coût d'une solution n'augmente pas plus d'un facteur ρ à l'interieur de chaque bloc. Autrement dit, si pour tout $S \subseteq \mathcal{I}$ et pour tout $i \in \{1, \cdots, a\}$, on a $cost(Comp(B_i^S)) \leq \rho \cdot cost(B_i^S)$. L'utilité d'un

ρ-compactage est de pouvoir calculer l'augmentation du coût à l'intérieur de chaque bloc seulement, comme l'illustre le lemme suivant.

Lemme 29. *Si Comp est un ρ-compactage, alors pour tout solution $S \subseteq \mathcal{I}$, on a $cost(Comp(S)) \leq \rho \cdot cost(S)$.*

Preuve. Par définition de la décomposition utilisée, on a :

$$\begin{aligned} cost(Comp(S)) &= \sum_{i=1}^{a} cost(Comp(B_i^S)) + \sum_{i=1}^{a-1} cost(Comp(R_i^S), Comp(L_{i+1}^S)) \\ &\leq \sum_{i=1}^{a} \rho.cost(B_i^S) + \sum_{i=1}^{a-1} cost(Comp(R_i^S), Comp(L_{i+1}^S)) \end{aligned}$$

Il est ainsi suffisant de montrer que :

$$\sum_{i=1}^{a-1} cost(Comp(R_i^S), Comp(L_{i+1}^S)) \leq \sum_{i=1}^{a-1} cost(R_i^S, L_{i+1}^S)$$

Soit $I_R \in R_i^S$ et $I_L \in L_{i+1}^S$ tels que I_R et I_L ne s'intersectent pas. Par définition d'un compactage, on a $r(Comp(I_R)) \leq r(I_R)$ et $l(I_L) \leq l(Comp(I_L))$. Ainsi, les intervalles $Comp(I_R)$ et $Comp(I_L)$ ne s'intersectent pas, ce qui prouve le résultat. □

Ainsi, d'après le lemme précédent, il est suffisant de se concentrer sur chaque bloc, et de trouver un compactage qui va préserver les coûts à l'intérieur de ceux-ci. Étant donné $\epsilon > 0$ fixé, le but est de trouver un $(1+\epsilon)$-compactage ayant un structure simple.

Lemme 30. *Pour tout $P \in \mathbb{N}$ fixé, il existe un $(1+\frac{4}{P})$-compactage tel que pour tout $X \subseteq \mathcal{I}$, $Comp(X)$ peut être décrit par $(2P+2)$ variables, chacune appartenant à $\{0, \cdots, n\}$.*

Preuve. D'après le Lemme 29, nous ne décrivons $Comp(X)$ que pour $X \subseteq B_i$, quel que soit $i \in \{1, \cdots, a\}$. On décompose $X = X_L \cup X_R$, avec $X_L \subseteq L_i$ et $X_R \subseteq R_i$. On pose également $x_L = |X_L|$ et $x_R = |X_R|$. Enfin, on effectue la division euclidienne de x_L et x_R par P : $x_L = q_L P + r_L$ et $x_R = q_R P + r_R$, avec $r_L, r_R < P$.
La première étape consiste à découper X_L et X_R en P sous-ensembles d'intervalles consécutifs, formant ainsi deux suites d'ensembles $(G_t^L)_{t=1,\cdots,P}$ et $(G_t^R)_{t=1,\cdots,P}$, avec pour tout $t \in \{1, \cdots, P\}$:

$$|G_t^L| = \begin{cases} q_L + 1 & \text{si} \quad t \in \{1, \cdots, r_L\} \\ q_L & \text{si} \quad t \in \{(r_L+1), \cdots, P\} \end{cases}$$

$$|G_t^R| = \begin{cases} q_R + 1 & \text{si} \quad t \in \{1, \cdots, r_R\} \\ q_R & \text{si} \quad t \in \{(r_R+1), \cdots, P\} \end{cases}$$

Puis, pour tout $t \in \{1, \cdots, P\}$, on note I_t^L (resp. I_t^R) le dernier (resp. premier) intervalle de G_t^L (resp. G_t^R). Le principe du compactage est, informellement, de faire « glisser » les intervalles de G_t^L (resp. G_t^R) vers la droite (resp. gauche) (on rappelle que les intervalles de G_t^L et G_t^R appartiennent à la solution X).
Plus formellement, pour tout $t \in \{1, \cdots, P\}$, $Comp(G_t^L)$ est défini comme les $|G_t^L|$ derniers intervalles I tels que $r(I) \leq r(I_t^L)$. Similairement $Comp(G_t^R)$ est défini comme les $|G_t^R|$ derniers intervalles I tels que $r(I_t^R) \leq r(I)$. La construction, pour un sous-bloc L_i, est résumé par la Figure 29.

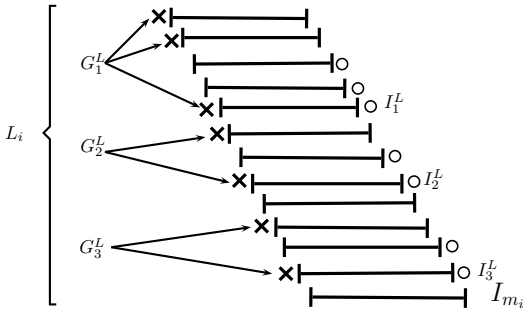


FIGURE 29 – Exemple de compactage d'un ensemble X pour un sous-bloc L_i, avec $P = 3$, et $x_L = 7$. Les intervalles signalés par une croix représentent X, alors que les intervalles signalés par un cercle représentent $Comp(X)$.

Par définition, on peut vérifier que $Comp$ ainsi défini est un compactage. De plus, étant donnés x_L, r_L, x_R, r_R et I_t^L, I_t^R, il est possible de construire $Comp(X)$ en temps polynomial. Il suffit donc maintenant de montrer que la solution n'est pas trop dégradée, *i.e.* que c'est bien un $(1 + \frac{4}{P})$-compactage.
On peut facilement vérifier les remarques suivantes :

(i) tous les intervalles de L_i forment une clique, de même que les intervalles de R_i.

(ii) pour tout $t_1, t_2 \in \{1, \cdots, P\}$, avec $t_1 \neq P$ et $t_2 \neq 1$, si un intervalle de $Comp(G_{t_1}^L)$ intersecte un intervalle de $Comp(G_{t_2}^R)$, alors pour tout $s_1 \in \{(t_1 + 1), \cdots, P\}$ et tout $s_2 \in \{1, \cdots, (t_2 - 1)\}$, tous les intervalles de $G_{s_1}^L$ intersectent tous les intervalles de $G_{s_2}^R$.

Pour tout $t \in \{1, \cdots, P\}$, on note $x_t^L = |G_t^L| = |Comp(G_t^L)|$ et $x_t^R = |G_t^R| = |Comp(G_t^R)|$. Par construction, et par la remarque (i), on a :

$$cost(Comp(X)) \leq \binom{x_L}{2} + \binom{x_R}{2} + \sum_{t=1}^{P} cost(Comp(G_t^L), Comp(X) \cap R_i)$$

$$cost(X) \geq \binom{x_L}{2} + \binom{x_R}{2} + \sum_{t=1}^{P} cost(G_t^L, X \cap R_i)$$

Pour tout $t \in \{1, \cdots, P\}$, soit $\lambda_t \in \{0, 1, \cdots, P\}$ le plus grand entier s tel qu'un intervalle de $Comp(G_t^L)$ intersecte un intervalle de $Comp(G_s^R)$ (si un tel intervalle d'existe pas, alors on pose $\lambda_t = 0$). Par la remarque (ii), pour tout $t \in \{1, \cdots, P\}$, on a

$$cost(Comp(G_t^L), Comp(X) \cap R_i) \leq x_t^L \sum_{u=1}^{\lambda_t} x_u^R$$

et, pour tout $t \in \{2, \cdots, P\}$, étant donné que certains intervalles de G_{t-1}^L intersectent certains intervalles de $G_{\lambda_{t-1}}^R$, tous les intervalles de G_t^L intersectent tous les intervalles de $G_{\lambda_{t-1}-1}^R$. Ainsi :

$$cost(G_t^L, C \cap R_i) \geq x_t^L \sum_{u=1}^{\lambda_{t-1}-1} x_u^R.$$

En combinant les inégalités précédentes, on obtient :

$$cost(Comp(X)) \leq \binom{x_L}{2} + \binom{x_R}{2} + \sum_{t=1}^{P} x_t^L \sum_{u=1}^{\lambda_t} x_u^R$$

$$cost(X) \geq \binom{x_L}{2} + \binom{x_R}{2} + \sum_{t=2}^{P} x_t^L \sum_{u=1}^{\lambda_{t-1}-1} x_u^R$$

Ainsi, on a :

$$\Delta = cost(Comp(X)) - cost(X) \leq x_1^L \sum_{u=1}^{\lambda_1} x_u^R + \sum_{t=2}^{P} x_t^L \sum_{u=\lambda_{t-1}}^{\lambda_t} x_u^R.$$

Comme dans notre cas, on a $x_t^L \leq (q_L + 1)$, cela implique :

$$\Delta \leq (q_L + 1)(\sum_{u=1}^{\lambda_P} x_u^R + \sum_{u=1}^{P} x_{\lambda_u}^R) \leq 2(q_L + 1)x_R \leq 2(\frac{x_L}{P} + 1)x_R$$

Il ne reste plus qu'à considérer les cas particuliers, selon les valeurs de x_L et x_R :

- Si $x_L \geq P$, alors $2(\frac{x_L}{P} + 1)x_R \leq \frac{4}{P} x_L x_R$, et :

$$\frac{\Delta}{cost(X)} \leq \frac{\frac{4}{P} x_L x_R}{(x_L - 1)x_L + (x_R - 1)x_R} \leq \frac{\frac{4}{P} x_L x_R}{\frac{1}{2}(x_L^2 + x_R^2)} \leq \frac{4}{P}$$

  (remarquons que dans un premier temps $cost(X)$ a été majoré par $\binom{x_L}{2} + \binom{x_R}{2}$, et que dans un second temps $(x_R - 1)$ a été minoré par $\frac{x_R}{2}$, étant donné que les cas avec $x_R \leq 1$ ne peuvent donner qu'un meilleur rapport d'approximation).

- Si $x_L < P$, alors on pose dans ce cas $Comp(X \cap L_i) = X_L$ (autrement dit, on garde la partie gauche inchangée). Si $x_R < P+1$, alors on pose $Comp(X) = X$, et on obtient un 1-compactage, toujours constructible en temps polynomial. Si $x_R \geq P + 1$. On peut alors améliorer la minoration précédente, et écrire

$$cost(X) \geq \frac{(x_L - 1)x_L}{2} + \frac{(x_R - 1)x_R}{2} + \sum_{t=1}^{P} x_t^L (\sum_{u=1}^{\lambda_t - 1} x_u^R)$$

En effet, dans ce cas pour tout $t \in \{1, \cdots, x_L\}$, l'ensemble G_t^L est un singleton (et $G_t^L = \emptyset$ pour $t \geq x_L + 1$). Ainsi, l'intervalle de G_t^L intersecte certains intervalles de $G_{\lambda_t}^R$, ce qui implique qu'il intersecte tous les intervalles de $G_{\lambda_t - 1}^R$. On a donc

$$\Delta \leq \sum_{t=1}^{P} x_t^L x_{\lambda_t}^R \leq \sum_{t=1}^{x_L} x_{\lambda_t}^R \leq x_R$$

et

$$\frac{\Delta}{cost(X)} \leq \frac{2x_R}{(x_L - 1)x_L + (x_R - 1)x_R} \leq \frac{2}{P}$$

Ce qui termine la preuve.

□

L'algorithme

Comme dit précédemment, l'algorithme est une programmation dynamique sur une instance ne comportant qu'une seule composante connexe (nous verrons plus tard comment résoudre le cas de plusieurs composantes connexes). Soit S^* une solution optimale au problème, $P \in \mathbb{N}$ un entier fixé, et $Comp$ le $(1 + \frac{4}{P})$-compactage précédemment défini. L'algorithme construit une solution au moins aussi bonne que $Comp(S^*)$ en énumérant, pour chaque bloc B_i, toutes les formes possibles de $Comp(X)$ pour $X \subseteq B_i$, et en particulier $Comp(S^* \cap B_i)$. Décrivons plus formellement les paramètres de la programmation dynamique :

- le premier paramètre est $k' \leq k$, le nombre d'intervalles à prendre.
- le second paramètre est i, l'index du bloc, signifiant que les k' sommets doivent être choisis parmi $\bigcup_{l=i}^{a} B_l$.
- enfin, B_{i-1}^S représente l'ensemble des $2P + 4$ variables encodant l'ensemble d'intervalles X_{i-1} choisis parmi le bloc B_{i-1}. Étant donné qu'il est possible de construire X_{i-1} à partir de B_{i-1}^S en temps polynomial, nous utiliserons directement B_{i-1}^S pour désigner X_{i-1} pour plus de clarté.

Étant donnés k', i et B^S_{i-1}, l'algorithme DP construit d'abord l'ensemble Ω de toutes les formes possibles de solutions pour le bloc i qui utilisent k' intervalles ou moins, en énumérant toutes les choix possibles pour $x_L, r_L, x_R, r_R,$, $(I^L_t)_{t=1,\cdots,P}$ et $(I^R_t)_{t=1,\cdots,P}$. Enfin, on retourne :

$$DP(k', i, B^S_{i-1}) = \arg min_{B\in\Omega} cost(B^S_{i-1} \cup B \cup DP(k' - |B|, i+1, B)$$

Le cas de base $i = a+1$ correspond au cas où il n'y a plus d'intervalles dans l'instance, et est donc trivial. Comme communément dans les programmations dynamiques, l'algorithme mémorise les précédentes étapes, évitant ainsi les calculs de la fonction pour des paramètres identiques. De la même manière, l'algorithme tel que décrit précédemment ne retourne que la valeur de la solution. Après des modifications nécessaires que nous ne détaillerons pas ici, il est facile de retourner la solution correspondante.

Lemme 31. *Pour tout $P \in \mathbb{N}$ fixé, $DP(k, 1, \emptyset)$ retourne une $(1 + \frac{4}{P})$-approximation pour* Sparsest k-Subgraph *dans les graphes d'intervalles propres en temps $\mathcal{O}(n^{\mathcal{O}(P)})$.*

Preuve. D'après le Lemme 30, il suffit pour cela de montrer que $cost(DP(k, 1, \emptyset)) \leq cost(Comp(S^*))$, où $Comp$ est le $(1 + \frac{4}{P})$-compactage précédemment défini.
Pour tout $i \in \{1, \cdots, a\}$, on définit $B^*_i = Comp(S^*) \cap B_i$, et $k^*_i = |\bigcup_{l=i}^{a} B^*_i|$. On montre par induction sur i (en partant de $i = a+1$) que $cost(B^*_{i-1} \cup DP(k^*_i, i, B^*_{i-1})) \leq cost(Comp(S^*) \cap \bigcup_{l=i-1}^{a} B_l)$.
Supposons que l'inégalité est vraie pour $i+1$, et montrons la pour i. En considérant l'itération où l'algorithme choisit $B = B^*_i$, on a :

$$cost(B^*_{i-1} \cup DP(k^*_i, i, B^*_{i-1})) \leq cost(B^*_{i-1}) + cost(B^*_{i-1}, B^*_i) + DP(k^*_i - |B^*_i|, i+1, B^*_i)$$

(on rappelle que $cost(X_1, X_2) = |\{(I_l, I_{l'}) \in E, I_l \in X_1, I_{l'} \in X_2\}|$). Et l'hypothèse d'induction permet de conclure.
La dépendance en P du temps d'exécution provient des n^{2P+4} valeurs possibles pour l'ensemble des paramètres, et du branchement en n^{2P+4} lorsque l'algorithme énumère les ensembles pour B^S_i. □

Il ne nous reste maintenant plus qu'à étendre le résultat précédent aux instances comportant au moins deux composantes connexes. Il suffit en fait de définir un nouvel algorithme de programmation dynamique qui va parcourir chaque composante connexe, en décidant à chaque fois combien d'intervalles prendre. Nous définissons brièvement ce principe, qui est expliqué plus en détails dans [98] pour Densest k-Subgraph.
Supposons que pour pour $k' \leq k$, il existe un algorithme $A(k', X)$ qui retourne une ρ-approximation pour Sparsest k-Subgraph sur un ensemble d'intervalles propres X connexe.
Soit $(C_i)_{i=1,\cdots,x}$ les composantes connexes de notre instance de Sparsest k-Subgraph. Il est alors suffisant de définir une programmation dynamique $DP'(k', i)$ calculant

une ρ-approximation de la solution avec k' intervalles sur $\bigcup_{t=i}^{x} C_t$, en gardant la meilleure solution parmi $A(l, C_i) + DP'(k' - l, i + 1)$ pour tout $l \in \{1, \cdots, k'\}$.
On a enfin le résultat :

Théorème 39. *Il existe une $\mathcal{PTAS}$ pour* Sparsest k-Subgraph *dans les graphes d'intervalles propres, s'exécutant en temps $n^{\mathcal{O}(\frac{1}{\epsilon})}$ pour tout $\epsilon > 0$ fixé.*

4.6.5 Algorithme $\mathcal{FPT}$ pour la paramétrisation standard dans les graphes d'intervalles

Introduction et intuition

Nous proposons maintenant un autre algorithme pour Sparsest k-Subgraph, dans les graphes d'intervalles (potentiellement non propres) cette fois-ci, retournant une solution optimale en temps $\mathcal{FPT}$ pour le paramètre standard : la valeur de la solution (*i.e.* le nombre d'arêtes). Étant donné que le nombre d'arêtes de la solution pouvant être arbitrairement inférieur à k, ce résultat est plus fort que celui présenté en Section 4.4 dans le cas des graphes chordaux si celui-ci était restreint aux graphes d'intervalles.
Pour cela, nous résolvons en fait la version de décision du problème. L'instance est donc toujours un ensemble d'intervalles $\mathcal{I}$ triés par extrémités droites croissantes, un entier $k \leq |\mathcal{I}|$ correspondant au nombre d'intervalles à sélectionner, ainsi qu'un entier $\mathcal{C} \leq \binom{k}{2}$. Le problème est ainsi de trouver un ensemble de k intervalles qui induisent au plus $\mathcal{C}$ intersections.
Comme c'est souvent le cas lors des algorithmes sur les intervalles, nous supposerons sans perdre de généralité que pour tout intervalle $I \in \mathcal{I}$, ses extrémités $r(I)$ et $l(I)$ sont des entiers, et que les extrémités de tous les intervalles de l'instance sont distincts.

Notations et préliminaires

Étant donné $x \in \mathbb{R}$, on définit $\mathcal{I}_{\geq x} = \{I \in \mathcal{I} : x \leq l(I)\}$ l'ensemble des intervalles qui « commencent » après x, $\mathcal{I}_{=x} = \{I \in \mathcal{I} : l(I) \leq x \leq r(I)\}$ l'ensemble des intervalles qui intersectent x, et $\mathcal{I}_{\leq x} = \{I \in \mathcal{I} : r(I) \leq x\}$ l'ensemble des intervalles qui « terminent » avant x.
Informellement, l'algorithme repose sur des règles de « domination » entre intervalles, permettant de dire, pour une situation donnée, qu'il est toujours plus intéressant de choisir des intervalles qui terminent le moins loin. Ainsi, pour reconstruire une solution, nous avons seulement besoin de connaître le nombre d'intervalles choisis à chaque étape. Ces règles de domination sont détaillées dans les Lemmes 32, 33 et 34.
On rappelle que dans ce qui suit, « le premier intervalle de S » signifie toujours le premier intervalle selon l'ordre considéré sur les intervalles (tri selon les extrémités droites croissantes).

Lemme 32. *Soit $S \subseteq \mathcal{I}$ une solution, et $s \in \mathbb{R}$ tel que $l(S) \leq s \leq r(S)$ et $S \cap \mathcal{I}_{=s} = \emptyset$. Soit $\tilde{I}$ le premier intervalle de $S \cap \mathcal{I}_{\geq s}$, et I^* le premier intervalle de $\mathcal{I}_{\geq s}$. Si $\tilde{I} \neq I^*$, alors on peut échanger ces deux intervalles afin d'obtenir une solution $S' = (S \setminus \{\tilde{I}\}) \cup \{I^*\}$ telle que $cost(S') \leq cost(S)$.*

Preuve. En effet, soit $I \in S$ tels que $I \neq \tilde{I}, I^*$. On montre que si I intersecte I^*, alors il intersecte également $\tilde{I}$. Par définition de $\tilde{I}$ et de S, on a $r(I^*) < r(\tilde{I}) < r(I)$, et si I intersecte I^*, on a $I \in \mathcal{I}_{=r(I^*)}$, et I doit forcément intersecter $\tilde{I}$ (*c.f.* Figure 30) □

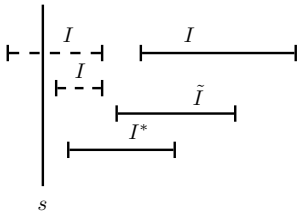


FIGURE 30 – Différentes positions de I dans le Lemme 32. Les intervalles en pointillés représentent les positions interdites.

Lemme 33. *Soit $S \subseteq \mathcal{I}$ une solution, $I_{i_1} \in S$ et $s \in \mathbb{R}$ tels que :*

(i) I_{i_1} est le premier intervalle de $S \cap \mathcal{I}_{=s}$,

(ii) $\exists \tilde{I} \in S \cap \mathcal{I}_{\geq s}$ tel que $\tilde{I}$ intersecte I_{i_1}.

Soit I^* le premier intervalle de $\mathcal{I}_{\geq s}$. Si $I^* \neq \tilde{I}$, alors on peut échanger ces deux intervalles afin d'obtenir une solution $S' = (S \setminus \{\tilde{I}\}) \cup \{I^*\}$ telle que $cost(S') \leq cost(S)$.

Preuve. Soit $I \in S$ tels que $I \neq \tilde{I}, I^*$. On montre que si I intersecte I^*, alors il intersecte également $\tilde{I}$. On distingue deux cas :

- Si $I \in \mathcal{I}_{=s}$, alors par définition de I_{i_1}, on a $r(I_{i_1}) < r(I)$, et puisque $s < l(\tilde{I}) < r(I_{i_1})$, I doit intersecter $\tilde{I}$.
- Si $I \in \mathcal{I}_{\geq s}$, alors comme dans la preuve du lemme précédent, par définition de $\tilde{I}$, on a $r(I^*) < r(\tilde{I}) < r(I)$, et si I intersecte I^*, on a $I \in \mathcal{I}_{=r(I^*)}$, et donc I intersecte également $\tilde{I}$.

□

Lemme 34. *Soit $S \subseteq \mathcal{I}$ une solution, et $s, s' \in R$ avec $s < s'$ tels que $\forall I \in S$ on a $r(I) \notin [s, s']$. Soit $\tilde{X} = S \cap \mathcal{I}_{\geq s} \cap \mathcal{I}_{=s'}$, et soient X^* les $|\tilde{X}|$ premiers intervalles de $\mathcal{I}_{\geq s} \cap \mathcal{I}_{=s'}$. Si $\tilde{X} \neq X^*$, alors il est possible d'échanger les intervalles de ces deux ensembles afin d'obtenir une solution $S' = (S \setminus \tilde{X}) \cup X^*$ de coût $cost(S') \leq cost(S)$.*

Preuve. Soit $\tilde{X} = \{\tilde{I}_1, \cdots, \tilde{I}_{|\tilde{X}|}\}$, et $X^* = \{I^*_1, \cdots, I^*_{|X^*|}\}$, où ces deux ensembles sont triés de manière croissante selon leur extrémité droite. Soit j_0 le plus grand index tel que $\tilde{I}_{j_0} \neq I^*_{j_0}$, et soit $I \in S \setminus (\tilde{(X)} \cup X^*)$ (on a ainsi $r(I^*_{j_0}) < r(\tilde{I}_{j_0})$). On montre que si I intersecte $I^*_{j_0}$, alors il intersecte également $\tilde{I}_{j_0}$. Pour cela on distingue deux cas :

- Si $s' < r(I)$, alors puisque $r(I^*_{j_0}) < r(\tilde{I}_{j_0})$, il est clair que I intersecte $\tilde{I}_{j_0}$ (*c.f.* Figure 31).
- Si $r(I) < s'$, alors par définition $r(I) < s$, et I ne peut pas intersecter $I^*_{j_0}$ ce qui contredit les hypothèses (et donc finalement prouve que ce cas est impossible).

□

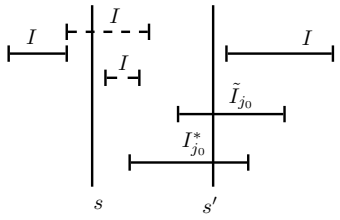


FIGURE 31 – Différentes positions de I dans le Lemma 33. Les intervalles en pointillés représentent les positions interdites.

L'algorithme

On rappelle que l'objectif de l'algorithme est de décider s'il existe un ensemble de k intervalles qui induisent C arêtes ou moins. Nous proposons la programmation dynamique $\mathcal{A}$ suivante (Algorithme 5), qui, étant donnés $s \in \mathbb{R}$, $k' \leq k$, et $C' \leq C$, retourne un ensemble S de k' sommets de $\mathcal{I}_{\geq s}$ qui induit au plus C' arêtes si un tel ensemble existe, et « *NON* » sinon. Comme on peut l'observer, l'algorithme utilise un ensemble $\Gamma_s(C)$. Cet ensemble est en fait un sous-ensemble de l'ensemble $\Omega_s(C)$ suivant.

Définition 36. *Pour $s \in \mathbb{R}$ et $C \in N$, nous définissons $\Omega_s(C) \subseteq \mathcal{P}(\mathcal{I})$ (où $\mathcal{P}(\mathcal{I})$ est l'ensemble des parties de $\mathcal{I}$) tel que pour tout $Q \in \Omega_s(C)$, on a :*

- *Q induit une composante connexe,*

- $cost(Q) \leq C$,
- $l(Q) = l(\mathcal{I}_{\geq s})$.

Autrement dit, $\Omega_s(C)$ est l'ensemble de toutes les composantes connexes de coût au plus C qui commencent après s. Nous montrons en fait que toute solution de $\Omega_s(C)$ peut être restructurée en une solution « bien formée » en utilisant les lemmes précédents. De plus, nous montrerons dans le Lemme 36 que l'ensemble de toutes les solutions bien formées (l'ensemble $\Gamma_s(C)$ de l'algorithme) est énumérable en temps $\mathcal{FPT}$. Ainsi, étant donnés s et C', l'algorithme branchera sur $\Gamma_s(C')$: toutes les solutions restructurées de $\Omega_s(C')$.

Algorithme 5 $\mathcal{A}(s,k',C')$

construire $\Gamma_s(C')$ (*c.f.* Définition 37)
si $\Gamma_s(C') = \emptyset$ **alors**
  retourner NON
sinon si $\exists Q \in \Gamma_s(C')$ avec $|Q| \geq k'$ **alors**
  retourner k' intervalles de Q
sinon
  retourner $\arg\max_{Q \in \Gamma_s(C')} cost(\{Q \cup \mathcal{A}(r(Q),k' - |Q|,C' - cost(Q))\})$
fin si

Tout d'abord, montrons comment restructurer une solution $Q \in \Omega_s(C)$. Comme indiqué précédemment, la restructuration consiste à utiliser les Lemmes 32, 33 et 34. Pour cela, nous définissons (récursivement sur s) la fonction $restruct(s, C, i)$ qui transforme $C \cap \mathcal{I}_{\geq s}$ (*i.e.* la partie de C qui est après s) en une solution restructurée (voir l'Algorithme 6 et la Définition 37). Le paramètre i (ainsi que les variables y_i correspondantes) seront utilisées dans le Lemme 35 afin de montrer qu'une solution restructurée peut être encodée de manière efficace.

Définition 37. *Étant donnés $s \in R$, $C \in \mathbb{N}$ et $Q \in \Omega_s(C)$, on définit :*

- $WSS(Q) = restruct(l(C) + \frac{1}{2}, C, 0)$ *la solution restructurée à partir de Q,*
- $\Gamma_s(C) = \{WSS(Q), Q \in \Omega_s(C)\}$ *l'ensemble des solutions restructurées associées aux composantes connexes de coût au plus C et commençant après s.*

Remarquons qu'à chaque étape de la programmation dynamique de l'Algorithme 5, on branche sur $\Gamma_s(C')$, qui est l'ensemble des solution restructurées Q telles que $l(Q) = l(\mathcal{I}_{\geq s})$. Par le Lemme 32, on peut supposer que pour toute solution optimale, la composante connexe commençant après s contient le premier intervalle de $\mathcal{I}_{\geq s}$. Ainsi, le terme $\frac{1}{2}$ dans la Définition 37 force la solution à prendre cet intervalle.

Le lemme suivant montre qu'il est suffisant pour l'Algorithme 5 de ne brancher que sur $\Gamma_w(C)$, évitant ainsi d'énumérer naïvement toutes les composantes connexes de $\Omega_s(C)$ qui impliquerait un temps exponentiel.

Algorithme 6 *restruct*(s,Q,i)

si $Q \cap \mathcal{I}_{\geq s} \neq \emptyset$ **alors**
  $I_{i_1} \leftarrow$ premier intervalle de $\mathcal{I}_{=s} \cap Q$
  si $\nexists I \in Q \cap \mathcal{I}_{\geq s}$ qui intersecte I_{i_1} **alors**
    $y_i \leftarrow 0$
    restruct$(r(I_{i_1}),Q,i+1)$
  sinon
    // on restructure un premier intervalle en utilisant le Lemme 33
    $\tilde{I} \leftarrow$ premier intervalle de $Q \cap \mathcal{I}_{\geq s}$ qui intersecte I_{i_1}
    $I^* \leftarrow$ premier intervalle de $\mathcal{I}_{\geq s}$ qui intersecte I_{i_1}
    $Q \leftarrow (Q \backslash \{\tilde{I}\}) \cup \{I^*\}$
    // on restructure un ensemble d'intervalles en utilisant le Lemme 34
    $s' \leftarrow \min(r(I^*),r(I_{i_1}))$
    $\tilde{X} \leftarrow Q \cap \mathcal{I}_{\geq s} \cap \mathcal{I}_{=s'}$
    $X^* \leftarrow$ les $|\tilde{X}|$ premiers intervalles de $\mathcal{I}_{\geq s} \cap \mathcal{I}_{=s'}$
    $Q \leftarrow (Q \backslash \tilde{X}) \cup X^*$
    $y_i \leftarrow |X| + 1$
    restruct$(s',Q,i+1)$
  fin si
fin si

Lemme 35. *Pour tout $s \in \mathbb{R}$ et tout $Q \in \Omega_s(C)$, on a :*

- $|WSS(Q)| = |Q|$,
- $|r(WSS(Q)) \leq r(Q)$,
- $cost(WSS(Q)) \leq cost(Q)$.

Preuve. Le premier point est clairement vrai étant donné que l'Algorithme 6 n'effectue que des échanges d'ensemble d'intervalles de même taille. Le second point est également vrai car chaque intervalle de Q n'est échangé qu'avec un intervalle dont l'extrémité droite est inférieure. Enfin, concernant le dernier point, dans les deux cas où l'algorithme modifie Q, les hypothèses des Lemmes 33 et 34 sont respectivement vérifiées. Ainsi, d'après ces lemmes le coût de la solution obtenue ne peut être qu'inférieur. □

Temps d'exécution

Le lemme suivant montre que l'énumération de $\Gamma_s(C)$ peut se faire en temps $\mathcal{FPT}$.

Lemme 36. *Pour tout $s \in \mathbb{R}$ et $C \in \mathbb{N}$, $|\Gamma_s(C)| \leq (\sqrt{2C}+2)^{C+1}$.*

Preuve. Soit $Q \in \Omega_s(C)$. L'argument principal repose sur le fait que $WSS(Q)$ est entièrement déterminé par les valeurs y_i définies par la procédure *restruct*. Ainsi, à la solution restructurée $WSS(Q)$ nous pouvons associer le vecteur $Y = (y_0, \cdots, y_{|Y|})$, et il suffit de borner valeur de chaque y_i ainsi que $|Y|$ afin de prouver le résultat demandé.
Tout d'abord on a déjà $y_i \leq \sqrt{(2C)} + 2$ En effet, dans les deux cas de la restructuration ($s' = r(I^*)$ ou $s' = r(I_{i_1})$), les $|X^*|$ intervalles intersectent tous s', qui correspond à l'extrémité droite d'un autre intervalle (I^* ou I_{i_1}). Ceci implique donc une clique de taille $y_i = |X^*| + 1$ dans la solution, dont le coût est borné par C.
Bornons maintenant la taille du vecteur Y. Pour cela, montrons qu'à chaque étape $i \in \{0, \cdots, |Y| - 1\}$, étant donné le paramètre s correspondant, on peut trouver $I \in \mathcal{I}_{=s}$ et $I' \in \mathcal{I}_{\geq s}$ qui s'intersectent, et tels que dans l'appel récursif suivant (avec le paramètre s'), soit I ou I' appartient à $\mathcal{I}_{\leq s'}$, évitant ainsi de compter plusieurs fois la même intersection, et impliquant par la suite que $|Y| - 1 \leq C$. Soit $i \in \{0, \cdots, |Y|-1\}$. Si $y_i \neq 0$, alors par définition de I^*, I_{i_1}, ces deux intervalles d'intersectent. Puis, étant donné que l'appel récursif a pour paramètre $s' = r(I_{i_1}) + 1$, et comme $i \neq |Y|$, on sait qu'il existe $I_{i_2} \in \mathcal{I}_{s'}$, qui implique que I_{i_2} intersecte I_{i_1}. Enfin, il est clair que $I_{i_1} \in \mathcal{I}_{\leq s'}$ ce qui conclu la preuve comme indiqué précemment. □

Théorème 40. SPARSEST k-SUBGRAPH *peut être résolu en temps* $\mathcal{O}(n^2k^3C(\sqrt{2C}+2)^{C+1})$ *dans les graphes d'intervalles.*

4.7 Conclusion, problèmes ouverts

Nous nous sommes intéressés dans ce chapitre à des problèmes d'optimisation à cardinalité fixée, et plus précisément à la recherche d'un sous-graphe induisant le moins ou le plus d'arêtes possibles (SPARSEST k-SUBGRAPH et DENSEST k-SUBGRAPH). Ces problèmes, en tant que généralisations de problèmes classiques tels que INDEPENDENT SET ou CLIQUE, ont été très largement étudiés ces dernières années, principalement par le biais d'algorithmes d'approximation, dans les graphes généraux ou des classes restreintes. En particulier, le problème DENSEST k-SUBGRAPH a soulevé beaucoup de questions. Tout d'abord pour son approximation dans le cas général, où le meilleur algorithme obtient un ratio un peu plus petit que $n^{1/4}$, alors que l'unique borne inférieure est $1 + \epsilon$. Mais aussi dans des classes particulières, telles que les graphes chordaux, où le problème reste $\mathcal{NP}$-difficile mais admet des algorithmes d'approximation à facteur constant, ou les graphes d'intervalles, où il est toujours ouvert de savoir si le problème est $\mathcal{NP}$-difficile ou polynomial.

Nous avons complété tous ces travaux en prouvant des résultats similaires pour la version SPARSEST k-SUBGRAPH : dans les graphes chordaux, d'abord, nous avons montré que le problème restait $\mathcal{NP}$-difficile et qu'il admettait un algorithme 2-approché, un algorithme $\mathcal{FPT}$, mais pas de noyau polynomial (sous certaines hypothèses). Concernant DENSEST k-SUBGRAPH, ces deux derniers résultats n'étaient

pas connus mais sont relativement faciles à obtenir. Dans le cas des graphes d'intervalles, nous laissons également ouverte la question de la complexité comme pour Densest k-Subgraph. Malgré cela, nous avons présenté un algorithme $\mathcal{FPT}$ (pour une paramétrisation plus fine que dans le cas des graphes chordaux), et une $\mathcal{PTAS}$ dans le cas des graphes d'intervalles propres. Nous avons également montré que l'algorithme qui atteint un rapport d'approximation de 3 pour Densest k-Subgraph dans les graphes chordaux a un ratio non borné même dans les graphes d'intervalles pour Sparsest k-Subgraph, mais atteint un ratio de 2 dans les graphes d'intervalles propres.

Nous proposons maintenant des problèmes ouverts ainsi que des pistes de recherches intéressantes concernant ceux-ci. Tout d'abord, probablement le problème le plus difficile concerne l'approximation de Densest k-Subgraph dans le cas général :

Problème ouvert 10. *Améliorer l'algorithme $n^{1/4}$-approché [12] pour* Densest k-Subgraph *dans le cas général, ou la borne inférieure de $(1+\epsilon)$ [83] .*

Un autre problème difficile et non moins motivant concerne la complexité de Densest k-Subgraph et Sparsest k-Subgraph dans les graphes d'intervalles et les graphes d'intervalles propres. Alors que le cas de Densest k-Subgraph est largement ouvert depuis de nombreuses années, le cas de Sparsest k-Subgraph est peut-être moins connu, mais semble d'après nous tout autant difficile.

Problème ouvert 11. Densest k-Subgraph *est-il polynomial dans les graphes d'intervalles (et/ou les graphes d'intervalles propres) ? Même question pour* Sparsest k-Subgraph*.*

Une étape intermédiaire pour le cas de Densest k-Subgraph dans les graphes d'intervalles serait peut-être la recherche d'un noyau polynomial. En effet, comme expliqué en début de Section 4.2, l'existence d'un noyau polynomial impliquerait que le problème n'est problablement pas $\mathcal{NP}$-difficile.

Problème ouvert 12. *Existe-t-il un noyau polynomial pour* Densest k-Subgraph *paramétré par k dans les graphes d'intervalles ?*

Puis, toujours concernant le statut ($\mathcal{NP}$-difficile *versus* Polynomial) de Densest k-Subgraph mais dans les graphes planaires (le résultat est immédiat pour Sparsest k-Subgraph, étant donné que Independent Set reste $\mathcal{NP}$-difficile). On rappelle que le résultat de [82] n'est valable que pour la version connexe du problème.

Problème ouvert 13. Densest k-Subgraph *est-il polynomial dans les graphes planaires ?*

On peut aussi se poser la question de l'adaptation de la technique de B. Baker [10] pour obtenir des algorithmes approchés dans le cas des graphes planaires.

Enfin, les autres problèmes sont les questions naturelles liées à l'amélioration ou l'adaptation des résultats présentés précédemment. Concernant les graphes chordaux d'abord.

Problème ouvert 14. *Peut-on améliorer le rapport d'approximation de deux de l'algorithme présenté pour* Sparsest k-Subgraph *dans les graphes chordaux ? Peut-on améliorer le rapport d'approximation de trois connu pour* Densest k-Subgraph *? Est-il possible de trouver une borne inférieure ?*

Problème ouvert 15. *Peut-on simplifier (en terme d'écriture ou de complexité) l'algorithme $\mathcal{FPT}$ présenté en Section 4.4 pour* Sparsest k-Subgraph *dans les graphes chordaux ?*

Problème ouvert 16. Maximum k-Coverage *est-il $\mathcal{FPT}$ paramétré par k dans les graphes chordaux ?*

Concernant ce dernier problème, l'algorithme pour Sparsest k-Subgraph ne donne pas immédiatement le résultat, étant donné que le paramètre devient $n - k$. Une autre question naturelle intéressante serait de généraliser ces résultats aux graphes parfaits :

Problème ouvert 17. Sparsest k-Subgraph *et* Densest k-Subgraph *sont-ils approximables à facteur constant dans les graphes parfaits ? Sont-ils $\mathcal{FPT}$?*

On termine enfin par des questions d'amélioration des algorithmes présentés dans les graphes d'intervalles et les graphes d'intervalles propres :

Problème ouvert 18. *Peut-on généraliser la $\mathcal{PTAS}$ de* Sparsest k-Subgraph *aux graphes d'intervalles (non propres) ?*

Problème ouvert 19. *Existe-t-il un algorithme $\mathcal{FPT}$ pour* Sparsest k-Subgraph *ou* Densest k-Subgraph *dans les graphes chordaux, paramétré par la valeur de la solution ?*

Notons qu'il semble difficile de montrer que Sparsest k-Subgraph ou Densest k-Subgraph munis de la paramétrisation ci-dessus est $\mathcal{W}[1]$-difficile dans les graphes chordaux. En effet, étant donné que le problème est $\mathcal{FPT}$ paramétré par k, il faudrait de ce fait une réduction qui permette d'obtenir une instance (G, k, C) où C ne dépendrait que du paramètre du problème de départ, mais pas k.

Conclusion

Les objectifs de cette thèse étaient de deux natures. D'une part, étudier et appliquer les techniques algorithmiques connues pour établir des résultats positifs et négatifs sur des problèmes précis, du point de vue de l'approximation et de la complexité paramétrée. D'autre part, de s'intéresser à la combinaison de techniques issues de ces deux paradigmes dans le but de développer de nouveaux outils algorithmiques pour traiter les problèmes difficiles.

Concernant le premier objectif, nous avons tout d'abord étudié la complexité algorithmique d'un problème de compaction de graphes, une famille de problèmes liés à ceux de partitions, homomorphismes ou modifications de graphes. Plus précisément, nous avons établi la difficulté du problème Sparsest k-Compaction du point de vue de la complexité classique, approchée et paramétrée, et nous avons proposé plusieurs algorithmes dans le cas général ainsi que dans des cas particuliers. La principale question ouverte reste de savoir si le problème admet un algorithme polynomial pour tout k fixé, *i.e.* si le problème est dans $\mathcal{XP}$ (Problème ouvert 6). Il paraît également intéressant, d'un point de vue théorique et pratique, de continuer l'étude de la complexité algorithmique d'autres problèmes de compactions, en définissant de nouvelles fonctions objectif. Nous avons par exemple établi la $\mathcal{NP}$-difficulté du problème Bounded Degree k-Compaction, et la même question de l'appartenance à $\mathcal{XP}$ peut se poser pour celui-ci, lorsque paramétré par $(k+C)$. Dans ce chapitre, nous avons également exhibé un lien entre Sparsest k-Compaction et Sparsest k-Subgraph, en notant qu'une solution pour ce dernier donnait une solution approchée pour le premier. Ceci nous a conduit vers une étude approfondie de Sparsest k-Subgraph, qui fit l'objet d'une seconde étude.

Le problème Sparsest k-Subgraph appartient à une classe de problèmes prenant en entrée un graphe G et un entier k, et dont l'objectif est de trouver un ensemble de k sommets optimisant une certaine fonction objectif. Des exemples classiques de telles fonctions objectif sont d'optimiser (maximiser ou minimiser) le nombre d'arêtes couvertes par ces sommets, ou le nombre d'arêtes induites. Ces problèmes sont des généralisations naturelles de problèmes classiques tels que Clique,

Independent Set ou Vertex Cover, et sont donc, de ce fait, $\mathcal{NP}$-difficiles. Durant cette thèse, nous avons plus précisément étudié la complexité de Sparsest k-Subgraph, une généralisation de Independent Set où l'objectif est de minimiser le nombre d'arêtes induites. Partant du constat de la difficulté de Independent Set dans les graphes généraux, nous avons étudié Sparsest k-Subgraph dans des classes de graphes dans lesquelles Independent Set est polynomial, telles que les graphes parfaits et ses sous-classes : graphes chordaux et graphes d'intervalles. Nous avons notamment établi la $\mathcal{NP}$-difficulté du problème dans les graphes chordaux et proposé, toujours dans cette classe un algorithme 2-approché et un algorithme $\mathcal{FPT}$ (en prouvant l'abscence de noyau polynomial, sous certaines hypothèses). Puis, nous avons étudié le problème dans les graphes d'intervalles et les graphes d'intervalles propres, en fournissant plusieurs algorithmes d'approximation et paramétrés. Pour résumer, les travaux réalisés dans cette thèse sur ce sujet ont permis d'obtenir un paysage similaire à celui du problème de maximisation correspondant : Densest k-Subgraph. En effet, dans les graphes chordaux ce dernier a été montré $\mathcal{NP}$-difficile [41] et plusieurs algorithmes approchés ont été développés [35, 89], alors qu'il est trivialement $\mathcal{FPT}$ et n'admet pas de noyau polynomial (sous certaines hypothèses, *c.f.* Théorème 35). Dans les graphes d'intervalles, la complexité des deux problèmes (polynomial ou $\mathcal{NP}$-difficile) est toujours inconnue (*c.f.* Problème ouvert 11), et reste l'un des problèmes ouverts majeurs sur ce sujet. Un autre challenge serait de trouver un éventuel lien entre Sparsest k-Subgraph et Densest k-Subgraph dans ces classes de graphes (non stables par passage au complémentaire), ou bien de trouver une classe où les deux problèmes se comportent différemment. Enfin, il serait également pertinent d'arriver à généraliser les algorithmes approchés et $\mathcal{FPT}$ au cas des graphes parfaits, dans lesquels ils restent $\mathcal{NP}$-difficiles alors que Clique et Independent Set y sont polynomiaux.

Enfin, concernant le second objectif, nous avons présenté un bref état de l'art des résultats récents combinant les principes de la théorie de l'approximation et de la complexité paramétrée. Puis, nous nous sommes concentrés sur les relations entre algorithmes de noyaux et approximation, en proposant d'étudier en détails la notion de noyau fidèle développée par M. Fellows *et al.* [58]. Alors que ceux-ci les utilisent pour obtenir des algorithmes paramétrés approchés avec un rapport d'approximation meilleur que dans le cas polynomial, nous proposons de les utiliser d'une manière indépendante aux algorithmes approchés paramétrés. En effet, de la même manière que les algorithmes de noyaux classiques par rapport aux algorithmes paramétrés, nous essayons d'établir des bornes supérieures et inférieures sur la taille des noyaux fidèles qu'il est possible d'obtenir. Par exemple, concernant le problème Clique qui n'admet pas de noyau polynomial lorsque paramétré par la taille d'un vertex cover du graphe d'entrée (sous certaines hypothèses), nous avons vu qu'il admettait un noyau fidèle linéaire avec une précision aussi petite que voulue pour toute une hiérarchie de paramètres généralisant le vertex cover. Nous avons également adapté aux noyaux fidèles les méthodes d'établissement de bornes inférieures, et avons montré

par exemple, toujours pour le problème Clique, que celui-ci ne pouvait admettre de noyau fidèle lorsque paramétré par la tree-width du graphe d'entrée, quelle que soit la précision espérée (sous certaines hypothèses), complétant ainsi d'une certaine manière les résultats positifs obtenus. Nous sommes convaincus que la recherche de noyaux fidèles pour d'autres problèmes pourra aider à une meilleure compréhension de la complexité intrinsèque des problèmes, tout en étoffant le cadre théorique de la complexité paramétrée permettant (entre autres) d'analyser la performance de règles de pré-traitement.

Bibliographie

[1] H. Abdeen, S. Ducasse, and H. A. Sahraoui. Modularization metrics : Assessing package organization in legacy large object-oriented software. In Proceedings of the 17th Working Conference on Reverse Engineering, pages 394–398, 2011.

[2] F. N. Abu-Khzam. A kernelization algorithm for d-hitting set. J. Comput. Syst. Sci., 76(7) :524–531, 2010.

[3] P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. Algorithmica, 33(2) :201–226, 2002.

[4] J.L. Ramírez Alfonsín and B. Reed. Perfect Graphs. Wiley and Sons, 2001.

[5] N. Apollonio and A. Sebő. Minconvex factors of prescribed size in graphs. SIAM Journal of Discrete Mathematics, 23(3) :1297–1310, 2009.

[6] N. Apollonio and B. Simeone. The maximum vertex coverage problem on bipartite graphs. Discrete Applied Mathematics, (in press), 2013.

[7] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In Proceedings of the 33th annual symposium on Foundations Of Computer Science, pages 13–22, 1992.

[8] S. Arora and S. Safra. Probabilistic checking of proofs : a new characterisation of np. In Proceedings of the 33th annual symposium on Foundations Of Computer Science, pages 2–13, 1992.

[9] J. Backer and J.M. Keil. Constant factor approximation algorithms for the densest k-subgraph problem on proper interval graphs and bipartite permutation graphs. Information Processing Letters, 110(16) :635–638, 2010.

[10] B. S. Baker. Approximation algorithms for np-complete problems on planar graphs. J. ACM, 41(1) :153–180, 1994.

[11] C. Bazgan. Schémas d'approximation et complexité paramétrée. Mémoire de DEA, 1995. Université Paris Sud.

[12] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities : an $\mathcal{O}(n^{1/4})$ approximation for densest k-subgraph. In Proceedings of the 42nd ACM symposium on Theory of Computing, pages 201–210. ACM, 2010.

[13] C-E Bichot and P. Siarry. Graph Partitioning. Wiley-ISTE, 2011.

[14] H. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. Journal of Computer and System Sciences, 75(8) :423–434, 2009.

[15] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM Journal of Computing, 25(6) :1305–1317, 1996.

[16] H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk. An $(c^k n)$ 5-approximation algorithm for treewidth. In Proceedings of the 54th annual Symposium on Foundations of Computer Science, pages 499–508, 2013.

[17] H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Cross-composition : A new technique for kernelization lower bounds. SIAM Journal of Discrete Mathematics, 28(1) :277–305, 2014.

[18] H.L. Bodlaender, J.R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. Journal of Algorithms, 18(2) :238 – 255, 1995.

[19] E. Bonnet, B. Escoffier, E. J. Kim, and V. Paschos. On subexponential and fpt-time inapproximability. In Proceedings of the 8th International Symposium on Parameterized and Exact Computation, pages 54–65, 2013.

[20] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. Journal of Computer and System Sciences, 13(3) :335–379, 1976.

[21] M. Bougeret, N. Bousquet, R. Giroudeau, and R. Watrigant. Parameterized complexity of the sparsest k-subgraph problem in chordal graphs. In Proceedings of the 40th International Conference on Current Trends in Theory and Practice of Computer Science, pages 150–161, 2014.

[22] N. Bourgeois, A. Giannakos, G. Lucarelli, I. Milis, and V. T. Paschos. Exact and approximation algorithms for densest k-subgraph. In WALCOM, pages 114–125, 2013.

[23] L. Brankovic and H. Fernau. A novel parameterised approximation algorithm for minimum vertex cover. Theoretical Computer Science, 511(0) :85 – 108, 2013. Exact and Parameterized Computation.

[24] H. Broersma, P. A. Golovach, and V. Patel. Tight complexity bounds for FPT subgraph problems parameterized by clique-width. In Proceedings of the 6th International Conference on Parameterized and Exact Computation, IPEC'11, pages 207–218, Berlin, Heidelberg, 2012. Springer-Verlag.

[25] M. Bruglieri, M. Ehrgott, H. W. Hamacher, and F. Maffioli. An annotated bibliography of combinatorial optimization problems with fixed cardinality constraints. Discrete Applied Mathematics, 154(9) :1344–1357, 2006.

[26] N. Bshouty and L. Burroughs. Massaging a linear programming solution to give a 2-approximation for a generalization of the vertex cover problem. In Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science, pages 298–308. Springer, 1998.

[27] H. J. Böckenhauer, J. Hromkovic, J. Kneis, and J. Kupke. On the parameterized approximability of tsp with deadlines. Journal of Theory of Computing Systems, 41(3) :431–444, 2007.

[28] L. Cai. Parameterized complexity of cardinality constrained optimization problems. Computer Journal, 51(1) :102–121, 2008.

[29] L. Cai, S. Chan, and S. Chan. Random separation : A new method for solving fixed-cardinality optimization problems. In H. L. Bodlaender and M. A. Langston, editors, Parameterized and Exact Computation, volume 4169 of LNCS, pages 239–250. Springer Berlin Heidelberg, 2006.

[30] L. Cai and J. Chen. On fixed-parameter tractability and approximability of NP optimization problems. Journal of Computer and System Sciences, 54(3) :465 – 474, 1997.

[31] L. Cai and X. Huang. Fixed-parameter approximation : conceptual framework and approximability results. In Proceedings of the International Workshop on Parameterized and Exact Computation, pages 96–108, 2006.

[32] L. Cai and D. W. Juedes. Subexponential parameterized algorithms collapse the w-hierarchy. In Proceedings of the 28th International Colloquium on Automata, Languages and Programming, pages 273–284, 2001.

[33] B. Caskurlu and K. Subramani. On partial vertex cover on bipartite graphs and trees. CoRR, abs/1304.5934, 2013.

[34] ML Cesati and L. Trevisan. On the efficiency of polynomial time approximation schemes.

[35] DannyZ. Chen, Rudolf Fleischer, and Jian Li. Densest k-subgraph approximation on intersection graphs. In K. Jansen and R. Solis-Oba, editors, Workshop

on Approximation and Online Algorithms, volume 6534 of LNCS, pages 83–93. Springer Berlin Heidelberg, 2011.

[36] Y. Chen, M. Grohe, and M. Gruber. On parameterized approximability. In Proceedings of the International Workshop on Parameterized and Exact Computation, pages 109–120, 2006.

[37] R. Chitnis, M. T. Hajiaghayi, and G. Kortsarz. Fixed-parameter and approximation algorithms : A new look. In Proceedings of the 8th International Symposium on Parameterized and Exact Computation, pages 110–122, 2013.

[38] M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas. The strong perfect graph theorem. ANNALS OF MATHEMATICS, 164 :51–229, 2006.

[39] V. Chvátal. On certain polytopes associated with graphs. Journal of Combinatorial Theory, Series B, 18(2) :138–154, 1975.

[40] S. A. Cook. The complexity of theorem-proving procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing, pages 151–158, 1971.

[41] D.G. Corneil and Y. Perl. Clustering and domination in perfect graphs. Discrete Applied Mathematics, 9(1) :27–39, 1984.

[42] B. Courcelle. The monadic second-order logic of graphs I. recognizable sets of finite graphs. Information and Computation, pages 12–75, 1990.

[43] H. Dell. And-compression of np-complete problems : Streamlined proof and minor observations. In Proceedings of the 9th International Symposium on Parameterized and Exact Computation, 2014. to appear.

[44] H. Dell and D. van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In Proceedings of the 42nd ACM Symposium on Theory of Computing, pages 251–260. ACM, 2010.

[45] E. D. Demaine and M. Hajiaghayi. Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 840–849, 2004.

[46] E. D. Demaine, M. Hajiaghayi, and K. Kawarabayashi. Algorithmic graph minor theory : Decomposition, approximation, and coloring. In Proceedings of the 46th annual symposium on Foundations Of Computer Science, pages 637–646, 2005.

[47] R. Diestel. Graph theory. Springer-Verlag, 2010.

[48] M. Dom, D. Lokshtanov, and S. Saurabh. Incompressibility through colors and ids. In Proceedings of the 36th International Colloquium on Automata, Languages and Programming, pages 378–389, 2009.

[49] R. Downey, M. Fellows, , and C. McCartin. Parameterized approximation problems. In Proceedings of the International Workshop on Parameterized and Exact Computation, pages 121–129, 2006.

[50] R. G. Downey, V. Estivill-Castro, M. R. Fellows, E. Prieto, and F. A. Rosamund. Cutting up is hard to do : The parameterised complexity of k-cut and related problems. Electronic Notes in Theoretical Computer Science, 78(0) :209 – 222, 2003.

[51] R. G. Downey and M. R. Fellows. Fundamentals of Parameterized Complexity. Texts in Computer Science. Springer, 2013.

[52] A. Drucker. New limits to classical and quantum instance compression. 54th Annual Symposium on Foundations of Computer Science, 0 :609–618, 2012.

[53] L. Euler. Solutio problematis ad geometriam situs pertinentis. Commentarii academiae scientiarum Petropolitanae, 8 :128–140, 1741.

[54] T. Feder, P. Hell, S. Klein, and R. Motwani. Complexity of graph partition problems. In Proceedings of the 31st annual Symposium on Theory Of Computing, pages 464–472, 1999.

[55] U. Feige, S. Goldwasser, L. Lovàsz, S. Safra, and M. Szegedy. Approximating clique is almost np-hard. In Proceedings of the 32th annual symposium on Foundations Of Computer Science, pages 2–12, 1991.

[56] U. Feige, G. Kortsarz, and D. Peleg. The dense k-subgraph problem. Algorithmica, 29 :2001, 1999.

[57] M. R. Fellows, D. Hermelin, F. A. Rosamond, and H. Shachnai. Tractable parameterizations for the minimum linear arrangement problem. In Proceedings of the 21st European Symposium on Algorithms, pages 457–468, 2013.

[58] M. R. Fellows, A. Kulik, F. Rosamond, and H. Shachnai. Parameterized approximation via fidelity preserving transformations. In Proceedings of the 39th International Colloquium on Automata, Languages and Programming, pages 351–362, 2012.

[59] J. Flum and M. Grohe. Parameterized Complexity Theory. Springer, 2006.

[60] L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct pcps for np. Journal of Computer and System Sciences, 77(1) :91–106, 2011.

[61] M. R. Garey and D. S. Johnson. Computers and Intractability : A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979.

[62] F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. SIAM Journal on Computing, 1(2) :180 – 187, 1972.

[63] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. Journal of Combinatorial Theory, Series B, 16(1) :47–56, 1974.

[64] O. Goldreich. Computational Complexity : A Conceptual Perspective. Cambridge University Press, 2008.

[65] O. Goldschmidt and D. S. Hochbaum. Polynomial algorithm for the k-cut problem. In Proceedings of the 29th annual symposium on Foundations Of Computer Science, pages 444–451, 1988.

[66] O. Goldschmidt and D. S. Hochbaum. A polynomial algorithm for the k-cut problem for fixed k. Math. Oper. Res., 19(1) :24–37, 1994.

[67] M. C. Golumbic. Algorithmic graph theory and perfect graphs. Academic Press, New York, USA, 1980.

[68] T. Gonzalez. On the computational complexity of clustering and related problems. In System Modeling and Optimization, volume 38, pages 174–182. 1982.

[69] M. Grohe. Local tree-width, excluded minors, and approximation algorithms. Combinatorica, 23(4) :613–632, 2003.

[70] J. Guo, R. Niedermeier, and S. Wernicke. Parameterized complexity of generalized vertex cover problems. In Algorithms and Data Structures, volume 3608 of LNCS, pages 36–48. Springer Berlin Heidelberg, 2005.

[71] J. Guo, R. Niedermeier, and S. Wernicke. Parameterized complexity of vertex cover variants. Theory of Computing Systems, 41(3) :501–520, 2007.

[72] M. T. Hajiaghayi, R. Khandekar, and G. Kortsarz. Fixed parameter inapproximability for clique and set cover in time super-exponential in opt. CoRR, abs/1310.2711, 2013.

[73] P. Hansen and M. Delattre. Complete-link cluster analysis by graph coloring. Journal of the American Statistical Association, 73(362) :pp. 397–403, 1978.

[74] J. Hrastad. Clique is hard to approximate within $n^{1-\epsilon}$. In Proceedings of the 37th annual symposium on Foundations Of Computer Science, pages 627–636, 1996.

[75] P. Hell. Graphs and Homomorphisms. Oxford University Press, 2004.

[76] D. S. Hochbaum and A. Pathria. Analysis of the greedy approach in covering probems. Naval Research Quarterly, 45 :615–627, 1998.

[77] Dorit S. Hochbaum. Approximation Algorithms for NP-Hard Problems. Course Technology Inc, 1996.

[78] E. Isnard. Communication privée, 2014.

[79] B. M. P. Jansen. The Power of Data Reduction : Kernels for Fundamental Graph Problems. PhD thesis, Utrecht University, 2013.

[80] G. Joret and A. Vetta. Reducing the rank of a matroid. CoRR, abs/1211.4853, 2013.

[81] R. M. Karp. Reducibility among combinatorial problems. In Complexity of Computer Computations, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

[82] J. M. Keil and T.B. Brecht. The complexity of clustering in planar graphs. Journal of Combinatorial, 9 :155–159, 1991.

[83] S. Khot. Ruling out ptas for graph min-bisection, dense k-subgraph, and bipartite clique. SIAM Journal of Computing, 36 :1025–1071, 2004.

[84] S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2-epsilon. In Proceedings of the 18th Annual IEEE Conference on Computational Complexity, pages 379–386. IEEE, 2003.

[85] J. Kneis, A. Langer, and P. Rossmanith. Improved upper bounds for partial vertex cover. In Proceedings of the 34th Workshop of Graph Theoretic Concepts in Computer Science, pages 240–251. Springer, 2008.

[86] M. Koivisto. An $O(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion-exclusion. In Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, pages 583–590, 2006.

[87] L. A. Levin. Universal search problems. Problems of Information Transmission, 9(3), 1973.

[88] M. Liazi, I. Milis, F. Pascual, and V. Zissimopoulos. The densest k-subgraph problem on clique graphs. Journal of Combinatorial Optimization, 14(4) :465–474, 2007.

[89] M. Liazi, I. Milis, and V. Zissimopoulos. A constant approximation algorithm for the densest k-subgraph problem on chordal graphs. Information Processing Letters, 108(1) :29–32, 2008.

[90] D. Marx. Minimum sum multicoloring on the edges of planar graphs and partial k-trees. In Proceedings of the 2nd International Workshop on Approximation and Online Algorithms, pages 9–22, 2004.

[91] D. Marx. Parameterized complexity and approximation algorithms. Comput. J., 51(1) :60–78, 2008.

[92] L. Mathieson. A proof checking view of parameterized complexity. CoRR, abs/1206.2436, 2012.

[93] G. B. Mertzios. A polynomial algorithm for the k-cluster problem on the interval graphs. Electronic Notes in Discrete Mathematics, 26(0) :111–118, 2006. Combinatorics 2006.

[94] G. B. Mertzios. Communication privée, 2014.

[95] N. Misra, V. Raman, and S. Saurabh. Lower bounds on kernelization. Discrete Optimization, 8(1) :110–128, 2011.

[96] A. Natanzon, R. Shamir, and R. Sharan. Complexity classification of some edge modification problems. Discrete Applied Mathematics, 113(1) :109 – 128, 2001.

[97] R. Niedermeier. Invitation to Fixed-Parameter Algorithms. OIJP Oxford, 2006.

[98] T. Nonner. PTAS for densest k-subgraph in interval graphs. In Proceedings of the 12th International Conference on Algorithms and Data Structures, pages 631–641. Springer, 2011.

[99] V. T. Paschos. Complexité et Approximation Polynomiale. Hermes Science Publications, 2004.

[100] S. B. Patkar and H. Narayanan. An efficient practical heuristic for good ratio-cut partitioning. In Proceedings of the 16th International Conference on VLSI Design, pages 64–69, 2003.

[101] E. Petrank. The hardness of approximation : Gap location. Computational Complexity, 4 :133–157, 1994.

[102] B. Reed. Finding approximate separators and computing tree width quickly. In Proceedings of the 24th annual symposium on Theory of computing, pages 221–228, 1992.

[103] N. Robertson and P. D. Seymour. Graph minors. xiii. Journal of Combinatorial Theory, Series B, 63(1) :65–110, 1995.

[104] D. J. Rose. Triangulated graphs and the elimination process. Journal of Mathematical Analysis and Applications, 32(3) :597 – 609, 1970.

[105] H. Saran and V. V. Vazirani. Finding k cuts within twice the optimal. SIAM Journal of Computing, 24(1) :101–108, 1995.

[106] Y. Shibata. On the tree representation of chordal graphs. Journal of Graph Theory, 12 :421–428, 1988.

[107] P. Turán. An extremal problem in graph theory. Matematikai és Fizikai Lapok, 48 :436–452, 1941.

[108] V. Vazirani. Approximation Algorithms. Springer, 2002.

[109] N. Vikas. Computational complexity of compaction to reflexive cycles. SIAM Journal of Computing, 32(1) :253–280, 2002.

[110] N. Vikas. Computational complexity classification of partition under compaction and retraction. In Proceedings of the 10th international Computing and Combinatorics Conference, pages 380–391, 2004.

[111] N. Vikas. A complete and equal computational complexity classification of compaction and retraction to all graphs with at most four vertices and some general results. Journal of Computer and System Sciences, 71(4) :406–439, 2005.

[112] N. Vikas. Algorithms for partition of some class of graphs under compaction. In Proceedings of the 17th international Computing and Combinatorics Conference, pages 319–330, 2011.

[113] R. Watrigant, M. Bougeret, and R. Giroudeau. Approximating the sparsest k-subgraph in chordal graphs. In Proceedings of the 11th Workshop on Approximation and Online Algorithms, pages 73–84, 2013.

[114] R. Watrigant, M. Bougeret, and R. Giroudeau. Approximating the sparsest k-subgraph in chordal graphs. Theory of Computing Systems (in press), 2014.

[115] R. Watrigant, M. Bougeret, R. Giroudeau, and J.-C. König. Sum-max graph partitioning problem. In Proceedings of the 2nd International Symposium on Combinatorial Optimization, pages 297–308, 2012.

[116] R. Watrigant, M. Bougeret, R. Giroudeau, and J.-C. König. On the sum-max graph partitioning problem. Theoretical Computer Science, 540-541 :143–155, 2014.

[117] T. A. Wiggerts. Using clustering algorithms in legacy systems remodularization. In Proceedings of the 4th Working Conference on Reverse Engineering, pages 33–44, 1997.

[118] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. Theory of Computing, 3(1) :103–128, 2007.

Table des figures

Résumé.

La théorie de la $\mathcal{NP}$-complétude nous apprend que pour un certain nombre de problèmes d'optimisation, il est vain d'espérer un algorithme efficace calculant une solution optimale. Partant de ce constat, un moyen pour contourner cet obstacle est de réaliser un compromis sur chacun de ces critères, engendrant deux approches devenues classiques. La première, appelée approximation polynomiale, consiste à développer des algorithmes efficaces et retournant une solution proche d'une solution optimale. La seconde, appelée complexité paramétrée, consiste à développer des algorithmes retournant une solution optimale mais dont l'explosion combinatoire est capturée par un paramètre de l'entrée bien choisi. Cette thèse comporte deux objectifs. Dans un premier temps, nous proposons d'étudier et d'appliquer les méthodes classiques de ces deux domaines afin d'obtenir des résultats positifs et négatifs pour deux problèmes d'optimisation dans les graphes : un problème de partition appelé Sparsest k-Compaction, et un problème de recherche d'un sous-graphe avec une cardinalité fixée appelé Sparsest k-Subgraph. Dans un second temps, nous présentons comment les méthodes de ces deux domaines ont pu se combiner ces dernières années pour donner naissance au principe d'approximation paramétrée. En particulier, nous étudierons les liens entre approximation et algorithmes de noyaux.

Mots clés : optimisation combinatoire, approximation, complexité paramétrée, problèmes de graphes.

Université Montpellier 2.
Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier.
LIRMM UMR 5506. 161 rue Ada, 34095 Montpellier Cedex 5 - France.

Abstract.

The theory of $\mathcal{NP}$-completeness tells us that for many optimization problems, there is no hope for finding an efficient algorithm computing an optimal solution. Based on this, two classical approaches have been developped to deal with these problems. The first one, called polynomial-time approximation, consists in designing efficient algorithms computing a solution that is close to an optimal one. The second one, called parameterized complexity, consists in designing exact algorithms which combinatorial explosion is captured by a carefully chosen parameter of the instance. The goal of this thesis is twofold. First, we study and apply classical methods from these two domains in order to obtain positive and negative results for two optimization problems in graphs: a partitioning problem called Sparsest k-Compaction, and a cardinality constraint subgraph problem called Sparsest k-Subgraph. Then, we present how the different methods from these two domains have been combined in recent years in a concept called parameterized approximation. In particular, we study the links between approximation and kernelization algorithms.

Keywords : combinatorial optimization, approximation, parameterized complexity, graph problems.

Université Montpellier 2.
Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier.
LIRMM UMR 5506. 161 rue Ada, 34095 Montpellier Cedex 5 - France.