Université d'Orléans

Habilitation à diriger des recherches
Spécialité : Informatique

présentée et soutenue publiquement le 1er décembre 2006 par

Ioan TODINCA

Décompositions arborescentes de graphes : calcul, approximations, heuristiques

Rapporteurs :

M. Hans BODLAENDER	Utrecht University
M. Victor CHEPOI	Université Aix-Marseille II
M. Michel HABIB	Université Paris 7

Membres du jury :

M. Hans BODLAENDER	Utrecht University
M. Cyril GAVOILLE	Université Bordeaux 1
M. Michel HABIB	Université Paris 7
M. Jean-Xavier RAMPON	Université de Nantes
M. Stéphan THOMASSÉ	Université Montpellier II
M. Henri THUILLIER	Université d'Orléans

Table des matières

Chapitre 1

Introduction

Les travaux présentés dans ce mémoire s'inscrivent dans le cadre de l'étude structurelle et algorithmique des graphes. De nombreux problèmes classiques en algorithmique des graphes, comme la coloration, le voyageur de commerce ou le calcul d'autres paramètres sont connus pour être difficiles, au sens où il n'existe pas d'algorithme polynomial pour résoudre ces problèmes sauf si P=NP, cf. [64]. Ce phénomène est encore plus flagrant lorsque l'on regarde les questions pratiques, comme les nombreuses variantes de la coloration, issues des télécommunications.

Les techniques pour répondre aux questions NP-difficiles sont au cœur de la recherche en algorithmique, ainsi nous disposons de plusieurs approches pour aborder les problèmes d'optimisation : heuristiques, algorithmes d'approximations, résolution des problèmes pour des cas particuliers, voire même algorithmes de complexité exponentielle. Les heuristiques sont des algorithmes souvent rapides, mais sur lesquels on ne peut donner aucune garantie théorique. Elles sont jugées de manière assez empirique, par la qualité des résultats obtenus sur des jeux de tests. Les algorithmes d'approximation sont des algorithmes polynomiaux pour lesquels on peut garantir que la solution obtenue est proche de l'optimum – souvent à une constante multiplicative près. Très fortement étudiés depuis les années '80, ils apportent une réponse satisfaisante à bon nombre de questions. Néanmoins, il a été prouvé que beaucoup de problèmes n'admettent pas d'approximation à facteur constant près, sauf si P=NP. Par exemple, pour la coloration des graphes ou la recherche d'un stable (ensemble de sommets deux à deux non reliés) de cardinal maximum aucune approximation "raisonnable" ne peut être envisagée. La troisième approche consiste à résoudre des problèmes difficiles pour des entrées particulières, dans notre cas pour des classes de graphes particulières. La bonne question consiste alors à se demander quelle est l'information *suffisante* que l'on doit posséder pour résoudre un problème ou une catégorie de problèmes. En restreignant cette information, nous résoudrons nos problèmes sur des entrées de plus en plus larges. Notons enfin que ces techniques pour aborder des problèmes difficiles ne sont en aucun cas disjointes : les observations menant à des solutions optimales dans des cas particuliers servent d'ingrédient de base pour des heuristiques, les heuristiques se transforment parfois en algorithmes d'approximation etc.

L'une des méthodes pour aborder des problèmes difficiles sur des entrées particulières consiste à décomposer les graphes. L'idée est de découper le graphe en plusieurs morceaux, qui s'agencent selon certaines règles. Si l'on peut résoudre notre problème sur chacun des morceaux, il suffira de recombiner les solutions partielles pour obtenir une solution globale, sur tout le graphe. Dans la pratique, les morceaux sont décomposés à leur tour, et de manière récursive on obtient une décomposition selon une structure d'arbre. L'un des grands avantages est que de *larges classes* de problèmes difficiles peuvent être résolus efficacement pour les graphes admettant une "bonne" décomposition.

L'idée de décomposer les graphes n'est pas nouvelle. Les travaux de Gallai [63] sont à l'origine de la *décomposition modulaire*. On décompose le graphe en modules, un module étant un ensemble de sommets qui ont le même comportement vis à vis du reste du graphe. La décomposition modulaire d'un graphe est unique, et de plus elle est calculable en temps linéaire en utilisant par exemple par l'algorithme de [45]. Pour résoudre nos problèmes difficiles sur le graphe en entrée, il suffira de résoudre ces problèmes (ou des version généralisées) sur les sous-graphes premiers, c'est-à-dire indécomposables pour la décomposition modulaire.

Les *décompositions arborescentes* (en anglais : tree decompositions) et la notion associée de *largeur arborescente* (treewidth) ont été introduites au début des années 80 par Robertson et Seymour [117, 118], dans leur fameuse étude sur les mineurs des graphes. Elles sont fortement liées aux concept de k-arbre partiel, dont l'origine remonte aux travaux de Dirac et de Rose [120], dans les années 70. Très sommairement, l'idée est que tout graphe peut être vu comme un arbre généralisé. La largeur arborescente d'un graphe est une sorte de d'écart, mesurant la distance entre le graphe et la classe des arbres : plus la largeur arborescente d'un graphe est petite, plus sa structure est proche d'un arbre. Bien qu'introduits afin de résoudre un problème combinatoire, ces outils ont été très vite utilisés de point de vue algorithmique. Beaucoup de problèmes classiques de l'algorithmique des graphes, qui sont NP-difficiles en général, peuvent être résolus en temps polynomial, voire linéaire, pour les graphes ayant une largeur arborescente bornée par une constante. On y retrouve par exemple la coloration, le problème du stable maximum ou du cycle hamiltonien.

Les décompositions modulaires et les décompositions arborescentes sont très complémentaires, car ce ne sont pas les mêmes graphes qui se décomposent bien selon les deux méthodes. Les *cliques-décompositions* [41, 44], ou, de manière presque équivalente, les décompositions NLC [134] généralisent à la fois les décompositions modulaires et les décompositions arborescentes : tous les graphes qui se décomposent bien par l'une des deux dernières méthodes ont également une "bonne" clique-décomposition, i.e. une largeur de clique (cliquewidth) petite. Là aussi, il a été montré que pour les graphes qui one une largeur de clique bornée, de larges classes de problèmes peuvent êtres résolus en temps polynomial en utilisant des outils logiques [42] ou de la programmation dynamique [134, 94, 54, 125]. Cependant, nous ne disposons pas encore d'algorithmes calculant une bonne clique-décomposition (cf. [108] pour des résultats d'approximation de la largeur de clique). Même une fois la décomposition calculée, les algorithmes actuels sont beaucoup plus coûteux en temps que les algorithmes similaires pour la décomposition arborescente.

Pour donner un dernier exemple de l'intérêt des techniques de décomposition, rappelons que la preuve de la conjecture de Berge, selon laquelle un graphe est parfait si et seulement si il ne contient pas de trou ou d'anti-trou impair, est basée sur une approche par décomposition [37]. Ce nouveau type de décomposition, appelé *décomposition antisymétrique* (skew partition) est également le fondement de l'algorithme de reconnaissance des graphes parfaits donné par Chudnovsky et al. [36].

Dans ce document nous nous intéressons aux décompositions arborescentes, ainsi qu'à deux autres types de décompositions, introduits par Robertson et Seymour dans la même série de travaux : les *décompositions en branches* [119] et les *décompositions linéaires* [116].

1.1 Décompositions arborescentes et largeur arborescente

On ne peut parler des décompositions arborescentes et de la largeur arborescente sans évoquer les travaux de Robertson et Seymour sur les mineurs de graphes, travaux qui les ont conduit à définir ces notions. Cette théorie est fort bien illustrée par le plongement des graphes non orientés sur certains types de surfaces. Prenons comme exemple le cas des graphes planaires, c'est-à-dire les graphes que l'on peut le dessiner dans le plan sans que deux arêtes se croisent. Considérons trois opérations sur les graphes : la première consiste à enlever une arête du graphe, la deuxième, à lui enlever un sommet et les arêtes adjacentes, et enfin la troisième est de contracter une arête en un seul sommet. Si l'on applique ces opérations à un graphe planaire, il est assez facile de se convaincre que nous obtenons toujours un graphe planaire. Plus formellement, un graphe H est appelé *mineur* de G si H est obtenu à partir de G par une suite d'opérations de ce type. Comme nous l'avons remarqué, la minoration préserve la planarité. Nous dirons que les graphes planaires constituent une classe de graphes fermée pour la minoration.

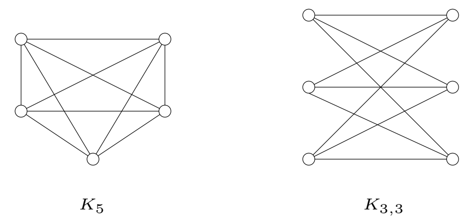

FIG. 1.1 – Ensemble d'obstructions des graphes planaires

Les plus petits graphes non planaires sont le graphe complet à cinq sommets, noté K_5, et le graphe biparti complet $K_{3,3}$. Un théorème bien connu dû à Kuratowski nous assure que tout graphe G non planaire a K_5 ou $K_{3,3}$ comme mineur. D'un autre côté, un graphe planaire ne peut pas avoir comme mineur un de ces deux graphes. En d'autres termes, les graphes planaires peuvent être identifiés comme la classe des graphes n'ayant pas de mineur dans l'ensemble $\{K_5, K_{3,3}\}$. Les deux mineurs interdits forment ce que l'on nomme *ensemble d'obstructions* pour les graphes planaires.

Etant donnée une classe de graphes $\mathcal{G}$ fermée par minoration, existe-t-il toujours un ensemble fini de graphes $ob(\mathcal{G})$ tel qu'un graphe G appartient à $\mathcal{G}$ si et seulement si aucun graphe de $ob(\mathcal{G})$ n'est mineur de G ? La réponse est affirmative, et c'est l'un des résultats principaux de la longue série d'articles que Robertson et Seymour ont consacrée à l'étude des mineurs de graphes. Le résultat est la conséquence d'un théorème plus général qui était connu sous le nom de la conjecture de Wagner : dans toute suite infinie de graphes, il existe deux graphes tels que l'un d'entre eux soit mineur de l'autre. Si l'on se restreint à la classe des arbres, ce théorème avait été prouvé par Kruskal [97]. C'est dans ce cadre-là qu'est apparue la notion de *largeur arborescente*, dans le deuxième article de la série. Elle a permis d'étendre le théorème de Kruskal à toute classe de graphes de largeur arborescente bornée, puis à tous les graphes.

La largeur arborescente est une mesure de connexité : si un graphe a une largeur arborescente petite cela indique que le graphe est "peu connexe", qu'il a une structure proche d'un arbre, ou plutôt d'une notion d'arbre généralisé que nous détaillerons plus tard. La largeur arborescente a été introduite afin de caractériser certaines classes de graphes fermées par minoration, ainsi que pour les reconnaître de façon algorithmique. Un des premiers résultats importants est que si l'on se donne un graphe planaire H, tout graphe G n'ayant pas H comme mineur a une largeur arborescente bornée par une constante ne dépendant que de H [117]. Les graphes de largeur arborescente au plus k sont eux-mêmes une classe fermée par minoration.

Les algorithmiciens se sont vite aperçu de l'intérêt algorithmique de ces outils. On peut dire sans hésiter que la plupart des problèmes NP-difficiles classiques peuvent être résolus en temps polynomial pour les graphes de largeur arborescente bornée. Il existe des résultats qui identifient des classes de problèmes résolubles en temps linéaire pour des graphes de largeur arborescente bornée. Les travaux de Courcelle [40], Arnborg et al. [5], Courcelle et Mosbah [43] ont abouti à la conclusion que tous les problèmes exprimables par des formules monadiques étendues de second ordre peuvent être résolus en temps linéaire. Schématiquement, il s'agit des problèmes de graphes que l'on peut formuler à l'aide des opérateurs logiques ($\wedge$, $\vee$, $\neg$, $\Rightarrow$), des quantificateurs sur des sommets, arêtes, ensembles de sommets et ensemble d'arêtes (comme $\exists W \subseteq V$, $\forall e \in E$), de tests d'appartenance et d'adjacence ($v \in W, xy \in E$) et certaines extensions. Les extensions permettent de prendre en considération des problèmes d'optimisation, comme le problème du stable maximum.

De point de vue pratique, la résolution de ces problèmes est plutôt basée sur de la programmation dynamique (voir [15, 72] pour des exemples). Elle permet de faire des algorithmes "sur mesure" même pour des problèmes qui ne rentrent pas dans le cadre logique décrit ci-dessus. De plus, les algorithmes obtenus sont plus rapides en pratique. Les complexités sont souvent de type $\mathcal{O}(2^k n)$, où n est le nombre de sommets du graphe et k est la largeur de la décomposition dont nous disposons. Ceci donne une idée sur la borne supérieure que l'on peut accepter pour k de sorte à rester dans des temps de calcul raisonnables.

Les décompositions arborescentes ont été dès le début un puissant outil théorique pour montrer que certains problèmes sont polynomiaux sur des instances particulières, mais également dans le contexte de la tractabilité à paramètre fixe (FPT - Fixed Parameter Tractability) [46] ou pour concevoir des schémas d'approximation [53]. Depuis quelques années nous assistons à une réelle entrée de ces outils dans les applications pratiques. On peut citer leur utilisation dans les réseaux probabilistes, qui ont souvent une petite largeur arborescente [99, 26], dans la résolution d'instances difficiles pour divers problèmes d'optimisation [38, 95, 1], ainsi que dans la programmation par contraintes [83, 82, 67].

Le point faible des décompositions arborescentes vient du fait que le calcul de la largeur arborescente est NP-difficile [4], même pour des classes restreintes comme les graphes de degré borné [28], les graphes bipartis ou les compléments des graphes bipartis [70]. Il existe un algorithme qui, prenant en entrée un graphe G à n sommets et une constante k, décide en temps $\mathcal{O}(n)$ si la largeur arborescente de ce graphe est au plus k. Cet algorithme est dû à Bodlaender [16]. Malheureusement, la constante du $\mathcal{O}$ cache une fonction surexponentielle en k ; ce procédé ne peut pas être employé dans la pratique, même pour des constantes k très petites.

Pour bon nombre de classes de graphes particulières, il existe des algorithmes polynomiaux qui calculent la largeur arborescente des graphes respectifs : graphes de permutation [25], graphes d'intervalles circulaires [128, 93], graphes bipartis cordaux [89], graphes distance-héréditaires [34], graphes HHD-free [35], graphes de cordes [85, 93], graphes faiblement triangulés [31]. Pour toutes ces classes, les algorithmes sont basés sur les notions de *triangulations* et de *séparateurs minimaux*.

Un graphe H est dit *triangulé* si tout cycle de H ayant quatre sommets ou plus possède une *corde*, c'est-à-dire une arête entre deux sommets non consécutifs du cycle. Etant donné un graphe quelconque G, une triangulation H de G est un graphe triangulé ayant le même ensemble de sommets et tel que G soit un sous-graphe de H. La largeur arborescente de G est le minimum, parmi toutes les triangulations H de G, de $\omega(H) - 1$, où $\omega(H)$ dénote la taille de la clique de cardinal maximum de H (cf. chapitre suivant pour les définitions). Ainsi, calculer la largeur arborescente de G revient à trouver une triangulation de G, tout en minimisant la clique maximum de cette triangulation. La triangulation optimale H se trouve clairement parmi les *triangulations minimales* de G ; une triangulation H est minimale si aucun sous-graphe strict de H n'est une triangulation de G.

Les algorithmes mentionnés ci-dessus sont basés sur les liens forts établis entre les triangulations minimales et les séparateurs minimaux [110]. Nous renforçons ces liens à travers la notion de clique maximale potentielle, et nous montrons entre autres que le calcul de la largeur arborescente se fait en temps polynomial par rapport au nombre de séparateurs minimaux du graphe en entrée. Nous étudions également le calcul exact de la largeur arborescente et de la largeur de branches, et nous donnons plusieurs techniques d'approximation de la largeur arborescente.

1.2 Décompositions linéaires et largeur linéaire

Les *décompositions linéaires* (path decompositions) des graphes sont un cas particulier de décompositions arborescentes, où l'arbre de décomposition se réduit à une chaîne. Le paramètre associé, la *largeur linéaire* (pathwidth) est donc au moins égal à la largeur arborescente.

Plus formellement, la largeur linéaire du graphe G est le minimum, parmi tous les complétions d'intervalles H de G, de $\omega(H) - 1$. Par complétion d'intervalles de G on entend un graphe d'intervalles H avec le même ensemble de sommets, et tel que toute arête de G soit également une arête de H. Là aussi on peut se restreindre aux *complétions d'intervalles minimales*.

La largeur linéaire a été introduite dans le premier article de la série de Robertson et Seymour sur les mineurs de graphes [116]. Ce paramètre a été redéfini et étudié dans divers contextes sous des noms différents. Ainsi, la largeur linéaire est égale nombre de séparation (vertex separation number), ou encore à l'épaisseur d'intervalles ou au nombre de poursuite (node search number), moins un. Le lecteur intéressé trouvera une présentation de ces équivalences dans [17].

Les décompositions linéaires permettent de résoudre des problèmes d'optimisation particulièrement difficiles [49], pour lesquelles les décompositions arborescentes ne suffisent pas. Si l'on dispose d'une décomposition linéaire de petite largeur, les algorithmes sont beaucoup moins gourmands en mémoire que lorsque nous disposons seulement d'une décomposition arborescente de même largeur [130]. Enfin, à travers les jeux de poursuite, la largeur linéaire modélise certains phénomènes de décontamination des réseaux [7].

Le calcul de la largeur linéaire est un problème de la classe FPT : Bodlaender et Kloks présentent dans [24] un algorithme de complexité linéaire qui décide, pour tout k fixé, si la largeur linéaire du graphe en entrée est au plus k. Leur algorithme calcule d'abord une décomposition arborescente (pas forcément linéaire) de largeur minimum, et utilise cette décomposition arborescente afin de vérifier si la largeur linéaire est au plus k. La complexité de l'algorithme dépend très fortement de k, par une fonction surexponentielle. Remarquons que les meilleurs algorithmes pour l'approximation de la largeur linéaire procèdent également en approximant d'abord la largeur arborescente ; on utilise ensuite le fait que le ratio largeur linéaire/largeur arborescente est au plus $\log n$.

Par [24], la largeur linéaire est calculable en temps polynomial pour toute classe de graphes de largeur arborescente bornée. Il existe plusieurs algorithmes calculant la largeur linéaire des arbres, d'abord en temps $\mathcal{O}(n \log n)$ [105, 52], et seulement depuis peu en temps linéaire [123]. Même les graphes ayant un seul cycle (c'est-à-dire un arbre plus une arête) nécessitent des techniques élaborées pour obtenir une complexité en temps de $\mathcal{O}(n \log n)$ [51]. Il existe quelques autres classes de graphes pour lesquelles nous disposons d'algorithmes polynomiaux pour le calcul de la largeur linéaire, comme les graphes de permutation, mais ceci grâce au fait que pour ces classes la largeur linéaire est égale à la largeur arborescente. Grosso modo, presque tout ce que l'on sait sur la largeur linéaire (et ce n'est vraiment pas beaucoup !) se base sur les liens entre ce paramètre et la largeur arborescente. Encore plus surprenant, même pour des classes de graphes de largeur arborescente très petite, comme les graphes planaires extérieurs (deux-connexes), on trouve des algorithmes d'approximation de la largeur linéaire [19]. Bien que le paramètre soit calculable en temps polynomial pour ces classes, par l'algorithme de [24], les temps de calcul sont grands et les techniques de programmation dynamique de [24] n'ont pas pu être transformées en algorithmes simples et combinatoires.

Lorsque l'on s'intéresse au calcul de la largeur linéaire, on espère pouvoir compter sur des outils comme ceux qui ont fait leur preuves pour le calcul de la largeur arborescente : nous aimerions avoir une connaissance fine des complétions d'intervalles minimales. A ma connaissance, avant le début de ces travaux il n'y avait même pas un algorithme polynomial calculant une complétion d'intervalles minimale d'un graphe quelconque. Nous avons ainsi initié une étude des complétions d'intervalles minimales des graphes, en donnant plusieurs techniques pour calculer ce type d'objets. Les connaissances acquises nous ont permis de calculer en temps polynomial la largeur linéaire pour les graphes d'intervalles circulaires – et c'est la première classe pour laquelle un tel algorithme existe, alors que la largeur arborescente de ces graphes n'est ni bornée par une constante ni égale à la largeur linéaire.

Ces travaux concernant les complétions d'intervalles minimales et la largeur linéaire sont au cœur de la thèse de Karol Suchan, que j'encadre.

1.3 Structure du mémoire

Chapitre 2. Préliminaires

Ce chapitre introduit, en plus de la terminologie et des notions classiques, plusieurs outils pour aborder le calcul de la largeur arborescente. Nous verrons une heuristique de calcul d'une décomposition arborescente en découpant récursivement le graphe à l'aide de ses séparateurs minimaux. Nous définissons les *cliques maximales potentielles* d'un graphe et montrons que ses objets sont suffisants pour calculer la largeur arborescente. Enfin, nous présentons l'adaptation de ces outils au calcul de la largeur de branches.

Chapitre 3. Cliques maximales potentielles : énumération et applications

Les cliques maximales potentielles ont été introduites lors de mes travaux de thèse [132, 31]. Nous avions montré qu'elles suffisent pour calculer la largeur arborescente, mais nous laissions ouverte la question de leur énumération. Dans ce chapitre nous présentons les résultats de [32], qui donnent une nouvelle caractérisation de ces objets ainsi qu'un algorithme d'énumération. Ceci résout par l'affirmative une conjecture de Kloks et al. [87] : il existe un algorithme calculant la largeur arborescente, en temps polynomial par rapport au nombre de séparateurs minimaux du graphe en entrée.

Nous verrons ensuite des algorithmes exacts (en temps exponentiel) pour le calcul de la largeur arborescente [59] et de la largeur de branches [60] des graphes quelconques.

Enfin, nous donnons une nouvelle preuve, courte et purement combinatoire, de la conjecture de Robertson et Seymour selon laquelle, pour tout graphe planaire, sa largeur arborescente et celle de son dual diffèrent d'au plus une unité [30].

Chapitre 4. Approximation de la largeur arborescente

Nous étudions une heuristique très naturelle pour le calcul des décompositions arborescentes, heuristique qui découpe le graphe en utilisant à chaque étape le plus petit séparateur possible. Nous montrons que, pour les graphes quelconques, cette heuristique ne garantit pas une approximation de la largeur arborescente à facteur constant près. Cependant, une telle approximation est garantie lorsque l'on se restreint aux graphes de nombre astéroïde borné. Ces résultats sont basés sur [29]. Pour la classe des graphes sans triplet astéroïde, nous donnons un algorithme d'approximation de la largeur arborescente à facteur deux [33]. Enfin, nous montrons que la largeur arborescente peut être approximée à facteur $\mathcal{O}(\log OPTIMUM)$ pour les graphes quelconques [29].

Chapitre 5. Complétions d'intervalles minimales

Nous avons donné dans [75] un premier algorithme polynomial calculant une complétion d'intervalles minimale d'un graphe quelconque. Un deuxième algorithme, plus simple et de meilleure complexité, est extrait de [126]. Nous présentons également un algorithme de complexité linéaire qui calcule une complétion d'intervalles *propres* minimale [113]. Ces deux derniers sont basés sur une caractérisations des graphes d'intervalles (et d'intervalles propres) par des ordres spéciaux sur les sommets.

Chapitre 6. Pliage de graphes : principes et applications

Le dernier chapitre technique introduit une nouvelle approche pour représenter les décompositions linéaires par le *pliage* du graphe en entrée. Si le graphe est "trop plié", nous nous donnons des mécanismes pour le déplier, afin d'obtenir une meilleure décomposition. Nous obtenons dans [77] un algorithme qui, prenant en entrée un graphe G et une complétion d'intervalles (pas forcément minimale) H de G, extrait une complétion d'intervalles minimale H', avec $G \subseteq H' \subseteq H$. Enfin, nous donnons un algorithme de complexité quadratique, calculant la largeur linéaire des graphes d'intervalles circulaires [127].

Le mémoire se termine par les **Conclusions et perspectives** de ces travaux.

Chapitre 2

Préliminaires

2.1 Graphes, classes de graphes et décompositions

Tout au long de ce document nous considérerons des graphes simples (sans boucles ou arêtes multiples), finis et non orientés. De plus, nos graphes seront connexes, car tous les problèmes algorithmiques qui nous intéressent peuvent être résolus en traitant séparément chacune des composantes connexes.

2.1.1 Graphes quelconques

Un graphe sera noté $G = (V, E)$, où V est l'ensemble de sommets et E est l'ensemble d'arêtes de G. Réciproquement, $V(G)$ et $E(G)$ représentent l'ensemble de sommets, respectivement d'arêtes du graphe G. Le nombre de sommets, respectivement d'arêtes sera noté n, respectivement m. Nous noterons simplement xy une arête $\{x, y\}$ de G.

Un graphe $H = (W, F)$ tel que $W \subseteq V$ et $F \subseteq \{xy \in E | x, y \in W\}$ est appelé *sous-graphe (partiel)* de G. Nous écrirons également $H \subseteq G$. Si $W \subseteq V$ est un ensemble de sommets de G, le *sous-graphe induit* par les sommets W est $G[W] = (W, F)$, ou $F = \{xy \in E | x, y \in W\}$. Un graphe $H = (V, F)$ tel que $E \subseteq F$ est dit *sur-graphe* de G. Un graphe est *complet* si tout couple de sommets distincts sont adjacents. Un ensemble de sommets qui induit un sous-graphe complet est appelé *clique*. Une *clique maximale* sera une clique de G, maximale par inclusion. On note $\omega(G)$ le nombre maximum de sommets d'une clique de G.

Si x est un sommet de G, on notera $N_G(x)$ les sommets adjacents à x dans G. Si $W \subseteq V$ est un ensemble de sommets, $N_G(W)$ désigne le *voisinage* de W, c'est-à-dire l'ensemble des sommets de $V \backslash W$ ayant au moins un voisin dans W. Le voisinage fermé d'un sommet x (respectivement d'un ensemble de sommets W) est $N_G[x] = N_G(x) \cup \{x\}$ ($N_G[W] = N_G[W] \cup W$). Comme d'habitude, l'indice est omis s'il n'y a pas d'ambiguïté.

Une suite de sommets distincts $[x_1, \ldots, x_p]$ de G est appelée *chaîne de x_1 à x_p* si $x_1x_2, \ldots, x_{p-1}x_p$ sont des arêtes de G. Si de plus x_px_1 est une arête, alors $[x_1, \ldots, x_p]$ s'appelle *cycle* de G. Dans les deux cas, on appelle *corde* une arête qui relie deux sommets non consécutifs de la chaîne ou du cycle. Lorsque la chaîne ou le cycle n'ont pas de corde on parle de *chaîne (cycle) sans corde* ou de *chaîne induite* ou *cycle induit*. Dans tout ce document, nous considérerons des chaînes et des cycles *élémentaires*, c'est-à-dire qui ne passent pas plusieurs fois par un même sommet. Le graphe G est dit *connexe* si tout couple de sommets est relié par une chaîne. Un ensemble de sommets qui induit un sous-graphe connexe de G et qui est maximal par inclusion pour cette propriété s'appelle *composante connexe* de G. Un graphe connexe sans cycles est appelé *arbre*.

Si K est un ensemble de sommets de G, le graphe $G[V \setminus K]$ sera noté simplement $G-K$. Nous noterons $\mathcal{C}_G(K)$ l'ensemble des composantes connexes de $G - K$.

2.1.2 Décompositions de graphes et paramètres associés

Nous nous intéressons ici à trois types de décompositions de graphes : les décompositions arborescentes (tree decompositions), les décompositions linéaires (path decompositions) et les décompositions en branches (branch decompositions) et aux paramètres de décomposition associés : largeur arborescente (treewidth), largeur linéaire (pathwidth) et largeur de branches (branchwidth). Il y a dans la littérature un grand nombre de définitions équivalentes pour ces objets, parfois antérieures à la version que nous verrons dans cette section. Nous suivons les définitions de Robertson et Seymour [116, 118, 119], à qui nous devons la notoriété de ces techniques de décomposition.

Définition 2.1 ([118]) *Une* décomposition arborescente *d'un graphe $G = (V, E)$ est un couple $(\{X_i, i \in I\}, \mathcal{T} = (I, E_T))$ où les X_i sont des ensembles de sommets de G, appelés* sacs de la décomposition*, et $\mathcal{T}$ est un arbre tel que :*

1. $\bigcup_{i \in I} X_i = V$,
2. *pour chaque arête xy de G, il existe un $i \in I$ tel que $x, y \in X_i$,*
3. *pour tout $x \in V$, l'ensemble T_x des sommets $i \in I$ de $\mathcal{T}$ tels que $x \in X_i$ induit dans $\mathcal{T}$ un sous-arbre.*

La largeur *d'une décomposition $(\{X_i, i \in I\}, \mathcal{T} = (I, E_T))$ est $\max_{i \in I} |X_i| - 1$. La* largeur arborescente *(*treewidth*, en anglais) de G, notée $\mathrm{tw}(G)$, est la largeur minimum de toutes les décompositions arborescentes de G.*

Pour éviter les confusions entre le graphe G et l'arbre de décomposition $\mathcal{T}$, les sommets de $\mathcal{T}$ seront appelés *nœuds* et ses arêtes seront appelées *branches*.

La figure 2.1 a, b montre un graphe et l'une de ses décompositions arborescentes, de largeur 3. En l'occurrence, c'est aussi la largeur arborescente de ce graphe. Notons qu'il est facile de trouver une décomposition arborescente d'un graphe : on peut toujours prendre un arbre avec un seul noeud et lui mettre comme étiquette l'ensemble V. La largeur de cette décomposition est $n - 1$, si n est le nombre de sommets de V.

Regardons la largeur arborescente de quelques graphes simples. La largeur arborescente d'un graphe complet est $n-1$, c'est donc "le pire cas" car on a toujours $\mathrm{tw}(G) \leq n-1$. Il est assez facile de voir que les graphes de largeur arborescente 1 sont exactement les forêts – ou les arbres, si l'on se restreint aux graphes connexes. Ceci est censé justifier la constante "-1"

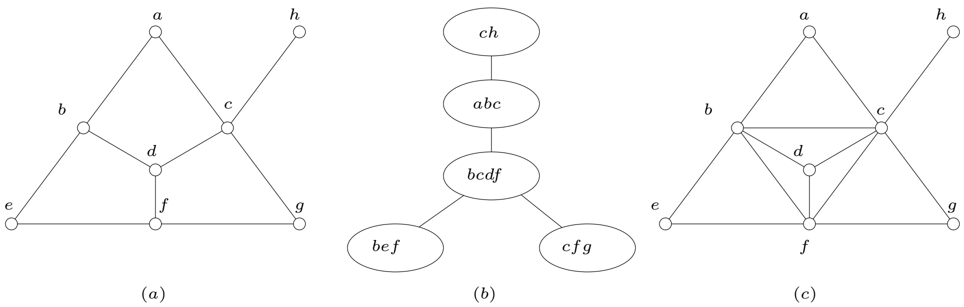

FIG. 2.1 – Décomposition arborescente et triangulation associée

de la définition de la largeur arborescente. On peut voir aussi que la largeur arborescente des cycles est 2. Plus difficile, la largeur arborescente d'une grille de taille $a \times b$ est $\min(a, b)$. Comme on pouvait s'y attendre, le calcul de la largeur arborescente est NP-difficile, même pour les graphes co-bipartis (les compléments des graphes bipartis) [70].

Un cas particulier intéressant de décomposition arborescentes est lorsque l'arbre de décomposition est une chaîne.

Définition 2.2 ([116]) *Une* décomposition linéaire *d'un graphe G est une décomposition arborescente $(\{X_i, i \in I\}, \mathcal{T} = (I, E_T))$ telle que $\mathcal{T}$ est une chaîne. La* largeur linéaire *(*pathwidth*, en anglais) de G, notée $\mathrm{pw}(G)$, est la largeur minimum parmi toutes les décompositions linéaires de G.*

Le calcul de la largeur linéaire est NP-difficile, même pour les graphes triangulés [68].

Définition 2.3 ([119]) *Une* décomposition en branches *du graphe $G = (V, E)$ est une paire $(\mathcal{T}, \tau)$ dans laquelle $\mathcal{T} = (V_T, E_T)$ est un arbre ternaire (dont chaque nœud interne est de degré trois) et τ est une function surjective*[1] *associant à chaque feuille de l'arbre $\mathcal{T}$ une arête de G.*

*Soient $T_1(e)$ et $T_2(e)$ les sous-arbres de $\mathcal{T}$ obtenus à partir de $\mathcal{T}$ en supprimant la branche $e \in E_T$. Nous définissons l'*étiquette *de e, $\mathrm{lab}(e)$, comme étant l'ensemble des sommets de G incidents à la fois à des arêtes de G associées à des feuilles de $T_1(e)$ et à des arêtes correspondant à des feuilles de $T_2(e)$. Le maximum $\big\{|\,\mathrm{lab}(e)|,\, e \in E_T\big\}$, est appelé la* largeur *de la décomposition en branches. La* largeur de branches *du graphe G $\big($notée $\mathrm{bw}(G)\big)$ est la largeur minimum parmi toutes les décompositions en branches de G.*

La définition d'une décomposition en branches peut sembler compliquée à première vue, cependant elle représente tout simplement un découpage récursif de l'ensemble des arêtes du graphe. Imaginons que l'on partitionne l'ensemble d'arêtes E en deux parties,

[1] Dans la définition classique, les feuilles de $\mathcal{T}$ sont en bijection avec les arêtes de G et la version surjective est appelée décomposition relaxée. Cependant les deux notions sont équivalentes et la décomposition relaxée est plus adéquate pour nos résultats.

et que chaque partie est récursivement partitionnée en deux jusqu'à ce que chaque classe contienne exactement une arête. La trace de cet algorithme de partitionnement récursif est précisément un arbre ternaire, dont les feuilles sont en bijection avec les arêtes du graphe.

La largeur de branches est fortement liée à la largeur arborescente. Robertson et Seymour ont prouvé que les deux paramètres diffèrent d'un facteur au plus 1.5. Plus précisément, pour tout graphe G on a les inégalités $\mathrm{bw}(G) \leq \mathrm{tw}(G) + 1 \leq 1.5\,\mathrm{bw}(G)$. Si G est un graphe complet, sa largeur arborescente est $n-1$, alors que sa largeur de branches est $\lceil 2n/3 \rceil$ (cf. [119]). Le calcul de ce paramètre est NP-difficile même pour les graphes split, une sous-classe des graphes triangulés [88]. Dans l'autre sens, le calcul de la largeur de branches est polynomial pour les graphes planaires [122], alors que l'on ignore toujours s'il existe un algorithme polynomial calculant la largeur arborescente des graphes planaires.

Seymour et Thomas considèrent que les décompositions en branches sont plus naturelles et plus pratiques que les décompositions arborescentes, dans la conception d'algorithmes. Certains résultats récents [80, 50] appuient cette intuition.

2.1.3 Classes de graphes

Nous n'épuiserons pas dans cette section la description des classes de graphes que nous aurons à traiter par la suite. Mais comme nous rencontrerons dans cet ouvrage plusieurs classes de graphes d'intersection, nous en donnons ici le principe.

On considère une famille $\mathcal{F}$ d'ensembles non vides. On lui associe le *graphe d'intersection* $G_{\mathcal{F}}$ de $\mathcal{F}$ obtenu comme suit : chaque ensemble de $\mathcal{F}$ est représenté par exactement un sommet de $G_{\mathcal{F}}$ et deux sommets de $G_{\mathcal{F}}$ sont adjacents si et seulement si les ensembles correspondants de $\mathcal{F}$ s'intersectent. On dira que la famille $\mathcal{F}$ est un *modèle d'intersection* de $G_{\mathcal{F}}$. Quand la famille $\mathcal{F}$ est formée d'ensemble particuliers, on obtient des classes de graphes intéressantes.

Les *graphes d'intervalles* sont les graphes d'intersection des intervalles d'un ordre total (de la droite réelle, par exemple). Les *graphes d'intervalles propres* sont les graphes ayant un modèle d'intervalles tel qu'aucun de ces intervalles ne soit strictement contenu dans un autre.

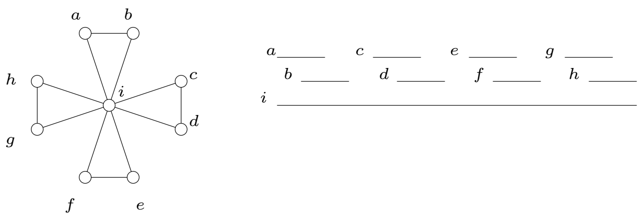

FIG. 2.2 – Graphe d'intervalles

Les *graphes de cordes* sont obtenus comme intersections des cordes d'un cercle. Lorsque la famille $\mathcal{F}$ est formée d'arcs de cercle, on obtient les *graphes d'intervalles circulaires*.

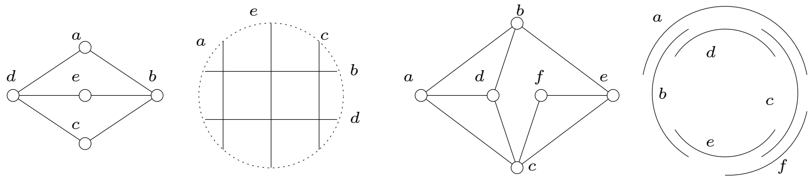

FIG. 2.3 – Graphes de cordes et graphes d'intervalles circulaires

Les *graphes triangulés*, qui s'obtiennent aussi par un modèle d'intersection, ainsi que les graphes d'intervalles, sont au cœur de ce document. Nous leur consacrons la section suivante.

2.2 Graphes triangulés et graphes d'intervalles

Il est difficile de savoir à quelle époque remonte la définition des graphes triangulés ; ces graphes apparaissent déjà en 1958 dans les travaux de Hajós.

Définition 2.4 *Un graphe est dit* triangulé *si tout cycle ayant plus de trois sommets possède une corde.*

Dans la littérature anglo-saxonne, on trouve les dénominations de *triangulated graphs*, *chordal graphs* ou encore *rigid circuit graphs*. On voit facilement que cette classe de graphes est héréditaire : tout sous-graphe induit d'un graphe triangulé est également triangulé.

Cette classe de graphes a été intensément étudiée dans la littérature et connaît un grand nombre de caractérisations équivalente. La brève présentation qui suit est basée notamment sur le livre de Golumbic [66] et un article de Blair et Peyton [14].

Le plus important pour la suite est de voir ces graphes comme des "arbres généralisés". Notons $\mathcal{K}_G = \{\Omega_1, \dots, \Omega_p\}$ l'ensemble des cliques maximales de G. Soit $\mathcal{T}$ un arbre sur $\mathcal{K}_G$, i.e. chaque clique de $\mathcal{K}_G$ correspond à un noeud de $\mathcal{T}$. Nous dirons aussi que les sommets de $\mathcal{T}$ sont étiquetés par les cliques maximales de G et nous désignerons simplement par Ω le noeud de $\mathcal{T}$ étiqueté par la clique maximale Ω. Nous dirons que $\mathcal{T}$ est un *arbre de cliques* de G s'il satisfait la *propriété d'intersection des cliques* : pour tout couple de cliques distinctes Ω et Ω', l'ensemble $\Omega \cap \Omega'$ est contenu dans toutes les cliques sur la chaîne reliant Ω et Ω' dans l'arbre $\mathcal{T}$. Remarquons qu'un arbre de cliques de G est également une décomposition arborescente de G, dont les sacs sont précisément les cliques maximales de G. En particulier la largeur de cette décompositoin est $\omega(G) - 1$.

Le théorème suivant est dû à Gavril [65] :

Théorème 2.5 ([65]) *Un graphe G est triangulé si et seulement si il admet un arbre de cliques.*

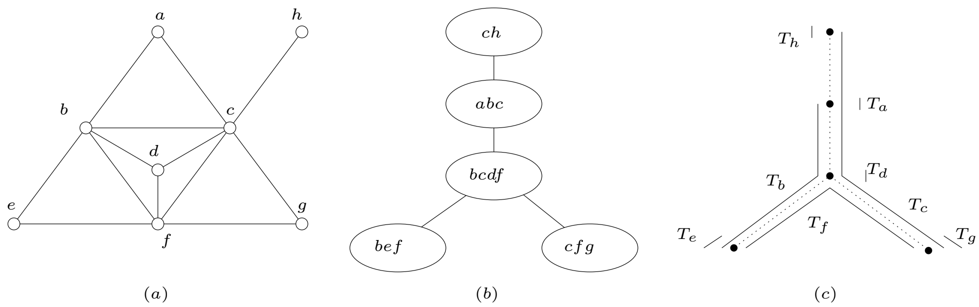

FIG. 2.4 – Graphe triangulé et représentation arborescentes

La figure 2.4 a, b montre un exemple de graphe triangulé, avec un de ses arbres de cliques.

Notons qu'à un graphe triangulé G on peut en général associer plusieurs arbres de cliques, non isomorphes. On trouve par exemple dans Paul [112] une étude sur la formation de tous ces arbres et des relations entre eux.

D'un point de vue technique, nous utiliserons beaucoup la notion d'arbre de cliques. Mais en fait, cette notion est fortement liée à la représentation des graphes triangulés comme des graphes modélisant l'intersection des sous-arbres d'un arbre. Cette dernière a l'avantage d'être plus intuitive.

Théorème 2.6 ([65]) *Un graphe est triangulé si et seulement si il est le graphe d'intersection d'une famille de sous-arbres d'un arbre.*

Une autre caractérisation des graphes triangulés est donnée par les *ordres d'élimination simplicielle*.

Définition 2.7 *Soit $G = (V, E)$ un graphe. Un sommet x est dit* simpliciel *si son voisinage $N_G(x)$ est une clique. Un ordre $x_1, \dots x_n$ sur les sommets de G tel que pour tout $i, 1 \leq i \leq n$ le sommet x_i est simpliciel dans $G[\{x_i, \dots, x_n\}]$ est appelé* ordre d'élimination simplicielle.

La preuve de ce théorème se trouve par exemple dans [66].

Théorème 2.8 *Un graphe G est triangulé si et seulement si il admet un ordre d'élimination simplicielle.*

Toujours pour le graphe de la figure 2.4, un ordre d'élimination simplicielle est par exemple h, a, e, g, d, b, c, f.

Cette propriété est à la base des algorithmes linéaires de reconnaissance des graphes triangulés tels que LexBFS et MCS, qui fonctionnent, d'ailleurs, de manière assez surprenante.

Tout graphe triangulé est le graphe d'intersection des sous-arbres d'un arbre. Si l'arbre support est une chaîne, les sous-arbres deviennent des sous-chaînes et le graphe sera un graphe d'intervalles. Clairement, les graphes d'intervalles sont une sous-classe des graphes triangulés. Pour les graphes d'intervalles, les arbres de cliques deviennent des chaînes de cliques.

Théorème 2.9 *Un graphe G est d'intervalles si et seulement si il possède une chaîne de cliques, c'est-à-dire un arbre de cliques tel que l'arbre se réduit à une chaîne.*

Nous verrons le moment venu d'autres caractérisations des graphes d'intervalles, par exemple une caractérisation utilisant un ordre spécial sur les sommets, à la façon de l'ordre d'élimination simplicielle.

2.3 Problèmes de triangulation

Dans cette partie nous redéfinirons la largeur arborescente et la largeur linéaire d'un graphe, en montrant que leur calcul est exprimable comme un "problème de triangulation". Le calcul de ces paramètres pour un graphe G reviendra à trouver un "bon" surgraphe triangulé (ou d'intervalles) H de G.

Définition 2.10 *Soit $G = (V, E)$ un graphe quelconque. On dit que le graphe $H = (W, F)$ est une* triangulation *de G si $V = W$, $E \subseteq F$ et H est un graphe triangulé. En d'autres termes, H est un sur-graphe triangulé de G.*

Le graphe $H = (V, F)$ est une triangulation minimale *de G si H est une triangulation de G et pour tout F' tel que $E \subseteq F' \subset F$, le graphe $H' = (V, F')$ n'est pas une triangulation de G.*

En utilisant la caractérisation des graphes triangulés par des arbres de cliques, on prouve :

Proposition 2.11 *La largeur arborescente de G est égal au minimum, parmi toutes les triangulations H de G, de $\omega(H) - 1$.*

Pour calculer la largeur arborescente d'un graphe G il faut donc chercher des triangulations de G, tout en minimisant la taille de la clique maximum. On peut donc se limiter à chercher parmi les *triangulations minimales* de G. C'est pourquoi les algorithmes que nous présenterons par la suite pour le calcul de ce paramètre seront basés sur des caractérisations algorithmiques des triangulations minimales d'un graphe.

De manière similaire, la largeur linéaire s'exprime comme un problème de plongement dans un graphe d'intervalles.

Définition 2.12 *Une* complétion d'intervalles *d'un graphe quelconque $G = (V, E)$ est un graphe d'intervalles $H = (V, F)$, avec $E \subseteq F$. Si de plus aucun sous-graphe strict de H n'est une complétion d'intervalles de G, nous dirons que H est une* complétion d'intervalles minimale *du graphe G.*

Proposition 2.13 *La largeur linéaire de G est égal au minimum, parmi toutes les complétions d'intervalles H de G, de $\omega(H) - 1$.*

Là aussi, pour calculer ce minimum nous pouvons restreindre nos recherches aux complétions d'intervalles minimales.

Rose, Tarjan et Lueker donnent une caractérisation simple des triangulations minimales [121] :

Lemme 2.14 *Soit H une triangulation d'un graphe G. H est une triangulation minimale de G si et seulement si, pour toute arête e de $E(H) \backslash E(G)$, le graphe $H - e = (V, E(H) \backslash \{e\})$ n'est pas une triangulation de G.*

Cette caractérisation mène à un algorithme polynomial très simple calculant une triangulation minimale du graphe en entrée. Malheureusement la propriété ne s'étend pas aux complétions d'intervalles minimales.

2.4 Séparateurs minimaux

Les algorithmes de calcul de la largeur arborescente et de la complétion minimale passent souvent par la notion de *séparateur minimal*. Nous définissons ici ces objets et nous étudions plus en détail les séparateurs minimaux des graphes triangulés.

2.4.1 Généralités sur les séparateurs minimaux

Définition 2.15 *Soit $G = (V, E)$ un graphe et soient a et b deux sommets de G. Un ensemble de sommets $S \subseteq V$ est appelé a, b-*séparateur *(ou simplement* séparateur*) si a et b se trouvent dans des composantes connexes différentes de $G - S$. Un a, b-séparateur est dit* a, b-séparateur minimal *s'il ne contient pas un autre a, b-séparateur. L'ensemble de sommets $S \subseteq V$ est appelé* séparateur minimal *de G s'il existe un couple de sommets a, b tel que S soit un a, b-séparateur minimal.*

Remarquons qu'un séparateur minimal peut être strictement contenu dans un autre séparateur minimal, voir figure 2.5 où le h, g-séparateur minimal $\{c\}$ est strictement inclus dans le a, g-séparateur minimal $\{c, f\}$.

Si S est un ensemble de sommets de G (pas forcément un séparateur), rappelons que $\mathcal{C}_G(S)$ est l'ensemble des composantes connexes de $G - S$. Soit $C \in \mathcal{C}_G(S)$ une telle composante connexe. Si tout sommet de S a au moins un voisin dans C, nous dirons que C est une *composante connexe pleine associée à S* ou que C est une *composante connexe pleine par rapport à S* dans G. Sur la figure 2.5, les composantes connexes de $G - T$ sont $\{a, b, e, d\}$, $\{g\}$ et $\{h\}$, et les deux premières sont pleines par rapport à T.

Lemme 2.16 *S est un séparateur minimal de G si et seulement si $G - S$ a au moins deux composantes pleines par rapport à S.*

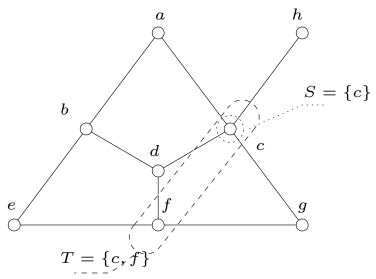

FIG. 2.5 – Séparateurs minimaux

Les séparateurs minimaux d'un graphe peuvent être énumérés en temps polynomial par séparateur. Un premier algorithme de ce type a été donné dans [90], la complexité a été améliorée dans [12]. Le nombre de séparateurs minimaux d'un graphe à n sommets peut être exponentiel par rapport à n. Cependant, ce résultat est intéressant car beaucoup de classes de graphes (graphes de cordes, graphes d'intervalles circulaires, graphes faiblement triangulés, ...) ont un nombre polynomial de séparateurs minimaux.

Théorème 2.17 ([12]) *L'ensemble Δ_G des séparateurs minimaux de G peut être calculé en temps $\mathcal{O}(n^3|\Delta_G|)$.*

2.4.2 Séparateurs minimaux des graphes triangulés

Dirac a donné en 1961 une première caractérisation des graphes triangulés à l'aide des séparateurs minimaux [48] :

Théorème 2.18 *Un graphe G est triangulé si et seulement si tous ses séparateurs minimaux sont des cliques.*

Le théorème de Dirac est expliqué de manière beaucoup plus structurelle par le très important théorème qui suit, qui se trouve dans les travaux de Ho et Lee [81] et Lundquist [101] :

Théorème 2.19 *Soit G un graphe triangulé et soit $\mathcal{T}$ un arbre de cliques* quelconque *de G. Un ensemble de sommets S est un séparateur minimal de G si et seulement si S s'écrit sous la forme $\Omega \cap \Omega'$, où Ω et Ω' sont des cliques maximales de G, adjacentes dans l'arbre de cliques $\mathcal{T}$.*

Ainsi à chaque branche $\Omega\Omega'$ d'un arbre de cliques $\mathcal{T}$ est associé le séparateur minimal $S = \Omega \cap \Omega'$, et chaque séparateur minimal correspond à une ou plusieurs branches de l'arbre.

2.5 Triangulations minimales, séparateurs minimaux et cliques maximales potentielles

2.5.1 Triangulations minimales et séparateurs minimaux

Définition 2.20 *Nous dirons que deux séparateurs minimaux S et T* se croisent *si S sépare deux sommets de T. Dans le cas contraire, ils sont* parallèles*. Ces relations sont symétriques [110].*

Le théorème suivant, dû à Parra et Scheffler, est un outil très puissant pour l'étude des triangulations minimales. Il indique que les triangulations minimales d'un graphe sont obtenues en choisissant un certain ensemble de séparateurs minimaux et en complétant chacun de ces séparateurs en une clique.

Notation 2.21 *Soit $\Gamma = \{S_1, \ldots, S_p\}$ un ensemble d'ensembles de sommets du graphe G. Notons G_Γ le graphe obtenu à partir de G en complétant chaque élément de Γ en une clique. Si Γ est formé d'un seul élément S, nous écrirons simplement G_S à la place de $G_{\{S\}}$.*

Théorème 2.22 ([110])

Soit $\Gamma \subseteq \Delta_G$ un ensemble maximal de séparateurs deux à deux parallèles de G. Le graphe $H = G_\Gamma$ est une triangulation minimale de G et $\Delta_H = \Gamma$.

Soit H une triangulation minimale d'un graphe G. Alors Δ_H est un ensemble maximal de séparateurs deux à deux parallèles de G et $H = G_{\Delta_H}$.

Soulignons que les séparateurs minimaux d'une triangulation minimale H de G se comportent de façon homogène dans les deux graphes, c'est-à-dire que pour un tel séparateur S, les composantes de $H - S$ sont les mêmes que dans $G - S$. De plus, les composantes pleines par rapport à S sont les mêmes pour les deux graphes.

Par le théorème 2.22, toute triangulation minimale de G est obtenue en complétant certains de ses séparateurs minimaux. La proposition ci-dessous nous indique que ces séparateurs peuvent être complétés un à un.

Proposition 2.23 ([110]) *Soit Γ' un ensemble de séparateurs deux à deux parallèles du graphe G et $H' = G_{\Gamma'}$. S est un séparateur minimal de H' si et seulement si S est un séparateur minimal de G, parallèle dans G à tous les séparateurs $T \in \Gamma'$.*

Il en découle l'algorithme suivant, calculant une triangulation minimale d'un graphe quelconque.

$H := G$
tantque H n'est pas triangulé
  choisir $S \in \Delta_H$ tel que $H[S]$ ne soit pas une clique
  $H := H_S$
returner H

Non seulement cet algorithme produit toujours une triangulation minimale du graphe en entrée, mais de plus chaque triangulation minimale de G peut être obtenue par un choix approprié de séparateurs. Nous n'avons pas encore discuté de la façon dont on peut choisir un séparateur minimal qui n'est pas une clique. La question sera réglée par une autre version du même algorithme, dans laquelle chaque séparateur choisi découpe le graphe en plusieurs graphes. Nous y verrons plus clairement en quoi les triangulations sont bien des *décompositions* du graphe.

La nouvelle version, donnée dans [9], est une combinaison entre les résultats de Parra et Schaeffler cités ci-dessus et les travaux de Tarjan sur les séparateurs cliques. Nous avons besoin d'introduire la notion de *bloc*, qui sera appelée à jouer un rôle important tout au long de ce document.

Définition 2.24 *Un ensemble de sommets $B \subseteq V$ de G est appelé* bloc *si, pour chaque composante connexe C_i de $G - B$,*
- *son voisinage $S_i = N(C_i)$ est un séparateur minimal ;*
- *$B \backslash S_i$ est non vide et est contenu en une unique composante connexe de $G - S_i$, pleine par rapport à S_i.*

Nous dirons que les séparateurs minimaux S_i bordent *le bloc B et notons par $\mathcal{S}(B)$ l'ensemble de ces séparateurs.*

Soit $\mathcal{B}_G$ l'ensemble des blocs de G. Notons que V est un bloc avec $\mathcal{S}(V) = \emptyset$.

Définition 2.25 *Soit B un bloc du graphe G. On appelle* réalisation *du bloc B le graphe $G_{\mathcal{S}(B)}[B]$ obtenu à partir de $G[B]$ en complétant chaque séparateur bordant B en une clique.*

L'algorithme de découpage maintient une liste de blocs appelée lb. Initialement $lb = \{V\}$. A chaque étape, on découpe un bloc B en choisissant un séparateur minimal S de $R(B)$. Le bloc B est alors remplacé par des blocs de la forme $S_i \cup C_i$, où C_i sont les composantes connexes de $R(B) - S$ et $S_i = N_G(C_i)$ (on peut montrer que ces derniers sont également des séparateurs minimaux). Le processus s'arête lorsque les réalisations de tous les blocs de lb sont des cliques. La sortie est le graphe $H = G_{lb}$, obtenu à partir de G en complétant chaque élément de lb en une clique.

Voici le pseudo-code de l'algorithme `DecoupageEnBlocs` :

Théorème 2.26 ([9], propriété 7.5) *L'algorithme `DecoupageEnBlocs` produit une triangulation minimale H de G, et les cliques maximales de H sont précisément les blocs de lb.*

L'algorithme `DecoupageEnBlocs` est clairement polynomial, sa complexité dépendant de la façon dont on calcule les nouveaux séparateurs. Nous discuterons dans le chapitre 4 une version qui choisit un séparateur de cardinal minimum.

Afin de mieux comprendre les blocs, regardons les séparateurs qui les bordent.

`DecoupageEnBlocs`

Entrée : $G = (V, E)$
Sortie : une triangulation minimale de G
  $lb := \{V\}$ $/* lb$ une liste de blocs $*/$
  tantque il existe $B \in lb$ tel que $R(B)$ ne soit pas un graphe complet **faire**
    $S :=$ un séparateur minimal de $R(B)$
    calculer les composantes $C_1, \ldots C_p$ de $R(B) - S$ et leurs voisinages $S_i := N_G(C_i)$
    dans lb, remplacer B par les blocs $S_i \cup C_i$, $1 \leq i \leq p$
  fin_tantque
  returner $H = G_{lb}$

FIG. 2.6 – Algorithme de décomposition en blocs.

Remarque 2.27 *Soit B un bloc. Si parmi les éléments de $\mathcal{S}(B)$ il y en a un, noté S, qui contient tous les autres, alors B est de la forme $S \cup C$, où C est une composante pleine de $G - S$. Nous dirons dans ce cas que B est un 1-bloc (car bordé par un seul séparateur) et nous noterons ce bloc (S, C). Remarquons qu'à partir d'un 1-bloc B, l'unique couple (S, C) tel que $B = S \cup C$ peut être calculé facilement, en prenant $S = N(V \backslash B)$ et $C = B \backslash S$.*

Si nous avons un ensemble de séparateurs minimaux $\{S_1, \ldots, S_p\}$, tel que pour chaque séparateur S_i il existe un unique 1-bloc (S_i, C_i) contenant tous les autres, on définit de façon naturelle la partie entre *ces séparateurs comme étant* $\mathrm{PartieEntre}(S_1, \ldots S_i) = \bigcap_i (S_i, C_i)$. *Cette partie entre définit toujours un bloc. Un bloc B qui n'est ni V ni un 1-bloc est égal à la partie entre les séparateurs minimaux qui bordent B.*

2.5.2 Triangulations minimales et cliques maximales potentielles

Remarquons que même pour des graphes avec peu de séparateurs minimaux, le nombre d'ensembles maximaux de séparateurs deux à deux parallèles (qui est le même que le nombre de triangulation minimales) peut être très grand. Prenons comme exemple un cycle de longueur de longueur $2p$, noté $C_{2p} = [0, 1, \ldots, 2p - 1]$. Les séparateurs minimaux de ce cycle sont les ensembles $\{i, j\}$, avec $1 \leq i, j \leq 2p$ et $|i - j| > 1$, donc C_{2p} a moins de $(2p)^2$ séparateurs minimaux. Remarquons sur la figure 2.7 que deux séparateurs $\{i, j\}$ et $\{i', j'\}$ se croisent si et seulement si les segments $[ij]$ et $[i'j']$ se croisent, ce qui nous permet de visualiser facilement les familles de séparateurs parallèles. Considérons maintenant les séparateurs $S_i = \{i, 2p - i\}$, pour $1 \leq i \leq p - 1$, et les séparateurs $T_i = \{i, 2p - i - 1\}$ et $T'_i = \{i + 1, 2p - i\}$ pour $1 \leq i \leq p - 2$. Il est facile de voir que pour tout choix de $U_i \in \{T_i, T'_i\}$, $1 \leq i \leq p - 2$, la famille de séparateurs $\{S_1, U_1, S_2, U_2, \ldots, S_{p-2}, U_{p-2}, S_{p-1}\}$ est une famille maximale de séparateurs parallèles de C_{2p}. On voit donc que C_{2p} a plus de 2^{p-2} familles maximales de séparateurs deux à deux parallèles.

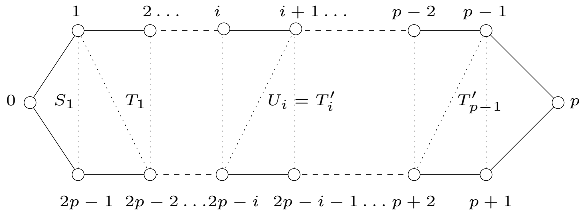

FIG. 2.7 – Séparateurs du cycle C_{2p}

Cependant, les cliques maximales de toutes ces triangulations sont formées par trois sommets – il y en a donc une quantité polynomiale. Or ce sont bien les cliques maximales des triangulations minimales qui nous intéressent. Nous verrons dans cette section que si nous disposons de tous ces objets, nous serons en mesure de calculer une triangulation optimale.

Définition 2.28 (clique maximale potentielle [31, 132]) *Un ensemble de sommets Ω d'un graphe G est appelé* clique maximale potentielle *s'il existe une triangulation minimale H de G telle que Ω soit une clique maximale de H.*

L'ensemble des cliques maximales potentielles du graphe G est noté Π_G.

Observons que l'on pourrait également caractériser les cliques maximales potentielles d'un graphe comme étant l'ensemble de ses blocs "non découpables", donc minimaux par inclusion.

Nous nous donnerons d'abord les moyens de reconnaître une clique maximale potentielle : nous répondrons à la question "étant donné un ensemble de sommets K d'un graphe G, existe-t-il une triangulation minimale H de G telle que K soit une clique maximale de H ?".

Théorème 2.29 *Soit $K \subseteq V$ un ensemble de sommets du graphe G. Soit $\mathcal{C}(K) = \{C_1, C_2, \ldots, C_p\}$ l'ensemble des composantes connexes de $G - K$. On note $S_i = N(C_i)$, pour tout i, $1 \leq i \leq p$, et $\mathcal{S}(K)$ l'ensemble de tous les S_i. K est une clique maximale potentielle si et seulement si*

- *S_i est strictement contenu dans K, pour tout $i, 1 \leq i \leq p$.*
- *$G_{\mathcal{S}(K)}[K]$, obtenu à partir de $G[K]$ en complétant en clique chaque élément de $\mathcal{S}(K)$, est un graphe complet.*

De plus, si K est une clique maximale potentielle, alors $\mathcal{S}(K)$ est l'ensemble des séparateurs minimaux de G, contenus dans K.

Ce théorème nous donne un algorithme simple pour reconnaître une clique maximale potentielle :

Corollaire 2.30 *Soit K un ensemble de sommets du graphe G. Nous pouvons vérifier en temps $\mathcal{O}(nm)$ si K est une clique maximale potentielle de G.*

Nous avons montré dans [31, 59] que, étant donné un graphe G et l'ensemble Π_G de ses cliques maximales potentielles, la largeur arborescente du graphe peut être calculée en $\mathcal{O}(n^3|\Pi_G|)$ (théorème 2.35). Ce résultat sera utilisé à plusieurs reprises dans ce mémoire, comme une boite noire. Néanmoins, pour le lecteur intéressé, nous présentons dans le reste de cette sous-section les principes de l'algorithme calculant la largeur arborescente à partir des cliques maximales potentielles.

Soit (S, C) un 1-bloc du graphe G. Rappelons que $R(S, C) = G_S[S \cup C]$ est la réalisation du bloc (S, C). La proposition suivante, due à Kloks, Kratsch et Spinrad [92] donne une relation entre les triangulations minimales d'un graphe et les triangulations minimales de certaines réalisations des blocs de G.

Proposition 2.31 ([92]) *Soit S un séparateur minimal de G, soient $C_1, \ldots, C_p$ les composantes connexes de $G - S$ et notons $S_i = N_G(C_i)$.*

Soit H une triangulation minimale de G ayant S comme séparateur minimal. Alors $H[S_i \cup C_i]$ est une triangulation minimale de $R(S_i, C_i)$, pour toute composante C_i de $G - S$.

Réciproquement, soit H_i une triangulation minimale de $R(S_i, C_i)$ pour chaque $i, 1 \leq i \leq p$. Alors le graphe $H = (V, E(H))$ avec $E(H) = \bigcup_{i=1}^{p} E(H_i)$ est une triangulation minimale de G.

On en déduit :

Corollaire 2.32 *Soit G un graphe non complet. Alors*

$$\mathrm{tw}(G) = \min_{S \in \Delta_G} \max_{C_i \in \mathcal{C}(S)} \mathrm{tw}(R(S_i, C_i))$$

où le maximum est pris sur toutes les composantes C_i de $G - S$, et $S_i = N_G(C_i)$.

Pour obtenir une triangulation minimale de la réalisation $R(S, C)$, nous devons choisir une clique maximale potentielle Ω avec $S \subset \Omega \subseteq (S, C)$ et compléter Ω. Comme Ω découpe (S, C) en blocs plus petits, il faut également trianguler ces derniers. Plus formellement :

Théorème 2.33 ([31]) *Soit (S, C) un 1-bloc dans G. Alors $H(S, C)$ est une triangulation minimale de $R(S, C)$ si et seulement si il existe une clique maximale potentielle Ω telle que $S \subset \Omega \subseteq (S, C)$ et $H(S, C) = (S \cup C, E(H))$ avec*

$$E(H) = \bigcup_{i=1}^{p} E(H_i) \cup \{xy | x, y \in \Omega\}$$

où $\{C_1, \ldots, C_p\}$ sont les composantes de $G - \Omega$ contenues dans C, $S_i = N_G(C_i)$, et les graphes H_i sont des triangulations minimales de $R(S_i, C_i)$.

Remarquons que les blocs (S_i, C_i) sont strictement contenus dans (S, C).

Corollaire 2.34 *Soit (S, C) un bloc plein de S dans G. Alors*

$$\mathrm{tw}(R(S,C)) = \min_{S \subset \Omega \subseteq (S,C)} \max_i (|\Omega| - 1, \mathrm{tw}(R(S_i, C_i)))$$

où le maximum est pris sur toutes les composantes C_i de $G - \Omega$ avec $C_i \subset C$ et $S_i = N_G(C_i)$.

Algorithme `LargeurArborescenteCMP`

Entrée : G, l'ensemble Π_G de ses cliques maximales potentielles et
  l'ensemble Δ_G de ses séparateurs minimaux
Sortie : $\mathrm{tw}(G)$
début
  calculer tous les 1-blocs (S, C) et les trier par nombre croissant de sommets
  pour chaque bloc (S, C) par ordre croissant
    $\mathrm{tw}(R(S,C)) := |S \cup C| - 1$ si (S, C) est minimal par inclusion
      et $\mathrm{tw}(R(S,C)) := \infty$ sinon
    pour chaque c.m.p. $\Omega \in \Pi_G$ telle que $S \subset \Omega \subseteq (S, C)$
      calculer les composantes C_i de $G - \Omega$ t.q. $C_i \subset C$
      prendre $S_i := N_G(C_i)$
      $\mathrm{tw}(R(S,C)) := \min(\mathrm{tw}(R(S,C)),$
        $\max_i(|\Omega| - 1, \mathrm{tw}(R(S_i, C_i))))$
    fin_pour
  fin_pour
  $\mathrm{tw}(G) := \min_{S \in \Delta_G} \max_{C_i \in \mathcal{C}(S)} \mathrm{tw}(R(S_i, C_i))$, où $S_i = N_G(C_i)$
fin

FIG. 2.8 – Algorithme calculant la largeur arborescente à partir des cliques maximales potentielles

L'algorithme est pratiquement celui donné dans [31, 132]. Cependant nous avons fortement amélioré sa complexité dans [59]. La complexité annoncée initialement était de l'ordre de $\mathcal{O}(n|\Delta_G| \cdot |\Pi_G|)$, venant des deux **pour** imbriquées. La nouvelle complexité est de $\mathcal{O}(n^3|\Pi_G|)$. Sachant que le nombre de séparateurs minimaux peut lui-même être exponentiel en la taille du graphe, cette amélioration est importante. De plus, nous en aurons absolument besoin un peu plus tard dans ce manuscrit, lorsque nous traiterons du calcul exact (en temps exponentiel) de la largeur arborescente. Puisque ce résultat de complexité est nouveau, nous donnerons les principaux éléments de la preuve.

Théorème 2.35 *Il existe un algorithme qui, étant donné un graphe G ainsi que l'ensemble Δ_G de ses séparateurs minimaux et l'ensemble Π_G de ses cliques maximales potentielles, calcule la largeur arborescente et de G en temps $\mathcal{O}(n^3\,|\Pi_G|)$. De plus, l'algorithme construit une triangulation optimale.*

Preuve. (éléments.) Montrons que, avec une implantation appropriée, le nombre total d'itérations de la boucle **pour** la plus interne est $\mathcal{O}(n|\Pi_G|)$ — alors que la structure des deux boucles imbriquées suggère que ce nombre pourrait être de l'ordre de $n|\Delta_G| \cdot |\Pi_G|$.

Un triplet (S, C, Ω), où (S, C) est un 1-bloc et Ω est une clique maximale potentielle avec $S \subset \Omega \subseteq S \cup C$ est appelé un *triplet utile*. Vérifions qu'il y a au plus $n|\Pi_G|$ triplets utiles. Fixons Ω et comptons le nombre de triplets utiles la contenant. Par le théorème 2.29, il y a au plus n séparateurs $S \subset \Omega$, chacun étant le voisinage d'une composante connexe de $G - \Omega$. Pour un tel S, la composante connexe C de $G - S$ telle que $\Omega \subseteq S \cup C$ est unique. Ceci montre que le nombre de triplets utiles n'excède pas $n|\Pi_G|$. Supposons que ces triplets utiles sont calculés lors d'une phase de préprocessage, et que chaque 1-bloc (S, C) pointe vers les cliques maximales potentielles Ω avec lesquelles (S, C, Ω) forment un triplet utile. Dans ce cas le nombre total d'itérations des deux boucles est au plus le nombre de triplets utiles, soit $\mathcal{O}(n|\Pi_G|)$.

Afin d'obtenir efficacement les triplets utiles on calcule, pour chaque clique maximale potentielle Ω, les composantes connexes de $G - \Omega$, ensuite leurs voisinages, correspondant aux séparateurs $S \subset \Omega$, et pour chaque séparateur de ce type on identifie l'unique 1-bloc (S, C) contenant Ω. On ajoute également un pointeur de (S, C) vers Ω. Le détail de l'implantation ainsi que les structures de données sont donnés dans [59]. On obtient ainsi la complexité désirée de $\mathcal{O}(n^3|\Pi_G|)$.

L'algorithme peut être transformé pour calculer non seulement la largeur arborescente du graphe en entrée, mais également une triangulation réalisant cet optimum. $\diamond$

2.5.3 Largeur de branches, triangulations efficaces et blocs

Dans sa thèse [102], F. Mazoit a modifié et adapté ces outils qui se sont avérés puissants pour le calcul de la largeur arborescente – à savoir les triangulations minimales et cliques maximales potentielles – au calcul de la largeur de branches. Précisons que cette adaptation est loin d'être triviale. Basé sur cet approche, nous donnerons dans le chapitre 3, section 3.3 un algorithme exponentiel de calcul de la largeur de branches. Dans un travail indépendant, C. Paul et J.A. Telle [111] ont initié une recherche allant vers la compréhension algorithmique de la largeur de branches et ont obtenu un certain nombre de résultats structurels similaires à ceux de Mazoit. En particulier, leur notion de k-troïka [111] est similaire aux résultats sur le calcul de la bloc-largeur de branches.

Une décomposition en branches $(\mathcal{T}, \tau)$ d'un graphe $G = (V, E)$ engendre elle aussi une triangulation de G. En effet, associons à chaque sommet x de G le sous-arbre T_x de l'arbre de décomposition $\mathcal{T}$ qui recouvre toutes les feuilles de $\mathcal{T}$ contenant une arête incidente à x, et qui est minimal pour cette propriété. Le graphe d'intersection de la famille d'arbres $\{T_x\}_{x \in V}$ est un graphe triangulé noté $H(\mathcal{T}, \tau)$. Clairement $H(\mathcal{T}, \tau)$ est une triangulation de G.

Remarquons que pour chaque branche e de l'arbre $\mathcal{T}$, son étiquette $\mathrm{lab}(e)$ induit une clique (pas forcément maximale) de $H(\mathcal{T},\tau)$. La décomposition en branches de G peut être étendue de façon naturelle en une décomposition en branches de $H(\mathcal{T},\tau)$:

Lemme 2.36 ([102, 60]) *Soit $(\mathcal{T},\tau)$ une décomposition en branches de G. Il existe une décomposition en branches $(\mathcal{T}',\tau')$ de $H(\mathcal{T},\tau)$ telle que :*

1. *$\mathcal{T}$ est un sous-arbre de $\mathcal{T}'$.*
2. *Pour chaque branche de $\mathcal{T}$, son étiquette dans $(\mathcal{T}',\tau')$ est la même que dans $(\mathcal{T},\tau)$.*
3. *La largeur des deux décompositions $(\mathcal{T}',\tau')$ et $(\mathcal{T},\tau)$ est la même.*

En particulier, si $(\mathcal{T},\tau)$ est une décomposition en branches optimales de G alors $\mathrm{bw}(G) = \mathrm{bw}(H(T,\tau))$.

L'une des propriétés fortement exploitées pour le calcul de la largeur arborescente venait du fait que, pour tout graphe G, il existe une triangulation *minimale* H de G telle que $\mathrm{tw}(G) = \mathrm{tw}(H)$. On pourrait espérer la même chose pour la largeur de branches ; la proposition suivante nous montre que, pour la largeur de branches, la situation n'est pas aussi simple.

Proposition 2.37 *Soit K_9^- le graphe obtenu à partir du graphe complet à 9 sommets en supprimant une seule arête ab. Quelle que soit une décomposition en branches $(\mathcal{T},\tau)$ optimale pour K_9^-, le graphe $H(T,\tau)$ n'ent pas une triangulation minimale de K_9^-.*

Preuve. Nous montrons que les sommets a et b sont adjacents dans $H(T,\tau)$. Comme K_9^- est triangulé, ceci implique le fait que la triangulation $H(\mathcal{T},\tau)$ n'est pas minimale.

Rappelons que la largeur de branches d'un graphe à n sommets est au plus $\lceil 2n/3 \rceil$ ([119]), donc $\mathrm{bw}(K_9^-) \leq 6$. Soit $(\mathcal{T},\tau)$ une décomposition en branches optimale pour K_9^-. Supposons que les sommets a et b sont non adjacents dans $H(T,\tau)$. Il existe alors une branche e de $\mathcal{T}$ séparant, dans $\mathcal{T}$, les sous-arbres T_a et T_b. Tout chemin reliant a et b dans K_9^- intersecte l'ensemble de sommets $\mathrm{lab}(e)$. Puisque $\mathrm{lab}(e)$ est un a,b-séparateur, $\mathrm{lab}(e)$ contient au moins 7 sommets, contredisant le fait que $\mathrm{bw}(K_9^-) \leq 6$. $\diamond$

Puisque les décompositions en branches optimales ne correspondent pas forcément à des triangulations minimales, nous perdons beaucoup d'outils parmi ceux dont nous disposions pour le calcul de la largeur arborescente. Une deuxième grande différence entre les deux paramètres provient du fait que le calcul de la largeur de branches reste NP-difficile même pour une sous-classe très restreinte des graphes triangulés, à savoir les graphes split [88] (alors que la largeur arborescente est calculable en temps linéaire pour tous les graphes triangulés). Cependant, Mazoit prouve que, pour tout graphe G, il existe une décomposition en branches optimale $(\mathcal{T},\tau)$ telle que la triangulation associée $(\mathcal{T},\tau)$ soit *efficace*. Les triangulations efficaces, définies ci-dessous, ont un comportement relativement similaire aux triangulations minimales.

Définition 2.38 *Une triangulation H de G est* efficace *si*

1. *chaque séparateur minimal de H est également un séparateur minimal de G ;*
2. *pour tout séparateur minimal S de H, les composantes connexes de $H - S$ sont aussi les composantes connexes de $G - S$.*

Les triangulations efficaces ont été en fait introduites dans [25] (et appelées, à l'époque... " triangulations minimales"). Toutes les triangulations minimales de G sont des triangulations efficaces, par le théorème 2.22.

Définition 2.39 *Une décomposition en branches $(\mathcal{T}, \tau)$ de $G = (V, E)$* respecte *un ensemble de sommets $S \subseteq V$ s'il existe une branche e de $\mathcal{T}$ telle que$S \subseteq \mathrm{lab}(e)$.*

Théorème 2.40 ([102, 103]) *Il existe une décomposition en branches optimale $(\mathcal{T}, \tau)$ de G telle que le graphe triangulé $H(\mathcal{T}, \tau)$ soit une triangulation efficace de G. De plus, $(\mathcal{T}, \tau)$ respecte chaque séparateur minimal de H.*

Les cliques maximales d'une triangulation efficace sont des blocs du graphe d'origine. Informellement, le rôle que jouaient les cliques maximales potentielles lors du calcul de la largeur arborescente sera repris ici par les blocs.

Lemme 2.41 ([102]) *Soit H une triangulation efficace de G. Toute clique maximale Ω de H est un bloc de G. Réciproquement, pour tout bloc B de G, il existe une triangulation efficace $H(B)$ de G telle que B soit une clique maximale de $H(B)$.*

Rappelons que la largeur arborescente du graphe G peut être exprimée par cette équation :

$$\mathrm{tw}(G) = \min_{H \text{ triangulation minimale de } G} \max\{|\Omega| - 1 \mid \Omega \text{ clique maximale de } H\}.$$

Nous cherchons une formule similaire pour la largeur de branches.

Définition 2.42 (bloc-largeur de branches) *Soit B un bloc de G et $K(B)$ le gaphe complet ayant B comme ensemble de sommets. Une décomposition en branches de $K(B)$ qui respecte chaque séparateur minimal $S \in \mathcal{S}(B)$ est appelée* bloc-décomposition en branches *du bloc B. La* bloc-largeur de branches $\mathrm{bbw}(B)$ *de B est la largeur minimum parmi toutes les bloc-décompositions en branches de G.*

De façon équivalente on peut définir la bloc-largeur de branches $\mathrm{bbw}(B)$ comme la largeur de branches de l'hypergraphe obtenu à partir du graphe complet $K(B)$ en rajoutant une hyperarête S pour chaque séparateur minimal S bordant B. La bloc-largeur de branches nous fournit la possibilité d'adapter à la largeur de branches l'équation "classique" pour la largeur arborescente.

Théorème 2.43 ([102])

$$\mathrm{bw}(G) = \min_{H \textit{ triangulation efficace de } G} \max\{\mathrm{bbw}(\Omega) \mid \Omega \textit{ clique maximale de } H\}.$$

Cette équation permet de transcrire l'algorithme `LargeurArborescenteCMP` (cf. théorème 2.35 et figure 2.8) en remplaçant "cliques maximales potentielles" par "blocs" et la taille des cliques maximales potentielles par la bloc-largeur de branches des blocs.

Théorème 2.44 ([60]) *Etant donné un graphe G, la liste $\mathcal{B}_G$ de tous ses blocs ainsi que la bloc-largeur de branches de chaque bloc, la largeur de branches de G peut être calculée en temps $\mathcal{O}(n^3|\mathcal{B}_G|)$.*

Chapitre 3

Cliques maximales potentielles : énumération et applications

Les cliques maximales potentielles ont été introduites afin de calculer et temps polynomial la largeur arborescente pour certaines classes de graphes. Nous avons vu dans le chapitre 2 qu'elles suffisent pour calculer ce paramètre, mais encore faut-il trouver un moyen de les énumérer. Ce sera l'objectif de la première section de ce chapitre, où nous verrons que le calcul des cliques maximales potentielles peut se faire en temps polynomial par rapport au nombre de séparateurs minimaux du graphe en entrée [32]. Ceci prouve par l'affirmative une conjecture de Kloks et al. [87, 86] : pour toute classe de graphes avec une quantité polynomiale de séparateurs minimaux, le calcul de la largeur arborescente se fait en temps polynomial.

Nous présenterons ensuite un algorithme exact de calcul de la largeur arborescente pour les graphes quelconques, basé sur le comptage et l'énumération des cliques maximales potentielles [59]. Comme on peut s'y attendre, l'algorithme est de complexité exponentielle. Sa complexité est de type $\mathcal{O}(\mathrm{Poly}(n) \cdot c^n)$, où c est une constante strictement inférieure à 2. Il s'agit du premier algorithme de ce type pour la largeur arborescente. Nous verrons également une approche similaire pour la largeur de branches [60].

Enfin, la caractérisation des cliques maximales potentielles des graphes planaires nous permet de donner (cf. [30]) une preuve simple d'une conjecture de Robertson et Seymour [117] : pour tout graphe planaire, la largeur arborescente du graphe et celle de son dual diffère d'au plus une unité.

Le chapitre se termine par une brève discussion sur l'utilisation des cliques maximales potentielles, ainsi que des problèmes ouverts autour de ces objets.

Ces résultats ont été obtenus en collaboration avec V. Bouchitté, F. Fomin, F. Mazoit et D. Kratsch.

3.1 Enumérations des cliques maximales potentielles

Rappelons que, si Ω est une clique maximale du graphe G, alors Ω est une clique du graphe $G_{\mathcal{S}(\Omega)}$, obtenu à partir de G en complétant les séparateurs minimaux qui bordent Ω (cf. théorème 2.29). Soit maintenant S l'un de ces séparateurs. A priori, si l'on complète seulement les séparateurs de $\mathcal{S}(\Omega)\backslash\{S\}$, on s'attend à ce que Ω ne soit pas une clique dans le nouveau graphe. Dans ce cas, nous dirons que Ω est une clique maximale potentielle active. Nous verrons que les cliques maximales potentielles actives ont une structure très simple, qui nous permettra de les énumérer efficacement. L'énumération sera ensuite étendue à toutes les cliques maximales potentielles du graphe en entrée.

Définition 3.1 *Soit Ω une clique maximale potentielle d'un graphe G et soit $S \subset \Omega$ un séparateur minimal de G. Si Ω n'est pas une clique dans le graphe $G_{\mathcal{S}(\Omega)\backslash\{S\}}$, obtenu à partir de G en complétant tous les séparateurs minimaux contenus dans Ω sauf S, nous dirons que S est un* séparateur actif *pour Ω. Une paire de sommets $x, y \in S$ non adjacents dans $G_{\mathcal{S}(\Omega)\backslash\{S\}}$ est appelée* paire active.

Une clique maximale potentielle active *est une clique maximale potentielle possédant au moins un séparateur actif.*

Théorème 3.2 ([32]) *Soit Ω une clique maximale potentielle active de G et $S \subset \Omega$ un séparateur minimal, actif par rapport à Ω. Soit (S, C) le 1-bloc associé à S contenant Ω et $x, y \in S$ une paire active. Alors $\Omega\backslash S$ est un x, y-séparateur minimal dans $G[C \cup \{x, y\}]$.*

Preuve. Remarquons que les sommets x et y, non adjacents dans $G_{\mathcal{S}(\Omega)\backslash\{S\}}$, existent par définition d'un séparateur actif. Puisque $G_{\Delta_G(\Omega)}$ nous avons $x, y \in S$.

Montrons d'abord que $\Omega\backslash S$ est un x, y-séparateur dans le graphe $G' = G[C \cup \{x, y\}]$. Supposons que x et y sont dans une même composante C_{xy} de $G' - (\Omega\backslash S)$. Il existe alors une composante D de $G - \Omega$, incluse dans C_{xy}, telle que son voisinage $T = N_G(D)$ contienne les deux sommets x et y. Par le théorème 2.29, T est un séparateur minimal de G, inclus dans Ω. Observons que $T \neq S$, sinon S séparerait D et Ω, contredisant le fait que $D \subset C$. Nous avons prouvé que T est un séparateur minimal contenu dans Ω, différent de S et contenant les deux sommets x et y. Ceci contredit le fait que x et y sont non adjacents dans $G_{\mathcal{S}(\Omega)\backslash\{S\}}$. Nous en déduisons que $\Omega\backslash S$ est un x, y-séparateur de G'.

Prouvons que $\Omega\backslash S$ est un x, y-séparateur minimal de G'. Montrons que, pour tout $z \in \Omega\backslash S$, il existe une chaîne μ de x à y dans G' qui intersecte $\Omega\backslash S$ uniquement dans z. Par le théorème 2.29, x et z sont adjacents dans $G_{\mathcal{S}(\Omega)}$, donc ils sont adjacents dans G ou reliés à travers une composante connexe C_i de $G - \Omega$. Dans le dernier cas, $C_i \subset C$, sinon C_i est également une composante de $G - S$, contredisant le fait que $z \notin S$. Dans les deux cas nous avons une chaîne μ' de x à z dans G', qui intersecte $\Omega\backslash S$ uniquement dans z. Pour les mêmes raisons, il existe une chaîne μ'' de z à y dans G' avec la même propriété. Les deux chaînes relient x et y, en intersectant $\Omega\backslash S$ uniquement dans z. Nous concluons que $\Omega\backslash S$ est un x, y-séparateur minimal de G'. $\diamond$

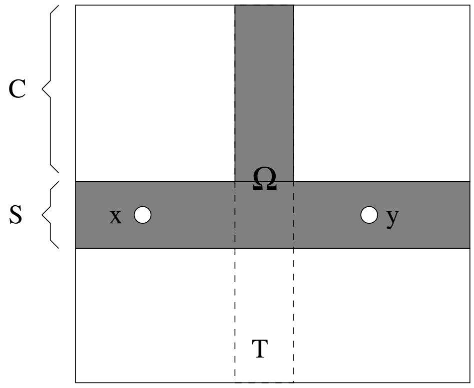

FIG. 3.1 – Clique maximale potentielle active.

Il n'est pas difficile de montrer que, dans ce cas, il existe un séparateur minimal T de G, croisant S et tel que $T \cap C = \Omega \backslash S$. Nous obtenons le théorème suivant qui montre que toute clique maximale potentielle active est identifiée par deux séparateurs minimaux du graphe.

Théorème 3.3 ([32]) *Soit Ω une clique maximale potentielle active, S un séparateur actif par rapport à Ω et (S, C) le 1-bloc associé à S contenant Ω. Il existe un séparateur minimal T de G, croisant S, tel que $\Omega = S \cup (T \cap C)$.*

On en déduit tout de suite que le nombre de cliques maximales potentielles actives est au plus $n|\Delta_G|^2$, le facteur n venant du fait que pour un séparateur S il y a au plus n composantes connexes de $G - S$. En utilisant l'algorithme de reconnaissance d'une clique maximale potentielle (corollaire 2.30), on peut énumérer toutes les cliques maximales potentielles actives en temps $\mathcal{O}(n^2 m|\Delta_G|^2)$.

Malheureusement toutes les cliques maximales potentielles d'un graphe ne sont pas forcément actives. Un exemple est donné dans la figure 3.2 (voir aussi [32]). Pour pouvoir énumérer toutes les cliques maximales potentielles d'un graphe, nous utiliserons une approche incrémentale. Soit $(v_1, v_2, \ldots, v_n)$ un ordre total quelconque sur les sommets de G et notons $G_i = G[\{v_1, \ldots v_i\}]$. Nous calculerons les cliques maximales potentielles de G_i en utilisant les cliques maximales potentielles de G_{i-1}, grâce au théorème suivant.

Théorème 3.4 ([32]) *Soit a un sommet de G et notons $G' = G - a$. Pour toute clique maximale potentielle Ω de G, nous avons l'une des propriétés suivantes :*

1. *soit Ω, soit $\Omega \backslash \{a\}$ est une clique maximale potentielle de G'.*
2. *$\Omega = S \cup \{a\}$, où S est un séparateur minimal de G.*
3. *Ω est active.*

Théorème 3.5 *Pour tout graphe G, le nombre de ses cliques maximales potentielles est au plus $\mathcal{O}(n^2|\Delta_G|^2)$. De plus, il existe un algorithme énumérant les cliques maximales potentielles an temps $\mathcal{O}(n^3 m|\Delta_G|^2)$.*

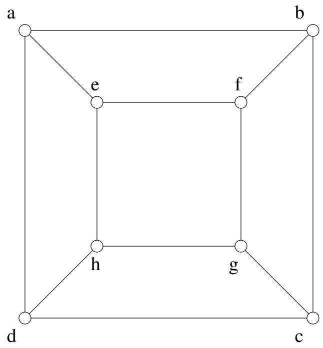

FIG. 3.2 – Clique maximale potentielle inactive : $\{a, c, h, f\}$.

Preuve. (éléments.) Considérons les graphes G_i définis comme ci-dessus. Une propriété (facile) que nous ne démontrons pas ici est que cette suite de graphes a un nombre croissant de séparateurs minimaux : $|\Delta_{G_{i-1}}| \leq |\Delta_{G_i}|$, pour tout i, $2 \leq i \leq n$. En particulier chacun de ces graphes a au plus $|\Delta_G|$ séparateurs minimaux.

Le graphe G_1 a une unique clique maximale potentielle, à savoir $\{v_1\}$. Supposons que nous ayons calculé toutes les cliques maximales potentielles de G_{i-1}, nous voulons calculer celles de G_i en utilisant le théorème 3.4. Pour chaque clique maximale potentielle Ω de G_{i-1}, nous vérifions (grâce au corollaire 2.30) si Ω ou $\Omega \cup \{v_i\}$ est une clique maximale potentielle de G_i, et si tel est le cas nous l'ajoutons au Π_{G_i}. Notons qu'au plus l'un des deux ensembles est une clique maximale potentielle de G_i, car un graphe ne peut avoir deux cliques maximales potentielles distinctes, comparables par inclusion. Nous calculons ensuite les cliques maximales potentielles de G_i de la forme $S \cup \{v_i\}$ (il y en a au plus au plus $n|\Delta_G|$) et enfin les cliques maximales potentielles actives, en utilisant le théorème 3.3 (il y en a au plus $n|\Delta_G|^2$). On obtient que $|\Pi_{G_i}| \leq |\Pi_{G_{i-1}}| + n|\Delta_G| + n|\Delta_G|^2$, en particulier $|\Pi_G| \leq n^2|\Delta_G|^2 + n^2|\Delta_G| + 1$. $\diamond$

Rappelons qu'il existe un algorithme d'énumération des séparateurs minimaux d'un graphe quelconque, en temps polynomial par séparateur. Une fois calculé Δ_G, on peut calculer l'ensemble de cliques maximales potentielles du graphe en entrée par le théorème 3.5. Enfin, le calcul de la largeur arborescente peut se faire en temps $\mathcal{O}(nm|\Pi_G|)$, par le théorème 2.35. On en conclut :

Théorème 3.6 *Soit $\mathcal{G}$ une classe de graphes et $\mathrm{Poly}_{\mathcal{G}}$ un polynôme. Supposons que tout graphe $G \in \mathcal{G}$ a au plus $\mathrm{Poly}_{\mathcal{G}}(n)$ séparateurs minimaux, où n est le nombre de sommets de G.*

Pour la classe $\mathcal{G}$, il existe un algorithme polynomial de calcul de la largeur arborescente.

Ce résultat unifie plusieurs algorithmes calculant la largeur arborescente pour diverses classes de graphes, comme les graphes de cordes [85, 93], les graphes d'intervalles circulaire [128, 93], les graphes faiblement triangulés [31], etc [25, 35, 34].

3.2 Calcul exact de la largeur arborescente

Les algorithmes exacts (en temps exponentiel) pour la résolution des problèmes NP-difficiles s'affirment comme une technique novatrice dans l'approche de ce type de questions. Bien que les premiers algorithmes non triviaux de ce type remontent à la fin des années '60, nous assistons depuis peu l'apparition de nombreux résultats de ce type, avec de nouvelles techniques et des améliorations spectaculaires. On peut citer comme "origine" de ces algorithmes l'approche de Held et Karp [79] pour résoudre le problème du voyageur de commerce en temps $\mathcal{O}^*(2^n)$ (la notation $\mathcal{O}^*$ ignore les facteurs polynomiaux en n, dont la croissance est dominée par la fonction exponentielle). Remarquons les grands progrès sur le problème du stable maximum, avec une complexité actuelle de $\mathcal{O}^*(1.221^n)$ pour les algorithmes utilisant un espace polynomial [58]. Voir également [57, 135] pour un état de l'art sur les algorithmes exponentiels.

Il y a plusieurs façons de calculer la largeur arborescente en temps $\mathcal{O}^*(2^n)$, par exemple en énumérant brutalement toutes les cliques maximales potentielles du graphe en entrée. Le même résultat peut être obtenu à l'aide de l'algorithme de Arnborg et al. [4], qui teste si un graphe est de largeur arborescente au plus k en utilisant tous les sacs possibles de taille au plus k. Ici, nous présentons un algorithme de complexité $\mathcal{O}^*(1.9601^n)$ pour la largeur arborescente (voir [59] pour les détails). Il a été amélioré par Villanger [133] à une complexité $\mathcal{O}^*(1.89^n)$.

Nous souhaitons donner une borne supérieure au nombre de cliques maximales potentielles d'un graphe d'ordre n – meilleure que la borne triviale de 2^n.

Pour illustrer la technique, comptons d'abord les séparateurs minimaux d'un graphe à n sommets. Pour tout séparateur minimal S il existe deux composantes connexes C et D de $G - S$ pleines par rapport à S. Ces composantes satisfont $N(C) = N(D) = S$. Les ensembles de sommets S, C et D étant disjoints, au moins l'un des trois est de taille au plus $n/3$. Ainsi, pour tout séparateur minimal S il existe un ensemble de sommets X, de taille au plus $n/3$, tel que $S = X$ ou $S = N(X)$. Il s'ensuit qu'un graphe d'ordre n possède au plus $2\binom{n}{n/3} = \mathcal{O}(1.89^n)$ séparateurs minimaux.

Par une technique non triviale, nous avons donné dans [59] le résultat suivant :

Théorème 3.7 *Un graphe à n sommets a au plus 1.73^n séparateurs minimaux.*

Nous allons montrer qu'un graphe d'ordre n a au plus $\mathcal{O}^*(\binom{n}{2n/5})$ cliques maximales potentielles. Comme pour l'énumération de ces objets, nous commençons par compter les cliques maximales potentielles actives. Grosso modo, l'idée est de montrer que chaque clique maximale potentielle peut être identifiée par un ensemble de sommets du graphe de taille au plus $2n/5$. L'algorithme calculant l'ensemble Π_G énumére tous les sous-ensembles de sommets de taille au plus $2n/5$ et, en appliquant une procédure de complexité polynomiale sur chacun de ces ensembles, génère toutes les cliques maximales potentielles.

Avant de rentrer dans les détails, donnons quelques intuitions sur la raison pour laquelle les cliques maximales potentielles peuvent être identifiées par des ensembles de taille au plus $2n/5$. La situation est décrite dans la figure 3.3. Considérons une clique maximale

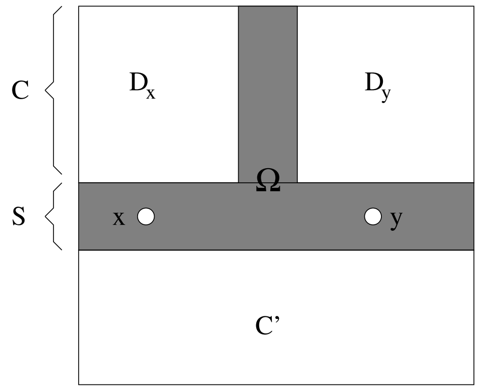

FIG. 3.3 – Représentation des cliques maximales potentielles par des petits ensembles.

potentielle Ω et un séparateur minimal S contenu dans Ω. Soit C l'unique composante de $G - S$ intersectant Ω et soit C' une composante pleine associée à S, différente de C. Nous prouvons dans le lemme 3.8 que (excepté quelques cas particuliers)

$$\Omega = N(C \backslash \Omega). \tag{3.1}$$

Considérons ensuite le cas où Ω est une clique maximale potentielle active, avec une paire active $x, y \in S$. Par le théorème 3.2, $\Omega \backslash S$ est un x, y-séparateur minimal de $G[C \cup \{x, y\}]$. En utilisant deux composantes pleines associées à $\Omega \backslash S$ dans $G[C \cup \{x, y\}]$, nous montrons dans le lemme 3.12 que $C \backslash \Omega$ contient deux sous-ensembles disjoints D_x et D_y, dont le voisinage contient $\Omega \backslash S$. Comme $S = N(C')$, nous avons :

$$\Omega = N(C' \cup D_x) = N(C' \cup D_y). \tag{3.2}$$

(En réalité la situation est plus compliquée, cf. lemme 3.12.) Par les équations 3.1 et 3.2, chacun des ensembles Ω, $D_x \cup D_y$, $C' \cup D_x$ et $C' \cup D_y$ est suffisant pour identifier Ω. Il suffit maintenant de constater qu'au moins l'un de ces ensemble est de taille au plus $2n/5$. C'est pourquoi le nombre de cliques maximales potentielles actives est au plus $\mathcal{O}^*(\binom{n}{2n/5})$. Le cas des cliques maximales potentielles inactives est traité séparément, un peu comme lors de l'énumération de ces objets.

Revenons aux résultats plus formels sur l'énumération des cliques maximales potentielles actives. La propriété suivante et le théorème 3.2 seront à la base de notre approche.

Lemme 3.8 *Soient Ω une clique maximale potentielle de G et S un séparateur minimal contenu dans Ω. Notons C l'unique composante connexe de $G - S$ qui intersecte Ω. Nous avons l'une des trois propriétés suivantes*

1. *$\Omega = N(C \backslash \Omega)$;*
2. *il y a un sommet $a \in \Omega \backslash S$ tel que $\Omega = N[a]$;*
3. *il y a un sommet $a \in S$ tel que $\Omega = S \cup (N(a) \cap C)$.*

Si nous sommes dans le second cas, le sommet a suffit pour identifier Ω ; il y a au plus n cliques maximales potentielles de ce type. Dans le dernier cas, la clique maximale potentielle Ω est identifiable par le sommet a, le séparateur S et un sommet $b \in C$. Nous dirons que le triplet (S, a, b) est une *représentation par séparateur* de Ω :

Définition 3.9 *Soit Ω une clique maximale potentielle de G. Le triplet (S, a, b) est appelée* représentation par séparateur *de Ω si S est un séparateur minimal de G, $a \in S$, $b \in V \backslash S$ et $\Omega = S \cup (N(a) \cap C_b(S))$, où $C_b(S)$ est la composante de $G - S$ contenant b.*

Le nombre de représentations par séparateurs possibles est au plus $n^2|\Delta_G|$, ce qui donne une borne supérieure pour le nombre de cliques maximales potentielles ayant une telle représentation.

Penchons-nous sur les cliques maximales potentielles qui satisfont la première condition et pour lesquelles S est actif. D'après le théorème 3.2, $\Omega \backslash S$ est un séparateur minimal du graphe $G' = G[C \cup \{x, y\}]$, où $x, y \in S$ forment une paire active. En prenant deux composantes pleines D_x et D_y de $G' - S$, contenant respectivement x et y, nous avons que $N_G(D_x \cup \{x\}) \cap C = N_G(D_y \cup \{y\}) \cap C = \Omega \backslash S$.

Lemme 3.10 *Soient Ω une clique maximale potentielle active, S un séparateur actif contenu dans Ω et $x, y \in S$ une paire active. Notons C la composante de $G - S$ qui intersecte Ω. Il existe deux sous-ensembles disjoints D_x et D_y de $C \backslash \Omega$ tels que $N(D_x \cup \{x\}) \cap C = N(D_y \cup \{y\}) \cap C = \Omega \backslash S$.*

Soit C' une composante pleine de $G - S$, différente de C, nous avons $S = N_G(C')$. Les triplets $(D_x \cup C', x, c)$ et $(D_y \cup C', y, c)$, où c est un sommet de C', forment une *représentation indirecte* de Ω (et suffisent pour calculer Ω) au sens suivant :

Définition 3.11 *Soit Ω une clique maximale potentielle de $G = (V, E)$. Nous dirons que le triplet (X, x, c), avec $X \subset V$, $x \in X$ et $c \notin X$ est une* représentation indirecte *de Ω si $\Omega = N(C_c) \cup N(D_x \cup \{x\}) \backslash C_c$, où*
- *C_c est la composante de $G[X]$ contenant c ;*
- *$D_x = X \backslash C_c$.*

On en déduit (voir [59] pour les détails) :

Lemme 3.12 *Avec les notations du lemme 3.10, pour chaque sommet $c \in C'$, où C' est une composante de $G - S$ pleine par rapport à S et différente de C, les triplets $(D_x \cup C', x, c)$ et $(D_y \cup C', y, c)$ sont des représentations indirectes de Ω.*

Nous avons tous les ingrédients pour le principal outil de comptage et d'énumération des cliques maximales potentielles actives.

Lemme 3.13 *Soit Ω une clique maximale potentielle active de G. Nous avons l'une des propriétés suivantes :*

1. *Il existe un sommet a de G tel que $\Omega = N[a]$;*
2. *Ω a une représentation par séparateur ;*
3. *$|\Omega| \leq 2n/5$;*
4. *$\Omega = N(D)$, pour un ensemble de sommets D de taille au plus $2n/5$;*
5. *Ω a une représentation indirecte (X, x, c) telle que $|X| \leq 2n/5$.*

Preuve. (éléments.) Les deux premières situations correspond aux cas 2 et 3 du lemme 3.8. Si nous sommes dans le premier cas du lemme 3.8, nous avons, avec les notations employées ci-dessus, $\Omega = N(C\backslash\Omega)$. D'après le lemme 3.12, les triplets $(D_x \cup C', x, c)$ et $(D_y \cup C', y, c)$ avec $c \in C'$ sont les représentations indirectes de Ω. Comme D_x, D_y, C' et Ω sont disjoints et que $D_x \cup D_y \subseteq V\backslash(\Omega \cup C')$, il est facile d'observer qu'au moins l'un des ensembles Ω, $C\backslash\Omega$, $D_x \cup C'$ et $D_y \cup C'$ est de taille $\leq 2n/5$. $\diamond$

En énumérant toutes les représentations par séparateurs et tous les voisinages $N[a]$, nous calculons les cliques maximales potentielles actives correspondant aus deux premiers points du lemme 3.13. Nous énumérons également tous les ensembles de sommets X, taille au plus $2n/5$, et nous calculons les cliques maximales potentielles de la forme X, $N(X)$, ou ayant (X, x, c) comme représentation indirecte, pour tout couple de sommets x, c. Nous en déduisons que le nombre total de cliques maximales potentielles actives de G est borné par n plus le nombre de représentations par séparateurs plus $\mathcal{O}^*(\binom{n}{2n/5})$, et ce dernier facteur domine les autres. De plus leur énumération se fait avec la même complexité en temps. Par une technique similaire au théorème 3.5, nous pouvons énumérer toutes les cliques maximales potentielles de G (et pas seulement les actives) sans changer brutalement la complexité. Nous obtenons :

Théorème 3.14 *Un graphe G à n sommets a au plus $\mathcal{O}(n^4 \cdot 1.9601^n)$ cliques maximales potentielles. Il y a un algorithme de complexité $\mathcal{O}^*(1.9601^n)$ énumérant toutes les cliques maximales potentielles du graphe en entrée.*

En utilisant l'algorithme de calcul de la largeur arborescente à partir des cliques maximales potentielles (théorème 2.35) on en déduit :

Théorème 3.15 *Il existe un algorithme de complexité $\mathcal{O}^*(1.9601^n)$ qui prend en entrée un graphe G quelconque et calcule la largeur arborescente de G.*

Avec le même type d'outils mais en affinant les représentations des cliques maximales potentielles par des petits ensembles, Villanger [133] améliore le temps d'énumération des cliques maximales potentielles à $\mathcal{O}^*(\binom{n}{n/3}) = \mathcal{O}^*(1.89^n)$. On obtient un algorithme d'énumération de la largeur arborescente avec la même complexité. Citons également le tout récent travail de Bodlaender et al. [21] sur le calcul exact de la largeur arborescente, en utilisant un espace polynomial : ils obtiennent un algorithme de complexité $\mathcal{O}^*(2.96^n)$.

3.3 Calcul exact de la largeur de branches

Afin d'obtenir un résultat similaire pour la largeur de branches en utilisant le théorème 2.44, nous avons besoin d'énumérer tous les blocs du graphe en entrée et de calculer, pour chaque bloc B, sa bloc-largeur de branches $\mathrm{bbw}(B)$ (cf. théorème 2.43).

Clairement un graphe d'ordre n a au plus 2^n blocs. Ils sont facilement énumérables en temps $\mathcal{O}^*(2^n)$, en énumérant chaque sous-ensemble de sommets et en testant s'il forme un bloc. Hélas la borne 2^n ne peut être amélioré de façon significative, comme nous verrons un peu plus tard.

La deuxième difficulté pour le calcul de la largeur de branches provient du fait que même le calcul de la bloc-largeur de branches est NP-difficile. Ce résultat peut être déduit directement à partir de la preuve de [88], qui montre que le calcul de la largeur de branches est NP-difficile pour les graphes split.

Soit $n(B)$ le nombre de sommets du bloc B et $s(B)$ le nombre de séparateurs minimaux qui bordent B. Observons que $s(B)$ est au plus égal au nombre de composantes connexes de $G - B$, en particulier $n(B) + s(B) \leq n$.

Le lemme suivant est dû à Mazoit [102], cependant il faut mentionner qu'un résultat très similaire a été formulé indépendamment par Paul et Telle [111]. Ces auteurs appellent leur décomposition une *troïka*. Une troïka correspond, par rapport aux ensembles de sommets ci-dessous, au triplet $(B\backslash A_1, B\backslash A_2, B\backslash A_3)$.

Lemme 3.16 *Pour tout bloc B du graphe G, nous avons* $\mathrm{bbw}(B) \leq p$ *si et seulement s'il existe une partition de B en quatre ensembles A_1, A_2, A_3, D tels que*

1. *pour tout $i \in \{1,2,3\}$, $|B\backslash A_i| \leq p$;*
2. *Chaque séparateur minimal $S \in \mathcal{S}(B)$ bordant B est contenu dans $B\backslash A_i$ pour au moins une valeur $i \in \{1,2,3\}$.*

Preuve. (éléments.) Supposons qu'il existe une partition de B comme dans le lemme. Pour chaque ensemble A_i, on construit un arbre ternaire T_i dont les feuilles sont étiquetées par les arêtes de la clique $K(B)$, n'ayant pas d'extrémité dans A_i (de sorte à ce que toute arête de $K(B) - A_i$ apparaisse sur au moins une feuille). Nous devons relier maintenant ces trois arbres. Pour chacun d'entre eux, on subdivise une branche e_i en lui rajoutant un nouveau nœud v_i. On crée un nouveau nœud u, relié à v_1, v_2, v_3. L'arbre $\mathcal{T}$ ainsi crée est une bloc-décomposition en branches de $G[B]$. En effet, chaque arête de la clique $K(B)$ correspond à au moins une feuille de l'arbre et l'étiquette $\mathrm{lab}(uv_i)$ est exactement $B\backslash A_i$. L'étiquette de chaque branche est contenue dans l'un des ensembles $B\backslash A_i$, pour un $i \in \{1,2,3\}$, ce qui permet de conclure que $\mathrm{bbw}(B) \leq p$.

Pour l'implication réciproque, voir [60]. $\diamond$

On en déduit deux résultats importants pour notre objectif.

Lemme 3.17 ([102]) *Il existe un algorithme calculant la bloc-largeur de branches d'un bloc B en temps $\mathcal{O}^*(3^{s(B)})$.*

Preuve. (éléments.) L'algorithme procède en calculant toutes les 3-partitions de $\mathcal{S}(B)$. Pour chaque 3-partition $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$, on fusionne les séparateurs de chaque $\mathcal{S}_i$ en un seul ensemble $\cup_{S \in \mathcal{S}_i} S$, en obtenant ainsi un nouveau bloc bordé par trois "super-séparateurs". Le lemme 3.16 permet de calculer la bloc-largeur de branches du nouveau bloc. Enfin, on prend la meilleure des solutions parmi toutes les 3-partitions possibles. $\diamond$

Lemme 3.18 ([60]) *Il existe un algorithme calculant la bloc-largeur de branches d'un bloc B en temps $\mathcal{O}^*(3^{n(B)})$.*

Preuve. Nous montrons que pour tout entier p, l'existence d'une partition de B comme dans le lemme 3.16 peut être vérifiée en temps $\mathcal{O}^*(3^{n(B)})$.

Pour ce faire, plutôt que de partitionner B en quatre parties, nous testons toutes les partitions de B en trois parties A_1, X, D, où X correspond à $A_2 \cup A_3$. Si $|B \backslash A_1| \leq p$, nous vérifions en temps polynomial si l'ensemble X peut être partitionné en A_2 etA_3 comme requis. Puisqu'il y a au plus $3^{n(B)}$ trois-partitions de B, ceci conduit à un algorithme de complexité $\mathcal{O}^*(3^{n(B)})$.

Nous dirons que deux sommets $x, y \in X$ sont équivalents s'il existe $z \in A_1$ et un séparateur minimal S bordant B contenant les trois sommets x, y, z. En particulier, $x \sim y$ implique que x et y doivent être tous les deux dans A_2 ou tous les deux dans A_3. Soient $X_1, \ldots, X_q$ les classes d'équivalence de X. L'ensemble X peut être partitionné en A_2 et A_3 comme dans le lemme 3.16 si et seulement si $\{|X_1|, \ldots, |X_q|\}$ peut être partitionné en deux parties ayant chacune une somme au plus égale à $p - |A_1| - |D|$. Considérons maintenant le problème EXACT SUBSET-SUM, dont l'instance est un ensemble d'entiers positifs $I = \{i_1, \ldots, i_q\}$ et un entier t, et le problème consiste à trouver une partie de I dont la somme est égale à t. Bien que NP-difficile en général, le problème EXACT SUBSET-SUM est polynomial lorsque t et les nombres i_j sont polynomialement bornés par n (voir par exemple le chapitre sur les approximations dans le livre de Cormen, Leiserson, Rivest [39]). En prenant $I = \{|X_1|, \ldots, |X_q|\}$ et en essayant toutes les valeurs de t entre 1 et n^2, nous pouvons tester en temps polynomial si l'ensemble de sommets X peut être partitionné comme requis. $\diamond$

Puisque $s(B) + n(B) \leq n$, au moins l'une des deux quantités est inférieure ou égale à $n/2$. Grâce aux lemmes 3.18 et 3.17, nous en déduisons :

Théorème 3.19 *Il existe un algorithme de complexité $\mathcal{O}^*(\sqrt{3}^n)$ calculant la bloc-largeur de branches d'un bloc de G.*

Les théorèmes 2.44 et 3.19 impliquent notre résultat principal sur le calcul exact de la largeur de branches.

Théorème 3.20 *La largeur de branches des graphes à n sommets peut être calculée en temps $\mathcal{O}^*((2 + \sqrt{3})^n)$, en utilisant un espace de taille $\mathcal{O}^*(2^n)$.*

Preuve. L'algorithme calcule tous les sous-ensembles de sommets B de G et teste si B est un bloc. Si tel est le cas, on calcule la bloc-largeur de branches $\mathrm{bbw}(B)$ à l'aide du théorème 3.19. Le nombre de blocs est au plus 2^n et pour chaque bloc nous calculons sa bloc-largeur de branches en temps $\mathcal{O}^*(\sqrt{3}^n)$. Globalement, cette phase coûte $\mathcal{O}^*((2+\sqrt{3})^n)$ en temps et $\mathcal{O}^*(2^n)$ en espace.

Nous utilisons ensuite le théorème 2.44 pour calculer la largeur de branches de G. Cette seconde phase coûte $\mathcal{O}^*(2^n)$ en temps et en espace. $\diamond$

Notre algorithme est basé sur l'énumération des blocs du graphe en entrée (en temps $\mathcal{O}^*(2^n)$) et sur le calcul de la bloc-largeur de branches (en temps $\mathcal{O}^*(\sqrt{3}^n)$ pour chaque bloc). Il est naturel de se demander si au moins l'une de ces étapes peut être améliorée.

Le calcul de la bloc-largeur de branches est le même problème que le calcul de la largeur de branches d'un hypergraphe avec n' sommets et s' arêtes de cardinalité trois ou plus (n' et s' correspondent à $n(B)$ et $s(B)$). Peut-on faire mieux que notre algorithme qui fonctionne en temps $O(\min(3^{n'}, 3^{s'}))$?

Remarquons que la borne supérieure triviale sur le nombre de blocs, 2^n, ne peut être améliorée de façon significative. Prenons l'union disjointe d'une clique K et d'un stable I, chacun ayant $n/2$ sommets, et ajoutons un couplage parfait entre K et I. Nous obtenons un graphe G_n avec la propriété que, pour tout $I' \subseteq I$, les sommets de $V(G_n) - I'$ forment un bloc. Ainsi G_n a au moins $\binom{n}{n/2} \geq 2^n/n$ blocs. La bonne question est de savoir si l'on peut définir une nouvelle classe de triangulations, plus restreintes que les triangulations efficaces, qui contienne le graphe $H(\mathcal{T}, \tau)$ pour au moins une décomposition en branches optimale du graphe en entrée.

3.4 Largeur arborescente, graphes planaires et dualité

Roberston et Seymour ont conjecturé que la largeur arborescente d'un graphe planaire et celle de son dual diffèrent d'au plus une unité. Lapoire [98] donne une preuve de cette conjecture, en utilisant des techniques algébriques. Nous avons donné dans [30] une preuve courte et combinatoire de ce résultat. Notre approche est basée sur une caractérisation simple des cliques maximales potentielles des graphes planaires et la notion de graphe radial, utilisée par Seymour et Thomas pour montrer que, pour tout graphe planaire, la largeur de branches du graphe est égale à celle de son dual.

Le résultat est valable pour tout graphe planaire, cependant pour éviter des détails techniques nous nous restreignons ici aux graphes 3-connexes. Dans un tel graphe G, pour tout séparateur minimal S, $G - S$ est formé d'exactement deux composantes connexes ; en particulier deux séparateurs minimaux distincts sont incomparables par inclusion.

Pour les notions classiques sur les graphes planaires, voir par exemple [47].

Définition 3.21 *Soit $G = (V, E)$ un graphe plan (un graphe planaire avec un dessin fixé). Notons F l'ensemble de faces de ce dessin. Le* graphe radial *$G_R = (V \cup F, E_R)$ a comme ensemble de sommets $V \cup F$. Nous plaçons une arête dans G_R entre toute paire de sommets*

v et f, où $v \in V$ est un sommet originel et $f \in F$ est un sommet face, tels que le sommet v est incident à la face f dans de dessin de G.

Le graphe radial G_R est clairement un graphe planaire. Il fournit un lien de passage de G à G^* :

Proposition 3.22 *Soit G un graphe plan 2-connexe. Le graphe radial G_R de G et le graphe radial G^*_R de son dual G^* sont égaux.*

Les séparateurs minimaux de S peuvent être vus comme des courbes de Jordan correspondant à des cycles de G_R. Rappelons qu'une courbe de Jordan $\tilde{\nu}$ découpe le plan Σ en exactement deux régions connexes. Nous dirons que $\tilde{\nu}$ sépare les régions de $\Sigma - \tilde{\nu}$.

Le résultat suivant apparaît dans une forme légèrement différente dans [53].

Proposition 3.23 (voir aussi [53]) *Soit G un graphe planaire 3-connexe. A chaque séparateur minimal S de G on peut associer un cycle élémentaire ν_S de G_R, tel que*
- *les sommets originels de ν_S sont exactement les éléments de S ;*
- *S sépare deux sommets x et y de G si et seulement si la courbe de Jordan $\tilde{\nu}_S$, dessinée par le cycle ν_S, sépare dans le plan les points correspondant à x et à y.*

Preuve. (éléments.) Soient C et D les deux composantes connexes de $G - S$. Clairement $N_G(C) = N_G(D) = S$. On contracte les sommets de C en un super-sommet x_C, et soit G' le graphe ainsi obtenu. Dans son graphe radial G'_R, les sommets incidents à x_C forment un cycle entourant x_C. Ce même cycle, noté $\nu_S(C)$, dessine dans G_R une courbe de Jordan $\tilde{\nu}_S(C)$ séparant chaque point de C de chaque point de D. Par construction, le cycle satisfait $\nu_S \cap V = N_G(C) = S$. $\diamond$

Nous dirons qu'un cycle ν_S comme dans la proposition 3.23 *représente* le séparateur S.

Nous pouvons associer aux courbes de Jordan une notion très naturelle de parallélisme et de croisement.

Définition 3.24 *Les courbes de Jordan $\tilde{\nu}_1$ et $\tilde{\nu}_2$ se* croisent *si $\tilde{\nu}_1$ intersecte les deux régions de $\Sigma \backslash \tilde{\nu}_2$. Dans le cas contraire, elles sont* parallèles. *Deux cycles élémentaires ν_1 et ν_2 de G_R se* croisent *si les deux courbes correspondantes $\tilde{\nu}_1$ et $\tilde{\nu}_2$ se croisent, sinon ils sont* parallèles.

Les relations de parallélisme entre courbes de Jordan et entre les cycles de G_R sont symétriques. Nous voudrions faire en sorte que, pour tout couple de séparateurs S et T de G, la relation de parallélisme (ou de croisement) entre eux soit la même que pour les courbes de Jordan et les cycles associés. Cependant ceci nous oblige à plus de précautions. Le cycle de la proposition 3.23 n'est pas unique. Le cycle $\nu_S(C)$ construit dans la preuve est de la forme $\nu_S(C) = [v_1, f_1, v_2, f_2, \ldots, v_p, f_p]$, où les f_i sont des sommets face et les v_i sont des sommets originels. Si deux sommets originels consécutifs v_i et v_{i+1} sont adjacents dans G, l'arête respective est incidente à deux faces f et f'. L'une d'entre elle, par exemple f est précisément la face f_i. Nous pouvons tout aussi bien choisir l'autre face, en mettant

$f_i = f'$, et le cycle garde les bonnes propriétés. Associons maintenant à chaque arête e de G une unique face $f(e)$ incidente à f. Nous dirons qu'un cycle $\nu = [v_1, f_1, v_2, f_2, \dots, v_p, f_p]$ de G_R est *bien formé* si, pour tout couple de sommets originels v_i et v_{i+1}, adjacents dans G, on a $f_i = f(v_i v_{i+1})$ (par convention, $v_{p+1} = v_1$). On peut alors prouver les résultats suivants :

Proposition 3.25 ([30]) *Soit G un graphe plan 3-connexe et S un séparateur minimal de G. Il existe un unique cycle bien formé ν_S de G_R qui représente le séparateur S.*

Proposition 3.26 ([30]) *Deux séparateurs minimaux S et T d'un graphe plan 3-connexe G sont parallèles si et seulement si les cycles bien formés correspondants ν_S et ν_T de G_R sont parallèles.*

Partant d'une famille de séparateurs parallèles de G, nous avons obtenu une famille de courbes de Jordan associées à ces séparateurs. Les séparateurs découpent le graphe en blocs, et les courbes découpent le plan en régions. Nous allons établir un lien fort entre les deux.

Soit $\tilde{\nu}$ une courbe de Jordan du plan Σ et R l'une des deux régions de $\Sigma \backslash \tilde{\nu}$. Comme pour la notion de 1-bloc dans les graphes, nous dirons que $(\tilde{\nu}, R) = \tilde{\nu} \cup R$ est une *région 1-bloc* du plan, bordée par $\tilde{\nu}$.

Définition 3.27 *Soit $\tilde{\mathcal{J}}$ un ensemble de courbes de Jordan tel que pour chaque $\tilde{\nu} \in \tilde{\mathcal{J}}$, il existe une région 1-bloc $(\tilde{\nu}, R(\tilde{\nu}))$ contenant toutes les courbes de $\tilde{\mathcal{J}}$. Nous définissons la* région entre *les éléments de $\tilde{\mathcal{J}}$ comme étant*

$$RegionEntre(\tilde{\mathcal{J}}) = \bigcap_{\tilde{\nu} \in \tilde{\mathcal{J}}} (\tilde{\nu}, R(\tilde{\nu}))$$

Nous dirons que la région entre les courbes de $\tilde{\mathcal{J}}$ est bordée *par $\tilde{\mathcal{J}}$.*

Définition 3.28 *Un ensemble de points du plan $BR \subseteq \Sigma$ est une* région bloc *s'il satisfait l'une des propriétés suivantes :*
- *$BR = \Sigma$.*
- *Il existe une courbe de Jordan $\tilde{\nu}$ telle que BR soit une région 1-bloc $(\tilde{\nu}, R)$.*
- *Il existe un ensemble de courbes $\tilde{\mathcal{J}}$ tel que $BR = RegionEntre(\tilde{\mathcal{J}})$.*

Remarquons que, d'après notre définition, les régions bloc sont toujours des ensembles fermés.

Dans la figure 3.4a, nous avons une région bloc (en gris) bordée par quatre courbes de Jordan. La figure 3.4b présente trois courbes $\tilde{\mu}_1, \tilde{\mu}_2$ et $\tilde{\mu}_3$, correspondant à trois chemins ayant les mêmes extrémités et dont les intérieurs sont mutuellement disjoints. Considérer les trois courbes de Jordan $\tilde{\nu}_1 = \tilde{\mu}_2 \cup \tilde{\mu}_3$, $\tilde{\nu}_2 = \tilde{\mu}_1 \cup \tilde{\mu}_3$ et $\tilde{\nu}_3 = \tilde{\mu}_1 \cup \tilde{\mu}_2$. La région entre $\tilde{\nu}_1, \tilde{\nu}_2$ et $\tilde{\nu}_3$ est exactement l'union de ces trois courbes. Nous verrons sous peu que, pour toute clique maximale potentielle Ω d'un graphe plan 3-connexe G, les sommets de Ω se trouvent sur une région bloc de ce type (que certains appellent une θ-structure, pour sa ressemblance graphique avec la lettre θ).

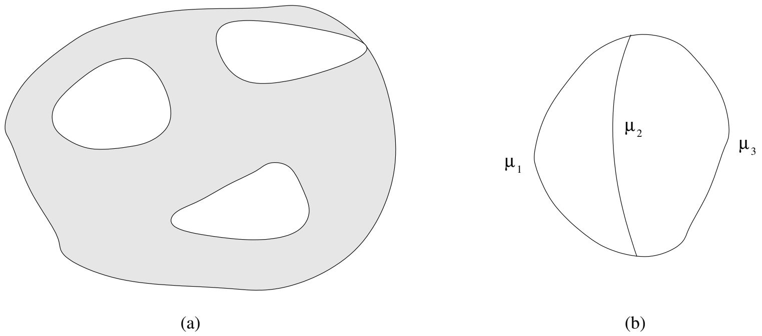

FIG. 3.4 – Régions bloc

Soit maintenant $\tilde{\mathcal{J}}$ un ensemble de courbes de Jordan deux à deux parallèles. Ces courbes découpent le plan en plusieurs régions bloc. Considérons l'ensemble de toutes les régions blocs bordées par des courbes de $\tilde{\mathcal{J}}$. Nous nous intéressons aux éléments minimaux par inclusion de cet ensemble, que nous appelons *régions bloc minimales* induites par $\tilde{\mathcal{J}}$. Le résultat suivant est une conséquence directe de la définition des régions bloc.

Proposition 3.29 *Soit $\tilde{\mathcal{J}}$ un ensemble de courbes de Jordan deux à deux parallèles dans Σ. Un ensemble de points A du plan est contenu dans une même région bloc minimale induite par $\tilde{\mathcal{J}}$ si et seulement si, pour toute courbe $\tilde{\nu} \in \tilde{\mathcal{J}}$, il existe une région 1-bloc $(\tilde{\nu}, R(\tilde{\nu}))$ contenant tout l'ensemble A.*

Soit H une triangulation minimale du graphe plan 3-connexe G. Par théorème 2.22, il existe un ensemble maximal de séparateurs deux à deux parallèles Γ de G tel que $H = G_\Gamma$. Soient $\mathcal{J}(\Gamma) = \{\nu_S | S \in \Gamma\}$ l'ensemble des cycles de G_R associés à ces séparateurs et $\tilde{\mathcal{J}}(\Gamma) = \{\tilde{\nu}_S | S \in \Gamma\}$ l'ensemble des courbes de Jordan correspondantes. Par la proposition 3.26, les cycles de $\mathcal{J}(\Gamma)$ sont deux à deux parallèles. Par conséquent, les courbes de $\tilde{\mathcal{J}}(\Gamma)$ découpent le plan en régions bloc. Nous montrons que chaque clique maximale Ω de H correspond à l'ensemble des sommets originels contenus dans une région bloc minimale formée par $\tilde{\mathcal{J}}(\Gamma)$.

Pour une région bloc BR, notons par BR_G l'ensemble des sommets de G contenus dans BR.

Théorème 3.30 *Soit $H = G_\Gamma$ une triangulation minimale d'un graphe plan 3-connexe $G = (V, E)$. $\Omega \subseteq V$ est une clique maximale de H si et seulement s'il existe une région bloc minimale BR définie par $\tilde{\mathcal{J}}(\Gamma)$ telle que $\Omega = BR_G$.*

Preuve. (éléments.) Si Ω est une clique de H, aucune courbe de $\tilde{\mathcal{J}}(\Gamma)$ ne sépare deux sommets de Ω. Il y aura donc, par la proposition 3.29, une région-bloc minimale BR avec $BR_G = \Omega$. Réciproquement, tout ensemble de sommets BR_G de ce type a la propriété qu'aucune courbe de $\tilde{\mathcal{J}}(\Gamma)$ ne sépare deux sommets de BR_G. On en déduit que BR_G induit une clique dans H. $\diamond$

A partir d'un ensemble $\mathcal{J}$ de cycles deux à deux parallèles de G_R, nous pouvons définir de façon naturelle une triangulation du graphe dual G^*.

Définition 3.31 *Soit $G = (V, E)$ un graphe plan 3-connexe et soit $G^* = (F, E^*)$ son dual. Considérons un ensemble $\mathcal{J}$ de cycles deux à deux parallèles du graphe radial G_R. Nous définissons le graphe $H^*(\mathcal{J}) = (F, E_{H^*})$ ayant F comme ensemble de sommets, en mettant une arête entre deux sommets face f et f' si et seulement si f et f' sont dans une même région bloc minimale définie par $\tilde{\mathcal{J}}$.*

Théorème 3.32 *$H^*(\mathcal{J})$ est une triangulation de G^*. De plus, pour chaque clique maximale Ω^* de $H^*(\mathcal{J})$ il existe une région bloc minimale BR définie par $\tilde{\mathcal{J}}$ telle que Ω^* est exactement l'ensemble des sommets face contenus dans BR.*

Preuve. (éléments.) La deuxième propriété est une conséquence directe de la construction de $H^*(\mathcal{J})$.

Pour montrer que $H^*(\mathcal{J})$ est un graphe triangulé, soit $[f_1, f_2, f_3, \ldots, f_p]$ un cycle de $H^*(\mathcal{J})$. Si le cycle est contenu dans une région bloc minimale, ses sommtes induisent une clique dans $H(\mathcal{J})$. Sinon, au moins deux sommets f_i et f_j de ce cycle sont séparés par une courbe $\tilde{\nu}$ de $\tilde{\mathcal{J}}$. Les sommets f_i et f_j découpent le cycle en deux chemins μ_1 et μ_2, dont l'intérieur est disjoint. Chacun de ces chemins à un sommet face sur la courbe $\tilde{\nu}$. Soient f_p et f_q ces sommets. Ils sont adjacents dans $H^*(\mathcal{J})$ (car aucune autre courbe ne croise $\tilde{\nu}$) et non consécutifs sur le cycle (car séparés par f_i et f_j), le cycle possède donc une corde. $\diamond$

A ce stade on pourrait envisager une preuve de la conjecture de Robertson et Seymour de la manière suivante. A partir d'une triangulation $H = G_\Gamma$ de G, on prend la famille de cycles $\mathcal{J}(\Gamma) = \{\nu_S | S \in \Gamma\}$ du graphe radial et on construit la triangulation $H^*(\mathcal{J}(\Gamma))$ de G^* comme dans la définition 3.31. Malheureusement ceci n'implique pas que $\mathrm{tw}(H^*(\mathcal{J}(\Gamma))) \leq \mathrm{tw}(H)$. Pour aboutir à ce résultat, nous devons ajouter des cycles à l'ensemble $\mathcal{J}(\Gamma)$, jusqu'à l'obtention d'un ensemble maximal de cycles deux à deux parallèles de G_R.

Théorème 3.33 *Soit $G = (V, E)$ un graphe plan 3-connexe et soit $\mathcal{J}$ un ensemble de cycles deux à deux parallèles de G_R, maximal par inclusion pour cette propriété. Considérons une région-bloc minimale BR définie par les courbes de $\tilde{\mathcal{J}}$ et soit BR_{G_R} l'ensembles de sommets de G_R contenus dans BR (BR_{G_R} contient des sommets face et des sommets originels). Nous avons ou bien $BR_{G_R} = \nu$ ou $BR_{G_R} = \nu \cup \mu$, où ν est un cycle de $\mathcal{J}$ et μ est une chaîne de G_R qui touche ν uniquement dans ses extrémités.*

Preuve. (éléments.) On montre (voir [30]) que si BR_{G_R} n'est pas un cycle, il existe une chaîne μ de G_R, contenue dans BR_{G_R} et qui intersecte ν uniquement dans ses extrémités x et y. Ces sommets x et y découpent le cycle ν en deux sous-chaînes ν_1 et ν_2. Nous avons ainsi trois cycles de G_R contenus dans BR_{G_R}, à savoir ν, $\mu\nu_1$ (concaténation de μ et de ν_1) et $\mu\nu_2$ (concaténation de μ et ν_2). Ces cycles sont parallèles à chaque cycle de $\mathcal{J}$, car inclus dans une région-bloc minimale. Par maximalité de $\mathcal{J}$, les trois cycles appartiennent à $\mathcal{J}$.

Observons qu'ils définissent une région bloc BR' qui est exactement $\tilde{\nu} \cup \tilde{\mu}$, comme dans la seconde partie de la figure 3.4. Puisque $BR' = \tilde{\nu} \cup \tilde{\mu} \subseteq BR$ et que BR est une région bloc minimale, nous concluons que $BR' = BR$. $\diamond$

Théorème 3.34 *Soi $G = (V, E)$ un graphe plan 3-connexe.*

$$\mathrm{tw}(G^*) \leq \mathrm{tw}(G) + 1$$

Preuve. (éléments.) Soit H une triangulation minimale de G de largeur minimum et soit $\Gamma = \Delta_H$ l'ensemble des séparateurs minimaux de H. On note $\mathcal{J}(\Gamma)$ l'ensemble de cycles de G_R correspondant à ces séparateurs. Considérons un ensemble maximal $\mathcal{J}$ de cycles deux à deux parallèles de G_R, contenant $\mathcal{J}(\Gamma)$, et le graphe triangulé $H^*(\mathcal{J})$. Montrons que pour toute clique maximale Ω^* de $H^*(\mathcal{J})$, il existe une clique Ω de H avec $|\Omega^*| \leq |\Omega| + 1$; ceci permettra de conclure que $\mathrm{tw}(H^*(\mathcal{J})) \leq \mathrm{tw}(H) + 1 = \mathrm{tw}(G) + 1$.

Par le théorème 3.32, il existe une région bloc minimale BR définie par $\mathcal{J}$ telle que Ω^* soit l'ensemble des sommets face contenus dans BR. Par le théorème 3.33, BR_{G_R} est ou bien un cycle ou alors un cycle plus une chaîne de G_R, avec les extrémités sur le cycle. Dans les deux cas, le graphe G_R étant biparti, le nombre de sommets face de BR_{G_R} est au plus égal au nombre de ses sommets originels, plus un. Mais l'ensemble des sommets originels de BR_{G_R} est également contenu dans une région bloc minimale définie par $\mathcal{J}(\Gamma) \subset \mathcal{J}$. Cet ensemble forme une clique Ω de H, d'après le théorème 3.30. $\diamond$

Avec un petit travail technique supplémentaire, le théorème précédent s'étend aux graphes planaires quelconques. On en déduit :

Corollaire 3.35 *Pour tout graphe planaire G,*

$$\mathrm{tw}(G) \leq \mathrm{tw}(G^*) \leq \mathrm{tw}(G) + 1.$$

3.5 Conclusion et problèmes ouverts

Les cliques maximales potentielles sont un outil important pour la compréhension et le calcul de la largeur arborescente. Nous avons montré qu'elles permettent de calculer la largeur arborescente en temps polynomial par rapport au nombre de séparateurs minimaux du graphe en entrée. Elles nous ont également servi pour le calcul exact de ce paramètre et ont permis de faire le lien entre la largeur arborescente d'un graphe planaire et celle de son dual. Remarquons que les résultats sur le calcul de la largeur arborescente présentés dans cette section, à l'exception de celui sur les graphes planaires, s'étendent au calcul du *remplissage minimum* du graphe en entrée. Le remplissage minimum d'un graphe G est le nombre minimum d'arêtes qu'il faut ajouter pour obtenir une triangulation de G. D'autres chercheurs ont étendu ces techniques au calcul de la largeur arborescente des graphes dont la décomposition modulaire présente de "bonnes" propriétés [27] (par exemple lorsque le nombre de séparateurs minimaux de tout graphe premier est polynomialement borné), ou au calcul d'autres paramètres lié aux décompositions arborescentes [20].

Récemment, Heggernes et al. [78] ont donné un algorithme calculant une triangulation minimale d'un graphe quelconque en temps $\mathcal{O}(n^\alpha \log n)$, où $\mathcal{O}(n^\alpha) \in \mathcal{O}(n^{2.37})$ est le temps nécessaire pour multiplier deux matrices de taille $n \times n$. Pendant près de 30 ans, la meilleure complexité pour ce problème est restée à $\mathcal{O}(nm)$ ([121]), donc $\mathcal{O}(n^3)$ pour les graphes denses. L'algorithme de Heggernes et al. procède en découpant le graphe en blocs, à l'aide de séparateurs calculés très rapidement. L'un des points importants pour sa complexité est qu'à certains moments l'algorithme "sait" que le bloc trouvé est une clique maximale potentielle, ce qui lui évite de retravailler sur ce bloc.

Les problèmes ouverts qui me paraissent importants sont liés à l'énumération des cliques maximales potentielles.

D'abord, il n'y a pas d'algorithme énumérant ces objets, avec un délai polynomial par clique maximale potentielle obtenue. L'algorithme que nous avons donné calcule bien les cliques maximales potentielles en $\mathcal{O}(|\Delta_G|^2 \operatorname{Poly}(n))$, mais le nombre de cliques maximales potentielles se situe entre $|\Delta_G|/n$ et $n^2|\Delta_G|^2$. Un algorithme avec délai polynomial permettrait un calcul exact de la largeur arborescente avec une meilleure borne que celle que nous avons donné, et de plus sa complexité serait polynomiale pour les classes de graphes avec "peu" de séparateurs minimaux.

Le deuxième problème serait de calculer la meilleure triangulation que l'on peut obtenir avec un sous-ensemble Δ' de l'ensemble des séparateurs minimaux de G. Si $\Delta' = \Delta_G$, nous pouvons énumérer toutes les cliques maximales potentielles de G et le problème est résolu. Par contre, si Δ' est un sous-ensemble strict des séparateurs minimaux de G nous ne savons pas énumérer les cliques maximales potentielles bordées uniquement par des séparateurs dans Δ'. Je ne connais même pas d'algorithme qui décide s'il existe une triangulation minimale H dont tous les séparateurs minimaux soient dans Δ'.

Citons enfin le problème, toujours ouvert, du calcul de la largeur arborescente des graphes planaires. Il existe un algorithme sous-exponentiel pour résoudre ce problème, mais nous ignorons si le paramètre peut être calculé en temps polynomial – alors que la largeur de branches des graphes planaires est calculable en temps polynomial par l'algorithme de [122].

Chapitre 4

Approximation de la largeur arborescente

Puisque le calcul de la largeur arborescente est un problème NP-difficile, il est naturel et important de s'intéresser à l'approximation de ce paramètre. Etant donné que les algorithmes qui utilisent les décompositions arborescentes pour résoudre des problèmes difficiles sont eux-mêmes exponentiels en la largeur de la décomposition dont on dispose, nous voyons clairement la nécessité d'avoir des algorithmes d'approximation calculant une décomposition très proche de l'optimum.

Au moment où nous avions commencé à aborder ce problème, le meilleur résultat dans ce domaine était l'algorithme de [22], qui approxime la largeur arborescente à facteur $\mathcal{O}(\log n)$. Nous présentons des algorithmes d'approximation de la largeur arborescente à facteur constant pour certaines classes de graphes [33, 29], ainsi qu'un algorithme d'approximation à facteur $\mathcal{O}(\log \text{OPTIMUM})$ pour les graphes quelconques. Nous évoquons en conclusion de ce chapitre les derniers progrès accomplis dans ce domaine ainsi que les questions ouvertes.

Ces résultats ont été obtenus en collaboration avec V. Bouchitté, D. Kratsch et H. Müller.

4.1 Une heuristique naturelle

Nous analysons ici une version très naturelle de l'algorithme `DecoupageEnBlocs`, de la figure 2.6. Lorsque l'on choisit un séparateur minimal S pour découper le graphe (ou, récursivement, l'un de ses blocs), on sait que S sera transformé en clique. Notre objectif étant de minimiser la taille de la plus grande clique obtenue en sortie, il est naturel de choisir à chaque étape un séparateur S de cardinal minimum. Nous appelons `DecSepMin` cette heuristique. Elle est de complexité polynomiale, puisque le calcul d'un séparateur de cardinal minimum peut se faire en temps polynomial par des techniques standard de flots [2].

Cette section contient un résultat négatif : l'heuristique `DecSepMin` n'est pas un algorithme d'approximation de la largeur arborescente à facteur constant, pour les graphes quelconques. Cependant, l'expérience prouve que l'heuristique est très bonne en pratique. Une partie de l'explication pourrait de trouver dans le résultat positif de cette section : pour certaines classes de graphes (ici, pour les classes de graphes de nombre astéroïde borné), `DecSepMin` fournit bien une décomposition de largeur $\mathcal{O}(k)$ où k est la largeur arborescente du graphe en entrée.

Montrons d'abors que, pour toute classe de graphes de nombre astéroïde au plus a, `DecSepMin` produit une triangulation de largeur au plus $8a\,\mathrm{tw}(G)$, G étant le graphe en entrée.

Définition 4.1 *Un* ensemble astéroïde *d'un graphe G est un stable A tel que pour tous sommet $v \in A$, il existe une composante connexe de $G - N(v)$ contenant tous les sommets de $A\backslash\{v\}$. Le* nombre astéroïde $\mathrm{an}(G)$ *du graphe G est la taille maximum d'un ensemble astéroïde de G. En particulier, un graphe est dit* sans triplet astéroïde *si son nombre astéroïde est au plus deux.*

Notons que le calcul de la largeur arborescente est NP-difficile même pour les graphes co-bipartis, une sous-classe des graphes sans triplet astéroïde [70].

Le nombre astéroïde est une borne supérieure pour le nombre de séparateurs maximaux par inclusion se trouvant sur le bord d'un bloc.

Proposition 4.2 ([92]) *Soit B un bloc de G et $\mathcal{EB}(B)$ l'ensemble des éléments maximaux parmi les séparateurs $\mathcal{S}(B)$, bordant B. Nous avons $|\mathcal{EB}(B)| \leq \mathrm{an}(G)$.*

On appelle *ensemble bloquant* l'ensemble des séparateurs $\mathcal{EB}(B)$. Nous avons vu que si G a un petit nombre astéroïde, chaque ensemble bloquant est de petite taille. Supposons que la largeur arborescente de notre graphe est au plus k. Nous montrerons que si un bloc B est de taille supérieure à $8k\,\mathrm{an}(G)$ et que chaque séparateur qui le borde est de taille au plus $4k$, la réalisation $R(B)$ peut encore être découpée par un séparateur de taille au plus $4k$. L'heuristique `DecSepMin` choisira donc des petits séparateurs (de taille au plus $4k$) jusqu'à ce que tous les blocs soient de taille au plus $8k\,\mathrm{an}(G)$. Nous en conclurons que l'heuristique produit de bonnes décompositions pour les graphes de nombre astéroïde borné. Nous avons besoin d'introduire quelques nouvelles notions.

Définition 4.3 *Soit B un bloc de G. Notons* $\mathrm{Bord}(B) = \{x \in B | x$ *a un voisin dans* $G - B\}$ *et* $\mathrm{Int}(B) = B\backslash\,\mathrm{Bord}(B)$.

Ainsi $\mathrm{Bord}(B)$ est l'union des séparateurs bordant le bloc B ; c'est également l'union des séparateurs de son ensemble bloquant.

Théorème 4.4 *Soit G un graphe de largeur arborescente au plus k. Soit B un bloc de G de taille supérieure à $4k$. Si $|\mathrm{Int}(B)| \geq |B|/2$, il existe un séparateur minimal $S \subseteq B$ de $R(B)$ tel que $|S| \leq 4k$.*

Preuve. (éléments.) Montrons d'abord qu'il existe un sommet $x \in \mathrm{Int}(B)$ tel que $|N_G(x)| \leq 4k$. Soit $m(B)$ le nombre d'arêtes de $G[B]$. Nous compterons $m(B)$ de deux façons différentes. Puisque $G[B]$ est un sous-graphe induit de G, sa largeur arborescente est au plus k. Il est bien connu que dans un graphe de largeur arborescente k, le nombre d'arêtes est au plus k fois le nombre de sommets, en particulier $m(B) \leq k|B|$. Supposons par absurde que chaque sommet x de $\mathrm{Int}(B)$ a strictement plus de $4k$ voisins dans G et montrons que $m(B) > k|B|$. Comme $x \in \mathrm{Int}(B)$, nous avons $N_G(x) \subseteq B$, donc le nombre d'arêtes de $G[B]$ incidentes à au moins un sommet de $\mathrm{Int}(B)$ est strictement supérieur à $(4k|\mathrm{Int}(B)|)/2 = 2k|\mathrm{Int}(B)|$. Mais $|\mathrm{Int}(B)| \geq |B|/2$, donc $m(B) > 2k|Int(B)| \geq k|B|$. Ceci contredit l'inégalité $m(B) \leq k|B|$.

Nous avons prouvé l'existence d'un sommet $x \in \mathrm{Int}(B)$ tel que $|N_G(x)| \leq 4k$. Soit y un sommet de $B \backslash N_G(x)$, différent de x (y existe parce que $|B| > 4k$). L'ensemble $N_G(x)$ sépare x et y dans G, donc il existe un séparateur minimal $S \subseteq N_G(x)$ séparant x et y dans G. Clairement $|S| \leq 4k$, et il n'est pas difficile de montrer que S est également un séparateur minimal de $R(B)$. $\diamond$

Corollaire 4.5 *Soit G un graphe de largeur arborescente k. Considérons un bloc B de G tel que tous les séparateurs bordant B soient de taille au plus $4k$. Si $|B| > 8k\,\mathrm{an}(G)$, il existe un séparateur minimal S de $R(B)$ tel que $|S| \leq 4k$.*

Preuve. Soit $\mathcal{EB}(B) = \{S_1, \ldots, S_p\}$ l'ensemble bloquant de B, par la proposition 4.2 $\mathcal{EB}(B)$ a au plus $\mathrm{an}(G)$ éléments. Nous avons $\mathrm{Bord}(B) = S_1 \cup \cdots \cup S_p$, et chaque S_i a au plus $4k$ sommets, donc $|\mathrm{Bord}(B)| \leq 4k\,\mathrm{an}(G)$. Par conséquent, $|\mathrm{Int}(B)| = |B| - |\mathrm{Bord}(B)| > |B|/2$. L'existence du séparateur S de $R(B)$ de taille au plus $4k$ est une conséquence directe du théorème 4.4. $\diamond$

Quand l'algorithme `DecSepMin` aura épuisé tous les séparateurs de taille au plus $4k$, il ne nous reste que des blocs de taille au plus $8k\,\mathrm{an}(G)$.

Corollaire 4.6 *Soit G un graphe de largeur arborescente k et de nombre astéroïde a. L'heuristique `DecSepMin` calcule une triangulation de G de largeur au plus $8ak$.*

Le calcul du nombre astéroïde d'un graphe est NP-difficile [91], ce qui semble être gênant pour notre algorithme. Remarquons que le corollaire 4.5 utilise simplement le fait que l'ensemble bloquant d'un bloc B a au plus $an(G)$ éléments. En testant à chaque étape le nombre d'éléments $\mathcal{ES}(B)$, l'algorithme peut être transformé pour que, prenant en entrée un graphe G et un nombre a, il produise l'une des sorties suivantes :

1. une triangulation de largeur au plus $8a\,\mathrm{tw}(G)$.
2. un avertissement indiquant que le nombre astéroïde de G est strictement supérieur à a.

Passons maintenant au résultat négatif concernant l'heuristique `DecSepMin`. Nous montrerons que, pour tout entier positif q, il existe une famille infinie de graphes pour lesquels la largeur de la triangulation calculée est supérieure à $q/2$ fois l'optimum. En d'autres termes, pour toute constante c, notre heuristique n'est pas une c-approximation de la largeur arborescente.

Considérons un graphe complet ayant q sommets $I_1, I_2, \ldots, I_q$. Divisons chaque arête I_iI_j de ce graphe en y rajoutant un sommet M_{ij}. Ajoutons un sommet C adjacent à $I_1, \ldots, I_q$. Remplaçons le sommet C par un module clique de taille $2p$ (chaque sommet de la clique sera adjacent aux voisins de C dans le graphe initial, ainsi qu'aux autres sommets de la clique). Enfin, remplaçons chaque sommet I_i par un module stable de taille p (chaque sommet du stable est adjacent aux anciens voisins de I_i). On prendra p "beaucoup plus grand" que q. Notons $G_{p,q}$ le graphe ainsi obtenu (voir la figure 4.1 pour $q = 3$).

Proposition 4.7 *L'heuristique du séparateur de cardinal minimum donne, pour le graphe $G_{p,q}$, une triangulation de largeur $(q+2)p-1$.*

Preuve. Les séparateurs de cardinal minimum de ce graphe sont exactement les séparateurs de la forme $I_i \cup I_j$, ayant tous une taille $2p$. De plus, ils sont deux à deux parallèles. C'est pourquoi notre algorithme choisira toujours un séparateur de ce type, jusqu'à ce qu'ils soient tous transformés en cliques. A ce stade, pour chaque bloc B du découpage, sa réalisation est une clique. L'un de ces blocs est $\Omega = C \cup I_1 \cup I_2 \cup \cdots \cup I_q$, de taille $(q+2)p$. ◇

Proposition 4.8 *La largeur arborescente de $G_{p,q}$ est au plus $2p + q(q-1)/2$.*

Preuve. Considérons le séparateur (non minimal) S formé par les sommets de C et tous les sommets M_{ij}. En complétant S en une clique nous obtenons un graphe H triangulé, plus exactement un graphe split dont S est la clique. Clairement toute clique maximale de H est de taille au plus $|S|+1$. ◇

Nous concluons que le ratio entre la largeur de la triangulation obtenue par notre algorithme et la largeur arborescente de $G_{p,q}$ est d'au moins $q/2$, pour p suffisamment grand. Ainsi notre heuristique n'est pas un algorithme d'approximation à facteur constant de la largeur arborescente.

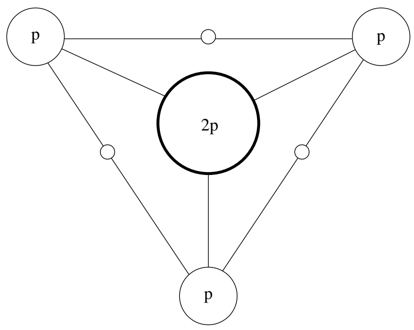

FIG. 4.1 – $G_{p,q}$

4.2 Graphes sans triplet astéroïde

Trois sommets (x, y, z) forment un *triplet astéroïde* de G si pour tout paire de sommets parmi les trois, il existe une chaîne qui les relies, en évitant le voisinage du troisième. Un graphe qui ne contient pas un tel triplet est appelé *sans triplet astéroïde*. De façon équivalente, un graphe est sans triplet astéroïde si son nombre astéroïde est au plus égal à 2.

D'après la proposition 4.2, dans un graphe sans triplet astéroïde l'ensemble bloquant $\mathcal{EB}(B)$ d'un bloc B contient au plus deux éléments. Si cet ensemble est vide, le bloc B est exactement l'ensemble des sommets du graphe. Si $\mathcal{EB}(B)$ a un seul élément, B est un 1-bloc. Enfin si $\mathcal{EB}(B)$ a deux éléments S et U, alors $B = \mathrm{PartieEntre}(S, U)$. Rappelons que étant donné deux séparateurs minimaux S et U, incomparables par inclusion, tels que U est contenu dans un 1-bloc $(S, C(S))$ et S est contenu dans un 1-bloc $(U, C(U))$, on définit la partie entre S et U comme $\mathrm{PartieEntre}(S, U) = (S \cup C(S)) \cap (U \cup C(U))$ (cf. figure 4.2).

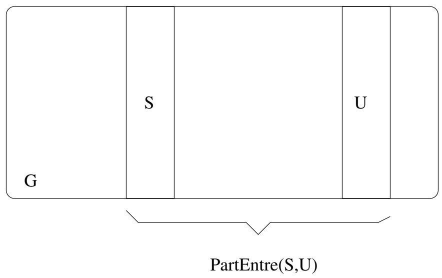

FIG. 4.2 – $\mathrm{PartieEntre}(S, U)$.

Commençons par une description informelle de l'algorithme `ApproxTW_ATf`. En entrée, nous avons un graphe sans triplet astéroïde G et un entier k. Nous voulons décider si $\mathrm{tw}(G) > k$ ou donner une triangulation de largeur au plus $2k$. La complexité en temps de notre algorithme ne dépend pas de k, il sera donc facile d'en obtenir une 2-approximation de la largeur arborescente des graphes sans triplet astéroïde.

L'algorithme s'appuie sur la propriété suivante, qui sera discutée un peu plus bas. Etant donné un graphe sans triplet astéroïde de largeur arborescente au plus k et un 1-bloc (S, C) de G avec $|S| \leq k$ et $|S \cup C| > 2k + 1$, il y a un séparateur minimal $U \subseteq (S, C)$ tel que $|\mathrm{PartieEntre}(S, U)| \leq 2k + 1$ et $|U| \leq k$. De plus, ce séparateur est calculable en temps polynomial.

Rappelons que $R(S, C) = G_S[S \cup C]$ est la réalisation de G. Sous l'hypothèse que $|S| \leq k$, nous calculerons une triangulation $H(S, C)$ de $R(S, C)$ de largeur au plus $2k$ ou bien nous déciderons que $\mathrm{tw}(G) > k$. Rappelons qu'une triangulation de G est obtenue en prenant $\bigcup_{C_i} H(S_i, C_i)$, où C_i sont les composantes de $G - S$ et $S_i = N(C_i)$ (cf. proposition 2.31).

L'algorithme procède comme suit. Si $|S \cup C| \leq 2k + 1$, on complète simplement le 1-bloc (S, C) en une clique. Sinon, nous cherchons un séparateur minimal $U \subseteq (S, C)$ de G tel que $|U| \leq k$ et $|\mathrm{PartieEntre}(S, U)| \leq 2k + 1$. Si un tel séparateur n'existe pas,

Algorithme `ApproxTW_ATf`

Entrée : un graphe sans triplet astéroïde G et un entier k
Sortie : $\mathrm{tw}(G) > k$ ou une triangulation H de G, de largeur au plus $2k$
debut
  calculer un séparateur minimal S de G de cardinal minimum
  si $|S| > k$ **alors**
    returner "$\mathrm{tw}(G) > k$"
  soient $C_1, \ldots, C_p$ les composantes de $G - S$ et $S_i = N_G(C_i)$
  pour chaque 1-bloc (S_i, C_i) associé à S `faire`
    $H(S_i, C_i) :=$ `trianguler_réalisation`(S_i, C_i, k)
    si l'appel retourne "$\mathrm{tw}(G) > k$" **alors**
      returner "$\mathrm{tw}(G) > k$"
  returner $\bigcup_{i=1}^{p} H(S_i, C_i)$
fin

FIG. 4.3 – 2-approximation de la largeur arborescente pour les graphes sans triplet astéroïde

nous affirmons que $\mathrm{tw}(G) > k$. Si U existe, observons que U découpe le bloc (S, C) en blocs plus petits, plus exactement en $\mathrm{PartieEntre}(S, U)$ et blocs de la forme (U', C'), où $C' \subset C$ sont les composantes de $G - U$ contenues dans C et $U' = N(C')$. Le processus est alors appliqué récursivement sur chaque sous-bloc (U', C'), en obtenant une triangulation $H(U', C')$ de $R(U', C')$ de largeur au plus $2k$ (ou en concluant que $\mathrm{tw}(G) > k$). Le bloc $\mathrm{PartieEntre}(S, U)$ sera complété en une clique. Pour trouver le premier séparateur minimal S de taille au plus k il suffit de calculer, par des techniques bien connues, un séparateur de cardinal minimum de G. La partie plus délicate est de trouver le prochain séparateur U, et pour ce faire nous utilisons une méthode assez indirecte. Nous considérerons le graphe co-biparti $G' = G_{\{S,C\}}[S \cup C]$ obtenu à partir de $G[S \cup C]$ en complétant chacun des ensembles S et C. Nous montrerons que si $\mathrm{tw}(G) < k$, il existe un séparateur minimal X de G' de taille au plus k. Réciproquement, si G' a un séparateur X tel que $|X| \leq k$, l'ensemble de sommets $S \cup X$ forme un bloc $\mathrm{PartieEntre}(S, U)$ de G, avec $U = N_G(C \backslash X)$. Le séparateur minimal U satisfait donc les conditions $|\mathrm{PartieEntre}(S, U)| \leq 2k + 1$ et $|U| \leq k$.

Le principal résultat technique est l'existence du séparateur U et la possibilité de le calculer efficacement.

Théorème 4.9 *Soit G un graphe sans triplet astéroïde, de largeur arborescente k. Considérons un bloc (S, C) de G, où S est de taille au plus k, et $|S \cup C| > 2k + 1$. Supposons que S n'est pas un séparateur minimal de la triangulation optimale H de G.*

Il existe un séparateur minimal U de G, contenu dans (S, C), tel que $|U| \leq k$ et $|\mathrm{PartieEntre}(S, U)| \leq 2k + 1$. De plus, un tel séparateur U peut être calculé en temps

Procédure `trianguler_réalisation`

Entrée : G, S, C et k
Sortie : $\mathrm{tw}(G) > k$ ou une triangulation $H(S, C)$ de $R(S, C)$ de largeur au plus $2k$
Fonction `calcul_U`
  $G' := G_{\{S,C\}}[S \cup C]$
  calculer un séparateur de cardinal minimum X de G'
  si$|X| > k$ **alors**
    retourner erreur
  sinon
    $U := N_G(C \backslash X)$
    retourner U

debut
  si $|S \cup C| \leq 2k + 1$ **alors**
    retourner le graphe complet $K(S \cup C)$
  $U :=$ `calcul_U`
  si erreur **alors**
    $/*$ on a encore la possibilité que $\mathrm{tw}(R(S, C)) \leq k$ $*/$
    retourner `ApproxTW_ATf`$(R(S, C), k)$
  pour chaque composante $C' \subseteq C$ de $G - U$ **faire**
    $U' := N_G(C')$
    $H'(U', C') :=$ `trianguler_réalisation`(U', C', k)
    si la sortie est “$\mathrm{tw}(G) > k$” **alors**
      retourner “$\mathrm{tw}(G) > k$”
  retourner$\bigcup_{C'} H'(U', C') \cup G_{\mathrm{PartieEntre}(S,U)}[S \cup U]$
fin

FIG. 4.4 – Triangulation de la réalisation d’un 1-bloc

polynomial.

Dans l’hypothèse de notre théorème, nous avons supposé que S n’est pas un séparateur de la triangulation optimale H. Si le séparateur minimal U n’est pas trouvé par notre algorithme, il reste une dernière chance : que S soit un séparateur minimal de H. Pour tester cette possibilité on complète S et on relance l’algorithme avec, en entrée, la réalisation $R(S, C)$ du bloc (S, C).

La preuve de ce théorème 4.9 est longue et fastidieuse, nous allons donner uniquement quelques intuitions allant en ce sens. Pour ce faire, nous allons utiliser comme cliques de la triangulation des blocs de type $S \cup (C \cap T)$, où T est un séparateur minimal qui croise S. Rappelons que, d’après le théorème 3.2, une large classe de cliques maximales potentielles est également de ce type.

Les deux observations qui suivent ne sont pas spécifiques aux graphes sans triplet astéroïde.

Lemme 4.10 *Soient (S, C) un bloc de G et T un séparateur minimal qui croise S. L'ensemble de sommets $S \cup (C \cap T)$ est également un bloc de G.*

La remarque suivante est une conséquence directe du théorème de Parra et Scheffler 2.22.

Remarque 4.11 *Soit G un graphe de largeur arborescente k et H une triangulation minimale optimale de G. Soit S un séparateur minimal de G, de taille au plus k. Si S n'est pas un séparateur minimal de H, il existe un séparateur minimal T de H qui croise S dans G.*

En particulier, si C est une composante de $G - S$ pleine par rapport à S, alors $S \cup (C \cap T)$ est un bloc de taille au plus $2k$.

En effet, puisque l'ensemble Δ_H des séparateurs minimaux de G est un ensemble maximal de séparateurs deux à deux parallèles de G et que $S \notin \Delta_H$, au moins un élément T de Δ_H croise S. Ce bloc $B = S \cup (C \cap T)$ est de taille au plus $2k$, et il découpe le bloc (S, C) en B et des 1-blocs de type $(N(D), D)$, où D est une composante de $G - B$ contenue dans C. Notre bloc B est un bon candidat pour être le bloc $\mathrm{PartieEntre}(S, U)$ du théorème 4.9.

Reconsidérons la structure des blocs évoqués dans le lemme 4.10 pour les graphes sans triplet astéroïde.

Proposition 4.12 *Soient G un graphe sans triplet astéroïde, (S, C) un 1-bloc et T un séparateur minimal de G qui croise S. Supposons que $C \backslash T \neq \emptyset$ et soit $U = N(C \backslash T)$. Alors U est un séparateur minimal et $S \cup (T \cap C) = \mathrm{PartieEntre}(S, U)$.*

Replaçons-nous sous les conditions du théorème 4.9. Si nous avons la chance que U soit de taille au plus k, le séparateur U satisfait toutes les conditions requises par le théorème – encore faut-il montrer qu'il peut être calculé en temps polynomial. La proposition suivante va dans ce sens.

Proposition 4.13 *Soient S, T et U trois séparateurs du graphe sans triplet astéroïde G et soit C une composante pleine de $G - S$, avec $S \cup (T \cap C) = \mathrm{PartieEntre}(S, U)$ comme dans la proposition 4.12.*

Soit G' le graphe co-biparti $G_{\{S,C\}}[S \cup C]$, obtenu à partir de $G[S \cup C]$ en complétant chacun des ensembles S et C en une clique. L'ensemble $X = (T \cap C) \cup U$ est un séparateur minimal de G'.

Réciproquement, si X est un séparateur minimal de G', alors $S \cup X$ est un bloc de la forme $\mathrm{PartieEntre}(S, U)$ *avec $U = N_G(C \backslash X)$.*

Afin de calculer U, nous procédons comme dans la fonction `calcul_U` de l'algorithme `trianguler_réalisation`. Nous construisons le graphe $G' = G_{\{S,C\}}[S \cup C]$ et nous calculons un séparateur de cardinal minimum X de G'. Si $|X| \leq k$, d'après la proposition 4.13, le bloc $B = S \cup X$ est de taille au plus $2k$. De plus, U est un sous-ensemble de X, impliquant que $|U| \leq k$. Nous aurons toutes les conditions du théorème 4.9. Cependant, pour l'instant nous ne sommes pas certains de l'existence d'un séparateur X de G de taille au plus k ; cette condition est prouvée uniquement si, par chance, $|(T \cap C) \cup U| \leq k$. La preuve complète du résultat dans le cas général est donnée dans [33]. Elle est plutôt longue (une dizaine de pages) et se base sur l'idée de redécouper l'ensemble $(T \cap C) \cup U$ par une clique maximale potentielle bien choisie de G.

Proposition 4.14 *Soit G un graphe sans triplet astéroïde, de largeur arborescente k. Considérons un 1-bloc (S, C) de G avec $|S| \leq k$ et $|S \cup C| > 2k + 1$. Supposons que S n'est pas un séparateur minimal de la triangulation optimale H de G.*

Soit G' le graphe co-biparti $G_{\{S,C\}}[S \cup C]$, obtenu à partir de $G[S \cup C]$ en complétant chacun des ensembles S et C en une clique. Il existe un séparateur minimal X de G', de taille au plus k.

Grâce à ce résultat et à la proposition 4.13, le séparateur minimal U comme dans le théorème 4.9 existe toujours et peut être calculé en temps polynomial par la procédure `calcul_U`. On en conclut :

Théorème 4.15 *Il existe un algorithme polynomial qui approxime à facteur 2 la largeur arborescente des graphes sans triplet astéroïde.*

4.3 Séparateurs équilibrés

Nous présentons dans cette section un algorithme polynomial qui, étant donné un graphe G, produit une décomposition arborescente de G ayant une largeur $\mathcal{O}(k \log k)$, où k est la largeur arborescente de G.

Bodlaender et al. proposaient dans [22] un algorithme d'approximation de la largeur arborescente, et dans leur cas la largeur de la décomposition obtenue est $\mathcal{O}(k \log n)$. L'algorithme est basé sur le calcul de *séparateurs équilibrés* ; le facteur $\log n$ provient du calcul de ces séparateurs, en utilisant une technique de Leighton et Rao [100]. Dans notre cas, nous employons une technique plus performante pour le calcul de séparateurs équilibrés, proposée par Even et al. [55], qui nous permet de ramener le facteur multiplicatif $\log n$ à un facteur $\log k$. Etant donné que les algorithmes qui utilisent les décompositions arborescente sont généralement exponentiels en la largeur de la décomposition dont on dispose, cette amélioration est très importante de point de vue théorique. Notons que, indépendamment de notre travail, Amir [3] donne un algorithme d'approximation de la largeur arborescente avec des performances similaires.

La preuve du résultat ci-dessous (voir par exemple [22]) est très semblable à la preuve du fait que, dans tout arbre, il existe un noeud qui découpe l'arbre en composantes de taille au plus $n/2$.

Lemme 4.16 *Soit $G = (V, E)$ un graphe de largeur arborescente k et W un ensemble de sommets de G. Il existe un séparateur T de G de taille au plus $k + 1$, tel que chaque composante connexe de $G - T$ contient au plus la moitié des sommets de W.*

Ce type de séparateur (pas forcément minimal – mais on peut toujours choisir une clique maximale potentielle !) est appelé séparateur 1/2-équilibré pour W.

Supposons que nous disposions d'un algorithme `SepEquilibre` qui, prenant en entrée un ensemble de sommets W, calcule un séparateur 1/2-équilibré pour W, de cardinal minimum. En le combinant avec l'algorithme `DecoupageEnBlocs`, il permettrait de calculer une décomposition arborescente de largeur au plus $3k + 2$ du graphe G, ou décider que $\mathrm{tw}(G) > k$. L'algorithme commence par chercher simplement un séparateur de G de taille au plus k, et avec ce séparateur on découpe le graphe en 1-blocs. Les blocs que nous générerons au cours de l'algorithme sont de deux types : des blocs de taille au plus $3k + 3$, que l'on complète simplement en cliques, et des 1-blocs de la forme (S, C), en assurant pour chacun de ces blocs que $|S| \leq 2k + 2$. Les blocs de la forme (S, C) sont découpés récursivement de la façon suivante. On calcule un séparateur $T :=$ `SepEquilibre`(S) qui soit 1/2-équilibré pour S. Soient $C_1, C_2, \ldots, C_p$ les composantes de $G - T$ contenues dans C et notons $S_i = N_G(C_i)$. Le bloc (S, C) est découpé en $S \cup (T \cap C)$ (de taille au plus $3k+3$) et des 1-blocs (S_i, C_i). Il reste à observer que $|S_i| \leq 2k+2$. En effet, $S_i \backslash T$ est contenu dans S, et tous ces sommets sont dans une même composante de $G-T$. Il y a donc au plus $|S|/2 \leq k + 1$ sommets de ce type. Nous avons donc $|S_i| \leq k + 1 + |T| \leq 2k + 2$. Cette technique est à la base des premiers algorithmes d'approximation de la largeur arborescente, qui fonctionnent en temps exponentiel en k [114].

Le calcul d'un 1/2-séparateur équilibré de cardinal minimum est NP-difficile. Nous utilisons des résultats d'approximation pour ce problème, qui nécessite une notion plus générale de séparateur équilibré.

Définition 4.17 *Soit $G = (V, E)$ un graphe et une W un ensemble de sommets de G. Considérons un paramètre ρ compris entre 0 et 1. Un séparateur ρ-équilibré pour W est un ensemble de sommets T tel que, pour chaque composante connexe de $G-T$, la composante contient au plus $\rho|W|$ sommets de W.*

Dans leur article, Even et al. travaillent surtout avec des arcs-séparateurs (les graphes sont orientés et les séparateurs sont des ensembles d'arcs et non pas de sommets). Comme ils le font remarquer, un sommet-séparateur d'un graphe non orienté correspond à un arc-séparateur dans un graphe orienté bien choisi. Nous reformulons leur résultats directement en termes de sommets-séparateurs.

Théorème 4.18 ([55]) *Soit G un graphe non orienté et W un ensemble de sommets de G. Soit τ la taille minimum (en nombre de sommets) d'un séparateur ρ-équilibré pour W, avec $1/2 \leq \rho < 1$. Etant donné un paramètre ρ', $\rho < \rho' \leq 1$, il existe un algorithme polynomial calculant un séparateur ρ'-équilibré pour W, de taille au plus $\frac{4.04\rho}{\rho'-\rho}(5 + \ln(\frac{\rho}{\rho'-\rho}\tau))\tau$.*

Remarquons que l'algorithme ne calcule pas un séparateur ρ-équilibré, mais bien un séparateur ρ'-équilibré avec $\rho' > \rho$. Le séparateur obtenu est "moins équilibré" que le séparateur optimal, et sa taille est également plus grande que l'optimum.

D'après le théorème 4.18, étant donné G, W et $\rho' > 1/2$, nous pouvons trouver en temps polynomial un séparateur T' de taille $\mathcal{O}(k \log k)$, ρ'-équilibré pour W. Par le même type d'algorithme que celui décrit ci-dessus, nous obtenons (voir aussi [22]) :

Théorème 4.19 ([29]) *Il existe un algorithme polynomial qui, prenant en entrée un graphe G et un entier k, calcule une décomposition arborescente de G de largeur $\mathcal{O}(k \log k)$ ou décide que* $\mathrm{tw}(G) > k$.

4.4 Approximation de la largeur arborescente : les bonnes et les mauvaises nouvelles

Le problème de l'approximation de la largeur arborescente est l'une de ces questions algorithmiques qui n'ont pas trouvé de réponse satisfaisante de point de vue théorique, alors que la situation est bien meilleure de point de vue pratique.

Nous avons donné dans ce chapitre un algorithme polynomial qui approxime la largeur arborescente d'un graphe quelconque à un facteur $\mathcal{O}(\log k)$, où k est la valeur optimale. Des travaux récents sur les séparateurs équilibrés [56] permettent même une approximation à facteur $\mathcal{O}(\sqrt{\log k})$. De point de vue théorique ces résultats pourraient paraître presque satisfaisants. Mais leur gros défaut ne vient pas, comme on pourrait le croire, du facteur $\log k$, mais de la constante cachée par le $\mathcal{O}$ qui est de l'ordre de 1000 ! Qui plus est, ils utilisent des outils puissants, mais compliqués, pour le calcul d'un séparateur équilibré. Le problème du séparateur $1/2$-équilibré est NP-difficile. Cependant, les nombreux travaux dans ce domaine pourraient aboutir à un algorithme d'approximation à facteur constant près, en se débarrassant du facteur $\log OPTIMUM$ (ou $\sqrt{\log OPTIMUM}$ actuel). Il est à noter qu'un tel algorithme a déjà été annoncé, mais il s'est avéré faux. En tout cas, cela aboutirait également à une approximation à facteur constant de la largeur arborescente.

Pour ma part j'aurais déjà aimé qu'on dispose d'un algorithme simple qui approxime la largeur arborescente à un facteur $\mathrm{Poly}(k)$, pour un polynôme "raisonnable". Là aussi, un tel algorithme [1] a été annoncé dans [131], mais il n'a jamais été publié (et, selon l'un des auteurs, le problème peut être considéré comme ouvert).

Au pôle opposé, il existe de nombreuses heuristiques pour la largeur arborescente (voir par exemple [18] pour une présentation synthétique de ces techniques). Sans garantie théorique, ces heuristiques ont été validées sur de nombreux jeux de tests, ce qui a d'ailleurs requis le développement d'autres heuristiques calculant des minorants de la largeur arborescente. Citons également le constat que Amir [3] fait dans la conclusion de son article. Après avoir implanté un algorithme (exponentiel) qui approxime la largeur arborescente des graphes quelconques à facteur 2, il constate que la plus simple et la plus connues des heuristiques, `MinimumDegree`, donne des résultats bien meilleurs que son algorithme sur

[1] pas forcément simple...

tous les jeux de tests qui l'intéressent. Bien entendu on peut trouver des exemples pour lesquels `MinimumDegree` se comporte mal, en particulier elle a exactement comportement que `DecSepMin` pour l'exemple de la section 4.1.

Chapitre 5

Complétions d'intervalles minimales

Les algorithmes que nous avons vu pour le calcul ou l'approximation de la largeur arborescente sont basés sur une connaissance très fine des triangulations minimales. Lorsque l'on s'intéresse aux décompositions linéaires et en particulier à la largeur linéaire (pathwidth), on espère pouvoir bénéficier d'outils similaires concernant les complétions d'intervalles minimales. Il se trouve que, à ma connaissance, il n'y avait dans la littérature aucun algorithme calculant une complétion minimale d'un graphe quelconque. Les approches naïves, consistant par exemple à partir d'une complétion en clique et à essayer de supprimer une arête à la fois ne mènent pas toujours à des complétions d'intervalles minimales.

Nous avons donné dans [75] un premier algorithme calculant une complétion d'intervalles minimale d'un graphe quelconque, algorithme présenté dans la première section de ce chapitre. Dans la deuxième section, je présenterai un autre algorithme, plus simple et plus rapide (voir aussi [126]) ainsi qu'un algorithme de complexité linéaire pour le calcul d'une complétion d'intervalles *propres* minimale [113].

Bien entendu de ces algorithmes ne suffisent pas, à eux seuls, à calculer de la largeur linéaire ou d'autres paramètres liés. Nous reviendrons sur le travail à faire et sur les progrès réalisés dans la conclusion de ce chapitre, ainsi que dans tout le chapitre suivant.

Les résultats de ce chapitre ont été obtenus en collaboration avec K. Suchan, Y. Villanger, P. Heggernes et I. Rapaport.

5.1 Une approche incrémentale

Le premier algorithme que nous avons donné pour le calcul d'une complétion d'intervalles minimale d'un graphe quelconque $G = (V, E)$ est basé sur une approche incrémentale [75].

Pour obtenir une complétion d'intervalles minimales du graphe G, nous partirons du graphe vide et nous ajouterons les sommets de G dans l'ordre $(v_1, v_2, \ldots, v_n)$. Etant donnée une complétion d'intervalles minimale H_i du graphe $G_i = G[\{v_1, v_2, \ldots, v_i\}]$, nous calculerons une complétion d'intervalles minimale H_{i+1} de G_{i+1} en rajoutant à H_i le sommet v_{i+1}, les arêtes entre v_{i+1} et ses voisins dans G_{i+1}, ainsi qu'un ensemble bien choisi de nouvelles arêtes, toutes incidentes à v_{i+1}. Cette approche est possible grâce à la propriété ci-dessous :

Lemme 5.1 *Soit H une complétion d'intervalles minimale d'un graphe $G = (V, E)$ quelconque. Soit $G' = G + x$ un graphe obtenu à partir de G en lui rajoutant un nouveau sommet x, ainsi qu'un ensemble d'arêtes incidentes à x. Il existe une complétion d'intervalles minimale H' de G' telle que $H' - x = H$.*

Preuve. (éléments.) Soit H'' le graphe obtenu à partir de H en ajoutant le sommet x et des arêtes entre x et chaque sommet de H. Le graphe H'' est un graphe d'intervalles : il suffit de considérer un modèle d'intervalles de H et de rajouter un nouvel intervalle, correspondant à x, qui intersecte tous les autres.

Prénons un graphe d'intervalles H' tel que $G' \subseteq H' \subseteq H''$, minimal pour cette propriété. Vérifions que H' est bien une complétion d'intervalles minimale de G'. Si ce n'est pas le cas, il existe une complétion d'intervalles $H'_1 \subset H'$. Par construction de H', il y a au moins une arête de $E(H') \backslash E(H'_1)$ avec les deux extrémités dans V. Par conséquent, $H'_1[V]$ est une complétion d'intervalles de G, strictement contenue dans H – contredisant la minimalité de H. $\diamond$

Désormais nous considérons le problème suivant (remarquer le changement de notation par rapport au lemme ci-dessus). En entrée nous avons un graphe d'intervalles $G = (V, E)$. Un nouveau sommet est ajouté à G, ainsi qu'un ensemble d'arêtes incidentes à x. Nous noterons G' ce nouveau graphe $G + x$, qui n'est pas nécessairement un graphe d'intervalles. Nous calculerons une complétions d'intervalles minimale H de G', telle que toute nouvelle arête de H par rapport à G' soit incidente à x. Nous dirons qu'une telle complétion d'intervalles minimale *respecte* G. D'après le lemme 5.1 et le commentaire qui le précède, en résolvant ce problème nous pourrons calculer de façon incrémentale une complétion d'intervalles minimale d'un graphe quelconque.

Rappelons qu'un graphe G est d'intervalles si et seulement si il possède une chaîne de cliques (théorème 2.9). Les nœuds de cette chaîne sont les cliques maximales de G. Nous noterons $\mathcal{P}_G$ (ou simplement $\mathcal{P}$) une chaîne de cliques de G.

Supposons que nous ayons calculé la complétion H et considérons une chaîne de cliques $\mathcal{P}_H$ de H. Par définition d'une chaîne de cliques, les cliques contenant x forment une sous-chaîne (connexe) P_x de $\mathcal{P}_H$. Revenons maintenant au graphe G. En supprimant x de tous les sacs de $\mathcal{P}_H$ et en supprimant si besoin tous les sacs qui ne correspondent plus à des cliques maximales de G, nous obtenons une chaîne de cliques $\mathcal{P}_G$ du graphe G. Nous dirons que $\mathcal{P}_G$ a été obtenue en *élaguant* le sommet x de $\mathcal{P}_H$.

Définition 5.2 *Une complétion d'intervalles H de G'* respecte *une chaîne de cliques $\mathcal{P}_G$ de G si $\mathcal{P}_G$ peut être obtenue en élaguant x à partir d'une certaine chaîne de cliques de H.*

Clairement les sacs qui contenaient x forment encore une sous-chaîne de $\mathcal{P}_G$. Notre objectif est de réaliser l'opération inverse : trouver une chaîne de cliques $\mathcal{P}_G$ de G et une sous-chaîne telles que, en ajoutant x aux sacs de cette sous-chaîne[1], on obtienne une complétion d'intervalles minimale H de G', qui respecte $\mathcal{P}_G$.

Une chaîne de cliques $\mathcal{P}_G$ est appelée *bonne* s'il existe une complétion d'intervalles minimales de G' qui la respecte. Intuitivement, une chaîne de cliques est bonne si la sous-chaîne formée par les sacs auxquels nous devons ajouter x est minimale. La difficulté sera de trouver une bonne chaîne de cliques de G. Notons en effet qu'un graphe d'intervalles peut avoir une quantité exponentielle de chaîne de cliques ; il nous faut une méthode pour en trouver une bonne, en temps polynomial. La figure 5.1 illustre deux chaînes de cliques d'un même graphe G, avec les intervalles (sous-chaînes) auxquels il faut rajouter x pour obtenir une complétion qui respecte chacune des chaînes (x est adjacent dans G' aux sommets a et c). Clairement la première des deux n'engendre pas une complétion d'intervalles minimale.

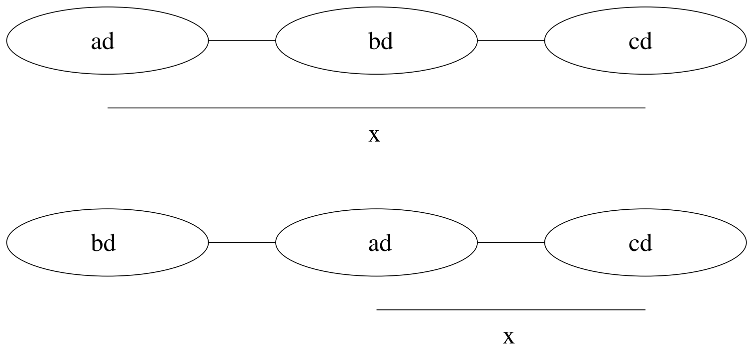

FIG. 5.1 – Chaînes de cliques de G et complétions de G'.

Il existe une façon bien connue pour représenter de manière compacte toutes les chaînes de clique d'un graphe d'intervalles, à l'aide de PQ-arbres (cf. par exemple [66] pour une description de ces outils). Nous avons vérifié, suite à une remarque de Christophe Paul, que cette structure de données peut servir de base pour la recherche d'une bonne chaîne de cliques. Cependant je donnerai ici une autre méthode pour "fouiller" parmi les chaînes de cliques, car cette méthode est simple et générale. Les PQ-arbres peuvent être considérés comme simples d'utilisation, mais uniquement si l'on s'en sert comme d'une boîte noire ; en réalité leur implantation est plutôt fastidieuse. L'algorithme que nous donnerons est basé uniquement sur la décomposition du graphe G en 1-blocs (calcul de composantes connexes), le calcul d'un préordre sur ces blocs (par des tests d'inclusion d'ensembles) et enfin sur une décomposition de Dilworth (en suites croissantes) de ce préordre, simple aussi puisqu'il s'agit d'un préordre de largeur deux.

Commençons par quelques observations sur les chaînes de cliques. Soit S un séparateur minimal du graphe d'intervalles G. Soient $\mathcal{C}(S) = \{C_1, \ldots, C_p\}$ les composantes connexes de $G - S$, et notons $S_i = N(C_i)$. Notons $\mathcal{B}(S) = \{S_1 \cup C_1, \ldots, S_p \cup C_p\}$ l'ensemble des 1-blocs associés à S. Chaque clique maximale de G est contenue dans exactement un de ces blocs, ainsi $\mathcal{B}(S)$ induit une partition sur les cliques maximales de G. La propriété suivante

[1]et si besoin en créant de nouveaux sacs correspondant aux séparateurs bordant la sous-chaîne, dans lesquels le sommet x serait ajouté

nous indique que, dans toute chaîne de cliques $\mathcal{P}_G$ de G, les cliques maximales appartenant à un bloc $B \in \mathcal{B}(S)$ apparaissent de façon contiguë.

Lemme 5.3 ([76]) *Soit B un bloc d'un graphe d'intervalles G. Pour toute chaîne de cliques $\mathcal{P}_G$, les cliques maximales de G contenues dans B forment une sous-chaîne connexe P_B de $\mathcal{P}_G$.*

Soit $\mathcal{P}_G$ une chaîne de cliques et S un séparateur minimal du graphe G. Il existe une branche e_S de $\mathcal{P}_G$ telle que S soit l'intersection des deux cliques incidentes à cette branche (cf. théorème. 2.19). D'après le lemme précédent, les blocs de $\mathcal{B}(S)$ partitionnent la chaîne de cliques en sous-chaînes $P_{B_1}, \ldots, P_{B_p}$. La branche e_S ne se trouve sur aucune de ses sous-chaînes, sinon S séparerait deux sommets d'un même bloc (voir [75] pour plus de détails). Nous partitionnons $\mathcal{B}(S)$ en deux listes, la liste L des blocs se trouvant à gauche de e_S et la liste R des blocs à droite de e_S. De plus les blocs sont ordonnés à partir des extrémités de la chaîne, vers e_S.

Définition 5.4 *Soient G un graphe d'intervalles, S un séparateur minimal de G, $\mathcal{P}_G$ une chaîne de cliques et e_S une branche de cette chaîne de cliques correspondant à S.*

Notons L la liste des blocs de $\mathcal{B}(S)$ situés à gauche de e_S, dans l'ordre dans lequel ils sont rencontré dans $\mathcal{P}_G$ en parcourant la chaîne de gauche à droite. De façon symétrique, notons R la liste des blocs à droite de e_S, dans l'ordre dans lequel ils sont rencontrés en parcourant la chaîne de droite à gauche.

Nous dirons que la paire (L, R) est compatible *avec la chaîne de cliques $\mathcal{P}_G$.*

Une partition de $\mathcal{B}(S)$ en deux ensembles totalement ordonnés (L, R) est appelée représentation ordonnée valide *de $\mathcal{B}(S)$ si (L, R) est compatible avec une chaîne de cliques de G.*

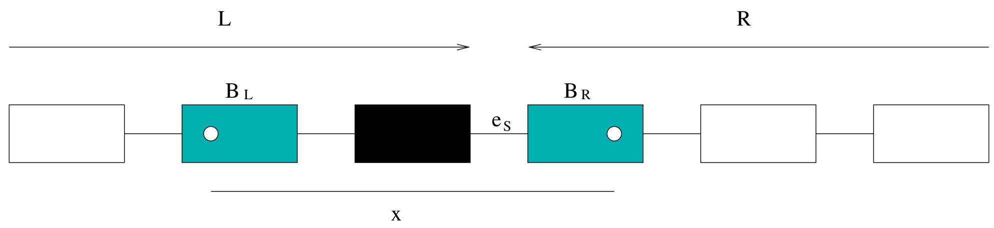

FIG. 5.2 – Représentation ordonnée valide et sa complétion optimale.

D'après les propriétés d'une chaîne de clique, pour qu'un bloc B' puisse apparaître entre un bloc B et la branche e_S, il faut que l'ensemble de sommets $B \cap S$ soit contenu dans toutes les cliques maximales de $G[B']$. Nous écrirons dans ce cas $B \preceq B'$, et il est facile de vérifier que cette relation définit un préordre sur $\mathcal{B}(S)$.

Définition 5.5 *Deux blocs B et B' de $\mathcal{B}(S)$ satisfont la relation $B \preceq B'$ si $B \cap S$ est contenu dans chaque clique maximale de $G[B']$.*

De manière équivalente, $B \preceq B'$ si et seulement si $B \cap S$ est contenu dans le voisinage de chaque sommet de B.

La condition $B \preceq B'$ est nécessaire pour que B' puisse apparaître entre B et e_S, mais pas suffisante. Le théorème suivant caractérise les représentations ordonnées valides. C'est le résultat important qui nous permettra par la suite de trouver efficacement une bonne chaîne de cliques.

Théorème 5.6 (représentations ordonnées valides) *Soient L, R deux listes avec des éléments dans $\mathcal{B}(S)$ telles que chaque bloc appartienne exactement l'une des deux. Le couple (L, R) est une représentation ordonnée valide de $\mathcal{B}(S)$ si et seulement si L et R sont des suites croissantes pour la relation $\preceq$ et le dernier bloc de chaque liste contient S.*

Preuve. (éléments.) La condition $B \preceq B'$ étant nécessaire pour que le bloc B' puisse apparaître entre B et e_S, les listes L et R doivent être des suites croissantes pour $\preceq$. De plus, pour le dernier bloc B de L (ou de R), la clique de $G[B]$ incidente à e_S contient S.

Réciproquement, soient $L = (B_1^L, \dots B_q^L)$ et $R = (B_1^R, \dots, B_r^R)$ deux suites croissantes qui partitionnent $\mathcal{B}(S)$. On peut montrer ([76]) que pour tout bloc $B \in \mathcal{B}(S)$, il existe une chaîne de cliques $\mathcal{P}_{G[B]}$ telle que la clique la plus à droite contient $B \cap S$. En concaténant $\mathcal{P}_{G[B_1^L]}, \dots \mathcal{P}_{G[B_q^L]}$ dans cet ordre avec $\mathcal{P}_{G[B_r^R]}, \dots \mathcal{P}_{G[B_1^R]}$, on obtient une chaîne de cliques de G. Puisque $S = B_q^L \cap B_r^R$, la branche faisant la jonction entre les blocs de gauche et les blocs de droite correspond bien au séparateur S. $\diamond$

En d'autres termes, une représentation ordonnée valide de $\mathcal{B}(S)$ est simplement une décomposition "à la Dilworth" (en suites croissantes) de la relation de préordre $\preceq$. Nous en déduisons que la largeur de la relation $\preceq$ est au plus deux. En effet deux blocs incomparables pour la relation $\preceq$ doivent se trouver de côtés opposés de la branche e_S. Par conséquent, il ne peut pas y avoir trois blocs mutuellement incomparables.

Avec cet outil de représentation des chaînes de cliques pourvu par le théorème 5.6, revenons à notre problème d'origine. Rappelons que nous devons rajouter au graphe d'intervalles G un sommet x, en obtenant ainsi un graphe G', et que nous souhaitons calculer une complétion d'intervalles minimale H de G' où toutes les arêtes ajoutées sont incidentes à x. Nous distinguerons un cas "simple", où le voisinage $N_{G'}(x)$ forme une clique dans G, et un cas plus "élaboré" où nous serons amenés à utiliser la décomposition de Dilworth.

Dans le premier cas, il suffira de rajouter toutes les arêtes entre x et un séparateur minimal S de G [76]. Pour trouver ce séparateur, il suffit de les essayer tous (il y en a au plus $n - 1$) et de prendre celui qui donne une complétion d'intervalles, minimale par inclusion parmi ces possibilités.

Théorème 5.7 ([76]) *Si $N_{G'}(x)$ induit une clique dans le gaphe G, il existe un séparateur minimal S de G tel que le graphe H obtenu à partir de G en rajoutant toutes les arêtes entes x et S soit une complétion d'intervalles minimale de G. De plus, ce séparateur peut être calculé en temps polynomial.*

Supposons désormais que $N_{G'}(x)$ n'est pas une clique de G. Il y a alors deux sommets $a, b \in N_{G'}(x)$, non adjacents dans G. Le théorème suivant nous indique que, pour tout a, b-séparateur minimal S de G, nous sommes obligés de rajouter toutes les arêtes entre x et S. Le théorème est énoncé en termes de triangulations de G', bien entendu toute complétion d'intervalles est également une triangulation.

Théorème 5.8 ([13]) *Soient deux sommets a et b de $N_{G'}(x)$, non adjacents dans G. Pour toute triangulation H_{tr} de G' telle que $H_{tr} - x = G$, S est contenu dans le voisinage de x dans H_{tr}.*

Fixons un séparateur minimal S de G, comme dans le théorème 5.8. Nous dirons qu'un bloc B de $\mathcal{B}(S)$ est *touché* si x a un voisin dans $B \backslash S$, dans le graphe G'. Sinon, le bloc est *non touché*.

Etant donnée une représentation ordonnée valide (L, R) de $\mathcal{B}(S)$, le lemme suivant caractérise les complétions d'intervalles H de G', ayant une chaîne de cliques compatible avec (L, R) (après élagage du sommet x) et qui sont minimales par inclusion pour cette propriété. Nous dirons dans ce cas que H est *optimale pour* (L, R) (voir également la figure 5.2).

Lemme 5.9 *Soit (L, R) une représentation ordonnée valide de $\mathcal{B}(S)$. Soient B_L et B_R les premiers blocs touchés de L, respectivement de R [2]. Une complétion d'intervalles H est optimale pour (L, R) si et seulement si :*

1. *Pour tout bloc B apparaissant après B_L dans L ou après B_R dans R, le bloc B est rempli dans H, i.e. $B \subseteq N_H(x)$.*
2. *Pour tout bloc B apparaissant avant B_L dans L ou avant B_R dans R, B reste non touché dans H, i.e. $B \cap N_H(x) \subseteq S$.*
3. *Pour tout $B \in \{B_L, B_R\}$, le graphe $H_d[B \cup \{x\}]$ est une complétion d'intervalles minimale de $G'_d[B \cup \{x\}]$. Ici $H_d[B \cup \{x\}]$ et $G'_d[B \cup \{x\}]$ sont obtenus à partir de $H[B \cup \{x\}]$ (resp $G'[B \cup \{x\}]$) en ajoutant un sommet fictif d, adjacent à $B \cap S$ et à x.*

Preuve. (éléments.) Soit H une complétion d'intervalles optimale pour (L, R) et $\mathcal{P}_G$ une chaîne de cliques de G compatible avec (L, R), obtenue à partir d'une chaîne de cliques de H après élagage de x. Par définition de B_L et B_R, x a un voisin $y_L \in B_L \backslash S$ et un voisin $y_R \in B_R \backslash S$, dans le graphe x. C'est pourquoi x a été présent, avant élagage, dans un sac contenant y_L et dans un sac contenant y_R. Ces sacs sont contenus dans les blocs B_L et B_R respectivement, donc x était dans tous les sacs des blocs dans l'intervalle ouvert entre B_L et B_R de $\mathcal{P}_G$. Ceci prouve le premier point.

Pour la deuxième propriété, soit H' une complétion d'intervalles quelconque qui respecte G et $\mathcal{P}_G$. En supprimant x de tous les sacs à l'extérieur de l'intervalle fermé entre

[2] Si B_R n'est pas défini, on peut l'imaginer comme ajouté à la fin de la liste R ; ainsi, il n'y a pas de blocs de R après B_R, et l'ensemble des blocs de R apparaissant avant B_R est R lui-même. Le cas où B_L n'est pas défini est traité de façon symétrique.

B_L et B_R de $\mathcal{P}_G$, nous avons toujours une complétion d'intervalles de G', satisfaisant le deuxième point. Notre complétion optimale H satisfait donc cette condition.

Le troisième point se montre en vérifiant que si $H_d[B \cup \{x\}]$ contient strictement une complétion d'intervalles H'_d de $G'_d[B \cup \{x\}]$, on peut supprimer dans H toutes les arêtes de $H_d[B \cup \{x\}]$ qui n'apparaissent pas dans H'_d. Pour les détails et pour l'implication inverse, voir [76]. $\diamond$

Par le lemme 5.9, nous devons calculer une représentation ordonnée valide (L, R) de sorte à maximiser l'ensemble des blocs qui apparaissent avant B_L dans L ou avant B_R dans R. Notre algorithme de calcul mettra au début des listes L et R le plus de blocs non touchés possibles. Il reste un détail à régler, lié aux blocs B_L et B_R, qui ont un statut particulier : nous voulons savoir si nous avons la possibilité d'épargner (de ne pas ajouter à la complétion) au moins une arête entre x et un sommet d'un de ces blocs.

Définition 5.10 *Nous dirons qu'un bloc B de $\mathcal{B}(S)$* peut être épargné *s'il existe une completion d'intervalles H_d de $G'_d[B \cup \{x\}]$ telle que x ne soit pas adjacent à tous les sommets de B, dans le graphe H_d. Comme dans le lemme 5.9, $G'_d[B \cup \{x\}]$ est le graphe obtenu à partir de $G'[B \cup \{x\}]$ e rajoutant un sommet fictif d, adjacent à x et à $B \cap S$.*

On peut vérifier en temps polynomial si un bloc peut être épargné [76]. Basé sur le lemme 5.9, le théorème suivant caractérise les bonnes représentations ordonnées (L, R), c'est-à-dire celles à partir desquelles nous pourrons construire une complétion d'intervalles minimale.

Théorème 5.11 *Soit (L, R) une représentation ordonnée valide de $\mathcal{B}(S)$. On note* $\mathrm{Ep}(L, R)$ *l'ensemble des blocs B apparaissant avant B_L dans L, avant B_R dans R, ou appartenant à $\{B_L, B_R\}$ et pouvant être épargnés.*

Le couple (L, R) respecte une bonne chaîne de cliques si et seulement si l'ensemble $\mathrm{Ep}(L, R)$ *est maximal par inclusion parmi toutes les représentations ordonnées valides. Dans ce cas, toute complétion d'intervalles H optimale pour (L, R) est une complétion d'intervalles minimale de G'.*

Notre algorithme calcule une représentation ordonnée valide (L, R) tout en maximisant de façon gloutonne l'ensemble $\mathrm{Ep}(L, R)$. Rappelons que (L, R) est une décomposition de en suites croissantes de la relation $\preceq$. Avant de le décrire, faisons l'observation suivante. Supposons que nous devons mettre dans la liste L trois blocs B_1, B_2, B_3, deux à deux comparables pour la relation $\preceq$. Supposons que B_1 est non touché, que B_2 est touché mais peut être épargné et B_3 est touché et ne peut pas être épargné. Toute liste L autre que (B_1, B_2, B_3) nous obligerait à rajouter des arêtes qui ne sont pas nécessaires. Nous affinons la relation $\preceq$ et mettant, en cas d'équivalence pour $\preceq$, les blocs non touchés avant les touchés que l'on peut épargner, eux-mêmes avant les touchés que l'on ne peut pas épargner. L'algorithme utilise, en plus de $\preceq$, cette relation affinée $\leq$.

L'algorithme `BonneRepresentationOrdonnee` produit une bonne représentation ordonnée (L, R) – c'est à dire une représentation ordonnée valide comme dans le théorème 5.11, à partir de laquelle on peut produire une complétion d'intervalles minimale. La preuve complète de l'algorithme est donnée dans [76].

Fonction `BonneRepresentationOrdonnee`

Entrée : $\mathcal{B}(S), \preceq, \leq_{top}$, les blocs associés à S, le préordre $(\mathcal{B}(S), \preceq)$ et l'extension linéaire $\leq_{top}$ du préordre raffiné
Sortie : (L, R) – une bonne représentation ordonnée de $\mathcal{B}(S)$
Variables : M – un tableau indiquant les positions possibles de chaque bloc.

$$M[B] = \begin{cases} \text{“}L\text{"} & \text{- } B \text{ doit être dans } L\,; \\ \text{“}R\text{"} & \text{- } B \text{ doit être dans } R\,; \\ \text{“}L \text{ ou } R\text{"} & \text{- } B \text{ peut être dans } L \text{ ou dans } R. \end{cases}$$

$MinToucheTrouve$ – indique si l'on a trouvé un premier bloc touché
$CoteMinTouche$ – le côté où le premier bloc touché a été placé, initialisé à L mais R aurait convenu également

Procédure forçage(B)
pour chaque $B' \in \mathcal{B}(S)$ t.q. (B incomp. avec B' pour $\preceq$) **faire**
  si ($B \leq_{top} B'$) et ($M[B'] =$ “L ou R") **alors**
    $M[B'] := oppose(M[B])$ $\{oppose(\text{“}L'') =''\ R''$ et vice-versa$\}$
    forcage(B')
fin_procédure

$MinToucheTrouve := faux$
$CoteMinTouche :=$ “L"
$L := \emptyset; R := \emptyset$
pour chaque $B \in \mathcal{B}(S)$ **faire** $M[O] :=$ “L ou R"
pour chaque $B \in \mathcal{B}(S)$ par odre topologique $\leq_{top}$ **faire**
  si ($M[B] =$ “L ou R") **alors**
    si B peut être épargné **alors**
      mettre B à la fin de la liste $oppose(CoteMinTouche)$ et $M[B] :=$
        $oppose(CoteMinTouche)$
      forçage(B)
    sinon
      mettre B à la fin de $CoteMinTouche$ et $M[B] := CoteMinTouche$
      forçage(B)
    fin_si
  sinon
    mettre B à la fin de $M[B]$
  fin_si
  si (non $MinToucheTrouve$ et B est touché) **alors**

$MinToucheTrouve := vrai$
$CoteMinTouche := M[B]$
fin_si
fin_pour
retourner (L, R)

Notre algorithme prend en entrée l'ensemble $\mathcal{B}(S)$, muni du préordre $\preceq$. Nous utilisons également en entrée une extension linéaire $\leq_{top}$ du préordre raffiné $\leq$. L'ordre total $\leq_{top}$ est calculé préalablement, par un tri topologique (arbitraire) de $(\mathcal{B}(S), \leq)$.

Théorème 5.12 *La paire (L, R) produite par l'algorithme* `BonneRepresentationOrdonnee` *est une bonne représentation ordonnée de $\mathcal{B}(S)$. L'algorithme est de complexité $\mathcal{O}(n^3)$.*

Preuve. (éléments.) L'algorithme considère les blocs de $\mathcal{B}(S)$ par ordre topologique $\leq_{top}$ et les traite un à un, avec l'objectif d'obtenir une représentation ordonnée (L, R) comme dans le théorème 5.11. Initialement tous les blocs sont marqués "L ou R", ce qui indique qu'ils peuvent être placés dans L ou dans R. Lorsque l'on décide de placer un bloc dans une liste, d'autres blocs peuvent être forcés d'apparaître dans la même liste ou du côté opposé, à cause des relations d'incomparabilité pour le préordre $\preceq$. Rappelons que si deux blocs sont incomparables par rapport à $\preceq$, ceci implique qu'ils doivent être de côtés opposés du séparateur S. La procédure `forçage` assure que, lorsqu'un bloc B est placé dans l'une des deux listes, tous les blocs B' dont le côté est forcé par B reçoivent la marque correspondante – "L" ou "R" selon la liste dans laquelle ils peuvent être placés.

On peut penser à cet algorithme comme ayant trois phases. Lors de la première, aucun des blocs rencontrés n'est touché. Les blocs sont placés dans l'une des listes L ou R selon les possibilités, de toute manière ils seront antérieurs aux premiers blocs touchés de chaque liste et donc ils resteront épargnés (voire non touchés). Lorsque nous rencontrons le premier bloc touché nous rentrons dans la seconde phase. Ce bloc sera placé dans l'une des deux listes, notée $CoteMinTouche$. Supposons sans perdre en généralité que cette liste est L. Nous restons dans cette seconde phase aussi longtemps que R ne contient aucun bloc touché. Pendant ce temps, si nous rencontrons un bloc B que l'on peut épargner nous essayons de le mettre dans R – car en le mettant dans L, après B_L, nous finirons par rajouter toutes les arêtes entre x et ce bloc. Au contraire, si B ne peut pas être épargné, nous le mettons dans L, en gardant la possibilité d'épargner d'autres blocs. Enfin, la troisième phase commence lorsque chaque liste contient un bloc touché. A partir de ce moment, quel que soit notre choix les blocs B rencontrés seront remplis (on ajoutera toutes les arêtes de x à B). $\diamond$

Une fois calculée la bonne représentation ordonnée (L, R), nous calculons une complétion H optimale pour (L, R) comme dans le lemme 5.9. Cela implique éventuellement de rappeler récursivement l'algorithme de calcul d'une complétion d'intervalles minimale d'un graphe quelconque, sur le ou les graphes $G'_d[B \cup \{x\}]$ avec $B \in \{B_L, B_R\}$ épargnable. Les détails sont donnés dans [76].

En appliquant l'approche incrémentale, nous concluons avec le résultat principal de cette section.

Théorème 5.13 *Il existe un algorithme incrémental qui prend en entrée un graphe quelconque et qui calcule, en temps polynomial, une complétion d'intervalles minimale de ce graphe.*

5.2 Une approche par l'ordonnancement des sommets

Les graphes d'intervalles peuvent être caractérisés par un certain type d'ordre total sur les sommets, de la même manière dont les graphes triangulés sont caractérisés par les ordre d'élimination simplicielle. Cette caractérisation nous permettra de donner un algorithme bien plus simple de calcul d'une complétion d'intervalles minimale, de complexité $\mathcal{O}(nm)$.

Avant de reparler de complétions d'intervalles minimales, nous illustrons cette approché sur les complétions d'intervalles propres minimales. Rappelons qu'un graphe H est dit graphes d'intervalles propres s'il possède une représentation d'intervalles telle qu'aucun intervalle ne soit strictement contenu dans un autre. Ce sont également les graphes d'intervalles unitaires (ayant un modèle d'intervalles de même longueur), ou encore les graphes d'intervalles sans griffe induite, la griffe étant le graphe biparti complet $K_{1,3}$ [66]. Comme d'habitude, nous dirons qu'un graphe H est une *complétion d'intervalles propre* de G si H est un graphe d'intervalles propres, obtenu en rajoutant des arêtes au graphe G. Le graphe H est une *complétion d'intervalles propres minimale* de G si aucun sous-graphe strict de H n'est une complétion d'intervalles propres de G.

Nous donnerons un algorithme de complexité linéaire qui prend en entrée un graphe quelconque G et qui calcule, en temps linéaire, une complétion d'intervalles propres minimale de G. Notre intérêt pour ce problème vient du problème de la *largeur de bande* (en anglais : bandwidth). La largeur de bande d'un graphe quelconque G, est le minimum, parmi toutes les complétions d'intervalles propres H de G, de $\omega(H) - 1$ [84] (noter la similarité frappante avec la largeur arborescente et la largeur linéaire). Ce problème est très classique en analyse numérique. En effet, si la matrice d'adjacence de G représente les éléments non nuls d'une matrice symétrique M, calculer la largeur de bande de G revient à effectuer une (même) permutation sur les lignes et les colonnes de M afin que tous ses éléments non nuls se trouvent sur une bande la plus étroite possible autour de la diagonale principale. Avec les notations de la section suivante, cette permutation est exactement l'ordre σ, bicompatible pour la complétion d'intervalles optimale H de G. Le problème du calcul de la largeur de bande est particulièrement difficile : il reste NP-complet même pour la classe des arbres [106]. C'est d'autant plus naturel de chercher des solutions heuristiques, et c'est précisément ce que nous faisons en calculant une complétion d'intervalles propres minimale.

5.2.1 Ordre d'intervalles propres

Définition 5.14 ([109]) *Soit $G = (V, E)$ un graphe et $\sigma = (v_1, v_2, \ldots, v_n)$ un ordre total sur ses sommets. Si à la fois σ et l'ordre renversé de σ sont des ordres d'élimination simplicielle pour le graphe G, alors σ est appelé* ordre bicompatible.

Théorème 5.15 ([109]) *H est un graphe d'intervalles propres si et seulement si il possède un ordre bicompatible.*

La proposition suivante est une caractérisation équivalente des ordres bicompatibles. Dans ce travail, nous utiliserons plutôt cette nouvelle propriété.

Lemme 5.16 ([109]) *Soit $H = (V, F)$ un graphe d'intervalles propres. Un ordre total sur ses sommets $\sigma = (v_1, v_2 \ldots, v_n)$ est bicompatible si et seulement si, pour chaque arête $v_i v_l \in F$, on a également $v_j v_k \in F$ pour tous i, j, k, l, $1 \leq i \leq j < k \leq l \leq n$.*

En d'autres termes, si les sommets v_i et v_l sont adjacents dans H, l'ensemble de sommets $\{v_i, v_{i+1}, \ldots, v_l\}$ forme une clique.

Pour un graphe quelconque, nous pouvons forcer n'importe quel ordre sur les sommets à devenir bicompatible, en rajoutant des arêtes et en obtenant ainsi une complétion d'intervalles propres.

Définition 5.17 *Soit $G = (V, E)$ un graphe quelconque et $\sigma = (v_1, \ldots, v_n)$ un ordre total sur ses sommets. Le graphe* $\mathrm{CIP}(C, \sigma) = (V, F)$ *est défini par*

$$F = \{v_j v_k \mid \textit{ il existe } i, l \textit{ tels que } 1 \leq i \leq j < k \leq l \leq n \textit{ et } v_i v_l \in E\}.$$

Comme conséquence immédiate du lemme 5.16 et du théorème 5.15, $\mathrm{CIP}(C, \sigma)$ est une complétion d'intervalles propres de G, mais pas forcément minimale. Cependant, toute complétion d'intervalles propres minimale peut être obtenue de cette façon.

Théorème 5.18 *Soit $G = (V, E)$ un graphe quelconque et $H = (V, F)$ une complétion d'intervalles propres de G. Il existe un ordre total σ sur les sommets de G tel que $H =$* $\mathrm{CIP}(C, \sigma)$.

Preuve. Il suffit de choisir un ordre σ bicompatible pour H, on aura $\mathrm{CIP}(C, \sigma) \subseteq H$ et la propriété suit par minimalité de H. $\diamond$

Définition 5.19 *Un ordre σ est appelé* CIP-minimal *si* $\mathrm{CIP}(C, \sigma)$ *est une complétion d'intervalles propres minimale de G. Tout préfixe de σ sera également appelé* préfixe CIP-minimal.

Notre objectif est de calculer un ordre CIP-minimal pour le graphe en entrée. Le premier sommet de l'ordre doit correspondre à une sorte d'extrémité du graphe. L'une des notions fortes d'extrémité d'un graphe est celle de *moplex*, définie par Berry et Bordat [11].

Un *module* du graphe G est un ensemble de sommets M tel que pour toute paire $x, y \in M$, nous ayons $N(x)\backslash M = N(y)\backslash M$. Un module clique est un module induisant un sous-graphe complet. Un module clique maximal par inclusion est simplement appelé *module clique maximal*.

Définition 5.20 ([11]) *Un* moplex *est un module clique maximal M, tel que $N(M)$ soit un séparateur minimal de G. Les sommets d'un moplex sont appelés* sommets moplexiens.

Nous prouverons que tout sommet appartenant à un moplex peut être utilisé comme sommet initial d'un ordre CIP-minimal. Par les résultats de [11], un tel sommet peut être obtenu en temps linéaire comme étant le dernier sommet numéroté par une exécution de `LexBFS`. Nous ne donnerons pas ici la déscription de l'algorithme `LexBFS` [121], par ailleurs très connu ; rappelons simplement qu'il s'agit d'un algorithme de parcours en largeur particulier, qui numérote les sommets du graphe de n à 1. Un sommet numéroté 1 par une exécution de `LexBFS` est appelé *sommet `LexBFS`-terminal*.

Théorème 5.21 ([11]) *L'algorithme `LexBFS` termine toujours sur un sommet moplexien.*

Un moplex M contenant un tel sommet est appelé *moplex `LexBFS`-terminal* (tous les moplexes ne sont pas `LexBFS`-terminaux).

Proposition 5.22 *Soit M un moplex de $G = (V, E)$ et $v \in M$. Il existe un ordre CIP-minimal σ commençant par v, tel que le voisinage de v dans $\mathrm{CIP}(C, \sigma)$ soit le même que dans G. Réciproquement, pour toute complétion d'intervalles minimale H de G telle que $N_G(v) = N_H(v)$, il existe un ordre σ', commençant par v, tel que $H = \mathrm{CIP}(C, \sigma')$.*

Preuve. Soit M un moplex contenant v (en fait ce moplex est unique [11]) et H' le graphe obtenu à partir de G en complétant $V\backslash M$ en une clique. Montrons d'abord que H' est un graphe d'intervalles propres. Notons $S = N_G(M)$. Par définition d'un moplex et par construction de H', ce dernier est formé de deux cliques, $M \cup S$ et $V\backslash S$. Leur intersection est S. Clairement H' est un graphe d'intervalles, sans ensemble stable de taille 3 ou plus. En particulier, H ne contient pas le graphe $K_{1,3}$ (la "griffe") comme sous-graphe induit. Par conséquent H est un graphe d'intervalles, sans griffe, il est donc un graphe d'intervalles propres (cf. [66]). En particulier il existe une complétion d'intervalles propres minimales de G, contenue dans H'.

Soit H une telle complétion, elle satisfait $N_G(v) = N_H(v)$. Par le théorème 5.15, il existe un ordre σ' tel que $H = \mathrm{CIP}(C, \sigma')$. Si tous les sommets apparaissant avant v dans σ' sont des éléments du moplex M, nous pouvons permuter v et le premier sommet sans modifier le graphe $\mathrm{CIP}(C, \sigma')$. De la même manière, si tous les sommets apparaissant après v dans σ' sont dans M, on peut permuter v et le dernier sommet, puis renverser σ', et nous obtenons un ordre total sur les sommets avec la propriété désirée.

Il reste à considérer la situation où il existe deux sommets $a, b \notin M$, tels que $a < v < b$ dans σ'. Le graphe $G - M$ est connexe, donc il existe une chaîne reliant a et b dans $G - M$. Considérons deux sommets a' et b' consécutifs sur cette chaîne, tels que $a' < v < b'$ dans

σ'. Comme a' et b' sont adjacents dans G, nous aurons dans $H = \mathrm{CIP}(C, \sigma')$ les arêtes va' et vb'. Par construction de H, nous avons $a', b' \in N_G(v)$, donc $a', b' \in S$. Rappelons que S est un séparateur minimal de G, il existe donc deux composantes C et D de $G - S$ pleines par rapport à S. Au moins l'une d'entre elles, par exemple C, est différente de M. Soit μ une chaîne de a' à b' dans $G[C \cup \{a', b'\}]$, qui n'utilise pas l'arête $a'b'$ (si une telle arête existe). Comme précédemment, il existe deux sommets a'' et b'', consécutifs dans μ et tels que $a'' < v < b''$ dans σ. Donc v est adjacent, dans H, aux deux sommets a'' et b''. Au moins l'un des deux sommets est dans C, contredisant le fait que $N_H(v) = N_G(v) = M \cup S$. $\diamond$

Maintenant que nous avons le premier sommet v_1 de notre ordre CIP-minimal, nous verrons que cet ordre peut être obtenu par un simple parcours en largeur du graphe G à partir de v_1 (notre parcours aura tout de même quelques règles supplémentaires pour le choix du prochain sommet). Cependant, le théorème caractérisant une famille d'ordres CIP-minimaux sera donné dans la forme la plus générale dont nous disposons. Nous allons procéder en partitionnant l'ensemble de sommets du graphe, les parties étant gardées de façon ordonnée. Cette partition ordonnée est affinée jusqu'à ce qu'elle devienne un ordre total sur V.

Définition 5.23 *Un tuple de sous-ensemble disjoints de V, $OP = (V_1, \ldots, V_k)$, dont l'union est exactement V, est appelé* partition ordonnée *de V.*

Un affinage *de OP est une partition ordonnée obtenue en remplaçant chaque ensemble V_i par une partition ordonnée de V_i.*

Définition 5.24 *Etant donnée une partition ordonnée $OP = (V_1, \ldots, V_k)$, tout tuple $OP' = (V_1, \ldots, V_j)$, avec $0 \leq j \leq k$, est appelé* préfixe *de OP. Nous utilisons la notation $\mathrm{V}(OP')$ pour l'ensemble $\bigcup\{V_i \mid 1 \leq i \leq j\}$.*

En particulier lorsque $OP = (V_1)$ est un singleton, nous écrirons simplement V_1. Si de plus V_1 a un seul sommet x, nous écrirons x plutôt que $\{x\}$. Etant donnés deux tuples $OP' = (V_1, \ldots, V_k)$ et $OP'' = (V_{k+1}, \ldots, V_{k+l})$, leur concaténation $OP = (V_1, \ldots, V_k, V_{k+1}, \ldots, V_{k+l})$ est notée $OP' \bullet OP''$. Remarquons qu'un ordre total $\sigma = (v_1, \ldots, v_n)$ de V est un cas particulier de partition ordonnée.

Notre algorithme de calcul d'un ordre CIP-minimal produira des préfixes CIP-minimaux, en rajoutant à chaque étape un nouveau sommet. Soit ρ le préfixe dont nous disposons à une étape fixée de notre algorithme. L'ensemble des sommets qui ne sont pas encore dans ρ seront gardés dans une partition ordonnée. Afin de décrire cette partition nous avons besoin d'introduire quelques notations.

Définition 5.25 *Soit ρ un préfixe non vide d'un ordre total sur les sommets de $G = (V, E)$. Notons $\mathrm{First}(\rho)$ le premier sommet de ρ ayant un voisin dans $V \setminus \mathrm{V}(\rho)$. Nous définissons le voisinage fort (noté $\mathrm{N_S}(\rho)$), le voisinage faible ($\mathrm{N_W}(\rho)$) et le non-voisinage ($\overline{\mathrm{N}}(\rho)$) comme suit :*

- $\mathrm{N_S}(\rho) = N(\mathrm{First}(\rho)) \setminus \mathrm{V}(\rho)$,
- $\mathrm{N_W}(\rho) = N(\mathrm{V}(\rho)) \setminus \mathrm{N_S}(\rho)$,

– $\overline{\mathrm{N}}(\rho) = V \setminus (\mathrm{V}(\rho) \cup \mathrm{N_S}(\rho) \cup \mathrm{N_W}(\rho))$.

En clair, nous avons partitionné les sommets à l'extérieur de ρ en trois catégories, ceux qui ont $\mathrm{First}(\rho)$ comme voisin (le voisinage fort $\mathrm{N_S}(\rho)$), ceux qui ont des voisins dans ρ, mais pas $\mathrm{First}(\rho)$ (le voisinage faible $\mathrm{N_W}(\rho)$) et ceux qui n'ont pas de voisins dans ρ (le non-voisinage $\overline{\mathrm{N}}(\rho)$).

Définition 5.26 *Nous dirons qu'un ordre σ* respecte *le préfixe ρ si σ est un affinage de $\rho \bullet (\mathrm{N_S}(\rho), \mathrm{N_W}(\rho), \overline{\mathrm{N}}(\rho))$.*

L'objectif est de montrer que si ρ est un préfixe CIP-minimal, il existe un ordre CIP-minimal qui le respecte. C'est un premier pas vers un algorithme qui étend le préfixe ρ, en lui rajoutant un nouveau sommet. Remarquons que tout ordre de parcours en largeur commençant par ρ respecte ce préfixe.

Le prochain lemme nous assure que, parmi les bonnes extensions de ρ, il en existe une dans laquelle ρ est immédiatement suivi par son voisinage fort.

Lemme 5.27 *Soient σ et σ' deux ordres totaux sur les sommets de G ayant ρ comme préfixe commun. Supposons que* $\mathrm{CIP}(C, \sigma') \subseteq \mathrm{CIP}(C, \sigma)$*. Si σ est un affinage de $\rho \bullet (\mathrm{N_S}(\rho), \mathrm{N_W}(\rho) \cup \overline{\mathrm{N}}(\rho))$, alors σ' est également un affinage de $\rho \bullet (\mathrm{N_S}(\rho), \mathrm{N_W}(\rho) \cup \overline{\mathrm{N}}(\rho))$.*

Preuve. Supposons que les deux ensembles $\mathrm{N_S}(\rho)$ et $\mathrm{N_W}(\rho) \cup \overline{\mathrm{N}}(\rho)$ sont non vides, sinon la conclusion est vraie pour tous σ, σ' commençant par ρ. Notons par v_f le sommet $\mathrm{First}(\rho)$.

Par absurde supposons qu'il existe deux sommets $a \in \mathrm{N_S}(\rho)$ et $b \in \mathrm{N_W}(\rho) \cup \overline{\mathrm{N}}(\rho)$ tels que $v_f < b < a$ dans l'ordre σ'. Par conséquent,$v_f b$ est une arête de $\mathrm{CIP}(C, \sigma')$. Cette arête doit être également présente dans $\mathrm{CIP}(C, \sigma)$, il existent donc deux sommets v' et a' adjacents dans G tels que $v' \leq v_f < b \leq a'$ dans l'ordre σ. Par définition de $v_f = \mathrm{First}(\rho)$ nous devons avoir $v' = v_f$. Donc $a' \in \mathrm{N_S}(\rho)$, contredisant le fait que $\mathrm{N_S}(\rho)$ apparaît avant b dans σ. ◇

Lemme 5.28 *Soient σ et σ' deux ordres avec un préfixe commun ρ et tels que* $\mathrm{CIP}(C, \sigma') \subseteq \mathrm{CIP}(C, \sigma)$*. Supposons que σ respecte ρ et soient $u \in \mathrm{N_W}(\rho)$, $w \in \overline{\mathrm{N}}(\rho)$. Alors u apparaît avant w dans σ'.*

Preuve. Par absurde, supposons que w se trouve avant u dans σ'. Soit $u' \in \mathrm{V}(\rho)$ un voisin de u dans G. L'arête wu' existe dans $\mathrm{CIP}(C, \sigma')$, puisque w est entre u' et u dans σ'. D'un autre côté, σ respecte ρ, donc w apparaît dans σ après $\rho \bullet (\mathrm{N_S}(\rho), \mathrm{N_W}(\rho))$. Aucun sommet de ρ n'est adjacent dans G à un sommet apparaissant après w dans σ. Par construction de $\mathrm{CIP}(C, \sigma)$, ce graphe ne contient pas l'arête wu'. ◇

Les lemmes 5.27 et 5.28 impliquent directement la proposition suivante, qui nous indique que, si ρ est un préfixe CIP-minimal, il existe une extension CIP-minimale qui respecte ρ.

Proposition 5.29 *Soient σ et σ' deux ordres totaux sur les sommets de G, ayant ρ comme préfixe commun, et tels que $\mathrm{CIP}(C, \sigma') \subseteq \mathrm{CIP}(C, \sigma)$. Si σ respecte ρ, alors σ' respecte ρ également.*

Le prochain résulta nous suggère que, pour choisir le prochain sommet dans $\mathrm{N_S}(\rho)$, on peut prendre un sommet u tel que $N_G(u) \cap \overline{\mathrm{N}}(\rho)$ soit minimal par inclusion.

Lemme 5.30 *Soient σ et σ' deux ordres totaux sur les sommets de G, ayant ρ comme préfixe commun, et soient deux sommets $u, w \in \mathrm{N_S}(\rho)$. Supposons que $\mathrm{CIP}(C, \sigma') \subseteq \mathrm{CIP}(C, \sigma)$. Si σ respecte $\rho \bullet u$ et w apparaît avant u dans σ', alors $N(w) \cap \overline{\mathrm{N}}(\rho) \subseteq N(u) \cap \overline{\mathrm{N}}(\rho)$.*

Preuve. Par contradiction, soit $w' \in (N(w) \cap \overline{\mathrm{N}}(\rho)) \setminus (N(u) \cap \overline{\mathrm{N}}(\rho))$. Si w' est entre $\mathrm{First}(\rho)$ et u dans σ', alors $\mathrm{CIP}(C, \sigma')$ contient l'arête u, w'. Sinon, u est entre w et w' dans σ' et nous avons la même propriété.

Puisque σ respecte $\rho \bullet u$, le sommet w' apparaît après $\rho \bullet (u, \mathrm{N_S}(\rho \bullet u), \mathrm{N_W}(\rho \bullet u))$. Aucun sommet de $\rho \bullet u$ n'est adjacent dans G à un sommet placé après w' dans σ. Les sommets u et w' ne sont pas adjacents dans $\mathrm{CIP}(C, \sigma)$, contredisant notre hypothèse. $\diamond$

Les résultats précédents nous fournissent une condition suffisante pour qu'un ordre σ soit CIP-minimal.

Théorème 5.31 *Soit $G = (V, E)$ un graphe. Considérons un ordre total $\sigma = (v_1, \ldots, v_n)$ sur les sommets de G tel que v_1 soit un sommet moplexien et, pour tout i, $1 \leq i < n$, nous ayons :*

1. *σ respecte le préfixe $\rho = (v_1, \ldots, v_i)$,*
2. *v_{i+1} est tel que $N(v_{i+1}) \cap \overline{\mathrm{N}}(\rho)$ soit minimal par inclusion parmi tous les sommets de $\mathrm{N_S}(\rho)$.*

L'ordre σ est CIP-minimal.

Preuve. (éléments.)

Supposons que σ n'est pas CIP-minimal, il existe donc un ordre σ' tel que $\mathrm{CIP}(C, \sigma')$ soit un sous-graphe stricte de $\mathrm{CIP}(C, \sigma)$. Prenons σ' avec cette propriété ayant le plus long préfixe commun avec σ. Soit $\rho = (v_1, \ldots, v_p)$ ce préfixe commun maximum.

Par construction de σ, les arêtes de $\mathrm{CIP}(C, \sigma)$ incidentes à v_1 sont des arêtes de G, et par la proposition Proposition 5.22, σ' commence également par v_1. Le préfixe ρ n'est pas vide.

On note $u = v_{p+1}$ le $p + 1$ème sommet de σ et w le $p + 1$ème sommet σ'. Nous allons montrer que l'ordre σ'', obtenu à partir de σ' en permutant u et w, satisfait $\mathrm{CIP}(C, \sigma'') = \mathrm{CIP}(C, \sigma')$. Le préfixe commun de σ'' et σ étant plus long que ρ, ceci mène à une contradiction.

Par la proposition 5.29, σ' respecte ρ. On montre facilement que les seules arêtes qui diffèrent entre $\mathrm{CIP}(C, \sigma'')$ et $\mathrm{CIP}(C, \sigma')$ ont une extrémité située dans l'intervalle entre u et w dans σ' (u et w compris). Soit I cet intervalle. Puisque σ' respecte ρ, nous avons $I \subseteq$

$\mathrm{N_S}(\rho)$. Par le lemme 5.30 et la condition 2 de notre théorème, $N(x) \cap \overline{\mathrm{N}}(\rho) = N(u) \cap \overline{\mathrm{N}}(\rho)$ pour tout $x \in I$. Pour tous les sommets de I, leur voisin dans G situé le plus à droite dans l'ordre σ' est le même. Le voisin le plus à gauche étant $\mathrm{First}(\rho)$, on vérifie que tous les sommets de I ont le même voisinage dans $\mathrm{CIP}(C, \sigma'')$ et $\mathrm{CIP}(C, \sigma')$. Nous en concluons que $\mathrm{CIP}(C, \sigma'') = \mathrm{CIP}(C, \sigma')$ – contradiction. $\diamond$

L'algorithme calculant un ordre CIP-minimal est une traduction du théorème 5.31. Il calcule v_1 par un `LexBFS`, puis lance un parcours en largeur à partir de v_1. Affin de calculer en temps constant un sommet $v \in \mathrm{N_S}(\rho)$ tel que $N_G(v) \cap \overline{\mathrm{N}}(\rho)$ soit minimal par inclusion, on garde pour chaque sommet v le nombre de ses voisins dans $\overline{\mathrm{N}}(\rho)$ (noté $\mathrm{d}_{\overline{\mathrm{N}}}(v)$). Avec les structures de données appropriées (les détails se trouvent dans [113]), nous obtenons un algorithme de complexité linéaire calculant un ordre CIP-minimal. A partir de cet ordre σ, on peut calculer par des techniques standard un modèle d'intervalles propres de la complétion d'intervalles propres minimale $\mathrm{CIP}(C, \sigma)$.

Théorème 5.32 *Il existe un algorithme de complexité linéaire qui, étant donné un graphe quelconque G, calcule un modèle d'intervalles propres d'une complétion d'intervalles propres minimale de G.*

Pour certains graphes, il existe des complétions d'intervalles propres minimales que l'algorithme ne peut pas obtenir. Par exemple, pour le biparti complet $K_{1,4}$, notre algorithme prend d'abord un sommet du stable (le sommet moplexien), puis complète les quatre autres sommets en une clique (cf. figure 5.4).

Une autre complétion d'intervalles propres minimale de $K_{1,4}$ est obtenue en rajoutant stable de taille quatre deux arêtes formant un couplage. Nous pouvons obtenir ce type de complétion si, au lieu de commencer par un sommet moplexien, nous découpons préalablement le graphe avec un séparateur minimal S, et nous décidons quelles sont les composantes de $G - S$ qui vont à droite de S dans l'ordre CIP-minimal et quelles sont celles qui vont à gauche. L'ordre CIP-minimal est alors calculé sur chacune des parties, commençant par les sommets de S.

5.2.2 Ordre d'intervalles

En nous basant sur la caractérisation des graphes d'intervalles par des ordres d'intervalles, nous adaptons la technique ci-dessus au calcul d'une complétion d'intervalles minimale. Remarquons cependant que la situation plus complexe que précédemment.

Définition 5.33 (Ordre d'intervalles [107]) *Un* ordre d'intervalles *sur les sommets d'un graphe $H = (V, F)$ est un ordre total $\sigma = (v_1, v_2, \ldots, v_n)$ de V tel que, pour tout $1 \leq i < j \leq k \leq n$, si $v_i v_k \in F$ alors $v_i v_j \in F$.*

En clair, si v_i est adjacent à v_k et $k > i$, alors v_i est également adjacent à tous les sommets se trouvant strictement entre v_i et v_k dans l'ordre σ.

Algorithme `OrdreCIP-minimal`

Entree : un graphe quelconque $G = (V, E)$;
Sortie : un ordre CIP-minimal de G ;
Structures de données :

$marque$: chaque sommet est marqué non atteint, atteint ou traité
Q est la queue des sommets atteints.
ρ est le préfixe courant (un ordre total sur les sommets traités).
$\mathrm{d}_{\overline{\mathrm{N}}}(v)$ est le nombre de voisins non atteints du sommet v.

Fonction `ChoixSommetSuivant` : choisit un sommet v dans la queue Q
tel que $v \in \mathrm{N}_{\mathrm{S}}(\rho)$ et $\mathrm{d}_{\overline{\mathrm{N}}}(v)$ soit minimum pour cette propriété.

début

calculer un sommet moplexien v_1 ; $\rho := (v_1)$
marquer tous les sommets comme non atteints
initialiser Q avec les voisins de v_1, les marquer comme
atteints, marquer v_1 comme traité
calculer $\mathrm{d}_{\overline{\mathrm{N}}}(x)$ pour chaque sommet x
pour $i := 2$ **à** n **faire**
$v_i :=$ `ChoixSommetSuivant()`
marquer v_i comme traité, $\rho := \rho \bullet v_i$
calculer l'ensemble N_i de voisins non atteints de v_i
marquer les éléments de N_i comme atteints et les enfiler dans Q
pour chaque $y \in N_i$ **pour** chaque $z \in N(y)$ **faire** $\mathrm{d}_{\overline{\mathrm{N}}}(z) := \mathrm{d}_{\overline{\mathrm{N}}}(z) - 1$
retourner ρ

fin

FIG. 5.3 – Algorithme de calcul d'un ordre CIP-minimal

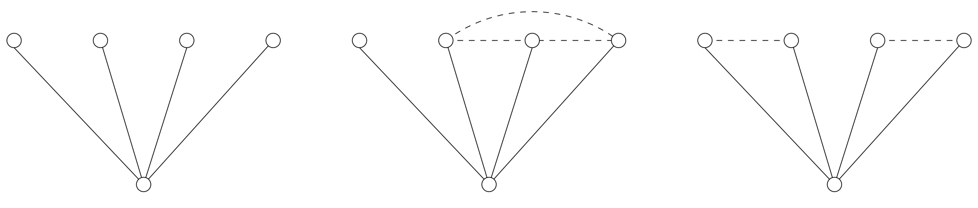

FIG. 5.4 – Complétions d'intervalle propres minimales du $K_{1,4}$.

Théorème 5.34 ([107]) *Un graphe $H = (V, F)$ est un graphe d'intervalles si et seulement s'il existe un ordre d'intervalles sur l'ensemble ses sommets.*

Remarque 5.35 *Etant donné un ordre d'intervalles $\sigma = (v_1, v_2 \dots, v_n)$ d'un graphe H, un modèle d'intervalles de H peut être obtenu en associant à chaque sommet v_i l'intervalle $[i, j]$, où $j > i$ est le plus grand indice tel que v_i et v_j sont adjacents dans H (ou $j = i$ si v_j n'existe pas).*

Réciproquement, à partir d'un modèle d'intervalles de H, il suffit de trier les sommets par début croissant et l'on obtient un ordre d'intervalles.

Définition 5.36 *Soit $G = (V, E)$ un graphe quelconque et soit $\sigma = (v_1, \dots, v_n)$ un ordre total quelconque de V. Le graphe $\mathrm{CI}(G, \sigma) = (V, F)$ est défini par*

$$F = \{v_i v_j \mid \textit{ il existe } k \textit{ tel que } 1 \leq i < j \leq k \leq n \textit{ et } v_i v_k \in E\}.$$

Par le théorème 5.34 $\mathrm{CI}(G, \sigma)$ est un graphe d'intervalles. De la même manière que pour le théorème 5.18, on déduit :

Théorème 5.37 *Soit $G = (V, E)$ un graphe quelconque et $H = (V, F)$ une complétions d'intervalles minimale de G. Il existe un ordre σ sur les sommets de G tel que $H = \mathrm{CI}(G, \sigma)$.*

Définition 5.38 *Un ordre total $\sigma = (v_1, \dots, v_n)$ est appelé* ordre CI-minimal *si* $\mathrm{CI}(G, \sigma)$ *est une complétion d'intervalles minimale de G. Tout préfixe $(v_1, \dots, v_k)$, $k \leq n$ d'un ordre CI-minimal est appelé* préfixe CI-minimal.

Notre objectif est de calculer un ordre CI-minimal pour pour le graphe G.

La première difficulté est de choisir un premier sommet pour notre ordre CI-minimal. Contrairement au cas des ordres CIP-minimaux, il ne suffit pas de prendre n'importe quel sommet moplexien. Cependant, nous montrerons que l'on peut prendre un sommet `LexBFS`-terminal. Nous avons besoin ici de propriétés plus fortes de l'algorithme `LexBFS`.

Lemme 5.39 ([11, 10]) *Soit M un moplex `LexBFS`-terminal et soit $S = N_G(M)$. Notons par $C_1, C_2, \dots, C_k$, avec $C_k = M$, les composantes connexes de $G - S$, dans l'ordre dans lequel elles ont été rencontrées par l'exécution de `LexBFS`. Nous avons les propriétés suivantes :*

- *$N(C_1) \subseteq N(C_2) \subseteq \cdots \subseteq N(C_k)$.*
- *Pour tout i, j, $1 \leq i < j \leq k$ et pour toute paire de sommets $x \in N(C_i)$, $y \in N[C_j] \backslash N(C_i)$, les deux sommets x et y sont adjacents.*

Lemme 5.40 *Considérons un graphe non complet $G = (V, E)$. Soit v un sommet du moplex `LexBFS`-terminal M et $S = N_G(M)$. Il existe une complétion d'intervalles minimale H de G telle que $N_G(v) = N_H(v)$. Pour tout H avec cette propriété, il existe une chaîne de cliques de H telle que $M \cup S$ soit l'une des cliques terminales.*

Preuve. (éléments.) Soient $C_1, C_2, \dots, C_k$, avec $C_k = M$, les composantes de $G - S$, dans l'ordre donné par l'exécution de `LexBFS` qui termine sur v. Elles satisfont le lemme 5.39.

Soit H' le graphe obtenu en transformant $N_G[C_i]$ en une clique, pour chaque i, $1 \leq i \leq k$. Par la première propriété du lemme 5.39, $(N_G[C_1], \ldots, N_G[C_k])$ est une chaîne de cliques de H', en particulier H' est un graphe d'intervalles. Le graphe H' contient une complétion d'intervalles minimale H de G, et clairement $N_G(v) = N_H(v)$.

Soit maintenant H une complétion d'intervalles minimale de G telle que $N_H(v) = N_G(v)$. On montre que S induit une clique dans H, en utilisant le fait que S est contenu dans un séparateur minimal de H et que tous ces séparateurs sont des cliques (voir [126] pour les détails). Par définition d'un moplex, $M \cup S$ induit également une clique dans H.

Pour chaque i, $1 \leq i \leq k$ notons $H_i = H[N_G[C_i]]$. Soit H'' l'union de tous les H_i. Par construction, $G \subseteq H'' \subseteq H$. Nous allons construire une chaîne de cliques $\mathcal{P}$ de H'', montrant que ce dernier est un graphe d'intervalles. Par minimalité de H, ceci implique que $H'' = H$. De plus, notre chaîne de cliques aura $M \cup S = N_G[M]$ comme clique terminale.

Soient $S_i = N_G(C_i)$. Par la deuxième propriété du lemme 5.39, chaque sommet de S_{i-1} est adjacent à chaque sommet de $N_G[C_i] \backslash S_{i-1}$ dans le graphe G – et par conséquent dans H_i. Nous avons que S_{i-1} est contenu dans chaque clique maximale de H_i. On construit ensuite, pour chaque H_i, une chaîne de cliques $\mathcal{P}_i$ telle que S_i est contenu dans la clique la plus à droite de $\mathcal{P}_i$ (voir [126] pour les détails). En concaténant les chaînes de cliques $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$ dans cet ordre, nous obtenons une chaîne de cliques $\mathcal{P}$ de H. Comme H_k est le graphe complet ayant $N_G[M] = S \cup M$ comme ensemble de sommets, la chaîne de cliques $\mathcal{P}$ a comme extrémité droite la clique $S \cup M$. $\diamond$

Théorème 5.41 *Soit G un graphe et v un sommet `LexBFS`-terminal de G. Pour toute complétion d'intervalles minimale H de G telle que $N_G(v) = N_H(v)$, il existe un ordre d'intervalles de H commençant par v.*

Preuve. La propriété est triviale si G est un graphe complet. Sinon, par le lemme 5.40, il existe une chaîne de cliques $\mathcal{P}$ de H telle que son extrémité droite soit $S \cup M$ (avec les notations du lemme 5.40). Par construction, v apparaît uniquement dans cette clique. Renversons $\mathcal{P}$ pour que v soit dans la clique la plus à gauche. Par construction d'un ordre d'intervalles à partir d'un modèle d'intervalles (cf. remarque 5.35), il existe un ordre d'intervalles de H commençant par v. $\diamond$

Comme dans le cas des ordres CIP-minimaux, nous allons voir qu'un ordre CI-minimal peut être obtenu par un parcours en largeur, à partir du premier sommet, avec quelques règles supplémentaires. L'une de ces règles nécessitera de relancer un `LexBFS` à chaque nouveau choix de sommet, ce qui fait que la complexité de l'algorithme est $\mathcal{O}(nm)$. Le résultat combinatoire qui donne une condition suffisante pour qu'un ordre soit CI-minimal est cependant plus large ; il englobe des ordres qui ne sont pas nécessairement des ordres de parcours en largeur.

Notation 5.42 *Soit $\rho = (v_1, \ldots, v_k)$ un préfixe et $R = V \setminus \mathrm{V}(\rho)$. Soit* Nxt *un sous-ensemble non vide de R, tel que* $\mathrm{Nxt} = N_G(v_i) \cap R$ *pour un certain dommet $v_i \in \mathrm{V}(\rho)$ et* Nxt *est minimal par inclusion pour cette propriété. L'ensemble $R \setminus \mathrm{Nxt}$ sera noté* Rst.

Le lemme suivant indique que, si ρ est un préfixe CI-minimal, les sommets de Nxt peuvent être mis de façon contiguë juste après ρ ; il restera à choisir leur ordre.

Lemme 5.43 *Soit σ un ordre total sur les sommets de G, obtenu comme affinage de $\rho \bullet \mathrm{Nxt} \bullet \mathrm{Rst}$, et σ' un ordre total, affinage de $\rho \bullet R$. Si $\mathrm{CI}(G, \sigma') \subseteq \mathrm{CI}(G, \sigma)$, alors σ' est aussi un affinage de $\rho \bullet \mathrm{Nxt} \bullet \mathrm{Rst}$.*

Preuve. Soit $v_i \in \mathrm{V}(\rho)$ tel que $\mathrm{Nxt} = R \cap N_G(v_i)$. Supposons que σ' n'est pas un affinage de $\rho \bullet \mathrm{Nxt} \bullet \mathrm{Rst}$, il y a donc un sommet $u \in \mathrm{Rst}$ et un sommet $w \in \mathrm{Nxt}$ tels que u apparaît avant w dans σ'. Puisque u apparaît dans σ après tous les sommets de Nxt, v_i et u de sont pas adjacents dans $\mathrm{CI}(G, \sigma)$. Dans σ', u est après v_i et avant w. Puisque $\mathrm{Nxt} \subseteq N_G(v_i)$, v_i et w sont adjacents dans G, donc v_i et u sont adjacents dans $\mathrm{CI}(G, \sigma')$ – contredisant notre hypothèse. ◇

Il faut maintenant choisir la meilleure permutation σ_{Nxt} de Nxt, de façon à ce que $\mathrm{CI}(G, \sigma_{\mathrm{Nxt}})$ soit minimal. De plus il faut tenir compte des sommets de Nxt qui ont des voisins dans Rst, et qui pour cette raison formeront une clique dans $\mathrm{CI}(G, \sigma)$. Au lieu d'utiliser $G[\mathrm{Nxt}]$ nous construirons un graphe G_{Nxt} qui permet de prendre en compte cette situation. Les sommets fictifs que nous avons ajoutés à $G[\mathrm{Nxt}]$ sont là pour représenter l'ensemble Rst.

Notation 5.44 *Soit ρ un préfixe. On note T l'ensemble de sommets de Nxt ayant un voisin dans Rst. Notons par G_{Nxt} le graphe obtenu à partir de $G[\mathrm{Nxt}]$ en ajoutant un sommet fictif d_1, adjacent à chaque sommet de T, et un deuxième sommet fictif d_2 adjacent uniquement à d_1.*

Théorème 5.45 *Soit σ un ordre total sur les sommets de G satisfaisant les propriétés suivantes :*

1. *σ commence par un sommet `LexBFS`-terminal v_1.*
2. *Pour chaque préfixe $\rho = (v_1, \ldots, v_i)$ de σ,*
   - *σ est un affinage de $\rho \bullet \mathrm{Nxt} \bullet \mathrm{Rst}$,*
   - *le sommet juste après ρ dans σ est un sommet `LexBFS`-terminal de G_{Nxt}, obtenu en lançant `LexBFS` à partir de d_2.*

L'ordre σ est CI-minimal.

Preuve. (éléments.) La preuve de ce théorème est relativement similaire à celle du cas des ordres CIP-minimaux (théorème 5.31). Cependant les choses sont plus compliquées à cause de la réutilisation de `LexBFS` lors de chaque itération. Signalons simplement que, en relançant `LexBFS` à partir du sommet d_2, c'est comme si nous calculions un ordre CI-minimal pour G_{Nxt}, tout en assurant que les sommets d_1, d_2 sont les derniers de cet ordre. Ainsi, Nxt est ordonné de façon optimale, tout en tenant compte du fait que les sommets de Rst apparaissent après Nxt dans l'ordre σ. La preuve complète se trouve dans [126]. ◇

Algorithme `OrdreCI-minimal`

Entrée : $G = (V, E)$ connexe
Sortie : un ordre σ sur les sommets de G, CI-minimal
soit v_1 le dernier sommet numéroté par une exécution de `LexBFS`sur G
$\rho := (v_1)$
$\mathrm{Nxt} = N_G(v_1); \mathrm{Rst} := V \backslash N_G[v_1]$
$OP := v_1 \bullet \mathrm{Nxt} \bullet \mathrm{Rst}$
pour $i := 2$ **à** n **faire**
    soit Nxt la classe apparaissant juste après ρ dans OP
    soit v_i le dernier sommet numéroté par `LexBFS`
        executé sur G_{Nxt} à partir de d_2 (cf le théorème 5.45)
    $\rho = \rho \bullet v_i$
    si $|\,\mathrm{Nxt}\,| \geq 2$ **alors**
        remplacer Nxt dans OP par $v_i \bullet (\mathrm{Nxt} \backslash \{v_i\})$
    soit C la dernière classe de OP telle que $N_G(v_i) \cap C \neq \emptyset$
    si $C \backslash N_G(v_i) \neq \emptyset$ **alors**
        remplacer C dans OP par $(C \cap N_G(v_i)) \bullet (C \backslash N_G(v_i))$
$\sigma := OP$
retourner σ

FIG. 5.5 – Algorithme `OrdreCI-minimal`

Théorème 5.46 *Il existe un algorithme de complexité $\mathcal{O}(nm)$, calculant une complétion d'intervalles minimale du graphe en entrée.*

Preuve. (éléments.) L'algorithme `OrdreCI-minimal` de la figure 5.5 calcule en $\mathcal{O}(nm)$ un ordre sur les sommets de G satisfaisant les conditions du théorème 5.45.

Le sommet v_1 est clairement `LexBFS`-terminal. L'initialisation assure que σ est un affinage de $v_1 \bullet N_G(v_1) \bullet V \setminus N_G[v_1]$.

L'algorithme maintient une partition ordonnée des sommets de G. Montrons qu'à chaque étape i, l'ensemble Nxt correspond bien au préfixe $\rho = (v_1, \ldots, v_i)$ conformément à la notation 5.42. Par contradiction supposons qu'il existe $j < i$ tel que $N(v_j)$ est strictement contenu dans Nxt. A l'étape j, la classe C de la partition contenant Nxt a été découpée dans $C \cap N_G(v_j)$ et $C \backslash N_G(v_j)$ – contredisant le fait que Nxt est une classe de OP a l'étape i.

Chaque itération de la boucle **pour** coûte $\mathcal{O}(m)$, à cause de l'utilisation de `LexBFS`, ce qui fait un temps global de $\mathcal{O}(nm)$. Remarquons que la mise à jour de la partition (trois dernières lignes de la boucle) se fait aisément en $\mathcal{O}(n)$. En utilisant des techniques d'affinage de partition plus élaborées [71, 69], le coût global de cette partie serait linéaire. Pour améliorer la complexité de l'algorithme, il suffirait de trouver une façon plus rapide pour calculer le premier sommet à choisir dans Nxt.

Le calcul d'un modèle d'intervalles de $\mathrm{CI}(G, \sigma)$ se fait en temps linéaire, comme dans la remarque 5.35. $\diamond$

5.3 Conclusion et problèmes ouverts

L'algorithme incrémental pour le calcul d'une complétion d'intervalles minimales a été le premier algorithme polynomial pour résoudre ce problème. Même si sa preuve est fastidieuse, l'algorithme reste simple. J'ai également la conviction que la représentation des chaînes de cliques par le préordre défini sur un ensembles de blocs du graphe est un outil intéressant en soi. La complexité de l'algorithme peut être ramenée à des valeurs raisonnables (sûrement $\mathcal{O}(n^4)$, vraisembleblement $\mathcal{O}(n^3)$) en utilisant des structures de données plus complexes pour la représentation du préordre [124] ou en remplaçant le préordre par des PQ-arbres. Nous avons vérifié que l'algorithme peut être adapté pour devenir une heuristique gloutonne pour la largeur linéaire ou le profile (le nombre minimum d'arêtes qu'il faut ajouter au graphe quelconque pour obtenir un graphe d'intervalles). En effet, avec de légères modifications, on peut faire qu'à chaque étape incrémentale, le nombre d'arêtes ajoutées ou la clique maximum du nouveau graphe soient minimisées. Par contre, l'ordre d'arrivée des sommets a une grande importance pour les deux paramètres, et nous n'avons pas à ce jour une technique pour choisir judicieusement cet ordre.

L'algorithme utilisant l'ordres d'intervalles est simple et assez efficace. Il n'est pas exclu que sa complexité puisse être améliorée. Plus que la complexité, le problème qui me semble important serait de caractériser *tous* les ordres CI-minimaux (et CIP-minimaux) d'un graphe. Il existe des ordres CI-minimaux qui ne satisfont pas les conditions du théorème 5.45. Par exemple, si le graphe en entrée est un cycle, tous les ordres CI-minimaux obtenus par notre théorème correspond au fait de rajouter un ensemble maximal de diagonales au cycle, sans les croiser et sans créer de triangle de diagonales. On obtient bien une complétion d'intervalles minimale, et toute complétion de ce type peut être obtenue par un ordre satisfaisant le théorème. Malheureusement nous avons des exemples de complétions minimales d'un cycle à huit sommets, qui ne sont pas de ce type.

Dans le chapitre qui suit, nous verrons une autre technique qui permet d'obtenir n'importe quelle complétion d'intervalles minimale d'un graphe quelconque.

Chapitre 6

Pliage de graphes : principes et applications

Nos recherches sur les complétions minimales d'intervalles sont motivées par le calcul des décompositions linéaires optimales (de largeur minimum). Pour avancer vers notre objectif, il serait important de *caractériser* les complétions d'intervalles minimales, en particulier de donner un algorithme capable d'obtenir n'importe quelle complétion de ce type. C'est ce que nous faisons dans ce chapitre, ou nous introduisons la notion de *pliage* d'un graphe. D'abord, nous répondrons à la question suivante : étant donnée une complétion d'intervalles H de G, extraire une complétion minimale $H' \subseteq H$. Nous utilisons ensuite la notion de pliage pour calculer, en temps quadratique, la largeur linéaire des graphes d'intervalles circulaires.

Les résultats de ce chapitre sont basés sur deux rapports de recherche [127, 77], non publiés à ce jour.

6.1 Pliage d'un graphe

Observons que si nous avons une décomposition linéaire $\mathcal{P}$ d'un graphe G, le graphe $H = \texttt{RemplDecLin}(\mathcal{P})$, obtenu en complétant en une clique chaque sac de la décomposition, est un graphe d'intervalles. Une chaîne de cliques de H est obtenue à partir de la décomposition linéaire en supprimant les sacs qui ne sont pas maximaux par inclusion.

Définition 6.1 *Soit G un graphe quelconque et $\mathcal{X} = \{X_1, \ldots, X_p\}$ un ensemble de cliques de G, pas nécessairement maximales. Si chaque arête de G est présente dans au moins une de ces cliques, nous dirons que $\mathcal{X}$ est une* couverture de G par des cliques.

Définition 6.2 *Soit $\mathcal{X}$ une couverture du graphe G par des cliques. Considérons un ordre total $\mathcal{Q}$ sur les cliques de $\mathcal{X}$. Nous dirons que le couple $(G, \mathcal{Q})$ est un* pliage *du graphe G. A chaque pliage on associe un graphe d'intervalles $\texttt{PliageCI}(G, \mathcal{Q})$, défini par l'algorithme `PliageCI` de la figure 6.1.*

Le graphe $\texttt{PliageCI}(G, \mathcal{Q})$ construit bien un graphe d'intervalles. Nous avons initialisé une décomposition linéaire en utilisant pour les sacs les ensembles de $\mathcal{Q}$, en respectant cet ordre. Pour chaque sommet x de G nous l'avons forcé apparaître dans des sacs contigus, en le rajoutant là où nécessaire. Enfin, nous avons rempli cette décomposition linéaire pour obtenir un graphe d'intervalles. Remarquons que l'algorithme produit, en plus de la complétion d'intervalles H, une décomposition linéaire $\mathcal{P}$ de ce graphe dont les sacs correspondent à des cliques.

Algorithme `PliageCI`

Entrée : un pliage (G, Q) ;
Sortie : Une complétion d'intervalles H de G ;

// On initialise une décomposition linéaire par les sacs de $\mathcal{Q}$,
   // en respectant l'ordre
$\mathcal{P} := \mathcal{Q}$;
pour chaque sommet x de G **faire**
   $s := \min\{i \mid x \in \mathcal{Q}(i)\}$;
      //$\mathcal{Q}(i)$ est le $i$ème ensemble de $\mathcal{Q}$
   $t := \max\{i \mid x \in \mathcal{Q}(i)\}$;
   pour $j := s + 1$ **à** $t - 1$ **faire**
      $\mathcal{P}(j) := \mathcal{P}(j) \cup \{x\}$;
fin_pour
$H :=$ `RemplDecLin`$(\mathcal{P})$;
retourner$H$

FIG. 6.1 – L'algorithme de pliage `PliageCI`.

Théorème 6.3 *Soit G un graphe et $\mathcal{X}$ une couverture de G par des cliques. Pour toute complétion d'intervalles H de G, il existe un ordre total $\mathcal{Q}$ sur $\mathcal{X}$ tel que $H = \texttt{PliageCI}(G, \mathcal{Q})$.*

Preuve. (éléments.) Chaque élément de $\mathcal{X}$ est contenu dans au moins une clique maximale de H. Considérons une chaîne de cliques $\mathcal{P}$ de H. On ordonne les éléments de $\mathcal{X}$ par l'ordre de leur première apparition (comme sous-ensemble d'un sac) dans $\mathcal{P}$, de gauche à droite. Si plusieurs éléments de $\mathcal{X}$ apparaissent pour la première fois comme sous-ensembles d'un même sac, on fixe un ordre total arbitraire entre ces éléments. On obtient de cette façon un ordre total $\mathcal{Q}$ sur $\mathcal{X}$. Il n'est pas difficile de vérifier que $H' = \texttt{PliageCI}(G, \mathcal{Q})$ est un sous-graphe de H. Par minimalité de ce dernier, nous avons $H = H'$. ◇

6.2 Extraction d'une complétion minimale

Supposons que nous ayons une complétion d'intervalles $H = (V, F)$ du graphe quelconque $G = (V, E)$. Nous voulons savoir si H est une complétion d'intervalles minimale de G, et si ce n'est pas le cas de construire une complétion minimale $H' \subset H$.

Nous pouvons essayer de supprimer une arête $e \in F \backslash E$, vérifier si le nouveau graphe $H - e$ est encore un graphe d'intervalles, puis recommencer jusqu'à ce qu'aucune arête ne puisse être supprimée. Ce processus s'arête après un nombre d'étapes au plus quadratique en $|F \backslash E|$. Par contre, la nouvelle complétion obtenue n'est pas forcément minimale (il existe des exemples simples pour montrer que cette situation peut arriver).

Définition 6.4 *Une complétion d'intervalles $H = (V, F)$ du graphe quelconque $G = (V, E)$ est appelée* quasi-minimale *si, pour toute arête $e \in F \backslash E$, le graphe $H - e$ n'est pas un graphe d'intervalles.*

Supposons dorénavant que la complétion H dont nous disposons en entrée soit quasi-minimale, nous voulons décider si elle est vraiment minimale. Sinon, il existe une complétion strictement contenue dans H. Donnons des noms différents à ces complétions. Nous appelons H_2 la complétion quasi-minimale en entrée, et soit H_0 une complétion minimale de G, contenue dans H (que nous ne connaissons pas ; remarquons cependant que H_2 est également une complétion d'intervalles quasi-minimale de H_0). Avant d'atteindre cette complétion minimale H_0, nous allons probablement découvrir d'autres complétions d'intervalles H_1, avec $H_0 \subseteq H_1 \subset H_2$. Comment calculer H_1 ? Le premier résultat important est que toute complétion quasi-minimale H_2 correspond à un pliage $(H_0, \mathcal{Q}_0)$ de H_0, où $\mathcal{Q}_0$ est une permutation des cliques maximales de Q_0. Il faudra ensuite "déplier" la permutation Q_0 pour obtenir, si ce n'est directement H_0, au moins un graphe d'intervalles H_1 comme ci-dessus. Nous recommençons ensuite ce processus sur H_1 à la place de H_2, et lorsque nous serons bloqués nous saurons que la complétion d'intervalles calculée est minimale.

Théorème 6.5 *Soit H_2 une complétion d'intervalles quasi-minimale du graphe d'intervalles H_0. Il existe un ordre total $\mathcal{Q}$ sur les cliques maximales de H_0 tel que H_2 correspond au pliage $(G, \mathcal{Q})$, i.e. $H_2 = \texttt{PliageCI}(H_0, \mathcal{Q})$.*

Preuve. (éléments.) La preuve que nous avons donné dans [77] se fait en plusieurs étapes. Clairement, chaque clique maximale de H_0 est contenue dans au moins une clique maximale de H_2. Tout d'abord, nous montrons que pour chaque clique maximale Ω de H_2, il existe une clique maximale K de H_0 qui est contenue *uniquement* dans Ω ; nous dirons dans ce cas que K est une *clique principale* de H_0 par rapport à H_2. Comme dans la preuve du théorème 6.3, nous considérons une chaîne de clique $\mathcal{P}_2$ de H_2 et nous trions les cliques principales de H_0 par l'ordre de leur première apparition comme sous-ensemble d'un sac de $\mathcal{P}_2$. Soit $\mathcal{Q}'$ cet ordre total sur les cliques principales de H_0. Nous montrons que H_2 est le graphe obtenu par l'algorithme `PliageCI` appliqué à $(H_0, \mathcal{Q}')$ (même si $\mathcal{Q}'$ n'est pas nécessairement une couverture par cliques de H_0). Ensuite nous étendons cet ordre $\mathcal{Q}'$ en un ordre total $\mathcal{Q}$ sur les cliques maximales de H_0, avec la propriété désirée. $\diamond$

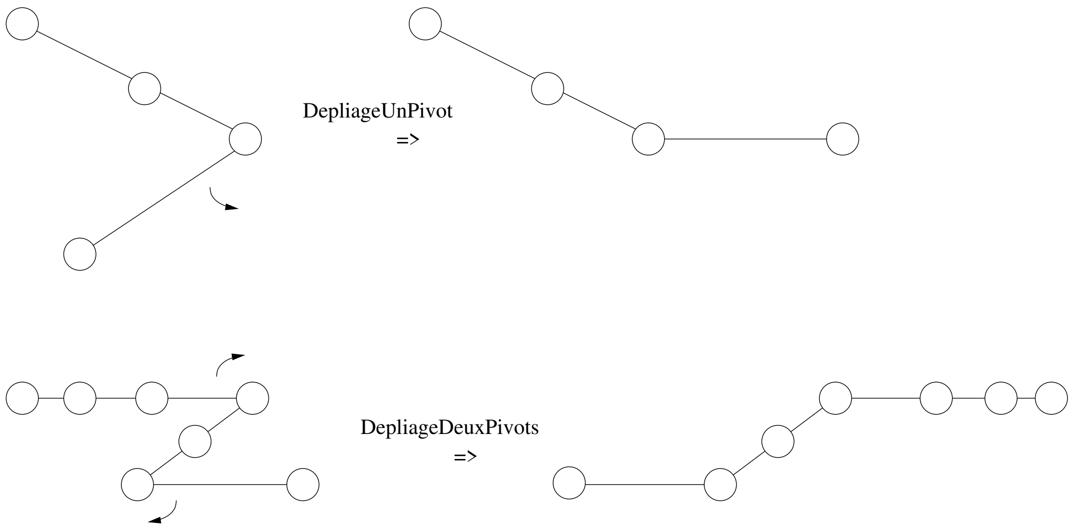

FIG. 6.2 – Chaque cercle représente une clique maximale du graphe H_0, et ces cliques sont reliées selon la chaîne de cliques P_0. La permutation $\mathcal{Q}$ est obtenue en prenant l'ordre des cliques, de gauche à droite sur l'abscisse. Les flèches indiquent l'opération que nous faisons pour déplier un ou deux pivots. La partie haute montre le cas du 1-pliage, où le pivot est la clique extrémale de la permutation. La partie en bas illustre le cas du 2-pliage, avec deux pivots qui sont le plus à gauche, respectivement le plus à droite selon la permutation $\mathcal{Q}$.

Soit donc $H_2 = \texttt{PliageCI}(H_0, \mathcal{Q})$ une complétion quasi-minimale de G. L'objectif est maintenant de *déplier* H_2 en H_0, ou du moins en un graphe d'intervalles H_1 tel que $H_0 \subseteq H_1 \subset H_2$. Fixons une chaîne de cliques $\mathcal{P}_0$ de H_0. Il est désormais préférable de voir le pliage comme un triplet $(H_0, \mathcal{P}_0, \mathcal{Q})$, où $\mathcal{Q}$ est comme précédemment un ordre total sur les cliques maximales de H_0. Ceci permet de prendre en compte la manière dont nous avons plié la chaîne de cliques $\mathcal{P}_0$.

Définition 6.6 *Un* pliage *du graphe d'intervalles H_0 par rapport à une chaîne de cliques $\mathcal{P}_0$ est un triplet $(H_0, \mathcal{P}_0, \mathcal{Q})$, où $\mathcal{Q}$ est une permutation des cliques maximales de H_0. Ce pliage définit un graphe d'intervalles $\texttt{PliageCI}(H_0, \mathcal{Q})$.*

Une clique maximale K de H_0 est un pivot *pour le pliage $(H_0, \mathcal{P}_0, \mathcal{Q})$ si les deux voisins de K dans $\mathcal{P}_0$ apparaissent du même côté de K dans l'ordre $\mathcal{Q}$.*

Rappelons que nous avons en entrée le graphe H_2 et que nous souhaitons calculer H_0. Pour ce faire, nous voulons "déplier" H_2. Un pliage $(H_0, \mathcal{P}_0, \mathcal{Q})$ peut à priori être compliqué, au sens où la permutation $\mathcal{Q}$ mélange fortement les cliques de H_0, par rapport à la permutation donnée par $\mathcal{P}_0$. Pour nos fins algorithmiques, nous préférons les pliages simples, avec un ou deux pivots.

Définition 6.7 *Soit $(H_0, \mathcal{P}_0, \mathcal{Q})$ un pliage de H_0. Si ce pliage contient un seul pivot, il sera appelé un* 1-pliage. *S'il contient exactement deux pivots, il sera appelé* 2-pliage.

Le théorème suivant nous assure que toute complétion d'intervalles quasi-minimale, mais non minimale H_2 de G provient d'un 1-pliage ou d'un 2-pliage d'une complétion H_1 de G, avec $H_1 \subset H_2$. Notons que H_1 n'est pas nécessairement la complétion minimale H_0.

Théorème 6.8 *Soit H_2 une complétion d'intervalles quasi-minimale, mais non minimale du graphe quelconque G. Il existe une complétion d'intervalles H_1 de G, strictement contenue dans H_2, et un 1-pliage ou un 2-pliage $(H_1, \mathcal{P}_1, \mathcal{Q}_1)$ de H_1, satisfaisant la propriété $H_2 = \texttt{PliageCI}(H_1, Q_1)$.*

Nous allons déplier le graphe H_2 en un sous-graphe d'intervalles. L'idée informelle de l'algorithme est décrite dans la figure 6.2. Si nous disposions de la chaîne de cliques $\mathcal{P}_2$ de H_2 produite par l'algorithme `PliageCI` appliqué à (H_1, Q_1), il suffirait pratiquement de deviner la position des pivots et de les "ouvrir" comme sur le dessin. Bien entendu, nous disposons uniquement du graphe H_2 et non pas de la "bonne" chaîne de cliques. Rappelons que le nombre de chaînes de cliques de H_2 peut être exponentiel en sa taille, pas question de tout essayer. Nous allons néanmoins pouvoir déplier le graphe H_2 en un graphe H_1' (pas forcément le H_1 du théorème), en nous assurant que ce dépliage supprime au moins une arête uv.

Considérons d'abord le cas, plus simple, où H_2 provient d'un 1-pliage de H_1. Nous allons deviner le pivot Ω, qui dans ce cas est à la fois une clique maximale de H_1 et de H_2 (les détails se trouvent dans [77]). Nous allons également deviner le sommet u, tel qu'une arête uv soit supprimée pendant le dépliage. Nous allons ensuite déplier la composante connexe C_u de $G - \Omega$, découpant le graphe H_2 en $H_2[N_G[C_u]]$ et $H_2 - C_u$. L'union de ces deux graphes sera le graphe H_1' que nous recherchons. Ce sera bien un graphe d'intervalles, car on peut coller une chaîne de cliques de $H_2[N_G[C_u]]$ avec une chaîne de cliques de $H_2 - C_u$ en identifiant les sacs correspondant à Ω de chaque côté. L'algorithme `DeplierUnPivot` essaye chaque clique maximale Ω et chaque sommet u. Si H_2 provient d'un 1-pliage, l'algorithme trouvera le graphe H_1', sous-graphe strict de H_2. Sinon, il retournera simplement H_2 et il nous faudra passer au cas du 2-pliage.

Lemme 6.9 *Soit $G = (V, E)$ un graphe quelconque. Considérons deux complétions d'intervalles H_1 et H_2 de G telles que H_1 soit un sous-graphe strict de H_2 et H_2 soit quasi-minimale. S'il existe un 1-pliage $(H_1, \mathcal{P}_1, \mathcal{Q}_1)$ de H_1 tel que $H_2 = \texttt{PliageCI}(H_1, \mathcal{Q}_1)$, alors l'algorithme $\texttt{DeplierUnPivot}(H_1, \mathcal{Q}_1)$ produit une complétion d'intervalles H_1' de G, avec $H_1' \subset H_2$.*

La construction qui permet de déplier simultanement deux pivots est sensiblement plus longue que dans le cas précédent. Cependant, le principe reste le même : on devine (en essayant toutes les possibilités) les cliques maximales Ω^l et Ω^r de H_2 qui contiennent les pivots, les sommets u et v tels que l'arête uv sera enlevée et quelques autres éléments qui ne dépendent que des graphes G et H_2. La preuve complète de l'algorithme est assez longue et peut être consultée dans [77], ainsi que l'algorithme `DeplierDeuxPivots`.

Nous avons le lemme suivant, similaire au cas du 1-pliage.

Algorithme `DeplierUnPivot`

Entree : Un graphe $G = (V, E)$, et une complétion d'intervalles H_2 de G
Sortie : Une complétion d'intervalles H_1' de G telle que
$H_1' \subset H_2$ si H_2 est défini par un 1-pliage d'un graphe H_1
$H_1' = H_2$ sinon

pour chaque paire (Ω, u) où Ω est une clique maximale de H et $u \in V \setminus \Omega$
  soit C_u la composante connexe de $G - \Omega$ contenant u
  $H_1' = H_2[N_G[C_u]] \cup (H_2 - C_u)$
  si H_1' est un graphe d'intervalles et $H_1' \subset H_2$ **alors**
    returner H_1'
  fin_si
fin_pour
returner H_2

FIG. 6.3 – Déplier un pivot.

Lemme 6.10 *Soit $G = (V, E)$ un graphe quelconque et soient H_1 et H_2 deux complétions d'intervalles de G. Supposons que $H_1 \subset H_2$ et que H_2 est une complétion quasi-minimale. S'il existe un 2-pliage $(H_1, \mathcal{P}_1, \mathcal{Q}_1)$ de H_1 tel que $H_2 =$ `PliageCI`$(H_1, \mathcal{Q}_1)$, alors le graphe H_1' obtenu par l'algorithme `DeplierDeuxPivots`(G, H_2) est une complétion d'intervalles de G, strictement contenue dans H_2.*

Les lemmes 6.9 et 6.10 impliquent le principal résultat de cette section.

Théorème 6.11 *Il existe un algorithme polynomial prenant en entrée un graphe quelconque G et une complétion d'intervalles H_2 de G, qui calcule une complétion d'intervalles minimale H_1 de G avec $H_1 \subseteq H_2$.*

Preuve. (éléments.) L'algorithme `ExtractionCIMinimale` d'extraction d'une complétion d'intervalles minimale est donné dans la figure 6.4. En partant de la complétion d'intervalles courante H_2, on extrait des arêtes une à une jusqu'à l'obtention d'une complétion quasi-minimale. On essaye ensuite les algorithmes de dépliage d'un, puis de deux pivots. Si l'un de ces algorithmes obtient une complétion strictement plus petite, nous recommençons avec la nouvelle complétion. Si les deux algorithmes ont été incapables de supprimer des arêtes, par le théorème 6.8 et les lemmes 6.9 et 6.10 c'est que la complétion actuelle est bien minimale. $\diamond$

Notons que, si l'algorithme utilise comme complétion initiale de $G = (V, E)$ le graphe

Algorithme `ExtractionCIMinimale`

Entrée : Un graphe $G = (V, E)$, et une complétion d'intervalles H_2 de G.
Sortie : Une complétion d'intervalles minimal H_1 de G, telle que $H_1 \subseteq H_2$.

$H_1 := H_2$
$H_0 := G$
tantque $H_0 \neq H_1$ **faire**
    $H_0 := H_1$
    pour chaque arête uv de $E(H_1) \setminus E(G)$ **faire**
        si $H_1 - uv$ est un graphe d'intervalles **alors**
            $H_1 := H_1 - uv$
    $H_1 :=$ `DeplierUnPivot`(G, H_1)
    $H_1 :=$ `DeplierDeuxPivots`(G, H_1)
retourner H_1

FIG. 6.4 – Extraction d'une complétion d'intervalles minimale.

complet $K(V)$, n'importe quelle complétion d'intervalles minimale de G peut être obtenue en sortie de l'algorithme [77].

6.3 Largeur linéaire des graphes d'intervalles circulaires

Rappelons qu'un graphe d'intervalle circulaire a comme modèle d'intersection une famille d'arcs de cercle d'un cercle. Cette classe généralise la classe des graphes d'intervalles.

Comme nous avons représenté les graphes d'intervalles par une chaîne de cliques, nous pouvons également représenter un graphe d'intervalles circulaires par un *cycle de cliques*.

Définition 6.12 *Un cycle de cliques $\mathcal{CC}$ d'un graphe d'intervalles circulaires G est un cycle dont les noeuds sont des cliques (pas nécessairement maximales) de G, de sorte que pour tout sommet x de G, l'ensemble des sacs contenant x forme une chaîne connexe dans $\mathcal{CC}$, et chaque arête de G est couverte par un sac.*

Etant donné un graphe G, il existe un algorithme de complexité linéaire qui permet de décider si G est un graphe d'intervalles et de calculer un modèle d'intervalles circulaires [104]. A partir de ce dernier, nous pouvons construire le cycle de cliques en temps linéaire. De plus le cycle a au plus $2n$ nœuds.

Fixons désormais un cycle de cliques $\mathcal{CC}$ du graphe G. Chaque arête de G apparaît dans au moins un sac de ce cycle, les sacs forment donc une couverture par cliques de G. Considérons une permutation $\mathcal{Q}$ de l'ensemble de sacs $\mathcal{CC}$. Comme dans la section précédente, nous allons étudier la permutation $\mathcal{Q}$ par rapport au cycle de cliques, c'est pourquoi le pliage du graphe G sera vu comme le un triplet $(G, \mathcal{CC}, \mathcal{Q})$.

Définition 6.13 *Un* pliage *du graphe d'intervalles circulaires G par rapport à un cycle de cliques $\mathcal{CC}$ est un triplet $(G, \mathcal{CC}, \mathcal{Q})$, où $\mathcal{Q}$ est une permutation des sacs de $\mathcal{CC}$. Ce pliage définit un graphe d'intervalles* `PliageCI`$(G, \mathcal{Q})$.

Le sac K de $\mathcal{CC}$ est un pivot *pour le pliage $(G, \mathcal{CC}, \mathcal{Q})$ si les deux voisins de K dans $\mathcal{CC}$ apparaissent du même côté de K dans l'ordre $\mathcal{Q}$.*

*Un pliage ayant exactement k pivots est dit k-*monotone.

Remarquons que tout pliage contient au moins deux pivots : le premier et le dernier élément de $\mathcal{Q}$ sont toujours des pivots. Notre principal résultat combinatoire est qu'il existe une complétion d'intervalles H de largeur minimum de G, obtenue par un pliage 2-monotone (ayant exactement deux pivots).

L'intersection de deux cliques Q, Q', consécutives dans le cycle de cliques, n'est pas forcément un séparateur de G. Nous dirons néanmoins que $Q \cap Q'$ est un semi-séparateur : l'union de deux semi-séparateurs est généralement un séparateur de G.

Regardons de plus près la complétion d'intervalles obtenue par un pliage.

Remarque 6.14 *Soit $(G, \mathcal{CC}, \mathcal{Q})$ un pliage du graphe d'intervalles circulaires G. Considérons la chaîne de cliques $\mathcal{P}$ produite par l'algorithme* `PliageCI`$(G, \mathcal{Q})$*. Observer que chaque sac de $\mathcal{P}$ est l'union d'une clique $Q \in \mathcal{Q}$, correspondant à ce sac au moment de l'initialisation, et les semi-séparateurs de la forme $Q \cap Q'$, où Q, Q' sont des cliques consécutives sur le cycle, mais séparées par Q dans la permutation $\mathcal{Q}$. Nous dirons que la clique S et ces semi-séparateurs ont été* fusionnés *par le pliage.*

Un pliage $(G, \mathcal{CC}, \mathcal{Q})$ définit de façon naturelle une "partie haute" et une "partie basse" sur le cycle de cliques. Soient Q_L et Q_R la première et respectivement la dernière clique de $\mathcal{Q}$. Notons $\mathcal{CC}^{up}$ (resp. $\mathcal{CC}^{down}$) la suite de cliques rencontrées sur le cycle à partir de Q_L et jusqu'à Q_R, en parcourant ce cycle dans le sens horaire (resp. dans le sens trigonométrique). On note $\mathcal{Q}^{down} = (Q_L = Q_{l_1}, Q_{l_2}, \ldots, Q_{l_r} = Q_R)$ la restriction de $\mathcal{Q}$ à $\mathcal{CC}^{down}$. Similairement, $\mathcal{Q}^{up} = (Q_L = Q_{u_1}, Q_{u_2}, \ldots, Q_{u_t} = Q_R)$ est la restriction de $\mathcal{Q}$ à $\mathcal{CC}^{up}$.

Définition 6.15 *Soit $(G, \mathcal{CC}, \mathcal{Q})$ un pliage 4-monotone de G. Notons Q_L et Q_R le premier, respectivement le dernier élément de $\mathcal{Q}$ et soient B_1, P les deux autres pivots, de gauche à droite dans $\mathcal{Q}$. Supposons sans perdre en généralité que B_1, P sont dans $\mathcal{Q}^{up}$. La chaîne du cycle $\mathcal{CC}$ qui commence par B_1 et, dans le sens trigonométrique, traverse P et continue tant qu'elle rencontre des sacs apparaissant après B_1 dans $\mathcal{Q}$, est appelée* anomalie *du pliage (voir aussi la partie haute de la figure 6.5).*

L'un de nous outils principaux, le théorème 6.17, montre que si $(G, \mathcal{CC}, \mathcal{Q})$ est un pliage 4-monotone auquel correspond une complétion d'intervalles $H =$ `PliageCI`$(G, \mathcal{Q})$, il existe un pliage 2-monotone $(G, \mathcal{CC}, \mathcal{Q}')$ tel que $H' =$ `PliageCI`$(G, \mathcal{Q}')$, la complétion

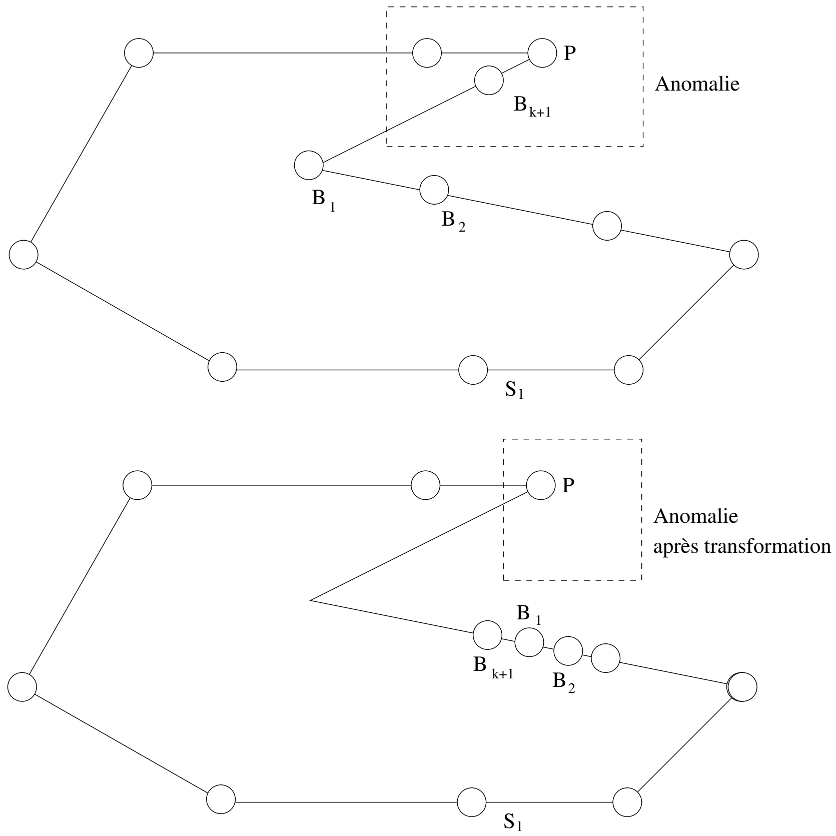

FIG. 6.5 – Anomalie (en haut) et transformation pour réduire cette anomalie

d'intervalles correspondant à ce pliage, a une largeur au plus égale à celle de H. Voici une ébauche informelle des principaux arguments la preuve. Considérons l'anomalie $\mathcal{A}$ du pliage 4-monotone $(G, \mathcal{CC}, \mathcal{Q})$. Supposons qu'elle est dans la partie haute du cycle, comme dans la figure 6.5.

Lorsque l'on restreint $\mathcal{CC}$ aux sacs qui ne sont pas dans l'anomalie (on enlève toutes les cliques $X \in \mathcal{A}$, en ajoutant une arête entre ses voisins), nous obtenons un cycle de cliques correspondant à un graphe $\tilde{G}$, qui est un sous-graphe induit de G. De plus, nous obtenons un pliage $(\tilde{G}, \tilde{\mathcal{CC}}, \tilde{\mathcal{Q}})$ de $\tilde{G}$, où $\tilde{\mathcal{Q}}$ est la restriction de la suite $\mathcal{Q}$ aux sacs qui ne sont pas dans $\mathcal{A}$. Ce pliage définit une complétion d'intervalles $\tilde{H} = \texttt{PliageCI}(\tilde{G}, \tilde{\mathcal{Q}})$.

Définition 6.16 *Soit $(G, \mathcal{CC}, \mathcal{Q})$ un pliage 4-monotone de G et soit $\mathcal{A}$ son anomalie. La* $\mathcal{A}$-largeur *de $(G, \mathcal{CC}, \mathcal{Q})$ est la largeur linéaire de $\tilde{H} = \texttt{PliageCI}(\tilde{G}, \tilde{\mathcal{Q}})$, où $\tilde{G}$ est le graphe d'intervalles circulaires défini par le cycle de cliques $\tilde{\mathcal{CC}}$ obtenu à partir de $\mathcal{CC}$ en supprimant l'anomalie et $\tilde{\mathcal{Q}}$ est la restriction de $\mathcal{Q}$ aux sacs de $\tilde{\mathcal{CC}}$.*

Soit $H = \texttt{PliageCI}(G, \mathcal{Q})$ une complétion d'intervalles optimale de G. Un pas de la procédure consiste a légèrement modifier le pliage $(G, \mathcal{CC}, \mathcal{Q})$ afin d'obtenir un pliage $(G, \mathcal{CC}, \mathcal{Q}')$, avec une anomalie $\mathcal{A}'$ avec strictement moins de sacs que $\mathcal{A}$, en assurant que la $\mathcal{A}'$-largeur de $(G, \mathcal{CC}, \mathcal{Q}')$ n'est pas supérieure à la largeur de H. Nous continuons jusqu'à l'obtention d'une anomalie vide. Ce dernier pliage $(G, \mathcal{CC}, \mathcal{Q}'')$ est 2-monotone. Son anomalie $\mathcal{A}''$ étant vide, la $\mathcal{A}''$-largeur du pliage est exactement la largeur linéaire de $H'' = \texttt{PliageCI}(G, \mathcal{Q}'')$, donc $\mathrm{pw}(H'') \leq \mathrm{pw}(H)$.

Donnons plus d'élements sur la construction de $(G, \mathcal{CC}, \mathcal{Q}')$ à partir de $(G, \mathcal{CC}, \mathcal{Q})$. Considerons les pivots B_1 et P de $(G, \mathcal{CC}, \mathcal{Q})$, comme dans la définition 6.15. Soit B_{k+1} l'unique voisin de B_1 sur le cycle qui appartient à l'anomalie. Soient $B_2, \ldots, B_k$ les cliques, n'appartenant pas à l'anomalie, qui sont après B_1 dans le cycle dans le sens horaires, mais avant B_{k+1} dans $\mathcal{Q}$. Soient $S_1, \ldots, S_{k+1}$ les semi-séparateurs correspondant aux arêtes de la partie inférieure du cycle, tels que S_i est fusionné avec le sac correspondant B_i dans $\texttt{PliageCI}(G, \mathcal{Q})$. Nous choisissons un semi-séparateur $S_l = Q \cap Q'$ correspondant à une arête QQ' en bas du cycle et nous modifions $\mathcal{Q}$ afin de placer tous les $B_{k+1}, B_1, \ldots, B_{l-1}$ (dans cet ordre) entre Q et Q'. Le choix de S_l fera que le nouveau pliage $(G, \mathcal{CC}, \mathcal{Q}')$ aura les propriétés requises. La construction est illustrée par la figure 6.5. Informellement, les sacs $B_{k+1}, B_1, \ldots, B_{l-1}$ glissent (sans sauter) le long du cycle, dans le sens horaire, et s'arrêtent au dessus de l'arête correspondant à S_l. La preuve complète n'est pas très longue mais reste assez technique ; elle se trouve dans [127].

Théorème 6.17 *Soit $(G, \mathcal{CC}, \mathcal{Q})$ un pliage 4-monotone de G et $H = \texttt{PliageCI}(G, \mathcal{Q})$. Il existe un pliage 2-monotone $(G, \mathcal{CC}, \mathcal{Q}'')$ tel que $H'' = \texttt{PliageCI}(G, \mathcal{Q}'')$ ait une largeur linéaire inférieure ou égale à celle de H.*

Par un argument inductif, on montre que tout pliage de G peut être transformé en un pliage 2-monotone, sans augmenter la largeur de la complétion d'intervalles associée.

Théorème 6.18 *Soit G un graphe d'intervalles circulaire de $\mathcal{CC}$ un cycle de cliques de G. Il existe un pliage 2-monotone $(G, \mathcal{CC}, \mathcal{Q})$ de G, tel que $H = \texttt{PliageCI}(G, \mathcal{Q})$ soit une complétion d'intervalles optimale (de largeur linéaire minimum) de G.*

L'algorithme pour le calcul de la largeur linéaire des graphes d'intervalles circulaires est très similaire à l'algorithme de [93] calculant le remplissage minimum pour cette même classe (ce dernier peut également être adapté au calcul de la largeur arborescente).

Considérons un cycle de cliques $\mathcal{CC}$ de G, obtenu à partir d'un modèle d'intervalles. Subdivisons chaque branche du cycle en lui rajoutant un nouveau sac, contenant le semi-séparateur correspondant à la branche. Nous obtenons un *cycle de cliques/semi-séparateurs* alternant les cliques des sacs originels et des sacs contenant des semi-séparateurs. Il est également utile de représenter ce nouveau cycle comme un *polygône (convexe) de points de contrôle* $\mathcal{PG}$ (nous suivons la notion de "polygon of scanpoints" de [93]). Les points de contrôle sont les sacs (cliques et semi-séparateurs) du cycle. On associe à chaque point de contrôle s le sac respectif, noté $V(s)$. Pour chaque triangle T formé par trois points s_1, s_2, s_3 , définissons la largeur $w(T)$ du triangle comme étant la cardinalité de l'union $V(s_1) \cup V(s_2) \cup V(s_3)$.

Définition 6.19 *Une* triangulation planaire linéaire *LP du polygone de points de contrôle $\mathcal{PG}$ est une triangulation planaire de ce polygone telle que chaque triangle de la triangulation contient au plus deux diagonales. La largeur $w(LT)$ de la triangulation est la largeur maximum de ses faces (triangles).*

Une telle triangulation est illustrée par la figure 6.6. Premièrement, nous montrons que la largeur linéaire de G est égale à la plus petite largeur d'une triangulation planaire linéaire de $\mathcal{PG}$, moins un. Par les mêmes techniques que [93], nous obtenons un algorithme quadratique de calcul da la largeur linéaire.

Théorème 6.20 *Pour tout graphe d'intervalles circulaires G, sa largeur linéaire est le minimum, parmi toutes les triangulations planaires linéaires LT de $\mathcal{PG}$, de $w(LT) - 1$.*

Preuve. (éléments.) Soit LT une triangulation planaire linéaire de $\mathcal{PG}$. Considérons un graphe dont les sommets sont les triangles de LT, et deux sommets sont adjacents si et seulement si les triangles respectifs partagent une diagonale. Il est facile de vérifier que ce graphe est une chaîne. Pour chaque nœud T de cette chaîne soient s_1, s_2, s_3 les points de contrôle qui forment le triangle. On associe à T le sac $V(s_1) \cup V(s_2) \cup V(s_3)$. On obtient ainsi une décomposition linéaire de G (cf. [127] pour les détails), par conséquent la largeur de cette décomposition est $w(LT) - 1$.

Réciproquement, soit $H = \texttt{PliageCI}(G, \mathcal{Q})$ une décomposition linéaire optimale de G, où $(G, \mathcal{CC}, \mathcal{Q})$ est un pliage 2-monotone de G. Nous allons construire une triangulation planaire linéaire LT de $\mathcal{PG}$ comme suit (voir également la figure 6.6). Pour chaque clique Ω du cycle de cliques $\mathcal{CC}$, il existe au plus une branche $\Omega'\Omega''$ de ce cycle telle que Ω se trouve entre Ω' et Ω'' dans l'ordre $\mathcal{Q}$ (cette propriété vient de la 2-monotonie du pliage). On ajoute dans $\mathcal{PG}$ une diagonale entre le point de contrôle représentant la clique Ω et celui

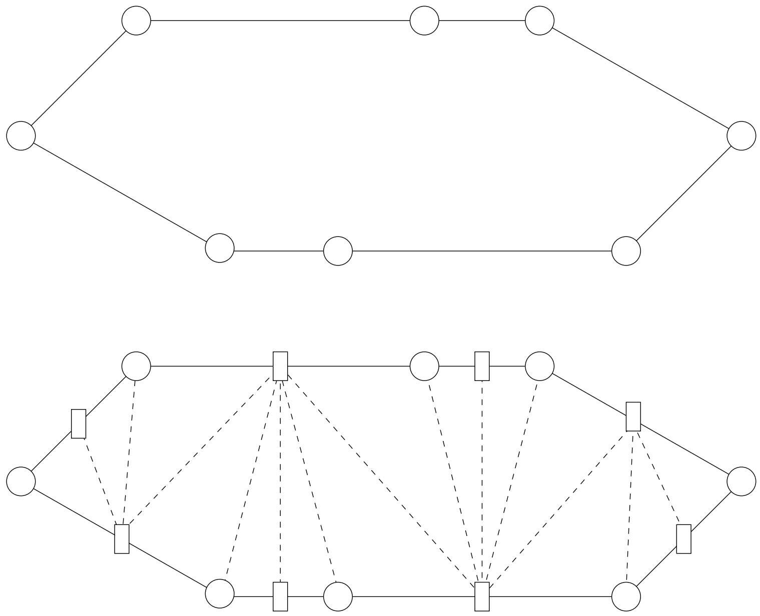

FIG. 6.6 – 2-pliage (haut) et la triangulation planaire linéaire associée (bas).

correspondant au semi-séparateur de la branche $\Omega'\Omega''$. De plus, on ajoute une diagonale entre le semi-séparateur $\Omega' \cap \Omega''$ et chacun des deux semi-séparateurs correspondant aux branches incidentes à Ω dans $\mathcal{CC}$. Nous obtenons ainsi une triangulation planaire linéaire LT de $\mathcal{PG}$, dont les triangles sont en bijection avec les sacs de la décomposition linéaire produite par l'algorithme `PliageCI`$(G, \mathcal{Q})$. En particulier, la largeur de LT est $\omega(H) = \mathrm{pw}(H) + 1$. $\diamond$

L'algorithme calculant une triangulation planaire linéaire de $\mathcal{PG}$, de largeur minimum, est décrit dans [127] et très similaire à celui de [93]. Nous obtenons :

Théorème 6.21 *Il existe un algorithme de complexité $\mathcal{O}(n^2)$, calculant la largeur linéaire des graphes d'intervalles circulaires.*

6.4 Conclusion et questions ouvertes

Nous avons donné un premier algorithme capable de produire n'importe quelle complétion d'intervalles minimale du graphe en entrée. De plus cet algorithme extrait une complétion minimale à partir d'une complétion non minimale. Il peut être utilisé en phase de post-processage de toute heuristique de calcul d'une décomposition linéaire (cette technique de post-processage est utilisée dans les problèmes de triangulations, cf. [18]). Il serait utile et intéressant de simplifier et d'améliorer la complexité de cet algorithme.

La technique de pliage introduite initialement pour l'extraction d'une complétion d'intervalles minimale nous a permis de calculer la largeur linéaire pour les graphes d'intervalles circulaires. C'est le premier algorithme polynomial résolvant ce problème pour une classe de graphes qui n'est pas de largeur arborescente linéaire bornée et qui ne satisfait pas la propriété que la largeur linéaire est toujours égale à la largeur arborescente. La question qui se pose tout naturellement est d'étendre ces résultats à d'autres classes de graphes et surtout d'exhiber une technique générique pour le calcul de la largeur linéaire.

Chapitre 7

Conclusion et perspectives

Nous avons étudié dans ce mémoire les décompositions arborescentes et les décompositions linéaires des graphes, avec les paramètres de largeur associés.

Pour mieux comprendre et mieux calculer la largeur arborescente, nous avons approfondi la notion de clique maximale potentielle. Ceci nous a permis de restreindre et de maîtriser l'ensemble des sacs qui sont utiles pour construire une décomposition arborescente optimale. Nous avons ainsi prouvé que le calcul de la largeur arborescente peut se faire en temps polynomial par rapport au nombre de séparateurs minimaux du graphe en entrée, unifiant ainsi un grand nombre d'algorithmes qui traitaient ce problème au cas par cas. Nous avons donné un algorithme exact de calcul de la largeur arborescente en temps $\mathcal{O}(1.9601^n)$, et de nouveaux outils pour comprendre la largeur arborescente des graphes planaires. Enfin, nous avons abordé la question de l'approximation de la largeur arborescente, en donnant des algorithmes d'approximation à facteur constant pour certaines classes de graphes, ainsi qu'un algorithme d'approximation à facteur $\mathcal{O}(\log k)$ pour les graphes quelconques, où k est la largeur arborescente du graphe en entrée.

Contrairement à la largeur arborescente, la largeur linéaire des graphes reste un paramètre très difficile à comprendre et à calculer. Les travaux (pourtant nombreux !) dans ce domaine n'ont abouti qu'à des résultats très partiels. Nous avons proposé une nouvelle façon d'aborder le problème, en passant par l'étude des complétions d'intervalles minimales des graphes. Nous présentons plusieurs algorithmes calculant des complétions d'intervalles minimales d'un graphe quelconque, convaincus du fait que la compréhension de ces complétions apportera les outils nécessaires à la maîtrise de la largeur linéaire. Cette approche nous a permis de calculer, en temps quadratique, la largeur linéaire des graphes d'intervalles circulaires.

Au cours de cette étude nous nous sommes posés un certain nombre de questions qui nous semblent être des axes de recherche importants et prometteurs. Elles seront résumées en quatre catégories :

– approximation de la largeur arborescente ;
– nouveaux paramètres de décompositions, modélisant plus finement les besoins des applications basées sur les décompositions arborescentes ;

– complétions de graphes ;
– paramètres de largeur et leur lien avec les jeux de poursuite.

Approximation de la largeur arborescente. L'approximation de la largeur arborescente à une constante multiplicative près reste un problème ouvert, sans doute le plus important lié au calcul de ce paramètre. Rappelons que le meilleur algorithme actuel produit une approximation de largeur $\mathcal{O}(k\sqrt{\log k})$ où k est la largeur arborescente du graphe en entrée ; malheureusement la constante cachée par le $\mathcal{O}$ est énorme.

Le principal frein que nous rencontrons dans la conception des algorithmes d'approximation provient de la complexité des outils qui montrent que la largeur arborescente d'un graphe est grande. Comme dans tout algorithme d'approximation, nous aurions besoins de bornes inférieures simples pour la largeur arborescente. Pour justifier qu'un graphe est de largeur arborescente au plus k, il suffit d'exhiber une décomposition arborescente de largeur au plus k ; de plus, cette décomposition occupe un espace mémoire $\mathcal{O}(n^2)$, sa taille ne dépend pas de k. Quel certificat utiliser pour prouver que la largeur arborescente d'un graphe est strictement supérieure à k ? Il existe un certain nombre d'objets conçus à ce sujet, comme les enchevêtrements (brambles) ou les havres (havens), voir [17, 47, 115] pour une synthèse. Ils sont à ma connaissance très peu utilisés de point de vue algorithmique (à quelques exceptions notables [122, 23]). La taille d'un tel certificat peut atteindre de l'ordre de n^k.

Il serait essentiel de pouvoir utiliser dans nos travaux algorithmiques une plus grande partie des puissants outils combinatoires développés dans la théorie des mineurs de graphes. A titre d'exemple, rappelons le fameux théorème du mineur grille : tout graphe qui ne contient pas la grille planaire $r \times r$ comme mineur a une largeur arborescente au plus $f(r)$. Actuellement la fonction f pour laquelle le théorème est prouvé est surexponentielle en r ; il n'existe aucun exemple où $f(r)$ excède $r^2 \log r$. Si cette deuxième borne était prouvée, la preuve fournirait sans doute un bon algorithme d'approximation de la largeur arborescente.

Paramètres de décomposition et programmation par contraintes. La programmation par contraintes est l'un des domaines où les décompositions arborescentes sont souvent utilisées, qu'il s'agisse de résultats théoriques ou appliqués [67, 83, 96]. Ces applications ont également mené à plusieurs généralisations des décompositions arborescentes et de la largeur arborescente, notamment en les élargissant aux hypergraphes (voir [67] pour une présentation synthétique de ces travaux).

L'analyse fine des algorithmes de résolution de contraintes basés sur des décompositions montrent que la largeur arborescente n'est pas le paramètre le plus pertinent pour étudier leur complexité. La notion de largeur arborescente valuée (définie en mettant un poids pour chaque sommet, et la largeur d'un sac devient le produit des poids de ses sommets) commence tout juste à être étudiée [6].

Une variante encore plus originale est proposé par l'heuristique BTD, comme "backtrack with tree decompositions" [83]. Les techniques classiques de programmation dynamique requièrent un espace mémoire de l'ordre $2^{\mathrm{tw}(G)}$ même dans le cas des problèmes avec des variables binaires. Si l'on peut se permettre ce genre de complexité en temps, l'utilisation d'une telle taille mémoire est prohibitive. L'heurisitique BTD est une heuristique de backtrack classique, guidée par l'arbre de décomposition. Elle mémorise seulement les instanciations des variables au niveau des séparateurs, et non pas des cliques de la triangulation. Ainsi l'espace mémoire est de l'ordre de 2^s, où s est la taille du plus grand séparateur utilisé dans la décomposition. La borne supérieure pour la complexité en temps reste $2^{\mathrm{tw}(G)}$, mais l'utilisation de techniques branch-and-bound rend cette heuristique beaucoup plus rapide en moyenne. Clairement, ce nouveau paramètre s peut être sensiblement plus petit que la largeur arborescente, et les jeux de tests évoqués dans [83] confirment que cette situation est courante dans les problèmes réels. Il serait donc très important de pouvoir calculer des décompositions (triangulations) où l'on essaye de minimiser deux paramètres, en priorité la taille du plus grand séparateur de la triangulation et seulement ensuite la clique maximum de la triangulation. Notons que, dans ce cas, les bonnes triangulations ne se trouvent pas forcément parmi les triangulations minimales. Par contre ce sont encore des triangulations efficaces, comme pour les décompositions en branches optimales.

Complétions de graphes Dans notre étude sur les complétions minimales de graphes, nous avons évoqué trois types de complétions d'un graphe quelconque : en graphes triangulés, en graphes d'intervalles et en graphes d'intervalles propres. Notons que d'autres chercheurs ont exploré la voie des complétions minimales vers d'autres classes de graphes, comme les graphes split [73] ou les graphes de comparabilité [74]. En plus des paramètres de largeur, il est également très classique de minimiser le nombre minimum d'arêtes ajoutées lors d'une telle complétion. Le tableau ci-dessous mentionne quelques paramètres classiques liés aux complétions.

$G=(V,E)\to H=(V,F)$	$\min \omega(H)-1$	$\min \lvert F\backslash E\rvert$
H triangulé	largeur arborescente	remplissage minimum
H d'intervalles	largeur linéaire	profile
H d'intervalles propres	largeur de bande	MPIC

Il reste beaucoup de questions ouvertes relativement à ces paramètres, par exemple on ignore toujours si le calcul du profile est un problème de la classe FPT. En clair, étant donné un graphe quelconque $G=(V,E)$ et un nombre k, peut-on décider s'il existe une complétion d'intervalles $H=(V,F)$ de G telle que $|F-E|\leq k$, en temps $\mathcal{O}(f(k)\,\mathrm{Poly}(n))$ (le polynôme en n ne doit pas dépentre de k) ?

De façon similaire se posent un grand nombre de questions sur la *distance de modification* (edit distance) entre un graphe quelconque G et une classe de graphes. Peut-on transformer le graphe G en un graphe de la classe $\mathcal{G}$ en lui rajoutant/supprimant un petit nombre d'arêtes/de sommets ? En quelque sorte on s'attend à ce que les graphes proches d'une classe simple soient simples eux aussi. Cependant des résultats récents dans ce domaine laissent penser que les choses pourraient être bien plus compliquées. Si l'on considère la classe des graphes obtenus à partir des graphes de comparabilité en rajoutant au plus deux arêtes, le problème de la coloration devient NP-difficile, alors qu'il est polynomial pour les graphes de comparabilité [129].

Largeur linéaire et jeux de poursuite Les *jeux de poursuite* (search games) dans un graphe consistent à imaginer que des agents poursuivent un fugitif dans ce graphe. Les agents et le fugitif sont toujours placés sur des sommets. Le fugitif et les agents se déplacent arbitrairement vite dans le réseau, le fugitif prend soin de ne pas passer par un sommet occupé par des agents. Les agents gagnent s'ils arrivent à capturer le fugitif (en occupant le même sommet), le fugitif gagne s'il arrive toujours à échapper. On étudie alors le nombre minimum d'agents qu'il faut pour attraper le fugitif. Dans la version la plus classique, les agents ne voient pas où se cache le fugitif ; dans ce cas le *nombre de poursuite* du graphe G, i.e. le nombre minimum d'agents nécessaires pour être certains de pouvoir attraper le fugitif, est exactement la largeur linéaire du graphe, plus un. Pour se faire une intuition de ce résultat, il suffit d'imaginer que l'on dispose d'une décomposition linéaire du graphe de largeur k. Au début les $k + 1$ agents se placent sur le sac le plus à gauche, en obligeant le fugitif d'aller à droite. A chaque étape, les agents se déplacent d'un sac vers droite dans la décomposition, en prenant soin de garder occupée l'intersection entre l'ancien et le nouveau sac. Les $k + 1$ agents finiront donc par attraper le fugitif. L'implication inverse, prouvant que l'on a besoin d'au moins $\mathrm{pw}(G) + 1$ agents, est bien moins facile. Si les agents ont la capacité de voir le fugitif, il faut à priori moins d'agents car ils savent vers quelle partie du graphe ils doivent s'orienter. Dans ce cas, le *nombre de poursuite avec fugitif visible* est égal à la largeur arborescente, plus un. Récemment, Barrière et al. [7, 8], ainsi que Nisse et Fraigniaud [62] ont introduit des versions *connectées* de ceux jeux, dans lesquelles la partie du graphe nettoyée par les agents doit rester connexe. Ces versions modélisent mieux la situation ou le fugitif est un agent de type virus, et les agents ont besoin de communiquer sur des canaux sûrs.

Il reste dans ce cadre énormément de questions ouvertes, sur le calcul des nouveaux types de nombres de poursuite [8, 61], ainsi que sur leur rapport avec les paramètres classiques de largeur. Ainsi, Fraigniaud et Nisse montrent [62] que la version "connectée avec fugitif visible" requiert parfois beaucoup plus d'agents que la largeur arborescente (version non connectée, avec fugitif visible). Ce ratio peut atteindre $\log n$. Dans le cas avec fugitif invisible, il n'existe pas d'exemple où ce ratio soit supérieur à 2, ni de preuve montrant que le ratio est toujours majoré par 2.

Bibliographie

[1] K. Aardal, C. van Hoesel, A. Koster, C. Mannino, and A. Sassano. Models and solution techniques for the frequency assignment problem. *4OR*, 1(4) :261–317, 2003.

[2] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows : theory, algorithms, and applications*. Prentice Hall, Englewood Cliffs NJ, 1993.

[3] E. Amir. Efficient approximation for triangulation of minimum treewidth. In *Proceedings 17th Conference on Uncertainty in Artificial Intelligence (UAI '01)*, 2001. http ://www.cs.berkeley.edu/eyal/papers/decomp-uai01.ps.

[4] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. on Algebraic and Discrete Methods*, 8 :277–284, 1987.

[5] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23 :11–24, 1989.

[6] E. Bachoore and H. Bodlaender. Weighted treewidth : Algorithmic techniques and results. Technical Report UU-CS-2006-013, Institute of Information and Computing Sciences, Utrecht University, 2006.

[7] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In *Proceedings 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 200–209. ACM, 2002.

[8] L. Barrière, P. Fraigniaud, N. Santoro, and D. Thilikos. Searching is not jumping. In *Proceedings 29th Workshop on Graph-theoretic Concepts in Computer Science (WG 2003)*, volume 2880 of *Lecture Notes in Computer Science*, pages 34–45. Springer, 2003.

[9] A. Berry. *Désarticulation d'un graphe*. PhD thesis, Université Montpellier II, 1998. In French.

[10] A. Berry and J. P. Bordat. Local LexBFS properties in an arbitrary graph. In *Proceedings of the Journées Informatiques Messines (JIM 2000)*, 2000. http ://www.isima.fr/berry/lexbfs.ps.

[11] A. Berry and J.P. Bordat. Separability generalizes Dirac's theorem. *Discrete Applied Mathematics*, 84(1-3) :43–53, 1998.

[12] A. Berry, J.P. Bordat, and O. Cogis. Generating all the minimal separators of a graph. In *Workshop on Graph-theoretic Concepts in Computer Science (WG'99)*, volume 1665 of *Lecture Notes in Computer Science*, pages 167–172. Springer-Verlag, 1999.

[13] A. Berry, P. Heggernes, and Y. Villanger. A vertex incremental approach for maintaining chordality. *Discrete Mathematics*, 306(3) :318–336, 2006.

[14] J. Blair and B. Peyton. An introduction to chordal graphs and clique trees. In *Graph Theory and Sparse Matrix Computations*, pages 1–29. Springer, 1993.

[15] H. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11 :1–23, 1993.

[16] H. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Computing*, 25 :1305–1317, 1996.

[17] H. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2) :1–45, 1998.

[18] H. Bodlaender. Treewidth : Characterizations, applications, and computations. In *Proceedings 32nd Workshop on Graph-theoretic Concepts in Computer Science (WG 2006)*, Lecture Notes in Computer Science. Springer-Verlag, 2006. To appear.

[19] H. Bodlaender and F. Fomin. Approximation of pathwidth of outerplanar graphs. *Journal of Algorithms*, 43(2) :190–200, 2002.

[20] H. Bodlaender and F. Fomin. Tree decompositions with small cost. *Discrete Applied Mathematics*, 145(2) :143–154, 2005.

[21] H. Bodlaender, F. Fomin, A. Koster, D. Kratsch, and D. Thilikos. On exact algorithms for treewidth. In *Proceedings 14th Annual European Symposium on Algorithms, ESA 2006*, volume 4168 of *Lecture Notes in Computer Science*, pages 672–683. Springer, 2006.

[22] H. Bodlaender, J. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize and shortest elimination tree. *Journal of Algorithms*, 18 :238–255, 1995.

[23] H. Bodlaender, A. Grigoriev, and A. Koster. Treewidth lower bounds with brambles. In G. S. Brodal and S. Leonardi, editors, *Proceedings 13th Annual European Symposium on Algorithms (ESA 2005)*, volume 3669 of *Lecture Notes in Computer Science*, pages 391–402. Springer, 2005.

[24] H. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2) :358–402, 1996.

[25] H. Bodlaender, T. Kloks, and D. Kratsch. Treewidth and pathwidth of permutation graphs. *SIAM J. on Discrete Math.*, 8 :606–616, 1995.

[26] H. Bodlaender, A. Koster, and F. van den Eijkhof. Pre-processing rules for triangulation of probabilistic networks. *Computational Intelligence*, 21(3) :286–305, 2005.

[27] H. Bodlaender and U. Rotics. Computing the treewidth and the minimum fill-in with the modular decomposition. *Algorithmica*, 36(4) :375–408, 2003.

[28] H. Bodlaender and D. Thilikos. Treewidth for graphs with small chordality. *Discrete Applied Mathematics*, 79(1-3) :45–61, 1997.

[29] V. Bouchitté, D. Kratsch, H. Müller, and I. Todinca. On treewidth approximations. *Discrete Applied Mathematics*, 136(2-3) :183–196, 2004.

[30] V. Bouchitté, F. Mazoit, and I. Todinca. Chordal embeddings of planar graphs. *Discrete Mathematics*, 273(1-3) :85–102, 2003. Special issue devoted to EuroComb'01.

[31] V. Bouchitté and I. Todinca. Treewidth and minimum fill-in : grouping the minimal separators. *SIAM J. on Computing*, 31(1) :212 – 232, 2001.

[32] V. Bouchitté and I. Todinca. Listing all potential maximal cliques of a graph. *Theoretical Computer Science*, 276(1-2) :17–32, 2002.

[33] V. Bouchitté and I. Todinca. Approximating the treewidth of AT-free graphs. *Discrete Applied Mathematics*, 131(1-5) :11–37, 2003. Special issue devoted to Journées de l'Informatique Messine (JIM'00).

[34] H. Broersma, E. Dahlhaus, and T. Kloks. A linear time algorithm for minimum fill-in and treewidth for distance hereditary graphs. *Discrete Applied Mathematics*, 99(1-3) :367–400, 2000.

[35] H.J. Broersma, E. Dahlhaus, and T. Kloks. Algorithms for the treewidth and minimum fill-in of HHD-free graphs. In *Workshop on Graph-theoretic Concepts in Computer Science (WG'97)*, volume 1335 of *Lecture Notes in Computer Science*, pages 109–117. Springer-Verlag, 1997.

[36] M. Chudnovsky, G. Cornuejols, X. Liu, P. Seymour, and K. Vuskovic. Recognizing Berge graphs. *Combinatorica*, 25 :143–186, 2005.

[37] M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas. The strong perfect graph theorem. *Ann. Math.*, 164 :51–229, 2006.

[38] W. Cook and P. Seymour. Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3) :233–284, 2003.

[39] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT press, 1990.

[40] B. Courcelle. The monadic second-order logic of graphs III : Treewidth, forbidden minors and complexity issues. *Informatique Théorique*, 26 :257–286, 1992.

[41] B. Courcelle, J. Engelfriet, and G. Rozenberg. Context-free handle-rewriting hypergraph grammars. In *Graph-Grammars and Their Application to Computer Science*, volume 532 of *Lectures Notes in Computer Science*, pages 253–268. Springer-Verlag, 1991.

[42] B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theoretical Computer Science*, 33(2) :125–150, 2000.

[43] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109 :49–82, 1993.

[44] B. Courcelle and S. Olariu. Upper bounds to the clique-width of graphs. *Discrete Applied Mathematics*, 101 :77–114, 2000.

[45] A. Cournier and M. Habib. A new linear algorithm for modular decomposition. In *Proceedings of Trees in Algebra and Programming - CAAP'94*, volume 787 of *Lecture Notes in Computer Science*, pages 64–84, 1994.

[46] E. Demaine, F. Fomin, M. Taghi Hajiaghayi, and D. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. of the ACM*, 52(6) :866–893, 2005.

[47] R. Diestel. *Graph Theory*. Springer, 1997.

[48] G.A. Dirac. On rigid circuit graphs. *Abh. Math. Sem. Univ. Hamburg*, 21 :71–76, 1961.

[49] K. Dohmen, A. Poenitz, and P. Tittmann. A new two-variable generalization of the chromatic polynomial. *Discrete Mathematics & Theoretical Computer Science*, 6(1) :69–90, 2003.

[50] F. Dorn. Dynamic programming and fast matrix multiplication. In *Proceedings 14th Annual European Symposiun on Algorithms (ESA 2006)*, volume 4168 of *Lectures Notes in Computer Science*, pages 280–291, 2006.

[51] J. A. Ellis and M. Markov. Computing the vertex separation of unicyclic graphs. *Information and Computation*, 192 :123–161, 2004.

[52] J. A. Ellis, I. H. Sudborough, and J. S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1) :50–79, 1994.

[53] D. Eppstein. Subgraph isomorphism in planar graphs and related problems. *Journal of Graph Algorithms and Applications*, 3(3) :1–27, 1999.

[54] W. Espelage, F. Gurski, and E. Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In *Workshop on Graph-theoretic Concepts in Computer Science (WG 2001)*, volume 2204 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2001.

[55] G. Even, J. Naor, S. Rao, and B. Schieber. Fast approximate graph partitioning algorithms. *SIAM J. on Computing*, 28(6) :2187–2214, 1999.

[56] U. Feige, M. T. Hajiaghayi, and J. R. Lee. Improved approximation algorithms for minimum-weight vertex separators. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC 2005)*, pages 563–572. ACM, 2005.

[57] F. Fomin, F. Grandoni, and D. Kratsch. Some new techniques in design and analysis of exact (exponential) algorithms. *Bulletin of the EATCS*, 87 :47–77, 2005.

[58] F. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer : a simple $O(2^{0.288n})$ independent set algorithm. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006)*, pages 18–25, 2006.

[59] F. Fomin, D. Kratsch, and I. Todinca. Exact (exponential) algorithms for treewidth and minimum fill-in. In *Proceedings 31st International Colloquium on Automatas,*

Languages and Programming (ICALP'04), volume 3142 of *Lecture Notes in Computer Science*, pages 568–580. Springer, 2004.

[60] F. Fomin, F. Mazoit, and I. Todinca. Computing branchwidth via efficient triangulations and blocks. In *Proceedings of the 31st Workshop on Graph-theoretic Concepts in Computer Science (WG 2005)*, volume 3787 of *Lecture Notes in Computer Science*, pages 374–384. Springer-Verlag, 2005.

[61] F. Fomin, D. Thilikos, and I. Todinca. Connected graph searching in outerplanar graphs. *Electronic Notes in Discrete Mathematics*, 22 :213–216, 2005. 7th International Colloquium on Graph Theory. Short communication.

[62] P. Fraigniaud and N. Nisse. Monotony properties of connected visible graph searching. In *Proceedings 32nd Workshop on Graph-Theoretic Aspects in Computer Science (WG 2006)*, Lectures Notes in Computer Science. Springer-Verlag, 2006. To appear.

[63] T. Gallai. Transitiv orienterbare graphe. *Acta Math. Acad. Sci. Hungar*, 18 :25–66, 1967.

[64] M.R. Garey and D.S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-completness*. Freeman, 1979.

[65] F. Gavril. The intersection graphs of a path in a tree are exactly the chordal graphs. *Journal of Combinatorial Theory*, 16 :47–56, 1974.

[66] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

[67] G. Gottlob, M. Grohe, N. Musliu, M. Samer, and F. Scarcello. Hypertree decompositions : Structure, algorithms, and applications. In *Proceedings of the 31st Workshop on Graph-theoretic Concepts in Computer Science (WG 2005)*, volume 3787 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2005.

[68] J. Gustedt. On the pathwidth of chordal graphs. *Discrete Applied Mathematics*, 45(3) :233–248, 2003.

[69] M. Habib, R. M. McConnell, C. Paul, and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(1-2) :59–84, 2000.

[70] M. Habib and R. Möhring. Treewidth of cocomparability graphs and a new order-theoretic parameter. *ORDER*, 1 :47–60, 1994.

[71] M. Habib, C. Paul, and L. Viennot. Partition refinement techniques : An interesting algorithmic tool kit. *International Journal of Foundations of Computer Science*, 10(2) :147–170, 1999.

[72] T. Hagerup. Dynamic algorithms for graphs of bounded treewidth. *Algorithmica*, 27 :292–315, 2000.

[73] P. Heggernes and F. Mancini. Minimal split completions of graphs. In *LATIN 2006 : Theoretical Informatics, 7th Latin American Symposium*, volume 3887 of *Lecture Notes in Computer Science*, pages 592–604. Springer-Verlag, 2006.

[74] P. Heggernes, F. Mancini, and C. Papadopoulos. Minimal comparability completions. In *Proceedings ISAAC 2006*, Lecture Notes in Computer Science, 2006. To appear.

[75] P. Heggernes, K. Suchan, I. Todinca, and Y. Villanger. Minimal interval completions. In *Proceedings of the 13th European Symposium on Algorithms (ESA 2005)*, volume 3669 of *Lecture Notes in Computer Science*, pages 403–414. Springer-Verlag, 2005.

[76] P. Heggernes, K. Suchan, I. Todinca, and Y. Villanger. Minimal interval completions. Technical Report RR2005-04, LIFO - University of Orléans, 2005. http ://www.univ-orleans.fr/sciences/lifo/rapports2005.

[77] P. Heggernes, K. Suchan, I. Todinca, and Y. Villanger. Characterizing minimal interval completions : towards better understanding of profile and pathwidth. Technical Report RR-2006-09, LIFO - Université d'Orléans, 2006.

[78] P. Heggernes, J. A. Telle, and Y. Villanger. Computing minimal triangulations in time $O(n^\alpha \log n) = o(n^{2.376})$. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005)*, pages 907–916. SIAM, 2005.

[79] M. Held and R. Karp. A dynamic programming approach to sequencing problems. *J. Soc. Indust. Appl. Math.*, 10 :196–210, 1962.

[80] I. Hicks, A. Koster, and E. Kolotoğlu. Branch and tree decomposition techniques for discrete optimization. In J. Cole Smith, editor, *TutORials 2005*, INFORMS TutORials in Operations Research Series, chapter 1, pages 1–29. INFORMS Annual Meeting, 2005.

[81] C.W. Ho and R.C.T. Lee. Counting clique trees and computing perfect elimination schemes in parallel. *Inform. Process. Letters*, 31 :61–68, 1989.

[82] P. Jégou, S. Ndiaye, and C. Terrioux. Computing and exploiting tree-decompositions for solving constraint networks. In *Proceedings of Constraint Programming (CP'95)*, pages 777–781, 2005.

[83] P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artif. Intell.*, 146(1) :43–75, 2003.

[84] H. Kaplan and R. Shamir. Pathwidth, bandwidth, and completion problems to proper interval graphs with small cliques. *SIAM Journal on Computing*, 25(3) :540–561, 1996.

[85] T. Kloks. Treewidth of circle graphs. *Intern. J. Found. Comput. Sci.*, 7 :111–120, 1996.

[86] T. Kloks, H. Bodlaender, H. Müller, and D. Kratsch. Erratum to the ESA'93 proceedings. In *Proceedings Second Annual European Symposium on Algorithms (ESA'94)*, volume 855 of *Lecture Notes in Computer Science*, page 508. Springer-Verlag, 1994.

[87] T. Kloks, H. Bodlaender, H. Müller, and D. Kratsch. Computing treewidth and minimum fill-in : all you need are the minimal separators. In *Proceedings First Annual European Symposium on Algorithms (ESA'93)*, volume 726 of *Lecture Notes in Computer Science*, pages 260–271. Springer-Verlag, 1993.

[88] T. Kloks, J. Kratochvil, and H. Müller. Computing the branchwidth of interval graphs. *Discrete Applied Mathematics*, 145(2) :266–275, 2005.

[89] T. Kloks and D. Kratsch. Treewidth of chordal bipartite graphs. *J. Algorithms*, 19(2) :266–281, 1995.

[90] T. Kloks and D. Kratsch. Listing all minimal separators of a graph. *SIAM J. Comput.*, 27(3) :605–613, 1998.

[91] T. Kloks, D. Kratsch, and H. Müller. Asteroidal sets in graphs. In *Workshop on Graph-theoretic Concepts in Computer Science (WG'97)*, volume 1335 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[92] T. Kloks, D. Kratsch, and J. Spinrad. On treewidth and minimum fill-in of asteroidal triple-free graphs. *Theoretical Computer Science*, 175 :309–335, 1997.

[93] T. Kloks, D. Kratsch, and C.K. Wong. Minimum fill-in of circle and circular-arc graphs. *J. Algorithms*, 28(2) :272–289, 1998.

[94] D. Kobler and U. Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Applied Mathematics*, 126(2-3) :197–221, 2003.

[95] A. Koster, C. van Hoesel, and A. Kolen. Optimal solutions for a frequency assignment problem via tree-decomposition. In *Graph-Theoretic Concepts in Computer Science (WG '99)*, volume 1665 of *Lecture Notes in Computer Science*, pages 338–349. Springer-Verlag, 1999.

[96] A. Koster, C. van Hoesel, and A. Kolen. Solving partial constraint satisfaction problems with tree decomposition. *Networks*, 40(3) :170–180, 2002.

[97] J.B. Kruskal. Well quasi-ordering, the tree theorem and vàrizsonys's conjecture. *Transactions of the American Mathematical Society*, 95 :210–225, 1960.

[98] D. Lapoire. Treewidth and duality in planar hypergraphs. http ://dept-info.labri.u-bordeaux.fr/~lapoire/papers/dual_planar_treewidth.ps.

[99] S.J. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their applications to expert systems. *The Journal of the Royal Statistical Society. Series B (Methodological)*, 50 :157–224, 1988.

[100] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6) :787–832, 1999.

[101] M. Lundquist. *Zero Patterns, Chordal Graphs and Matrix Completion*. PhD thesis, Clemson University, 1990.

[102] F. Mazoit. *Décompositions algorithmiques des graphes*. PhD thesis, Ecole normale supérieure de Lyon, 2004. In French.

[103] F. Mazoit. The branch-width of circular-arc graphs. In *LATIN 2006 : Theoretical Informatics, 7th Latin American Symposium*, volume 3887 of *Lecture Notes in Computer Science*, pages 727–736. Springer-Verlag, 2006.

[104] R.M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2) :93–147, 2003.

[105] N. Meggido, S. L. Hakimi, M.R. Garey, D.S. Johson, and C.H. Papadimitriou. The complexity of searching a graph. *Journal of the ACM*, 35 :18–44, 1988.

[106] B. Monien. The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 7 :505–512, 1986.

[107] S. Olariu. An optimal greedy heuristic to color interval graphs. *Information Processing Letters*, 37(1) :21–25, 1991.

[108] S. Oum and P. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory Series B*, 96(4) :514–528, 2006.

[109] B. S. Panda and S. K. Das. A linear time recognition algorithm for proper interval graphs. *Information Processing Letters*, 87(3) :153–161, 2003.

[110] A. Parra and P. Scheffler. Characterizations and algorithmic applications of chordal graph embeddings. *Discrete Appl. Math.*, 79(1-3) :171–188, 1997.

[111] C. Paul and J.A. Telle. New tools and simpler algorithms for branchwidth. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA 2005)*, volume 3669 of *Lecture Notes in Computer Science*, pages 379–390. Springer, 2005.

[112] C. Paul. *Parcours en Largeur Lexicographique : Un Algorithme de Partitionnement. Application aux Graphes et Généralisation*. PhD thesis, Université Montpellier II, 1998.

[113] I. Rapaport, K. Suchan, and I. Todinca. Minimal proper interval completions. In *Proceedings 32nd Workshop on Graph-Theoretic Aspects in Computer Science (WG 2006)*, Lectures Notes in Computer Science, 2006. To appear.

[114] B. Reed. Finding approximate separators and computing tree width quickly. In *Proceedings of the Twenty Fourth Annual ACM Symposium on Theory of Computing (STOC 92)*, pages 221–228. ACM, 1992.

[115] B. Reed. *Recent advances in algorithms and combinatorics*, chapter Algorithmic aspects of tree width, pages 87–162. CMS Books Math./ Ouvrages Math. Springer, 2003. SMC, 11.

[116] N. Robertson and P. Seymour. Graph minors. I. Excluding a forest. *Journal of Combinatorial Theory Series B*, 35 :39–61, 1983.

[117] N. Robertson and P. Seymour. Graphs minors. III. Planar tree-width. *Journal of Combinatorial Theory Series B*, 36 :49–64, 1984.

[118] N. Robertson and P. Seymour. Graphs minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7 :309–322, 1986.

[119] N. Robertson and P. Seymour. Graph minors X. Obstructions to tree decompositions. *Journal of Combinatorial Theory Series B*, 52 :153–190, 1991.

[120] D.J. Rose. On simple characterization of k-trees. *Discrete Mathematics*, 7 :317–322, 1974.

[121] D.J. Rose, R.E. Tarjan, and G.D. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5 :266–283, 1976.

[122] P. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2) :217–241, 1994.

[123] K. Skodinis. Construction of linear tree-layouts which are optimal with respect to vertex separation in linear time. *Journal of Algorithms*, 47(1) :40–59, 2003.

[124] K. Suchan. *Complétions d'intervalles minimales*. PhD thesis, Université d'Orléans, 2006. In preparation.

[125] K. Suchan and I. Todinca. Powers of graphs of bounded NLC-width (clique-width). To appear in Discrete Applied Mathematics.

[126] K. Suchan and I. Todinca. Minimal interval completions through graph exploration. In *Proceedings ISAAC 2006*, Lecture Notes in Computer Science, 2006. To appear.

[127] K. Suchan and I. Todinca. Pathwidth of circular-arc graphs. Technical Report RR-2006-10, LIFO - Université d'Orléans, 2006.

[128] R. Sundaram, K. Sher Singh, and C. Pandu Rangan. Treewidth of circular-arc graphs. *SIAM J. Discrete Math.*, 7 :647–655, 1994.

[129] Y. Takenaga and K. Higashide. Vertex coloring of comparability+ke and -ke graphs. In *Proceedings 32nd Workshop on Graph-Theoretic Aspects in Computer Science (WG 2006)*, Lectures Notes in Computer Science. Springer-Verlag, 2006. To appear.

[130] J. A. Telle. Tree-decompositions of small pathwidth. *Discrete Applied Mathematics*, 145(2) :210–218, 2005.

[131] R. Thomas. Tree-decompositions of graphs. http ://www.math.gatech.edu/ thomas/tree.ps.

[132] I. Todinca. *Aspects algorithmiques des triangulations minimales des graphes*. PhD thesis, École Normale Supérieure de Lyon, 1999.

[133] Y. Villanger. Improved exponential-time algorithms for treewidth and minimum fill-in. In *LATIN 2006 : Theoretical Informatics, 7th Latin American Symposium*, volume 3887 of *Lecture Notes in Computer Science*, pages 800–811. Springer-Verlag, 2006.

[134] E. Wanke. k-NLC graphs and polynomial algorithms. *Discrete Applied Mathematics*, 54(2-3) :251–266, 1994.

[135] G. Woeginger. Space and time complexity of exact algorithms : Some open problems (invited talk). In *Parameterized and Exact Computation, First International Workshop (IWPEC 2004)*, volume 3162 of *Lecture Notes in Computer Science*, pages 281–290. Springer, 2004.

Index