Techniques combinatoires pour les algorithmes paramétrés et les noyaux, avec applications aux problèmes de multicoupe.

Jean Daligault

▶ To cite this version:

Jean Daligault. Techniques combinatoires pour les algorithmes paramétrés et les noyaux, avec applications aux problèmes de multicoupe.. Discrete Mathematics [cs.DM]. Université Montpellier II - Sciences et Techniques du Languedoc, 2011. English. NNT : . tel-00804206

HAL Id: tel-00804206
https://theses.hal.science/tel-00804206

Submitted on 25 Mar 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ACADÉMIE DE MONTPELLIER

UNIVERSITÉ MONTPELLIER II

Sciences et **T**echniques du **L**anguedoc

THÈSE

présentée au Laboratoire d'Informatique de Robotique
et de Microélectronique de Montpellier pour
obtenir le diplôme de doctorat

Spécialité	:	**Informatique**
Formation Doctorale	:	**Informatique**
École Doctorale	:	**Information, Structures, Systèmes**

Techniques combinatoires pour les algorithmes paramétrés et les noyaux, avec applications aux problèmes de multicoupe.

Combinatorial Techniques for Parameterized Algorithms and Kernels, with Applications to Multicut.

par

Jean DALIGAULT

5 juillet 2011

Directeur de thèse
M. Stéphan THOMASSÉ, professeur .... LIRMM, Université Montpellier II

Rapporteurs
Mme Cristina BAZGAN, professeur .... LAMSADE, Université Paris Dauphine
M. Fedor V. FOMIN, professor .... Universitetet i Bergen

Examinateurs
M. Victor CHEPOI, professeur .... LIF, Université de Provence, Marseille
M. Cyril GAVOILLE, professeur .... LABRI, Université de Bordeaux
M. Alain JEAN-MARIE, Directeur de recherche INRIA .... LIRMM, Université Montpellier II
M. Ioan TODINCA, professeur .... LIFO, Université d'Orléans

Contents

Remerciements

Ces années de thèse ont constitué une expérience plaisante et enrichissante. Je le dois en grande partie à tous mes collègues de l'équipe AlGCo. Etre accueilli au sein d'une équipe aussi sympathique et dynamique était une grande chance pour moi, et j'adresse mille merci à tous ! En particulier à mes co-bureaux : Anthony, qui m'aura supporté jusqu'au bout, si longtemps que j'aurais juré (à tort) qu'il n'a jamais eu les cheveux longs. Nicolas, qu'on m'a un jour présenté comme le nouveau stagiaire de l'équipe, et qui a très vite inversé les rôles. Kévin, pour son aide précieuse, et sa petite famille : Sarah qui m'a appris à dire "Jean travaille une fois par semaine" en langage des signes, et Mala qui daigne jouer à Dobble avec moi quand il juge que j'ai besoin de compagnie.

Mais aussi à Daniel et Alex, qui ont accueilli avec beaucoup de gentillesse le timide nouveau thésard de l'équipe, et qui n'ont pas bronché quand je leur ai peut-être fait la peur de leur vie quelque part en Italie. Sans oublier les ex-Montpelliérains : Michael, qui à l'inverse m'a fait la peur de ma vie (devant un tableau) en m'aidant néanmoins à survivre au well-quasi-order et à ma première année de thèse. Bin, qui a changé cette roue avec brio au bord de l'autoroute, et Vincent qui a initié les marins d'eau douce que Mamadou et moi étions aux joies de la navigation en pleine mer.

Merci aussi à Stéphane et Emeric, qui du haut de leurs gabarits (respectivement) de rugbyman et de basketteur sont d'une gentillesse infinie. Benjamin, à qui je n'ai toujours pas pardonné le coup de la corde, et EunJung qui nous a brillamment libérés. Christophe, jamais en manque de traits d'esprits corrosifs, Marie-Catherine et Philippe. Sans oublier l'autre Philippe, chez qui la gentillesse le dispute à la curiosité (vous êtes-vous jamais demandé comment estimer le nombre de blagues carambar distinctes ? non ??).

I am also grateful to Anders Yeo (whom I would love to watch play badminton) and Gregory Gutin, who very kindly welcomed me to Royal Holloway and introduced me further to parameterized complexity.

Je ne trouverai pas les mots adéquats pour exprimer ma gratitude à l'égard de mon directeur de thèse, Stéphan Thomassé. Il y a quatre ans, Stéphan m'a convaincu en deux brèves rencontres de partir faire ma thèse dans une ville où je n'avais aucune attache, loin de ma famille et de mes amis. Et je me réjouis de cette décision depuis lors. Très peu de thésards (sinon aucun) n'ont la chance d'avoir un directeur à la fois aussi fort, sympathique, dynamique, disponible et gentil (même si sa gentillesse s'efface parfois temporairement, au moment de donner le fouet à un ou des étudiant(s) paresseux), et j'ai conscience de ma bonne fortune. J'ai profité de ses connaissances encyclopédiques pendant ces quatre années, qui m'ont donné la chance d'aborder de nombreux domaines. Mais plus que d'une somme de connaissances, c'est d'une vision globale des mathématiques discrètes dont Stéphan fait profiter sans relâche et avec passion les personnes qui l'entourent. Le regarder travailler, et trouver une multitude de liens entre une myriade de domaines, est la chose la plus impressionnante et enrichissante dont j'ai été témoin dans la recherche.

J'espère, Chef, ne pas t'avoir trop embêté en t'appelant "Chef". Je n'ai jamais été fort pour exprimer mes émotions, et j'ai peur de ne pas réussir à transmettre à quel point ma gratitude à ton égard est grande, et pas du seul point de vue de la recherche. S'il y a un jour un Oscar pour les directeurs de thèse, alors je nomine Stéphan.

Je suis reconnaissant à toutes celles et tous ceux dont je me suis éloigné géographiquement pendant cette thèse. Mes parents, et leur soutien sans faille, et qui se sont posé des questions quand leur fils leur a annoncé qu'il travaillait dans la "décomposition". Thomas, qui crapahute de par le monde, a fait et vu ces choses que je ne ferai et verrai jamais.

Et comment remercier cette bande d'affreux jojos qui me supportent en dépit de tout ? Alice (j'essaierai de faire de ma soutenance un petit spectacle de marionnettes, promis), et Aurel et Caro (come back folks!), et Bob (pardon, Lieutenant Bob), et Duche (qui m'a battu sur le fil, le salaud), et Fabien (non seulement Poitiers a battu Tours, mais en plus il a fallu que ça soit mentionné dans ma thèse ?), et Jim et Min (qui vont convoler à leur tour, avant de s'envoler vers d'autres cieux) et Laura (biblio-woman au volant de sa biblio-mobile ?), et les Nours (et leur Noursonne), et Pierre (comment prononce-t-on à Londres, Pieeeeeurrrr ?), et Spil (dont le groupie n°1 se désole à chaque concert raté).

Résumé de la thèse

Thématique de la thèse.

Cette thèse s'articule autour de trois thèmes étroitement liés : les algorithmes efficaces pour des problèmes de graphes, les propriétés combinatoires et structurelles des graphes, et la théorie de l'ordre. Cette trinité algorithmes / structures / ordres a fait ses preuves en théorie des graphes. Par exemple, le célèbre théorème des mineurs de Robertson et Seymour résulte d'une interaction entre une approche structurelle (largeur arborescente, construction de graphes excluant un mineur à partir de graphes plongés dans une surface), des propriétés d'ordre (le bel ordre de la relation de mineur) et des propriétés algorithmiques fortes (la solubilité à paramètre fixé du problème de 'linkage', la polynomialité de tous les problèmes clos par mineurs).

Les résultats principaux de cette thèse sont de nature algorithmique. Par exemple, le problème de Multicoupe est FPT (soluble à paramètre fixé), ou encore le problème consistant à trouver un arbre avec beaucoup de feuilles dans un graphe orienté admet un algorithme d'approximation à facteur constant. Les problèmes traités sont des problèmes difficiles (**NP**-complets), pour lesquels un angle d'attaque indirect doit être utilisé (approximation, algorithmes à paramètre fixé, noyaux polynomiaux). Ces deux dernières approches, appartenant au paradigme de la complexité paramétrée, sont les approches privilégiées dans cette thèse. Le domaine de la complexité paramétrée est relativement récent et très prometteur.

Les méthodes utilisées sont principalement de nature combinatoire et structurelle. Les décompositions arborescentes de graphes sont un outil central de ces travaux (clique-décomposition, décomposition en arbre). D'autres propriétés comme la structure des graphes avec grand grille-mineur et sans grand clique-mineur, ou la structure des graphes sans diamant induit sont également étudiées, à la fois pour elles-mêmes, mais aussi parce qu'elles induisent des améliorations algorithmiques.

Quand cela s'est avéré possible, ces travaux ont utilisé des notions liés aux ordres. Un chapitre est dédié à la théorie de l'ordre (le bel ordre de la relation de sous-graphe induit), et des outils comme le Théorème de Dilworth ou les '$s-t$ numberings' ont été très utiles pour affiner notre compréhension des structures.

Résultats de la thèse.

Le problème de graphes au centre de cette thèse est le problème de la Multicoupe. Etant donné un graphe et un ensemble de paires de sommets (les requêtes), on cherche à couper simultanément toutes les requêtes en supprimant le moins d'arêtes possible. C'est un problème difficile (NP-complet) et d'une importance fondamentale, qui généralise le problème basique de la coupe. Résoudre des problèmes de Multicoupe est aussi important en pratique, notamment pour l'étude et la conception de réseaux (réseaux de communication, VLSI, ...).

Les deux résultats centraux de cette thèse concernent ce problème de Multicoupe. En premier lieu, le problème de MULTICOUPE DANS LES ARBRES admet un noyau polynomial [1]. En d'autres termes, on peut réduire les instances de MULTICOUPE DANS LES ARBRES à une taille petite (en fonction de la taille de la solution désirée). Ce résultat est l'objet du Chapitre 2.

En second lieu, le problème de Multicoupe est soluble à paramètre fixé [2]. En d'autres termes, la partie exponentielle de la complexité du problème de Multicoupe (inévitable si **P**$\neq$**NP**) réside dans la taille de la solution désirée, et non pas dans la taille de l'entrée toute entière. Ce résultat est l'objet du Chapitre 7.

Les autres résultats principaux de cette thèse sont les suivants. Pour le problème (**NP**-dur) consistant à trouver un arbre avec un nombre maximum de feuilles dans un digraphe, nous avons fourni un noyau quadratique, un algorithme d'approximation à facteur constant [3], et un algorithme paramétré rapide [4]. Ces résultats sont l'objet des Chapitres 3 et 8 respectivement. Trouver un arbre avec beaucoup de feuilles, ou de manière équivalente trouver un ensemble dominant connecté, est un problème central dans le domaine des réseaux de communications, utilisé de manière systématique pour construire l'épine dorsale d'un réseau.

Les résultats précédents relèvent du domaine de la complexité paramétrée. Les travaux suivants, autour des graphes induits exclus, ont eux aussi des applications algorithmiques, mais cette fois directement vers des algorithmes polynomiaux. Le Chapitre 5 est consacré à une caractérisation partielle des classes 2-bel-ordonnées par sous-graphe induit [5]. La motivation principale de ce travail : tout problème de graphes clos par sous-graphe induit et restreint à une classe 2-bel-ordonnée admet automatiquement un algorithme polynomial.

1. Ce résultat a été présenté à STACS 2009.
2. Ce résultat sera présenté à STOC 2011.
3. Travaux présentés à IWPEC 2009.
4. Travaux publiés dans Journal of Computer and System Sciences (2010).
5. Travaux publiés dans Order (2010).

Enfin, cette thèse aborde les propriétés structurelles des classes de graphes géométriques. Le Chapitre 6 est consacré à prouver que les graphes de cercle de Helly sont exactement les graphes de cercle sans diamant induit [6].

Préliminaires

La théorie classique de la complexité mesure la difficulté d'un problème en fonction de la taille de l'entrée. La complexité paramétrée raffine cette analyse, en prenant en compte à la fois la taille de l'entrée et un second paramètre. Parmi les paramètres pertinents : une largeur du graphe (largeur d'arbre ou largeur de clique par exemple), la taille de la solution désirée, le degré maximum du graphe d'entrée, ...

Formellement, un problème est dit *Fixed-Parameter Tractable*, abbrégé en FPT, en français *soluble à paramètre fixé*, s'il admet un algorithme en temps $f(k)\mathrm{poly}(n)$ sur les entrées de taille n et de paramètre k. Le domaine de la complexité paramétrée est un domaine très prometteur et extrêmement dynamique depuis sa découverte dans les années 90. Il s'enorgueillit de nombreuses réussites pratiques, de multiples techniques pour créer des algorithmes paramétrés, et d'une théorie de la complexité très avancée avec notamment une théorie de la difficulté analogue à la NP-complétude en complexité classique.

Un point de vue parallèle aux algorithmes paramétrés est la notion de noyau, qui correspond à une compression sans perte des instances d'un problème.

Formellement, un *noyau* pour un problème paramétré est un algorithme en temps polynomial qui prend en entrée une instance et qui retourne une instance équivalente et de taille bornée par une fonction du paramètre. Si cette fonction est un polynôme, on parle de *noyau polynomial.* L'existence d'un noyau est équivalent à la solubilité à paramètre fixé, mais certains problèmes FPT n'ont pas de noyau polynomial.

Un noyau peut être vu comme un pré-traitement. Avant de tenter résoudre un problème difficile, on peut commencer par calculer un noyau pour réduire la taille de l'instance, avant de chercher une solution. C'est d'autant plus bénéfique que le noyau est petit (par exemple polynomial, et si possible linéaire ou quadratique).

Partie I : Noyaux

Les noyaux sont à l'intersection entre l'algorithmique et la combinatoire. Formellement, un noyau est un algorithme, mais en pratique, un noyau est obtenu à partir de règles de réductions, qui sont généralement de deux types:

- Remplacer une structure locale par une structure plus simple équivalente (propriété structurelle).
- Prouver que si un invariant est plus grand ou plus petit qu'une certaine valeur, alors il doit (ou il ne peut pas) exister une solution (propriété combinatoire).

Le Chapitre 2 est consacré au problème de MULTICOUPE DANS LES ARBRES. Etant donné un graphe, une *requête* est un couple de sommets (u, v) qu'on souhaite déconnecter. Dans

6. Travaux publiés dans Discrete Mathematics (2010).

un arbre, un seul chemin réalise une requête donnée, on peut donc assimiler la requête (u,v) à l'unique chemin entre u et v.

> MULTICOUPE DANS LES ARBRES :
> **Donnée** : Un arbre T, un ensemble de requêtes, un entier k.
> **Paramètre** : k.
> **Question** : Existe-t-il une multicoupe de taille au plus k, c'est-à-dire un ensemble S de k arêtes de T tel que toute requête contient une arête de S ?

Guo et Niedermeier [80] ont prouvé que ce problème était FPT, et ont posé la question de l'existence d'un noyau polynomial (de même que Fellows [11]). Le Chapitre 2, basé sur un travail commun avec Nicolas Bousquet, Stéphan Thomassé et Anders Yeo, fournit une réponse positive.

Nous utilisons des règles de réductions simples, essentiellement déjà exprimées dans [80], ainsi que deux règles de réduction plus compliquées et plus puissantes qui permettent de faire décroître la taille de toute instance jusqu'à un polynôme en k. Les règles simples :

(0) Une requête de longueur 1 force à utiliser son unique arête dans la solution.
(1) Il n'existe aucune solution s'il existe $k+1$ requêtes disjointes.
(2) Si toutes les requêtes partant d'une feuille ont la même direction (c'est-à-dire ont toutes la même seconde arête), alors il existe une solution optimale qui évite l'arête adjacente à cette feuille. Si toutes les requêtes partant d'un noeud interne sans feuille ont la même direction, c'est-à-dire ont toutes la même première arête (par exemple celle de droite), alors il existe une solution optimale qui évite l'autre arête adjacente à ce noeud (à gauche).
(3) Une requête qui en contient une autre peut être supprimée.

De manière informelle, ces règles ne laissent que peu de champ libre aux requêtes entre noeuds internes, puisqu'elles ne peuvent pas se contenir, par la règle (3). Elles peuvent soit être disjointes (mais il ne peut y avoir en tout que k requêtes disjointes par la règle (1)), soit s'intersecter strictement. Ce dernier cas sera essentiellement traité par la règle (4). Le difficulté du problème réside surtout dans les requêtes partant d'une feuille, puisque deux requêtes partant de deux feuilles distinctes ne peuvent se contenir. Les deux règles de réduction suivantes expriment en un sens une domination plus subtile que l'inclusion brute.

(4) S'il existe $k+2$ requêtes distinctes $R, R_1, \dots, R_{k+1}$, telles que pour tout $i \neq j$, $R_i \cap R_j \subseteq R$, alors on peut supprimer la requête R.
(5) Envergure Dominante : Si l'envergure d'une feuille contient $k+1$ requêtes dont les extrémités sont toutes distinctes, alors il existe toujours une solution optimale qui n'utilise pas l'arête incidente à cette feuille.

Sans entrer dans les détails techniques de cette règle (5), la règle la plus puissante et la plus complexe, l'envergure d'une feuille x est la partie de l'arbre à travers laquelle s'étend une requête la plus courte depuis x vers la gauche et une requête la plus courte depuis x vers la droite. La formulation ci-dessus de la règle (5) s'applique au cas où l'arbre est une *chenille* (en anglais un *caterpillar*), c'est-à-dire un chemin sur lequel sont attachées des feuilles. Dans le cas général, l'énoncé de la règle de l'envergure devient plus complexe.

Ces règles appliquées à une chenille réduisent la taille d'une instance à $O(k^5)$ (Théorème 11 dans la Section 2.1). Nous dédions une section entière au cas de la chenille, car il contient l'essence de notre preuve tout en restant relativement simple. Avec la généralisation de la règle (5) détaillée dans la Section 2.2, toute instance de MULTICOUPE DANS LES ARBRES est réduite à une taille $O(k^6)$ (Théorème 19).

Le troisième chapitre concerne la recherche d'un arbre orienté avec beaucoup de feuilles. Cette problématique est cruciale pour la conception et l'analyse de réseaux de communication. Un bon réseau possède souvent une "épine dorsale", c'est-à-dire un sous-arbre (si possible petit) qui domine le reste du graphe. Une telle épine dorsale correspond exactement aux noeuds internes d'un arbre avec beaucoup de feuilles.

L'existence et la recherche d'arbres avec beaucoup de feuilles est une problématique très étudiée, tant pour les graphes non-orientés que pour les graphes orientés. En non-orienté, ce problème, bien que NP-complet, est relativement simple, et des noyaux linéaires et algorithmes d'approximations à facteur constant étaient déjà connus. Nous nous sommes intéressés aux graphes orientés.

ARBRE ENRACINÉ AVEC BEAUCOUP DE FEUILLES :
Donnée : Un graphe orienté D, un sommet r appelé la *racine* de D, un entier k.
Paramètre : k.
Question : Existe-t-il un arbre orienté enraciné en r qui couvre D et qui possède au moins k feuilles ?

Un *arbre orienté* est un arbre où tout arc est orienté de la racine vers les feuilles, parfois appelé *arbre sortant*.

Dans le Chapitre 3, nous présentons un noyau quadratique pour le problème d'ARBRE ENRACINÉ AVEC BEAUCOUP DE FEUILLES (améliorant un noyau cubique de Fernau *et al.* [61]), ainsi qu'un algorithme d'approximation à facteur constant (l'unique algorithme d'approximation connu n'avait un facteur que $\sqrt{n}$ [50]).

Si la racine n'est pas précisée dans la donnée, alors le problème n'admet pas de noyau polynomial (sauf si la hiérarchie polynomiale s'écroule au troisième niveau) [61]. Cela signifie que la notion de noyau n'est pas purement algorithmique. En effet, le problème non-enraciné se réduit trivialement à n instances du problème enraciné (une par racine possible).

Le point de départ de notre travail est de se réduire aux graphes 2-connexes, c'est-à-dire sans sommet dont la suppression rend une partie du graphe inaccessible depuis la racine r. Ceci est exprimé par la règle (1) de la Section 3.2. Les graphes 2-connexes possèdent un ordre particulier sur les sommets appelé $r-r$ énumération. Dans un tel ordre (où la racine est dupliquée et apparait à la fois au début et à la fin), chaque sommet possède un voisin entrant avant et après lui. Dans la Section 3.1, nous utilisons une $r-r$ énumération pour exprimer des bornes combinatoires sur l'existence d'arbres avec beaucoup de feuilles. Notamment, nous montrons que dans un graphe orienté 2-connexe, il existe toujours un arbre couvrant avec au moins autant de feuilles que :

– une fraction constante du nombre de sommets de degré entrant au moins trois (Théorème 37).
– une fraction constante du nombre de sommets incidents à un arc entrant simple (Théorème 38).

Intuitivement, on aurait pu s'attendre à une borne dépendant du nombre de sommets de grand degré *sortant,* et non entrant comme dans le Théorème 37. Pour le Théorème 38, un arc (u,v) est dit *simple* si l'arc inverse, (v,u), n'apparait pas dans le graphe. L'existence de nombreux *2-circuits,* c'est-à-dire d'arcs (u,v) et (v,u), est l'obstruction principale à l'existence d'un arbre avec beaucoup de feuilles dans les graphes orientés 2-connexes.

Un noyau quadratique découle de ces résultats combinatoires couplés aux règles de réduction simples présentées dans la Section 3.2. Ce noyau pour le problème d'ARBRE ENRACINÉ AVEC BEAUCOUP DE FEUILLES est détaillé dans la Section 3.3, et un algorithme d'approximation à facteur constant est présenté dans la Section 3.4.

Cette approche donne un résultat plus satisfaisant du point de vue de l'approximation grâce au traitement possible des 2-circuits. Comme un noyau doit conserver la valeur exacte de la solution, nous ne pouvons pas nous débarrasser des longs chemins de 2-circuits. Dans le cadre de l'approximation au contraire, un long chemin de 2-circuits fournit soit une feuille soit deux feuilles dans un arbre, et il est donc facile de fournir une solution satisfaisante du point de vue de ces chemins de 2-circuits. Les résultats combinatoires de la Section 3.1 assurent par ailleurs une solution raisonnablement bonne sur les portions ne contenant pas de 2-circuits.

La question ouverte principale sur le problème d'ARBRE ENRACINÉ AVEC BEAUCOUP DE FEUILLES est l'existence d'un noyau linéaire. Notre noyau possède un nombre quadratique d'arêtes, et des résultats de borne inférieure correspondant exactement à un nombre quadratique d'arêtes existent désormais [45]. Une réponse négative proviendrait certainement de l'adaptation de ces techniques au problème d'ARBRE ENRACINÉ AVEC BEAUCOUP DE FEUILLES, et une réponse positive découlerait du traitement des longs chemins de 2-circuits.

Partie II : Structure et Décompositions de Graphes

La seconde partie de cet manuscrit se focalise sur la structure des graphes, et notamment sur les mineurs, les décompositions arborescentes et les classes de graphes à mineurs ou sous-graphes induits exclus. Ces thèmes sont étroitement liés, et souvent en rapport avec la théorie de l'ordre.

Un graphe H est un *mineur* d'un graphe G si H peut être obtenu à partir de G en effectuant une suite de contractions d'arêtes et de suppressions de sommets et d'arêtes. La relation de mineur est une relation plus riche que celle de sous-graphe.

Une *décomposition en arbre* (tree-decomposition en anglais) d'un graphe G est un arbre T et une collection $\{T_v | v \in V\}$ de sous-arbres de T telle que $T_u \cap T_v \neq \emptyset$ pour tout arête (u, v). La *largeur* d'un noeud de T est le nombre de sous-arbres T_v qui contiennent ce noeud, *moins un*. La *largeur* de la décomposition est le maximum des largeurs d'un noeud de T, et la *largeur d'arbre* (treewidth en anglais) d'un graphe G est le minimum des largeurs d'une décomposition en arbre de G.

Dans le Chapitre 4, nous nous intéressons aux multicoupes dans les graphes en général :

> MULTICOUPE :
> **Donnée** : Un graphe G, un ensemble de requêtes, un entier k.
> **Paramètre** : k.
> **Question** : Existe-t-il une multicoupe de taille au plus k, c'est-à-dire un ensemble S de k arêtes dont la suppression coupe toutes les requêtes ?

Nous réduisons le problème de MULTICOUPE aux graphes de largeur d'arbre bornée. Un graphe de grande largeur d'arbre admet une grande grille comme mineur (Théorème 57), et nous utilisons cette structure de grille-mineur pour extraire plus d'informations sur les multicoupes possibles.

Le coeur de notre approche réside dans l'étude de problèmes de connectivité sur trois sommets. Etant donné trois sommets x, y, z d'un graphe G, nous appelons $(zy|x)$-*coupe* de taille k un ensemble de k arêtes dont la suppression déconnecte y de x mais ne déconnecte pas y de z.

> SÉPARATION DE TRIPLET :
> **Donnée** : Un graphe G, trois sommets x, y, z de G, et un entier k.
> **Paramètre** : k.
> **Sortie** : Calculer une $(zy|x)$-coupe de taille au plus k s'il en existe.

Nous prouvons que ce problème de SÉPARATION DE TRIPLET est FPT (Théorème 63). Une notion plus complexe et plus pertinente vis-à-vis des multicoupes est celle de *lien fort*. Etant donné un ensemble de sommets T d'un graphe G, deux entiers k et k' et un sommet z de G, nous disons qu'un sommet $x \notin T$ est k'-*fortement* $(z|T)$-k-*lié* si pour tout sous-ensemble $S \subseteq T$ de grande taille, plus précisément tel que $|S| \geq |T| - k'$, il n'existe pas de $(zx|S)$-coupe de taille au plus k.

Un des résultats clés du Chapitre 4 est le Théorème 66, qui assure que quand un ensemble de sommets T est suffisamment grand (en fonction de k et de k'), il existe toujours un sommet $x \in T$ qui est k'-fortement $(z|T\backslash\{x\})$-k-lié. De ce résultat, nous déduisons deux règles de réductions pour MULTICOUPE. Une requête est *inutile* (*irrelevant* en anglais dans le Chapitre 4) si l'existence d'une solution ne change pas quand cette requête est supprimée.

Règle 1. *Un sommet incident à* $k^{O(k)}$ *requêtes est incident à une requête inutile.*

La Règle 1 est très commode, et la définition suivante a pour but d'étendre la recherche de requête inutile. Un ensemble de sommets T est dit *groupé* (*gathered* dans la version anglaise) si pour tout ensemble d'arêtes F, de taille au plus k, il existe au plus une composante connexe de $G \setminus F$ qui contient au moins deux sommets de T.

Règle 2. *Dans une instance réduite par la Règle 1, s'il existe un ensemble groupé* T *d'au moins* $k^{O(k)}$ *terminaux, alors on peut trouver en temps FPT une requête inutile.*

Cette règle de réduction est l'outil le plus puissant du Chapitre 4. Nous utilisons la structure donnée par une grande grille en mineur pour réorganiser l'instance jusqu'à pouvoir appliquer la Règle 2. Dans la Section 4.3, nous traitons le cas où l'instance admet, non seulement une grande grille, mais aussi une grande clique en mineur. Dans la Section 4.4, nous utilisons notamment un renforcement d'un résultat de Robertson et Seymour sur les grilles sans grande clique en mineur, le Lemme 85. Après un nettoyage approprié de l'instance, nous exhibons soit une arête inutile, car trop loin de tout terminal pour participer efficacement à une multicoupe, soit un grand ensemble groupé de terminaux. Dans les deux cas, l'instance de grande largeur d'arbre est réduite, et ce processus peut être répété jusqu'à l'obtention d'une instance de largeur d'arbre bornée en k.

Le Chapitre 5 concerne les sous-graphes induits. Le célèbre théorème des mineurs de Robertson et Seymour, brièvement décrit pages 64-65, établit que la relation de mineur est un *bel ordre*, c'est-à-dire que pour tout ensemble infini de graphes, il en existe un qui est un mineur d'un autre. Cet énoncé de théorie de l'ordre est d'une importance algorithmique cruciale, car il implique que toute classe de graphes close par mineur est exprimable par un nombre fini d'obstructions, et donc reconnaissable en temps polynomial.

Un résultat de ce type ne peut exister pour la relation plus contrainte de sous-graphe induit, qui n'est pas un bel ordre. Par exemple, l'ensemble des cycles est une *antichaine*, c'est-à-dire qu'aucun cycle n'est sous-graphe induit d'un cycle de taille différente.

Afin d'établir un méta-résultat, évidemment moins puissant que pour la relation de mineur, mais qui assure aussi la polynomialité dans le cas des sous-graphes induits, la notion de 2-bel ordre est intéressante. Une classe de graphes $\mathcal{S}$ est dite 2-bel-ordonnée (par sous-graphe induit) s'il n'existe pas d'antichaine infinie de graphes de $\mathcal{S}$ dont les sommets sont bicolorés, quand la relation de sous-graphe induit doit conserver les couleurs.

Un intérêt de cette notion de 2-bel ordre est qu'une classe de graphes 2-bel-ordonnée admet un ensemble fini de sous-graphes induits interdits, et est donc automatiquement reconnaissable en temps polynomial (Proposition 112). Le Chapitre 5 propose une caractérisation partielle des classes de graphes 2-bel-ordonnées. Nous utilisons la NLC-décomposition (notion équivalente à la décomposition en cliques). Etant donné deux graphes G et H, dont les sommets sont partitionnés en k classes de couleur, la NLC-décomposition permet de construire l'union disjointe des deux graphes G et H, et de choisir, pour chaque classe de couleur de G et chaque classe de couleur de H, si elles seront totalement adjacentes ou totalement non-adjacentes. Un graphe est dit de largeur NLC au plus k s'il est constructible à partir de sommets isolés, en utilisant cette opération d'union, et en s'autorisant à ré-étiqueter (c'est-à-dire essentiellement fusionner) les classes de couleur. Pour une définition formelle, nous invitons le lecteur à se référer au Chapitre 5.

Autoriser des ré-étiquetages quelconques permet de construire facilement des classes de graphes qui ne sont pas 2-bel-ordonnées, même avec seulement $k = 3$ classes de couleur. Nous étudions donc les restrictions de la largeur NLC où seules certaines opérations de ré-étiquetage sont autorisées. Nous caractérisons totalement les ensembles de ré-étiquetages autorisés qui mènent à des classes 2-bel-ordonnées, de plusieurs manières différentes.

Pour ce faire, nous définissons une relation $\leq$ sur les mots de fonctions (resp. sur les arbres de fonctions) qui correspond à la relation sous-mot (resp. sous-arbre) où la composition doit additionnellement être respectée. Nous définissons également une relation $\preceq$ sur les fonctions de ré-étiquetage basée sur les images : étant donné deux fonctions f et g, nous écrivons $f \preceq g$ quand $\mathrm{Im}(f \circ g) = \mathrm{Im}(f)$.

En dénotant par $\mathrm{NLC}_k^{\mathcal{F}}$ l'ensemble des graphes de largeur NLC au plus k quand les ré-étiquetages sont restreints aux fonctions appartenant à l'ensemble $\mathcal{F}$, nous montrons dans le Chapitre 5 que tous les énoncés suivants sont équivalents :

– L'ensemble de fonctions $\mathcal{F}$ est totalement ordonné par la relation $\preceq$.
– L'ensemble des mots sur $\mathcal{F}$ est bel-ordonné par $\leq$.
– L'ensemble des arbres sur $\mathcal{F}$ est bel-ordonné par $\leq$.
– L'ensemble de graphes $\mathrm{NLC}_k^{\mathcal{F}}$ est bel-ordonné (par sous-graphe induit).
– L'ensemble de graphes $\mathrm{NLC}_k^{\mathcal{F}}$ ne contient pas de chemins arbitrairement grands.
– L'ensemble de graphes $\mathrm{NLC}_k^{\mathcal{F}}$ est ∞-bel-ordonné.

Pour expliciter la dernière formulation, une classe de graphes est ∞-*bel-ordonnée* si elle est n-bel-ordonnée pour tout entier n, c'est-à-dire s'il n'existe pas d'antichaine infinie de graphes dont les sommets sont colorés avec n couleurs.

La conjecture qui a originellement motivé notre travail a été formulée par Pouzet [107] :

Conjecture 1. *Une classe de graphes close par sous-graphe induit est 2-bel-ordonnée si et seulement si elle est ∞-bel-ordonnée.*

En d'autres termes, utiliser plus que 2 couleurs n'apporterait aucune restriction supplémentaire. Nos résultats montrent que c'est bien le cas pour les classes de graphes $\mathrm{NLC}_k^{\mathcal{F}}$. Pour compléter la preuve de la conjecture de Pouzet, il suffirait donc de prouver la Conjecture 113 : toute classe de graphes $\mathcal{G}$ close par sous-graphe induit et 2-bel-ordonnée est-elle incluse dans une classe $\mathrm{NLC}_k^{\mathcal{F}}$ bel-ordonnée ? En d'autres termes, tout graphe de $\mathcal{G}$ est-il constructible par une décomposition NLC avec k couleurs où l'ensemble $\mathcal{F}$ des ré-étiquetages autorisés est totalement ordonné par $\preceq$, c'est-à-dire que pour toutes fonctions f et g de ré-étiquetage autorisées, $\mathrm{Im}(f \circ g) = \mathrm{Im}(f)$ ou $\mathrm{Im}(g \circ f) = \mathrm{Im}(g)$?

Dans le Chapitre 6, nous nous intéressons à une classe de graphes définie de manière géométrique, les graphes de cercle. Un graphe G est un *graphe de cercle* s'il existe un ensemble de cordes d'un cercle, en bijection avec les sommets de G, tel que deux sommets sont adjacents dans G si et seulement si les deux cordes correspondantes s'intersectent. L'ensemble de cordes est alors appelé un *modèle* de G

Une des sous-classes naturelles des graphes de cercle est la classe des graphes de cercle de Helly. Un ensemble d'objets a la *propriété de Helly* si tout sous-ensemble d'objets, qui s'intersectent deux à deux, possède un point en commun. Un graphe de cercle est dit de Helly s'il possède un modèle avec la propriété de Helly, c'est-à-dire où toute clique correspond à des cordes qui s'intersectent exactement au même point.

Tous les graphes de cercle ne sont pas des graphes de cercle de Helly. Par exemple, le diamant (une clique à quatre sommets à laquelle on enlève une arête) est un graphe de cercle qui n'admet pas de modèle avec la propriété de Helly. Le Chapitre 6 est consacré à prouver que le diamant est essentiellement la seule obstruction, c'est-à-dire que les graphes de cercle de Helly sont exactement les graphes de cercle sans diamant induit. Ce résultat avait été conjecturé par Durán [51]. Cette caractérisation fournit un algorithme efficace pour la reconnaissance des graphes de cercle de Helly. Cette reconnaissance peut s'effectuer avec la même complexité que la reconnaissance des graphes de cercle, en temps quadratique avec l'algorithme de Spinrad [119], et même en temps quasi-linéaire avec l'amélioration récente de Gioan *et al.* [54].

Notre preuve est constructive. Nous partons d'un modèle (sans nécessairement la propriété de Helly) d'un graphe de cercle G sans diamant, et nous faisons grandir un sous-graphe induit H de G tel que le modèle restreint à H possède la propriété de Helly. Nous modifions le modèle de G sans changer l'ordre circulaire des extrémités des cordes. Nous maintenons plusieurs autres propriétés pour ce sous-graphe H. Par exemple, H doit être *convexe*, c'est-à-dire qu'une corde qui se trouve entièrement entre deux cordes de H doit elle aussi appartenir à H. Tant que $H \neq G$, nous trouvons un sommet $u \notin H$ tel que $H \cup u$ est convexe, et dont le voisinage possède de bonnes propriétés d'ordre. Nous montrons qu'une notion naturelle de successeur et de prédécesseur peut être définie sur le voisinage de u et propagée. Ceci nous permet de trouver une séquence S de cordes contenant u et basée sur la relation successeur / prédécesseur, telle que de petites mouvements sur les extrémités des cordes de S peuvent être exécutés sans détruire le modèle, et de telle manière que le modèle sur $H \cup S$ vérifie la propriété de Helly.

Partie III : Algorithmes Paramétrés

La troisième partie de ces travaux concerne les algorithmes paramétrés. Dans le Chapitre 7, nous prouvons que MULTICOUPE est FPT (soluble à paramètre fixé). Le problème de MULTICOUPE était l'un des derniers problèmes naturels importants dont la solubilité à paramètre fixé restait ouverte.

La preuve exposée dans le Chapitre 7 n'utilise pas explicitement la réduction à largeur d'arbre bornée du Chapitre 4. Les outils principaux sont cependant issus des idées exposées dans le Chapitre 4, notamment les résultats de connectivité à base de séparation de triplets. L'autre idée clé, la traduction des requêtes en clauses 2-SAT, provient indirectement de la réduction à largeur d'arbre bornée, puisque cette idée est née en étudiant le problème de MULTICOUPE sur des instances très simples de petite largeur d'arbre. Nous donnons ici une synthèse des principales étapes de notre preuve, et nous renvoyons le lecteur au Chapitre 7 pour un cadre plus formel.

Le point de départ de notre preuve est l'obtention d'une *multicoupe de sommets*, c'est-à-dire d'un ensemble de sommets dont la suppression coupe toutes les requêtes, de taille au plus $k+1$. De plus, nous pouvons supposer que la multicoupe de sommets Y doit être *explosée* par la solution, c'est-à-dire que toute paire de sommets de Y doit être séparée par la solution. Un tel raisonnement avait été utilisé pour la première fois dans [99]. L'ensemble Y donne un début de structure à l'instance : on peut se focaliser sur les composantes connexes de $G \setminus Y$, que nous appelons les *Y-composantes*. Les sommets de Y adjacents à une Y-composante sont ses *points d'attachement*. Pour simplifier, nous regroupons toutes les Y-composantes adjacentes à un même unique point d'attachement $y \in Y$ en une seule Y-composante (que nous appelons une y-cerise).

Le nombre de Y-composantes est borné en k, nous pouvons donc brancher pour décider combien d'arêtes de la solution doivent se trouver dans chaque Y-composante.

Aucune requête n'a ses deux extrémités dans une même Y-composante avec au moins deux points d'attachement, et nous pouvons donc simuler chaque requête (u,v) par plusieurs *demi-requêtes* (u,y,v), où y est un point d'attachement commun des Y-composantes de u et de v. Informellement, couper la demi-requête (u,y,v) signifie couper tout chemin entre u et v qui passe par y.

Les demi-requêtes donnent une structure plus simple au problème : couper la demi-requête (u,y,v) revient soit à séparer u de y dans la composante de u, soit à séparer v de y dans la composante de v. Nous allons pouvoir nous focaliser séparément sur chaque composante, et exprimer de manière logique la contrainte liée à une demi-requête par une clause 2-SAT [7], à condition de pouvoir exprimer simplement (c'est-à-dire avec une variable 2-SAT) le fait que u soit séparé de y dans sa composante. C'est l'objectif principal du reste de notre preuve.

En utilisant des branchements extensifs, nous réduisons les composantes à trois sommets d'attachement ou plus, dans le Lemme 150. Il ne reste plus que des cerises et des Y-composantes à deux points d'attachement (que nous appelons alors 2-composantes).

Les cerises sont essentiellement traitées avec les résultats de connectivité, du type séparation de triplets, donnés au Chapitre 4. En effet, la séparation ou non d'un sommet avec l'unique point d'attachement de sa cerise est l'unique paramètre pertinent pour évaluer une coupe à l'intérieur d'une cerise. Ces résultats de connectivité sont reformulés d'une manière plus adaptée à notre propos et étendus à la Section 7.4.

7. 2-SAT est le problème de satisfiabilité de formules logiques dont les clauses n'ont que deux variables.

Pour donner plus de structure aux 2-composantes, qui constituent désormais le coeur du problème, nous montrons au Lemme 152 l'existence (après branchement) d'un chemin particulier appelé *l'épine dorsale* de la 2-composante. Ce chemin, entre les deux points d'attachement de la 2-composante, doit comporter exactement une arête de la solution. L'ensemble des multicoupes est alors partiellement linéairement ordonné, en fonction de l'arête de l'épine dorsale choisie par la multicoupe. Il est possible d'exprimer très simplement le fait qu'une multicoupe appartienne à une section finale ou initiale d'un ordre linéaire total. Notre but est donc d'assurer que les multicoupes qui séparent un sommet donné et un point d'attachement de sa 2-composante consistent exactement en une section finale ou initiale d'un ordre total.

L'instance, telle que transformée jusqu'ici, correspond au problème intermédiaire, défini à la Section 7.5, que nous appelons COMPONENT MULTICUT. Nous étudions ensuite une problème plus général que COMPONENT MULTICUT, que nous appelons BACKBONE MULTICUT dans la Section 7.6. Les instances de BACKBONE MULTICUT comportent notamment des clauses 2-SAT, ce qui nous permet d'enrichir les instances en traduisant au fur et à mesure des contraintes sur le type de solution recherchée en des formules logiques.

Nous transformons ensuite les instances avec des branchements extensifs jusqu'à ce que chaque sommet de l'épine dorsale soit un sommet séparateur entre les deux sommets d'attachement de la 2-composante. C'est une partie très technique de la preuve, qui se conclut avec le Lemme 158.

Avec la forme des 2-composantes légèrement simplifiée, nous atteignons finalement notre but : linéariser totalement (une fois de plus via des branchements) l'ensemble des multicoupes. Nous définissons à la Sous-section 7.6.6 un ordre pertinent $\preceq$ sur les multicoupes. Cet ordre total nous permet, avec les outils de connectivité spécifiques à la Section 7.4, de réduire les 2-composantes à seulement leur épine dorsale. C'est la seconde partie très technique de notre preuve.

A ce stade, l'instance se compose simplement de cerises attachées sur un sommet de Y et de chemins isolés entre deux sommets de Y. Il est alors possible d'exprimer totalement les contraintes associées aux requêtes par des clauses 2-SAT, et de finalement résoudre la formule 2-SAT obtenue en temps polynomial.

Cet algorithme à paramètre fixé pour MULTICOUPE a une complexité simplement exponentielle en k. Notre preuve étant complexe, il devrait cependant être très difficile d'implémenter l'algorithme en pratique. Trouver une preuve plus simple menant à un algorithme utilisable pour MULTICOUPE reste un problème très intéressant.

Dans le dernier chapitre, le Chapitre 8, nous abordons le problème de l'existence d'arbres orientés avec beaucoup de feuilles du point de vue des algorithmes paramétrés, contrairement au Chapitre 3 où un point de vue combinatoire avait mené à un noyau et à un algorithme d'approximation.

Le résultat principal du Chapitre 8 est un algorithme paramétré de temps d'exécution

$O^*(3.72^k)$ pour trouver un arbre orienté avec au moins k feuilles. Nous raffinons un algorithme de complexité $O^*(4^k)$ obtenu dans [88]. C'est un algorithme de branchement très simple, et qui a pourtant apporté une très importante amélioration par rapport à la longue liste des algorithmes crées auparavant, tant pour les graphes orientés que non-orientés.

L'idée est de faire grandir une solution partielle en branchant à chaque étape sur différentes manières d'étendre l'arbre partiel. Plus précisément, à chaque pas, une feuille x de l'arbre partiel est sélectionnée. Dans la première branche, on suppose que x sera une feuille dans la solution finale, et dans la deuxième branche, on suppose que x sera interne. Dans le premier cas, on gagne une feuille de la solution finale, et il est facile dans le deuxième cas d'assurer le gain d'au moins une feuille dans la solution partielle. Ces deux nombres de feuilles ne peuvent dépasser k, et la profondeur de l'arbre représentant les branchements de l'algorithme est donc au plus $2k$, ce qui donne un algorithme en temps $O^*(4^k)$.

Pour améliorer cette complexité, nous parvenons à trouver, dans les branchements les moins favorables pour la complexité de l'algorithme, un sommet extérieur à la solution partielle que l'on peut supposer être forcément une feuille dans la solution finale sans perte de généralité. C'est l'étape (4.2) de notre algorithme $\mathcal{B}(D, T, L)$. Assurer un gain supplémentaire dans les branchements non favorables nous permet de descendre strictement sous $O^*(4^k)$, ce qui entraine une amélioration pour le problème classique (non paramétré).

Pour calculer le nombre maximum de feuilles d'un arbre couvrant dans un graphe orienté ou non-orienté, l'algorithme évident de force brute consiste à énumérer tous les sous-ensembles de sommets, et à décider (en temps polynomial) s'il existe un arbre couvrant dont les feuilles sont exactement l'ensemble donné. Cet algorithme prend un temps $O^*(2^n)$, où n est le nombre de sommets du graphe. Aucun meilleur algorithme n'était connu. Dans la Section 8.3, nous utilisons notre algorithme paramétré pour décider l'existence d'un arbre couvrant avec k feuilles pour les entiers k depuis 1 jusqu'à une borne légèrement plus grande que $\frac{n}{2}$, et nous complétons l'analyse pour les entiers k significativement plus grand que $\frac{n}{2}$ par l'algorithme de force brute. Comme notre algorithme paramétré est significativement meilleur que $O^*(4^k)$, et comme l'algorithme de force brute n'est pas appliqué sur les valeurs de k voisines de $\frac{n}{2}$, nous cassons la barrière de 2^n. Plus précisément, cet algorithme hybride s'exécute en temps $O(1.9973^n)$. Cette complexité a depuis été significativement améliorée pour les graphes non-orientés [108].

Pour finir, la Section 8.4 est dédiée à un algorithme paramétré randomisé pour le problème dual qui consiste à trouver un arbre couvrant avec peu de feuilles. Nous utilisons le paradigme développé dans [89] pour décider en temps randomisé $O^*(2^k)$ s'il existe un arbre couvrant avec au moins k noeuds internes dans un graphe orienté ou non-orienté. Nous exhibons un polynôme multivarié correspondant à un graphe, tel que le graphe possède un arbre avec k noeuds internes si et seulement si le polynôme contient un monôme multilinéaire de degré k. Le résultat fondamental de [89], exprimé au Lemme 178, permet justement de décider l'existence d'un tel monôme multiliénaire de degré k en temps randomisé $O^*(2^k)$. Intuitivement, l'idée de ce type de transformation est d'associer une variable à chaque sommet du graphe, et de construire un polynôme de telle manière que les monômes correspondent à des sous-arbres du graphe, ou plus précisément à une "marche arborescente" sur le graphe. La multilinéarité (c'est-à-dire le fait que toute variable ait degré au plus un dans le monôme) permet d'assurer que le même sommet n'est pas utilisé deux fois, c'est-à-dire qu'on trouve bien une structure avec k sommets.

CHAPTER

1 Preliminaries

1.1 Introduction to the Thesis

Themes of the thesis.

This thesis revolves around three tightly linked themes: efficient algorithms for graphs problems, combinatorial and structural properties of graphs, and order theory. This trinity algorithms / structures / orders has proved successful in graph theory. For example, the celebrated minor theorem of Robertson and Seymour results of an interplay between a structural approach (tree decomposition, structure theorem for constructing graphs excluding a minor from graphs embedded in a surface), order properties (the well-quasi-order of the minor relation) and strong algorithmic properties (the Fixed-Parameter Tractability of the linkage problem, the polynomiality of all problems closed under minor).

The main results of this thesis are algorithmic statements. For example, the MULTICUT problem is FPT (Fixed-Parameter Tractable), and finding trees with many leaves in a digraph admits a constant-factor approximation algorithm. We are interested in hard (NP-complete) problems, which have to be tackled through a paradigm such as approximation algorithms, FPT algorithms or polynomial kernels. This thesis focuses on parameterized complexity, a rather recent and very promising domain.

Our methods are mostly combinatorial and structural. Tree decompositions (tree-width, clique-width) are a central tool in this work. We also study other properties such as the structure of graphs with large grid-minor and without large clique-minor, or the structure of diamond-free graphs, for themselves as well as for their algorithmic use.

When possible, we used order theory. A chapter is devoted to the well-quasi-order of the induced subgraph relation, and tools such as Dilworth's Theorem or $s-t$ numberings have been of great use to refine our understanding of structures.

Results of the thesis.

The graph problem central to this thesis is the MULTICUT problem. Given a graph and a set of pairs of vertices (the requests), we want to simultaneously cut all requests by removing as few edges as possible. This is a hard problem (NP-complete) of fundamental importance, which generalises the basic cut problem. Solving MULTICUT is also important in practice to design and study networks (communication networks, VLSI design, ...).

The two central results in our work concern this MULTICUT problem. First, the MULTICUT IN TREES problem admits a polynomial kernel [1]. In other words, instances of MULTICUT IN TREES can be reduced to a small size (as a function of the desired solution size). This result is the aim of Chapter 2.

Second, the MULTICUT problem is FPT [2]. In other words, the exponential part of the complexity of the MULTICUT problem (unavoidable if **P**$\neq$**NP**) lies in the size of the desired solution, and not in the whole size of the input graph. This result is expressed in Chapter 7.

The other main results of this thesis are the following. For finding a tree with many leaves in a digraph (an NP-hard problem), we provide a quadratic kernel, a constant factor approximation algorithm [3] and a fast parameterized algorithm [4]. This results are exposed in Chapter 3 and Chapter 8. Finding a tree with many leaves, or in other words in a small connected dominated set, is a central problem in the domain of communication networks, widely used to find the backbone of a network.

The previous results belong to the area of parameterized complexity. The following results on excluded induced graphs also have algorithmic applications, but directly to polynomial algorithms. Chapter 5 is devoted to a partial characterisation of graph classes 2-well-quasi-ordered under the induced subgraph relation [5]. The main motivation of this work is that every graph problem closed under induced subgraphs and restricted to a 2-well-quasi-ordered class admits a polynomial time algorithm. The last theme of this thesis is geometric graph classes. Chapter 6 is devoted to proving that Helly circle graphs are exactly the circle graphs without an induced diamond [6].

1.2 Fixed-Parameter Tractability

Standard complexity theory aims at establishing the difficulty of a problem in terms of the size of the instance. Many important problems are **NP**-complete, and thus are seen as equivalent in the usual complexity paradigm. Fixed-Parameter Tractability has been designed to discriminate these problems further, according to a second dimension: the

1. Result presented at STACS 2009.
2. To be presented at STOC 2011.
3. Results presented at IWPEC 2009.
4. Work published in Journal of Computer and System Sciences (2010).
5. Work published in ORDER (2010).
6. Result published in Discrete Mathematics (2010).

difficulty of a problem is expressed in terms of the instance size together with another parameter. Some relevant and commonly used parameters are: a width of the input graph (treewidth, cliquewidth...), the solution size, the maximum degree of the input graph, but a whole zoology of parameters has been appearing recently.

Such a two dimensional paradigm leads to an even larger diversity of complexity classes. This parameterized refinement is meaningful both theoretically and practically. Citing Flum and Grohe [62]: "Measuring complexity only in terms of the input size means ignoring any structural information about the input instances in the resulting complexity theory. Sometimes, this makes problems appear harder than they typically are".

More formally, a problem is said to be *Fixed-Parameter Tractable* with respect to a parameter if it admits an algorithm running in time $f(k) * poly(n)$ on instances of size n and of parameter k, for some computable function f. The parameter of an instance is also required to be polynomial-time computable. In other words, the algorithm is uniformly polynomial for fixed k. This is a more restrictive notion than the mere polynomiality for fixed k expressed by the complexity class **XP**. Indeed, algorithms of running time of the type $2^k * n$ and n^k have very different behaviors in practice.

Parameterized complexity has been originally developed during the 1990s by Downey, Fellows and their co-authors in a series of papers (see their book [49]).

From a practical standpoint, **NP**-hard problems are difficult to tackle without restriction on the possible inputs. Fortunately, practical instances are usually structured. For example, rather than having to answer queries to any VERTEX COVER instance, we might have to answer only questions of the type: does this instance admit a vertex cover of size at most k, where $k \leq 20$. Or does this graph of treewidth at most 20 admit a c-colouring. In these cases, a good parameterized algorithm for VERTEX COVER parameterized by the solution size, or for c-colouring parameterized by the treewidth should allow to tackle the practical problem satisfactorily. Indeed, Vertex Cover admits an algorithm running in time $1.28^k + kn$ [26].

Parameterized Techniques.

Among the most common parameterized techniques stand bounded search trees, colour coding, dynamic programming on tree decompositions, iterative compression, multilinear monomial randomized techniques, and kernels.

The use of a *bounded search tree* is a very natural branching technique. At each step, we branch towards a certain number of new instances. The whole process can be visualized as a tree of instances rooted in the original instance. When the maximum degree and the depth of the search tree are bounded in terms of the parameter k, and the amount of work required at each branching step (and to solve each leaf instance) is polynomial, the whole algorithm runs in time $f(k) * poly(n)$. The bounded search tree technique will be used in Chapter 8 to design a parameterized algorithm for finding directed trees with many leaves.

Colour coding is a probabilistic method introduced by Alon *et al.*[5]. Suppose you are looking for a path of length k in a graph. Colour coding consists in randomly k-colouring the vertices of the graph, and looking for a path where each colour is represented exactly once, which is a much easier task. The key is to prove that with a reasonably high probability, a k-path in the original graph becomes a colourful path. One can repeat this process until a solution has been found with a large enough probability (and this can be derandomized into a deterministic parameterized algorithm). Colour coding will not be used in this thesis.

Dynamic programming on tree decompositions can be seen as the origin of the parameterized framework. To tackle the k-Linkage problem, Robertson and Seymour designed tree-decompositions. Instances of large treewidth have a particular structure, containing a large grid minor, which can be used to solve the problem. On the other hand, instances of small treewidth can be tackled using their tree decomposition. The most straightforward use of a tree decomposition of small width is called dynamic programming. This dichotomy between tree decomposition of small width and large grid minor structure allowed Robertson and Seymour to show that k-linkage is FPT on their way to the graph minor theorem. This deep connection between tree decompositions and parameterized complexity is at the heart of this thesis, and in particular of Part II. The interplay between parameterized complexity and tree decomposition yielded the celebrated Courcelle's Theorem [33]: every problem definable in monadic second order logic is FPT when parameterized by the treewidth of the input graph. This implies that many natural and important **NP**-hard problems are FPT when parameterized by the treewidth of the input graph, such as INDEPENDENT SET, COLOURING, FEEDBACK VERTEX SET.

Iterative compression consists in, given a solution for a slightly smaller instance (for example, the graph minus a vertex or minus an edge), trivially growing it into a solution for the whole instance, and trying to compress this slightly too large solution into an admissible solution. If such a compression can be performed in FPT time, then growing the desired structure step by step from scratch (say, adding a vertex or an edge of the instance graph at each step) and repeatedly performing this compression leads to a parameterized algorithm. This technique will be used in Chapter 7.

Koutis and Williams [89] have developed a randomized technique based on finding a multilinear monomial in a polynomial presented as a circuit. This technique allows to find $O^*(2^k)$ [7] randomized algorithms for a number of problems, including testing the existence of a given tree on k nodes as a subgraph, obtaining better bounds than non-randomized algorithms. These randomized algorithms do not appear to be efficiently derandomizable though. We will discuss the applications of this technique for finding trees with many leaves or many internal vertices in Chapter 8.

Kernelization consists in reducing the size of an instance until it is small with respect to the parameter. It is not only a technique to design parameterized algorithms, but rather a

7. In the O^* notation we omit the polynomial terms, so that $O^*(f(k))$ stands for $O(P(n)f(k))$ for some polynomial P.

parallel framework interesting by itself. The next section gives an introduction on kernels, and Part I of this thesis is devoted to kernelization.

A Hardness Theory.

Fixed-Parameter Tractability has a well-developed hardness theory. The class **W[P]** plays the role of the class **NP** in the parameterized world, and is refined by the **W**-hierarchy. The **A**-hierarchy plays the role of the polynomial hierarchy. These two hierarchies coincide on their first level, and it is believed that the class FPT is strictly included in this first level **W[1]=A[1]**. The fact that **W[P]≠FPT** unless **P=NP** has not been proved, but a weaker version involving limited nondeterminism holds. Consider the class **NP[f]** of problems decided by an **NP** machine performing at most $f(n)$ nondeterministic steps on inputs of size n. We know that **W[P]≠FPT** unless **P=NP[**α**(n)log(n)]** where α is any computable unbounded function. Compare with the fact that **P=NP[log(n)]**. In other words, if supra-logarithmic nondeterminism gives more power than polynomial time, then **W[P]**-hard problems are indeed not FPT.

Notorious FPT problems and Open question(s).

- Minor testing (parameterized by the size of the target graph) and the k-Linkage problem are FPT by the graph minor series results.
- Topological minor testing and immersion testing are FPT as well [76].
- Computing an optimal tree decomposition (parameterized by the treewidth) is FPT [9].
- DIRECTED FEEDBACK VERTEX SET (removing k vertices to make a digraph acyclic) parameterized by the solution size is FPT [29].
- ALMOST 2-SAT (removing k clauses to make a collection of clauses with two literals satisfiable) is FPT [109].
- ODD CYCLES TRANSVERSAL (removing k vertices to make a graph bipartite) is FPT [110].
- MULTICUT (parameterized by the solution size) is FPT (Chapter 7 or [100]).[8]
- Is BICLIQUE (deciding the existence of a k by k complete bipartite subgraph) FPT?[9]

The FPT status of many important natural problems is now known, and the focus has shifted towards areas such as: above/below lower bound parameterizations, subexponential algorithms on planar graphs, meta-results.

8. This implies that ODD CYCLES TRANSVERSAL, among others, is FPT.
9. BICLIQUE has no polynomial kernel if **P≠NP** [30].

1.3 Kernelization

1.3.1 Introduction to Kernels

Reducing the size of an instance, via preprocessing or data reduction, is an important topic in computer science. Given an **NP**-hard problem, there can exist no polynomial time algorithm reducing the size of an instance by at least one bit if **P**$\neq$**NP** (otherwise, repeating this process gives a polynomial algorithm for the **NP**-hard problem). Kernelization is a more reasonable attempt in this direction: the goal is to reduce the size of the instance until it is small enough.

Formally, a *kernelization algorithm* for a parameterized problem Π is a polynomial time algorithm which receives as input an instance (I,k) of Π and outputs another instance (I',k') of Π such that, for some computable function f:

1. $k' \leq f(k)$.
2. $|I'| \leq f(k)$.
3. the instances (I,k) and (I',k') are both true or both false.

The reduced instance (I',k') is called a *kernel*. If I' is a substructure (say, a subgraph) of I, then I' can be seen as the core of the instance I for this problem, hence the denomination "kernel". However, I' can be totally unrelated to I.

The existence of a kernelization algorithm for a decidable problem clearly implies that the problem is FPT as one can kernelize the instance, and then solve the reduced instance (I',k'), no matter how slow the decision algorithm is. This whole process runs in time $poly(n) + g(k)$, for some function g.

The converse actually holds. Given an FPT problem, one can run a parameterized algorithm during a (large enough) polynomial number of steps, return a trivially true/false instance if the answer has been computed, otherwise return the original instance, whose size is then bounded in terms of the parameter.

Hence a problem is FPT if and only if it admits a kernelization algorithm. The size of the reduced instance given by this result is not necessarily small with respect to the parameter. A much more constrained condition is to be able to reduce to an instance of polynomial size in terms of k. We then say that the problem admits a *polynomial kernel*. Some problems are FPT but do not admit a polynomial kernel unless **P=NP**. More formally, a *polynomial kernelization algorithm* for a parameterized problem Π is a polynomial time algorithm which receives as input an instance (I,k) of Π and outputs another instance (I',k') of Π such that:

1. $k' \leq k$.
2. $|I'| \leq P(k)$, where **P** is a polynomial.
3. the instances (I,k) and (I',k') are both true or both false.

The above definition of a polynomial kernel is the most standard. Alternative notions do exist. For example, one can relax condition (1.) to $k' \leq \mathrm{poly}(k)$. Most polynomial kernels designed in the literature satisfy the above strong definition, but weaker notions should be of interest. Names such as "weak polynomial kernel" have been suggested for this type of relaxation. What is merely called a kernel here is sometimes referred to as a "strong kernel". The existence of a (strong) kernel is equivalent to the existence of a weak kernel, but there exist problems which have a weak polynomial kernel but no (strong) polynomial kernel [30].

Note that when a graph problem admits a polynomial kernel, rather than expressing the kernel size in terms of bitlength, it is usually expressed in terms of number of vertices.

A kernelization algorithm can be used as a preprocessing step to reduce the size of the instance before applying another algorithm. Being able to ensure that this kernel has actually polynomial size in k enhances the overall speed of the algorithm. See [81] for a review on kernelization.

1.3.2 Kernel Lower Bounds

Kernelization now has a hardness theory, which has been developed in the past few years. The most natural way to prove that a problem admits no polynomial kernel if **P**$\neq$**NP** is to prove that it has a parameter-decreasing self-reduction [30]. For example, SAT parameterized by the number of variables has a parameter-decreasing self-reduction: an instance is equivalent to the logical "OR" of the two instances obtained by replacing a given variable by *true* and *false* respectively. This reduction strictly decreases the parameter, and the size blow-up is polynomial (here, by a factor two). Interleaving such self-reductions with applications of a (strong) polynomial kernel would give a polynomial-time algorithm:

Theorem 2 ([30]). *An* **NP***-hard problem which has a parameter-decreasing self-reduction admits no (strong) polynomial kernel unless* **P=NP**.

SAT parameterized by the number of variables and k-Rooted Path have no (strong) polynomial kernel unless **P=NP** by Theorem 2. Indeed, k-Rooted Path (finding a path of length k starting at a given vertex in a graph or a digraph) admits a slightly less trivial parameter-decreasing self-reduction, which essentially consists in branching over the first edge of the desired path and gluing results together to form a single graph.

A breakthrough lower bound technique called OR-composition has been found in [12], based on a result by Fortnow and Santhanam [67]. A parameterized problem $\mathcal{P}$ is said to be *OR-composable* if there exists a polynomial-time algorithm which, given a set of instances of $\mathcal{P}$, computes an instance of $\mathcal{P}$ equivalent to their logical "OR", and whose parameter is polynomially bounded in the maximum parameter of an input instance.

Theorem 3 ([12])**.** *A parameterized* **NP***-hard problem which is OR-composable admits no polynomial kernel unless the polynomial hierarchy collapses to the third level.*

This even holds for weak polynomial kernels.

Theorem 3 can be made slightly more flexible, working with parameterized reductions to transfer lower bounds from a problem to another. Such a technique was used to prove that deciding the existence of k vertex-disjoint cycles admits no polynomial kernel [16] and that finding directed trees with at least k leaves admits no polynomial kernel [61].

A very recent strengthening of the OR-composition can be found in [15]. A problem $\mathcal{P}$ is said to *cross-compose* into $\mathcal{Q}$ if there exists a polynomial-time algorithm which, given a set of instances of $\mathcal{P}$, computes an instance of $\mathcal{Q}$ equivalent to their logical "OR", and whose parameter is polynomially bounded in the maximal size of an input instance. The cross-composition technique generalises the OR-composition technique in two ways: the parameter bound is relaxed, and one can reduce from a different problem.

Theorem 4 ([15])**.** *A parameterized problem into which an (unparameterized)* **NP***-hard problem cross-composes admits no polynomial kernel unless the polynomial hierarchy collapses to the third level.*

Dell and van Melkebeek refined the OR-composition technique to obtain precise lower bounds in [45]. They essentially show that instances of d-SAT cannot be compressed away from the trivial $O(n^d)$ bound, and infer the following result:

Theorem 5 ([45])**.** *An* **NP***-hard vertex-deletion problem closed by subgraphs does not admit a kernel of size* $O(k^{2-\epsilon})$ *for any* $\epsilon > 0$*.*

Note that size means bitlength (and not number of vertices) in this result. VERTEX COVER, FEEDBACK VERTEX SET and BOUNDED-DEGREE DELETION are such problems, and an edge-quadratic (and thus tight) kernel is known for each of them (see [24], [122] and [59] respectively).

1.3.3 Polynomial Kernels

The existence of a polynomial kernel can be a subtle issue. A result by Fernau *et al.* [61] shows that ROOTED MAXIMUM LEAF OUT-BRANCHING has a cubic kernel while MAXIMUM LEAF OUT-BRANCHING does not, unless the polynomial hierarchy collapses to the third level (using the OR-composition technique). So the notion of polynomial kernel is not purely algorithmic, since MAXIMUM LEAF OUT-BRANCHING can be solved by n applications of an algorithm for ROOTED MAXIMUM LEAF OUT-BRANCHING (one for each possible root).

On the positive side, Bodlaender *et al.* designed powerful meta-results in [14] to prove the existence of polynomial (or even linear) kernels in bounded genus graphs. These results apply to problems which can be expressed in Monadic Second-Order logic, and satisfy compactness properties. Extending this work, Fomin *et al.* showed that the bidimensionality framework can be used to establish linear kernels on classes excluding a minor [66].

Notorious problems admitting polynomial kernels and open questions.

- VERTEX COVER (hitting all edges of a graph with k vertices) admits a $2k$-vertex kernel [26].
- More generally, d-Hitting Set (finding a set of k elements hitting a collection of subsets of size d) admits a kernel of size $O(k^{d-1})$ [1].
- A number of problems including FEEDBACK VERTEX SET, DOMINATING SET, CONNECTED DOMINATING SET, INDEPENDENT SET, and MINIMUM LEAF SPANNING TREE admit linear kernels on planar graphs [14].
- MULTICUT IN TREES admits an $O(k^6)$-vertex kernel (see Chapter 2).
- Does DIRECTED FEEDBACK VERTEX SET admit a polynomial kernel?
- Does ODD CYCLES TRANSVERSAL admit a polynomial kernel?
- Does CHORDAL DELETION (removing k vertices to make a graph chordal, *i.e.* without induced cycles of length four or more) admit a polynomial kernel? [10]
- Does CLIQUE COVER (covering the edges of a graph by at most k edge-disjoint cliques) admit no polynomial kernel?

This was a brief introduction to Fixed-Parameter Tractability. For more on this topic, we refer the reader to the flourishing literature, for example to the three now classical books: [49, 62, 105].

1.4 Multicut

1.4.1 Introduction to Multicut

The central problem in this thesis is the MULTICUT problem, an important and natural (and **NP**-hard) generalisation of the standard (and polynomial time decidable) cut problem.

Given a graph G and a set R of *requests* between pairs of vertices (these vertices are called *terminals* or *endpoints*), a(n) (edge)-*multicut*[11] is a subset F of edges of G whose removal separates the two endpoints of every request (*i.e.* the two endpoints of a request lie in different connected components of $G \setminus F$).

MULTICUT:
Input: A graph $G = (V, E)$, a set of requests R, an integer k.
Parameter: k.
Output: TRUE if there is a multicut of size at most k, otherwise FALSE.

10. CHORDAL DELETION is FPT [97].
11. In this thesis, the term *multicut* stands for edge-multicut.

When parameterized by the solution size as above, MULTICUT was considered as one of the main open problems of the fixed parameterized complexity theory [46].

State of the art on MULTICUT.

MULTICUT and its variants have raised an extensive literature. These problems play an important role in network issues, such as routing and telecommunication (see [32]). For example, vertices of the graph could represent Urban Switch Centers in a telephone network, and (weighted) edges represent physical connections between vertices [23].

The MULTICUT problem is already hard when restricted to trees. Indeed, VERTEX COVER can be viewed as MULTICUT in stars. Hence MULTICUT IN TREES is **NP**-complete and **MaxSNP** hard, which implies that it admits no Polynomial-Time Approximation Scheme (PTAS) unless **P=NP**. Garg *et al.* [72] proved that MULTICUT IN TREES admits a factor 2 approximation algorithm, by showing that in trees the minimal multicut size is at most twice the maximal flow value, and using a primal-dual approach. Guo and Niedermeier [80] proved that MULTICUT IN TREES is FPT with respect to the solution size.

Another variant is the MULTIWAY CUT problem in which a set of terminals has to be pairwise separated. Parameterized by the solution size, MULTIWAY CUT has been proved FPT by Marx [96]. A faster $O^\star(4^k)$ algorithm is due to Chen *et al.* [28].

On general instances, Garg *et al.* gave an approximation algorithm for MULTICUT within a logarithmic factor in [71], proving that the minimum multicut size is within a factor $O(log(l))$ of the maximum multiflow value in general graphs, where l is the number of requests. However, MULTICUT has no constant factor approximation algorithm if Khot's Unique Games Conjecture holds [25]. The fact that MULTICUT is probably not tractable from an approximation standpoint motivates the study of the fixed parameterized tractability of MULTICUT.

Guo *et al.* showed in [79] that MULTICUT is FPT when parameterized by both the treewidth of the graph and the number of requests. Gottlob and Lee proved a stronger result in [74]: MULTICUT is FPT when parameterized by the treewidth of the input structure, *i.e.* the input graph whose edge set is augmented by the set of requests.

The graph minor theorem of Roberston and Seymour implies that MULTICUT is non-uniformly FPT when parameterized by the solution size and the number of requests. Marx proved that MULTICUT is (uniformly) FPT for this latter parameterization [96]. A faster algorithm running in time $O^*(8 \cdot l)^k$ was given by Guillemot [77]. Marx *et al.* [98] obtained FPT results for more general types of constrained MULTICUT problems through treewidth reduction results. However their treewidth reduction techniques do not yield FP-Tractability of MULTICUT when parameterized by the solution size only. Finally, Marx and Razgon obtained a factor 2 Fixed-Parameter-Approximation for MULTICUT parameterized by the solution size in [99].

1.4.2 Expressing other Problems in the Multicut Framework

The importance of MULTICUT does not only lie in its theoretical relevance and its practical network applications. Many graph problems can be expressed into the MULTICUT framework. The following is a non-exhaustive list of a few such problems.

Cluster Edition With Free Edges

MULTICUT is FPT-equivalent to CLUSTER EDITION WITH FREE EDGES, which consists in deciding whether a graph can be edited (through edge deletions and additions) into a cluster graph (a vertex-disjoint union of cliques), with cost at most k, where the edition of free edges has cost 0, while the edition of other edges has cost 1. This problem is called FUZZY CLUSTER EDITING in [13]. Its equivalence with MULTICUT has been mentioned several times in the literature, we will give a short proof in Subsection 7.3.1.

2-CNF≡ Deletion

2-CNF≡ Deletion is a variant of SAT where clauses are *equivalence-clauses*, expressing the equivalence of two literals. 2-CNF≡ Deletion consists in deleting as few clauses as possible to make a set of equivalence-clauses satisfiable. It can be reduced to MULTICUT where requests form a perfect matching as follows [86]. To an instance $\mathcal{F}$ of 2-CNF≡ deletion, associate a graph $G(\mathcal{F})$ whose vertices are literals, whose edges are described by the clauses and where there is a request between a variable and its negation. A formula is satisfiable if its associated graph has no connected component containing a variable and its negation. Deleting k clauses in $\mathcal{F}$ to make it satisfiable is equivalent to finding a k-multicut of $G(\mathcal{F})$. This simple transformation is approximation- and FPT-preserving.

2-CNF≡ deletion contains problems such as Minimum Edge-Deletion Graph Bipartisation. To each vertex, associate a variable which states to which of the two parts it will belong. Each edge of the graph can be expressed with an equivalence-clause stating that its endpoints must lie in different parts of the bipartition.

Other reductions to MULTICUT

A vertex cover is a set of vertices which hits all edges of a graph. VERTEX COVER can be expressed as MULTICUT restricted to stars. Indeed, given a instance $G = (V, E)$ of VERTEX COVER, let S be a star whose leaf set if V, and where (u, v) is a request if $(u, v) \in E$. A vertex cover of the graph G corresponds exactly to an edge-multicut of S.

Sparsest Multicut, which consists in finding a cut minimizing the ratio between the number of removed edges and the number of requests cut, can also be reduced to MULTICUT in an approximation-preserving way [117].

1.4.3 Multicut In Trees

In the (unweighted) MULTICUT IN TREES problem, we consider a tree T together with a set $\mathcal{R}$ of pairs of distinct nodes of T, called *requests*. Hence, a request can also be seen as a prescribed path between these two nodes. We will often identify the request and its path. A *multicut* of $(T, \mathcal{R})$ is a set S of edges of T which intersects every request in $\mathcal{R}$, *i.e.* every path corresponding to a request contains an edge of S.

A multicut corresponds to a covering by cliques of the vertices of the intersection graph of requests in $\mathcal{R}$. Note that this intersection graph is chordal. Equivalently, consider the bipartite graph B with vertex bipartition $(E, \mathcal{R})$ where E is the set of edges of T, and where a request in $\mathcal{R}$ is adjacent to an edge in E if and only if the edge belongs to the request. In this setting, a multicut of size k corresponds in B to a cover of the part $\mathcal{R}$ with k stars rooted in E.

MULTICUT IN TREES:
Input: A tree $T = (V, E)$, a set of requests $\mathcal{R}$, an integer k.
Parameter: k.
Output: TRUE if there is a multicut of size at most k, otherwise FALSE.

This problem is known to be FPT, see [78] or [80] for a branching algorithm and an exponential kernel. The existence of a polynomial kernel was asked in [11].

Our Results.
As of 2008, MULTICUT was only known to be FPT when restricted to trees. With Nicolas Bousquet, Stéphan Thomassé and Anders Yeo, we showed that MULTICUT IN TREES has actually a polynomial kernel. To obtain this kernel we exhibit reduction rules which can be applied until the instance size becomes polynomial in the solution size k. These rules essentially rely on the local structure of the instance. Some of these rules, as well as the analysis on the instance size bound, are very involved. This $O(k^6)$ kernel is presented in Chapter 2, based on [22].

It was shown only later, in 2010, that MULTICUT is FPT on general graphs (see Chapter 7). This means that the next natural question is whether MULTICUT restricted to larger classes than trees still has a polynomial kernel. No further progress has been made in this direction so far. Exhibiting a polynomial kernel for MULTICUT IN TREES was already very challenging, and FPT algorithms for the general MULTICUT problem are very involved, and require a heavy dose of branching. Hence one can be pessimistic about the existence of a polynomial kernel for the general MULTICUT problem. But no negative result has been found so far. If MULTICUT turned out not to have a polynomial kernel, finding where the line between polynomial kernel and no polynomial kernel stands for MULTICUT would be a very interesting (and probably very difficult) question.

Also, a more general presentation of MULTICUT IN TREES is to assign weights to edges, and ask for a multicut of minimal weight. Our technique to establish a polynomial kernel for the unweighted version does not seem to generalise to the weighted case.

1.5 Maximum Leaf Spanning Tree Problems

The MAXIMUM LEAF SPANNING TREE (in short MAXLEAF) problem consists in finding a spanning tree with the maximum number of leaves in an undirected graph. This is equivalent to finding a Connected Dominating Set of minimum size. Indeed, the set of internal nodes in a spanning tree corresponds to a Connected Dominating Set. This problem is **NP**-complete and $MaxSNP$-hard [70], which implies that it admits no PTAS unless $P = NP$.

Finding undirected trees with many leaves has many applications in the area of communication networks, see [47] or [124] for instance. An extensive literature is devoted to the paradigm of using a small connected dominating set as a backbone for a communication network.

MAXLEAF has been well-studied from a graph-theoretical point of view. Graphs with at least $n + \binom{k}{2}$ edges have a tree with k leaves [48]. On graphs with maximum degree d, Linial conjectured that there always exists a tree with at least $\frac{d-2}{d+1}n + c_d$ leaves, where c_d is a constant depending on d. This conjecture was proved up to $d = 5$ [120, 75, 87], but fails when $d \to \infty$ as observed by Alon due to results in [2]. Kleitman and West prove a guarantee of $(1 - \frac{5}{2}\frac{\ln(d)}{d})n$ leaves in [87], which is optimal up to the factor 5/2 by Alon's lower bound.

Many algorithms have also been designed for MAXLEAF. There is a factor 2 approximation algorithm for the MAXLEAF problem [118], and a $3.75k$ kernel [56]. An $O^*(1.94^n)$ exact algorithm was designed in [64]. In cubic graphs, a factor 3/2 approximation has been found [19].

The MAXLEAF problem has been studied from the parameterized complexity perspective as well and several authors [17, 57, 60] have designed fixed parameter tractable (FPT) algorithms for solving the parameterized version of MAXLEAF: given a graph G and an integral parameter k, decide whether G has a spanning tree with at least k leaves. This is equivalent to the existence of a CDS of size at most $n - k$. Hence such parameterized algorithms can be used to find an optimal CDS when there exist only large CDS. The most natural question would be: is there an FPT algorithm for finding a CDS of size at most k? Indeed an FPT algorithm for this problem would have a great practical impact on network problems. But the answer is negative: the existence of a Connected Dominating Set of size at most k is a W[2]-hard problem.

In this thesis, we are interested in the directed version of MAXLEAF, whose study has begun more recently, but has nonetheless fruited many results.

An *out-tree* is a tree where each edge has been oriented from the root towards the leaves. More formally, an *out-tree* is a connected digraph with a single vertex of in-degree 0, the root, where every other vertex has in-degree exactly 1. An *out-branching* of a digraph D is a spanning out-tree in D, *i.e.* a connected subgraph with no cycle where every vertex has in-degree at most one.

ROOTED MAXIMUM LEAF OUT-BRANCHING:

Input: A digraph D, an integer k, a vertex r of D.
Parameter: k.
Output: TRUE if there is an out-branching of D rooted at r with at least k leaves, otherwise FALSE.

This problem is equivalent to finding a Connected Dominating Set of size at most $|V(D)|-k$, connected meaning in this setting that every vertex is reachable by a directed path from r.

ROOTED MAXIMUM LEAF OUT-BRANCHING remains **NP**-complete even restricted to acyclic digraphs [4].

The best approximation algorithm known for MAXIMUM LEAF OUT-BRANCHING found a solution within the square root of the optimum. algorithm [50]. From the Parameterized Complexity viewpoint, Alon *et al.* showed that MAXIMUM LEAF OUT-BRANCHING restricted to a wide class of digraphs containing all strongly connected digraphs is FPT [3], and Bonsma and Dorn extended this result to all digraphs and gave a faster parameterized algorithm [18]. Recently, Kneis, Langer and Rossmanith [88] obtained an $O^*(4^k)$ algorithm for MAXIMUM LEAF OUT-BRANCHING, which is also an improvement for the undirected case over the numerous FPT algorithms designed for MAXIMUM LEAF SPANNING TREE (Chen and Liu have a similar algorithm in [27]). Fernau *et al.* [61] proved that ROOTED MAXIMUM LEAF OUT-BRANCHING has a polynomial kernel, exhibiting a cubic kernel. They also showed that the unrooted version of this problem admits no polynomial kernel, unless the polynomial hierarchy collapses to the third level, using a breakthrough lower bound result by Bodlaender *et al.* [12].

In this thesis, we provide a quadratic kernel for ROOTED MAXIMUM LEAF OUT-BRANCHING and a linear kernel for the acyclic subcase of ROOTED MAXIMUM LEAF OUT-BRANCHING in Chapter 3. We also exhibit a constant factor approximation algorithm for ROOTED MAXIMUM LEAF OUT-BRANCHING. These results are based on reduction rules and combinatorial bounds, using a notion called $s-t$ numberings. In Chapter 8 we design an $O^*(3.72^k)$ algorithm for ROOTED MAXIMUM LEAF OUT-BRANCHING, which in particular provides the first non-trivial exponential algorithm for finding an out-tree with maximal number of leaves in a digraph.

Part I: Kernels

Kernels lie at the intersection between algorithms and combinatorics. Strictly speaking, a kernel is an algorithm, but in practice, finding a kernel has more to do with understanding combinatorial and structural properties. To design a kernel, one effectively creates reduction rules, which are mostly of two types:

– Replace a substructure with another, and prove that it does not impact the existence of a solution (structural property).
– Show that if an invariant is smaller/greater than a given value, then there must/cannot exist a solution (combinatorial property).

The algorithmic part is quite often trivial (essentially proving that the reduction rules can be applied in polynomial time).

In Chapter 2, we provide a polynomial kernel for the MULTICUT IN TREES problem, answering an open question raised by Fellows in [11] and by Guo and Niedemeier in [80]. This chapter is based on [22], a joint work with Nicolas Bousquet, Stéphan Thomassé and Anders Yeo.

This work was my first contact with the area of parameterized complexity. MULTICUT IN TREES is simpler to approach than the general MULTICUT problem, as a request is realized by only one path. Hence finding a multicut in a tree simply consists in finding a hitting set for a prescribed set of paths in a tree. Our later attempts at transposing this idea to the general MULTICUT problem did not prove very fruitful. We will state a few conjectures on this topic in Subsection 7.8.3.

This simple setting gave us something to chew on during Nicolas' undergraduate internship, and we quickly came up with Rules (0) to (3), which had already been found in [80]. Rule (4) is actually quite simple as well, but this was not enough to obtain a polynomial kernel. We had to go through the pains of designing Rule (5), the Wingspan rule, which is quite technical and less natural than the others. In a sense, this rule expresses the fact that the requests cannot overlap too much, which is the key to bound the size of an instance. Indeed, requests cannot be too much spread out by Rule (1). Finding a more natural reduction rule to replace the wingspan rule(s) is an interesting open question.

In Chapter 3, we provide kernels for the ROOTED MAXIMUM LEAF OUT-BRANCHING problem. This chapter is mostly based on [42], a joint work with Stéphan Thomassé. The key starting point was reducing cut-vertices (Rule (1) in Section 3.2) to focus on 2-connected digraphs. Stéphan coming across the concept of $s-t$ numberings allowed us to better formalize our intuitions, leading to Section 3.1 and its clean combinatorial bounds. We will show that in a 2-connected digraph, one can always find a tree with as many leaves as:

– a constant fraction of the number of vertices of in-degree at least three (Theorem 37).
– a constant fraction of the number of vertices incident to a simple in-arc (Theorem 38).

The algorithmic consequences, a quadratic kernel in Section 3.3 and a constant factor approximation in Section 3.4, stemmed from these combinatorial results.

Section 3.5 gives a linear kernel in the acyclic case, based on [39], a joint work with Gregory Gutin, EunJung Kim and Anders Yeo. I am grateful to Gregory, EunJung and Anders for inviting me and for introducing me to MAXLEAF problems during the summer of 2008. This and the work on MULTICUT IN TREES convinced me to focus my thesis on parameterized complexity.

CHAPTER

2 A Polynomial Kernel for Multicut In Trees

We show that MULTICUT IN TREES has a polynomial $O(k^6)$ kernel. We use reduction rules from [78] and [80] along with other more powerful rules. In Section 2.1, we first illustrate our techniques when the tree T is a caterpillar. In Section 2.2 we extend the proof to general trees.

2.1 A Polynomial Kernel for Caterpillars

Recall that the input of MULTICUT IN TREES is an instance $(T, \mathcal{R}, k)$, where T is a tree, $\mathcal{R}$ is a set of requests and k is an integer. A node of T which is not a leaf is an *internal node*. The *internal tree* of T is the tree restricted to its internal nodes. We say that T is a *caterpillar* if its internal tree is a path. We consider the restriction of the MULTICUT IN TREES problem to caterpillars, as it contains the core of our proof in the general case. In particular, our key reduction rule, the wingspan rule, is easier to state and apply in the caterpillar setting.

Let us give some general definitions which will apply both for the caterpillar case and for the general case.

The *request graph* $R(T, \mathcal{R})$ of the instance $(T, \mathcal{R}, k)$ has vertex set $V(T)$, and edge set $\mathcal{R}$. We say that two nodes x and y are *R-neighbours* if xy is a request in $\mathcal{R}$, equivalently if x and y are adjacent in $R(T, \mathcal{R})$. A leaf x and an internal node y are *quasi-R-neighbours* if xy is a request, or if there exists a request xz, where z is a leaf rooted at y. An internal node with no leaf attached to it is an *inner node*. If x is a leaf, we denote by $e(x)$ the unique edge incident to x, and we denote by $f(x)$ the unique neighbour of x. A *group* of leaves is the set of leaves connected to the same internal node. A *group request* is a request xy where x and y belong to the same group. A leaf which is an endpoint of a group request is a *bad leaf*. A *leaf to leaf request* is a request between two leaves. An *internal request* is a request between two internal nodes. A request between an internal node and a leaf is a *mixed-request.* Two requests are *disjoint* if their edge sets are disjoint. Two requests x_1y_1 and x_2y_2 are *endpoint-disjoint* if x_1, y_1, x_2, y_2 are pairwise different.

The *internal path* of a request is the intersection between the path of the request and the internal tree. The *common factor* of two requests is the intersection of their paths. A request R_1 *dominates* a request R_2 if the internal path of R_1 contains the internal path of R_2.

Contracting an edge e in $(T, \mathcal{R}, k)$ means contracting e in T, and transforming each request of the form $(e_1, \dots, e_t, e, e_{t+1}, \dots, e_l)$ in $\mathcal{R}$ into $(e_1, \dots, e_t, e_{t+1}, \dots, e_l)$. *Deleting* an edge e means contracting e in T and removing every request containing e from $\mathcal{R}$ (so that the instance remains a tree).

Two requests of length at least 2 from a given leaf x *have the same direction* if the second edge of their path starting at x is the same. Two requests from an internal node x *have the same direction* if the first edge of their paths (starting at x) is the same. All the requests from x *have the same direction* if they pairwise have the same direction.

In the following, our instance T is assumed to be a caterpillar. We call the two extremities of the internal path the *left end* and the *right end* of T. The path between a node x and the right (resp. left) end will be called *right* and *left* relatively to x.

Let T' be the internal tree of the caterpillar T. The following five sets partition T:

- The set I_1 of leaves of T'.
- The set I_2 of degree two nodes of T'.
- The set L_1 of leaves rooted at I_1.
- The set L_2' of bad leaves rooted at I_2.
- The set L_2 of the other leaves rooted at I_2.

The *wingspan* W of a leaf x is the path between the closest quasi-R-neighbour on the right of x and the closest quasi-R-neighbour on the left of x (if no such neighbour exists, we take $f(x)$ by convention). The *size* of a wingspan is the number of L_2-leaves pending from it. The *subcaterpillar* $SC(W)$ of the wingspan W consists in the union of W and the leaves rooted at W. The wingspan W *dominates* a request yz if both y and z belong to $SC(W)$.

The usual way of exhibiting a kernel is to define a set of *reduction rules.* These rules should be *safe,* meaning that after applying a rule, the truth value of the problem on the instance does not change. Moreover the repeated application of the rules should take polynomial time. Finally, after iterating these rules on an instance, we want the *reduced instance* to be of size polynomial in k.

The reduction rules

We apply the following reduction rules to an instance:

(0) Unit Request: if a request R has length one, *i.e.* $R = e$ for some edge e of T, then we delete e and decrease k by one.

(1) Disjoint Requests: if there are $k+1$ disjoint requests in $\mathcal{R}$, then we return a trivially false instance.

(2) Unique Direction: if all the requests starting at a leaf x have the same direction, then contract $e(x)$. If all the requests starting at an inner node (*i.e.* an internal node with no leaf) x have the same direction, then contract the edge e adjacent to x which does not belong to any request starting at x.
(3) Inclusion: if a request R is included in another request R', then delete R' from the set of requests.
(4) Common Factor: if $k+2$ distinct requests $R, R_1, \dots, R_{k+1}$ are such that for every $i \neq j$, the common factor of R_i and R_j is a subset of R, then delete R from the set of requests.
(5) Dominating Wingspan: if x is an L_2-leaf with a wingspan dominating at least $k+1$ endpoint-disjoint requests, then contract $e(x)$.

Each iteration of the reduction consists in applying the first applicable rule, in the above order.

Lemma 6. *Rules Unit Request, Disjoint Requests, Unique Direction, Inclusion, Common Factor and Dominating Wingspan are correct.*

Proof. (1) Rules Unit Request and Disjoint Requests are obvious.
(2) For Rule Unique Direction, assume first that all the requests from a leaf x have the same direction, and that a multicut contains $e(x)$. Let e' be the second common edge of all these paths. As e' cuts all the requests cut by $e(x)$, if $e(x)$ is in a solution S then $S \backslash \{e(x)\} \cup \{e'\}$ is also a solution. So we can contract $e(x)$.
Now, assume that all the requests from an inner node x go to the right. If a solution S contains the edge e adjacent to x on the left then $S \backslash \{e\} \cup \{e'\}$, where e' is the edge incident to x on the right, is a solution since a request going through e also goes through e'.
(3) An edge cutting R also cuts all the paths containing R which proves Rule Inclusion.
(4) If there is a multicut of k edges, then one of these edges must intersect two requests among $R_1, \dots, R_{k+1}$. This edge lies in the intersection of two of these paths, hence in R, so the request R is cut in any multicut of $\mathcal{R} \setminus \{R\}$.
(5) Let x be an L_2-leaf with a wingspan W dominating $k+1$ endpoint-disjoint requests. If a multicut of size k exists, it contains an edge e which cuts two of these requests. As the requests are endpoint-disjoint, their intersection is included in the internal tree, hence in W. Assume, for example, that e is on the left of the leaf x. Then all the requests from x which go to the left go through e, and moreover x has no group request by definition. Thus, if a solution exists, there is a solution without $e(x)$, since $e(x)$ can be replaced by the edge e' which is on the right of $f(x)$. □

Lemma 7. *Deciding whether a rule applies and applying it takes polynomial time.*

Proof. Denote by n the number of nodes in T and by r the number of requests.

- The application of Rule Unit Request takes time $O(r)$. The maximum edge-disjoint paths problem in trees is polynomial, see [72], thus Rule Disjoint Requests is polynomial. Rule Unique Direction can be applied in time $O(rn^2)$, and Rule Inclusion can be applied in time $O(r^2n)$.
- For the running time of Rule Common Factor, consider a request R. Informally, we are looking for a large enough set of requests which intersect R, possibly leaving it at one or two places, such that the edges through which they leave are all distinct. More formally, let Z be the set of edges not in R but sharing a vertex with some edge in R. Let Y be the set of edges e in Z such that there exists a request starting at a node in R and going through e. We can assume without any loss that one request per such edge e is chosen. Let G be the graph whose vertex set is $Z - Y$ and whose edges are the pairs (e, e') such that there exists a request going through both e and e'. There exist $k+1$ paths as in Rule Common Factor if and only if G has a matching of size at least $k+1-|Y|$. As the matching problem is polynomial, the application of Rule Common Factor takes polynomial time.
- Let W be a wingspan, and consider the graph $R(T, \mathcal{R})_{|SC(W)}$ which is restriction of the request graph to $SC(W)$. In other words, the vertices of $R(T, \mathcal{R})_{|SC(W)}$ are the nodes in $SC(W)$ and two vertices are adjacent if there is a request between them. There exist $k+1$ endpoint-disjoint requests dominated by W if and only if $R(T, \mathcal{R})_{|SC(W)}$ has a matching of size $k+1$, thus Rule Dominating Wingspan is polynomial. □

Lemma 8. *The reduction process has a polynomial number of iterations.*

Proof. Each rule decreases the sum of the lengths of the requests, which is initially less than the number of requests times the number of nodes. □

In the following we consider an instance in which none of these rules can be applied, and prove that such a reduced instance has polynomial size in k.

Informally speaking, Rule Inclusion rules out many possible patterns for requests between internal nodes. Two such internal requests are either disjoint (and there cannot be more than k disjoint requests by Rule Disjoint Requests) or "overlapping". The complexity of the problem essentially lies in the request with a leaf endpoint. Rule Inclusion provides very little help to reduce this type of requests, which are not contained one in another provided their endpoints are disjoint. Rule Common Factor and Rule Dominating Wingspan are designed to deal with requests with a leaf endpoint in a slightly more complicated fashion than the trivial Inclusion rule (and additionally, to treat "overlapping" patterns which can occur with internal requests as well).

Let us introduce two graph theoretic lemmas which are used in our proof.

Lemma 9. *Let G be an undirected graph with m edges and of maximum degree $\Delta \geq 1$. Then G has a matching of size $\lceil \frac{m}{2\Delta - 1} \rceil$.*

Proof. Such a matching can be obtained by a greedy algorithm, as taking an edge uv in the matching forbids the edges adjacent to u and those adjacent to v (there are at most $2\Delta-1$ such edges, including uv). □

Lemma 10. *Let* H *be an undirected graph on* n *vertices and of maximum degree* $\Delta \geq 1$. *Then* H *has an independent set of size* $\lceil \frac{n}{\Delta+1} \rceil$.

Proof. Such an independent set can be obtained by a greedy algorithm, as taking a vertex u in the independent set forbids the vertices adjacent to u. □

Theorem 11. *The* MULTICUT IN CATERPILLARS *problem has a kernel of size* $O(k^5)$.

The rest of this section is dedicated to prove that an instance reduced under the above reduction rules has size $O(k^5)$. Let A be such an instance.

Observation 12. *A node has at most* $k+1$ *R-neighbours in each direction in* A.

Proof. If a node x has $k+2$ R-neighbours in, say, the right direction, then Rule Common Factor applies, where R is a longest right request of x. □

Claim 13. *There are at most* $2k(2k+1)$ *bad leaves in* A.

Proof. A bad leaf is connected to at most $k+1$ leaves of the same group, by Rule Common Factor. Let G be the undirected graph whose vertices are the bad leaves of T and where there is an edge between two leaves if there is a group request between them. The minimal degree in G is at least 1, and the maximal degree is at most $k+1$. If there are at least $2k(2k+1)+1$ bad leaves then there are at least $k(2k+1)+1$ edges in G. Thus by Lemma 9 there exist a matching of size $k+1$, which implies the existence of $k+1$ endpoint-disjoint (thus disjoint) group requests. In this case, Rule Disjoint Requests would apply. □

Claim 14. *A wingspan has size at most* $2k(4k+3)$ *in* A.

Proof. Let W be a wingspan of maximal size, *i.e.* such that $SC(W)$ contains a maximal number of L_2-leaves. As Rule Dominating Wingspan does not apply, W does not dominate $k+1$ endpoint-disjoint requests. Let W' be the set of leaves pending from W. Consider the restriction $G = R(T,\mathcal{R})_{|SC(W)}$ of the request graph to $SC(W)$. Finding $k+1$ endpoint-disjoint requests dominated by W is equivalent to finding a matching of size $k+1$ in G. The degree of a vertex u in G is at most $2k+2$ because there are at most $k+1$ requests in each direction for u in T (by Observation 12). Moreover, if u corresponds to a node of W', the degree of u is at least one. Indeed, since the wingspan W has maximal size, each L_2-leaf pending from W must have a request dominated by W.

If there are $2k(4k+3)+1$ L_2-leaves in W', then G contains at least $k(4k+3)+1$ edges, and so G has a matching of size $k+1$ by Lemma 9, which in turn means the existence of $k+1$ endpoint-disjoint requests. □

Claim 15. *There are* $O(k^3)$ L_2*-leaves in* A.

Proof. Let x be an L_2-leaf pending from a wingspan W. By the previous claim, there are less than $2k(4k+3)$ leaves pending from W. At most $2k(4k+3)+1$ L_2-leaves not pending from W have wingspans intersecting W from each direction, as the furthest L_2-leaf (on the right) whose wingspan intersects W has a wingspan whose subcaterpillar contains all other leaves whose wingspans intersect W from the right. Let H be the auxiliary graph with vertex set L_2, where two L_2-leaves are adjacent if their wingspans intersect. H has maximum degree less than $6k(4k+3)+2$ by the above discussion. By Lemma 10, if T has at least $(6k(4k+3)+2)k+1$ vertices, then H has a stable set of size $k+1$. Thus T would have $k+1$ disjoint wingspans, and thus $k+1$ disjoint requests, a contradiction.

Claim 16. *There are* $O(k^5)$ I_2*-nodes in* A.

Proof. By Claim 15, there are $O(k^3)$ I_2-nodes with leaves. Let us bound the number of inner nodes, *i.e.* of I_2-nodes without a leaf. Let I' be the set of inner nodes in T. Consider the graph G which is the restriction of the request graph to I'.

Because of Rule Inclusion, each inner node has degree at most two in G (one in each direction). Thus G is a disjoint union of paths, called *request paths.* The length of a request path is at most k by Rule Disjoint Requests. A node with degree 1 in G is an *extremal inner node.*

Each extremal inner node must be an $\mathcal{R}$-neighbour in T of a leaf or of an internal node with a leaf (otherwise it would be reduced by Rule Unique Direction). Denote by X the set of leaves and I_2-nodes with leaves. Each node in X has $O(k)$ $\mathcal{R}$-neighbours among the inner nodes, and $|X| = O(k^3)$, so there are $O(k^4)$ inner nodes with a neighbour in X (in particular, at most $O(k^4)$ extremal inner nodes). Each extremal inner node belongs to a unique request path of size at most k. Moreover each inner node with no neighbour in X must belong to a request path. So there are $O(k^5)$ inner nodes in T. □

There are $O(k^3)$ leaves and $O(k^5)$ internal nodes in a reduced instance. Thus our reduction rules provide a kernel of size $O(k^5)$ for MULTICUT IN CATERPILLARS, which concludes the proof of Theorem 11. □

2.2 General Trees

Should no confusion arise, we retain the terminology of the previous section.

Let $(T, \mathcal{R}, k)$ be an instance. Let T' be the tree obtained from T by deleting the leaves. We partition the set of nodes of T into the following seven sets:

- The set I_1 of leaves in T'.
- The set I_2 of degree 2 nodes in T'.
- The set I_3 of the other nodes in T'.
- The set L_1 of leaves rooted at I_1.
- The set L_2 of leaves rooted at I_2, endpoint of no group request.
- The set L'_2 of leaves rooted at I_2, endpoint of at least one group request.

– The set L_3 of leaves rooted at I_3.

We also denote by L the set of leaves of T.

We need a few technical definitions. A *caterpillar* of T is a maximal connected component of $T - I_3 - L_3$. The *backbone* of a caterpillar is the set of internal nodes of T in this caterpillar. A caterpillar C is *non-trivial* if the set of internal nodes in C seen as a caterpillar has size at least two. The *extremities* of a non-trivial caterpillar C are the two nodes of C which are I_2 or I_1-nodes of T and become I_1-nodes in C. A *minimal request* of a node x is a request having x as an endpoint and whose internal path is minimal for inclusion among all internal paths of requests with x as an endpoint. If several requests have the same internal paths, we arbitrarily distinguish one as minimal and will not consider the others as minimal. If xy is a minimal request of x then y is called a *closest R-neighbour* of x.

Let x and y be nodes in T. If z lies on the path between x and y, or is a leaf rooted at a node lying on the path between x and y, we say that z lies *toward* y from x (and we do not write "from x" should no confusion arise).

Assume that x is an L_2-leaf of a caterpillar C (that is, an L_2-leaf of T which belongs to C). Let $Gr(x)$ be the group of leaves pending from $f(x)$. Let $A(x)$ and $B(x)$ be the two connected components of $T - \{f(x)\} - Gr(x)$. Let $a(x)$ (resp. $b(x)$) be the extremity of C in $A(x)$ (resp. $B(x)$). If $A(x)$ (resp. $B(x)$) contains no extremity of C, that is if $f(x)$ is an extremity of C, then we define $a(x) = f(x)$ (resp. $b(x) = f(x)$). A *wingspan* W of x is the path between two closest R-neighbours of x lying respectively in $A(x)$ and $B(x)$. Observe that x can have several wingspans. The *subcaterpillar* of a wingspan W consists in W and the leaves rooted at W.

An L_2-leaf x *covers* a caterpillar C if either $x \notin C$ and there is a request starting at x and going through the whole backbone of C, or if $x \in C$ and a wingspan of x contains the whole backbone of C. Figure 2.1 provides an example of leaves covering a caterpillar.

We apply the following reduction rules to an instance: Rules (0), (1), (2), (3), and (4) are stated in the previous section. Rule Dominating Wingspan is split for convenience into two rules, one similar to the caterpillar case and the other more general, as follows:

(5a) Bidimensional Dominating Wingspan: if x is an L_2-leaf of a caterpillar C with a wingspan W such that $W \cap C$ dominates at least $k+1$ endpoint-disjoint requests, then we contract $e(x)$.

(5b) Generalised Dominating Wingspan: assume that x is an L_2-leaf of the caterpillar C, and that x covers C. Assume that for every closest quasi-R-neighbour z of x in $A(x)$, there exist $k+1$ endpoint-disjoint requests between a node lying toward $b(x)$ from $f(x)$ and a node toward z from $a(x)$. Then we contract $e(x)$. See Figure 2.2.

As an observation, Rule generalised Dominating Wingspan symmetrically applies with the roles of left and right (*i.e.* with the roles of $A(x)/a(x)$ and $B(x)/b(x)$) reversed. This was not formally stated above for clarity. Each iteration of the reduction consists in applying the first applicable rule, in the above order.

Lemma 17. *Rules (5a) and (5b) are correct.*

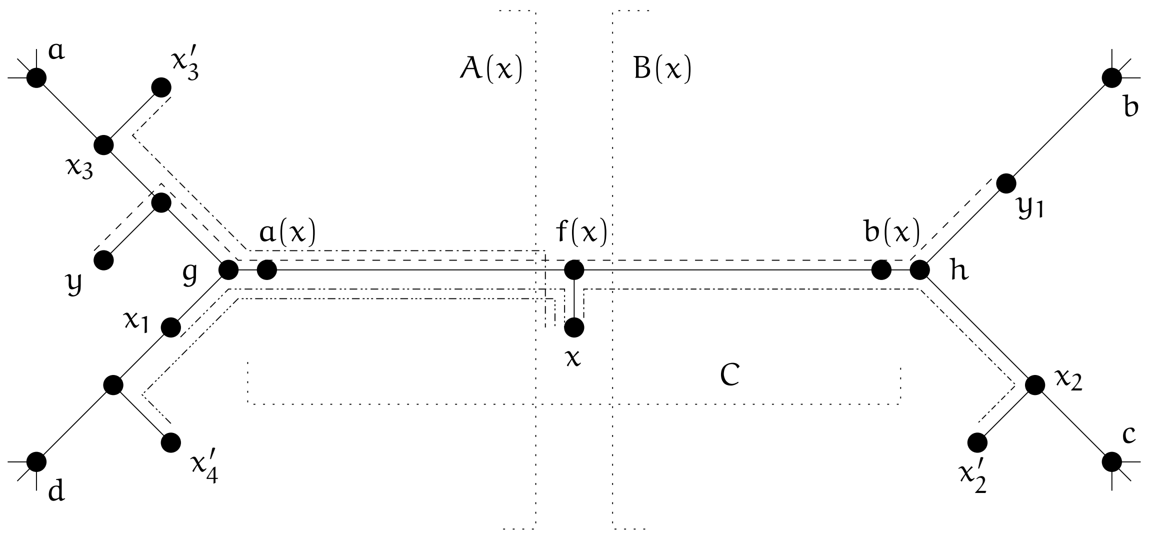


Figure 2.1: The nodes a, b, c, d are I_1-nodes and g, h are I_3 nodes. C is a caterpillar, and x is an L_2-leaf of C, with requests xx_1, xx_2', xx_3' and xx_4'. The closest quasi-R-neighbours of x are x_1, x_2 and x_3. The leaf x has two wingspans: the path between x_1 and x_2, and the path between x_3 and x_2. Either of these wingspans contains the whole backbone of C, so x covers C. Similarly, yy_1 is a request, so the L_2-leaf y, which does not belong to the caterpillar C, covers C as well.

Proof. Correction of Rule Bidimensional Dominating Wingspan follows from the correction proof of Rule Dominating Wingspan in the previous section.

Assume that Rule generalised Dominating Wingspan can be applied to x. Let $z_1, \dots, z_l$ be the closest R-neighbours of x in $A(x)$. For every $i \in \{1, \dots, l\}$, because of the $k+1$ endpoint-disjoint requests mentioned in the rule, any k-multicut contains an edge in the path between z_i and $b(x)$. Assume that a k-multicut S contains an edge e'' between x and $b(x)$. Let e' be the edge adjacent to $e(x)$ in the path between x and $a(x)$. If S contains $e(x)$, then $S - \{e(x)\} \cup \{e'\}$ is also a k-multicut. Indeed, any request x, u with $u \in A(x)$ is cut by e', and any request xv with $v \in B(x)$ is cut by e''. Assume now that a k-multicut S contains no edge between x and $b(x)$. Then for every $i \in \{1, \dots, l\}$, S must contain an edge e_i on the path between z_i and $f(x)$. Let e' be the edge adjacent to $e(x)$ in the path between x and $b(x)$. If S contains $e(x)$, then $S - \{e(x)\} \cup \{e'\}$ is a k-multicut. Indeed, any request xu with $u \in A(x)$ is cut by an edge e_i, and any request xv with $v \in B(x)$ is cut by e'. □

Proposition 18. *The repeated application of these rules on the instance until none can be applied takes polynomial time.*

Proof. The proof of the first five cases was made for general trees in the previous section. The polynomiality of Rule Bidimensional Dominating Wingspan follows from the proof

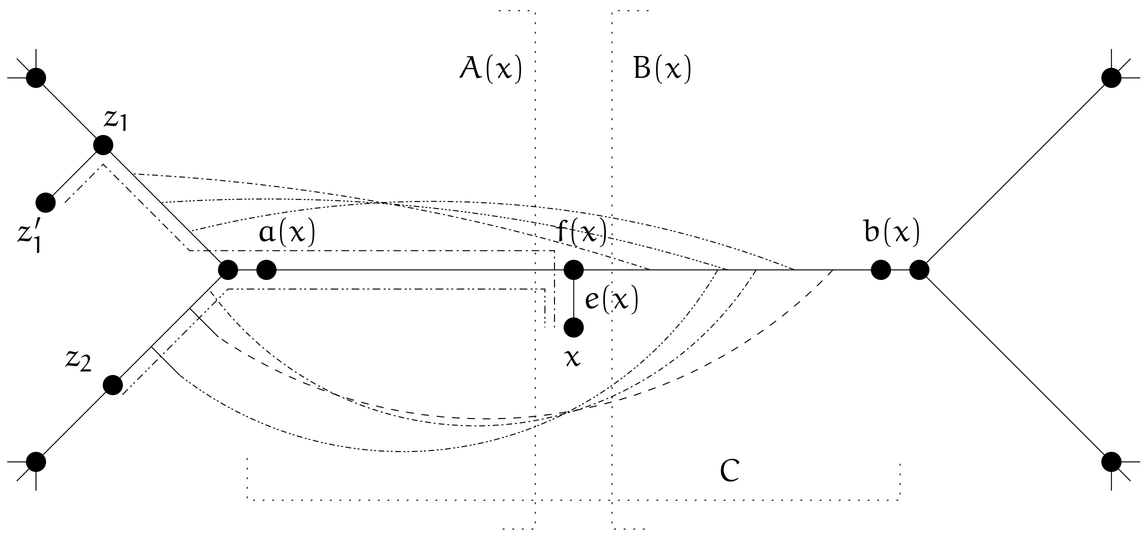


Figure 2.2: Rule generalised Dominating Wingspan, with $k = 2$, applies to the L_2-leaf x. Indeed, x has two closest quasi-R-neighbours in $A(x)$, namely z_1 and z_2. There are $k+1 = 3$ requests between the paths $[z_1, a(x)]$ and the path $[f(x), b(x)]$, and likewise with z_2.

of Rule Dominating Wingspan's polynomiality in the previous section. Deciding whether there exist $k+1$ endpoint-disjoint requests between prescribed areas can still be expressed as a matching problem as in Rule Dominating Wingspan's proof, so the application of Rule generalised Dominating Wingspan also takes polynomial time. □

Theorem 19. *The number of nodes in a reduced instance is* $O(k^6)$.

The rest of this section is devoted to the proof of Theorem 19.

Claim 20. $|I_1| = O(k)$

Proof. There are at most k groups of leaves with a group request, by the $k+1$ disjoint requests rule. Every group of L_1-leaves has a group request, otherwise a leaf of this group would be deleted by Rule Unique Direction. Every I_1-node has at least one L_1-leaf pending from it, thus $|I_1| \le k$. □

Claim 21. $|I_3| = O(k)$

Proof. In a tree, there are at most as many nodes of degree at least 3 as the number of leaves, so $|I_3| \le |I_1| \le k$. □

Claim 22. $|L_1| = O(k^2)$ *and* $|L'_2| = O(k^2)$

Proof. Each leaf in L_1 is a bad leaf by Rule Unique Direction, and each leaf in L'_2 is bad by definition. As in Claim 13 there are at most $2(k+1)(2k+1) - 1$ bad leaves in T. Thus $|L_1 \cup L'_2| = O(k^2)$ □

We now show that:
– $|L_3| = O(k^4)$
– $|L_2| = O(k^4)$
– $|I_2| = O(k^6)$

Claim 23. *The number of requests from a node* x *to a group of leaves is at most* $k+1$.

Proof. Otherwise Rule Common Factor would apply to these requests. □

Claim 24. *The number of requests from a node* x *to all the* L_2*-leaves in a given caterpillar* C *is at most* $2k+2$ *if* $x \in C$ *and* $k+1$ *if* $x \notin C$.

Proof. Otherwise there would be at least $k+2$ requests sharing the same direction between x and leaves in this caterpillar, and Rule Common Factor would apply to these requests. □

Claim 25. *There are at most* $2k(k+2)$ *requests between two groups of leaves.*

Proof. Let G be the bipartite graph whose vertices are the leaves of the two groups Y and Z, and where a leaf in Y and a leaf in Z are adjacent if there is a request between them. The maximum degree in G is at most $k+1$ by Claim 23, thus if there are $2k(k+2)+1$ requests between Y and Z, then by Lemma 9 there would be a matching of size $k+2$ in G. Thus there would be $k+2$ endpoint disjoint requests between Y and Z, and Rule Common Factor would apply. □

Claim 26. *The number of requests between a group of leaves* E *and the nodes in a given caterpillar* C *is at most* $4k(k+2)$.

Proof. Assume by contradiction that there are at least $4k(k+2)+1$ such requests. Let f be the node in which the leaves of E are rooted. If f belongs to C, then $C-f$ has two connected components, and we only consider the component C' to which there is the most requests from G. If f does not belong to C, then we let $C' = C$. There are at least $2k(k+2)+1$ requests between C' and E. Consider the undirected (bipartite) graph G whose vertices are the leaves of E and the nodes of C', and where there is an edge between a leaf from E and node of C' if there is a request between them. This graph has maximum degree $k+1$ by Rule Common Factor, thus by Lemma 9, G has a matching of size $k+2$. Thus there would be $k+2$ endpoint disjoint such requests, and Rule Common Factor would apply to them. □

Claim 27. *There are at most* $2k-1$ *caterpillars in* T.

Proof. There are at most $2k$ nodes in $I_1 \cup I_3$. Let us call them *separating nodes*. Let r be one of these separating nodes. Let us consider r as the root of T. Each caterpillar is adjacent to exactly two separating nodes. Let us associate to each caterpillar of T its adjacent separating node further away from the root r. This mapping is a bijection, and no caterpillar is mapped on r, thus there are at most $2k-1$ caterpillars. □

Claim 28.

$|L_3| = O(k^4)$

Proof. We have that $|I_3| = O(k)$ by Claim 21. Let X be an L_3-group rooted in $y \in I_3$. Because of Rule Disjoint Requests, at most $2k(k+1)$ leaves in X are endpoints of group requests (by Lemma 9 on the usual auxiliary request graph on X). Each leaf of X must be the endpoint of at least one request, so let us count the maximal number of requests contributed by each type of nodes. By Claim 25, and as there are at most k groups of L_1-leaves and k groups of L_3-leaves, at most $2k * 2k(k+2)$ leaves of X have a request toward an L_1-leaf or an L_3-leaf. There are at most $2k-1$ caterpillars in T by Claim 27, and leaves in X have in total at most $4k(k+2)$ R-neighbours in a given caterpillar by Claim 26. Thus $O(k^3)$ leaves in X are endpoints of a request toward a caterpillar node, and I_3 nodes can contribute for at most $O(k^2)$ requests, so $|X| = O(k^3)$. This gives $|L_3| = O(k^4)$. □

Claim 29.

$|L_2| = O(k^4)$

Proof. Assume by contradiction that $|L_2| \geq 3(2k-1)(k+1)(k+1)(4k+3)$. Let C be a caterpillar of T containing the maximum number of L_2-leaves. By Claim 27, there are at most $2k-1$ caterpillars in T, thus C contains at least $3(k+1)(k+1)(4k+3)$ L_2-leaves.

Assume first that C is not covered. We obtain a contradiction as in the caterpillar case. Consider x to be the L_2-leaf having a wingspan whose intersection $\tilde{W}$ with C has maximal size. Let C' be the subcaterpillar of backbone $\tilde{W}$. Then C' contains at least $(k+1)(4k+3)$ L_2-leaves, otherwise one would find $k+1$ disjoint wingspans by taking $\tilde{W}$, then a $\tilde{W}_1$ disjoint from $\tilde{W}$, then a $\tilde{W}_2$ disjoint from $\tilde{W}$ and $\tilde{W}_1$, ..., and finally a $\tilde{W}_k$ disjoint from $\tilde{W}, \tilde{W}_1, \dots, \tilde{W}_{k-1}$, as in Claim 15. Note that the caterpillars $W, W_1, \dots, W_k$ are themselves disjoint, as their intersections $\tilde{W}, \tilde{W}_1, \dots, \tilde{W}_k$ with C are disjoint and non-empty. Thus there would be $k+1$ disjoint requests, a contradiction. Since $\tilde{W}$ is maximal, each L_2-leaf y in C' is the endpoint of a request $r \subseteq C'$. The existence of $(k+1)(4k+3)$ L_2-leaves in C' means that there are at least $k+1$ endpoint-disjoint requests dominated by $\tilde{W}$, by Lemma 9 applied to the usual auxiliary request graph G on the L_2-leaves of C' (note that the maximum degree of G is at most $2k+2$). Thus Rule (5a) should apply, a contradiction.

Assume now that C is covered by some L_2-leaf x. If at least $(k+1)(4k+3)$ L_2-leaves in C do not cover C, then some wingspan of x dominates $(k+1)(4k+3)$ requests, and thus dominates at least $k+1$ endpoint-disjoint requests, by the usual application of Lemma 9. So Rule (5a) should apply, a contradiction. So at least $3(k+1)(k+1)(4k+3)-(k+1)(4k+3)$ L_2-leaves in C cover C, let X be the set of these leaves. Let a and b be the extremities of the caterpillar C, and denote by A and B the two corresponding connected components of $T-C$. Let $d_1, \dots, d_j$ be the I_1-nodes in A. Note that $j \leq k$. Consider the set X_i of $2(k+1)(4k+3)$ leaves in X whose closest R-neighbour toward d_i are the closest to a of all closest R-neighbours toward d_i of vertices in X. Note that a closest R-neighbour toward d_i of a vertex in X cannot belong to C, as leaves in X cover C. When less than $2(k+1)(4k+3)$ L_2-leaves in X have an R-neighbour toward d_i, remove X_i from X, mark d_i as *invalid*, and proceed. Note that at least one d_i must be valid, as $|X| > 2k(k+1)(4k+3)$.

Now we have a list of at most k sets (the sets X_i for d_i valid) of size $2(k+1)(4k+3)$. The union X' of these is of size at most $2k(k+1)(4k+3) < |X|$. Thus there exists an L_2-leaf z in $X-X'$. Consider the closest R-neighbour n_i of $\tilde{x}$ toward a valid d_i. Note that $\tilde{x}$ has no R-neighbour toward an invalid d_i, as such L_2-leaves has been removed from X. There are either $(k+1)(4k+3)$ L_2-leaves of X_i between $\tilde{x}$ and a or $(k+1)(4k+3)$ L_2-leaves of X_i between $\tilde{x}$ and b. Thus there are $k+1$ endpoint-disjoint requests either between the subcaterpillars of backbone $]\tilde{x},a[$ and $]a,n_i[$ or between the subcaterpillars of backbone $]b,\tilde{x}[$ and $]a,n_i[$, by Lemma 9 on the usual auxiliary request graph. In the former case Rule Common Factor applies to the request between $\tilde{x}$ and n_i (or a leaf pending at n_i) along with the above-mentioned $k+1$ endpoint disjoint requests. If the latter case applies for all valid d_i, then Rule generalised Dominating Wingspan applies to the L_2-leaf $\tilde{x}$. □

Claim 30.

$|I_2| = O(k^6)$

Proof. There are $O(k^4)$ internal nodes with leaves in T, by Claim 29. It remains to bound the cardinal of the set Z of inner nodes in I_2.

Let r be a given I_1-node of T, we now consider r as the root of T. Let u be an inner node in Z. Let $C(u)$ be the caterpillar containing u, denote by $a(u)$ and $b(u)$ its extremities, with $b(u)$ an ancestor of $a(u)$ with respect to r. Let $A(u)$ be the connected component of $T-\{u\}$ containing $a(u)$. If the node u has an R-neighbour in $A(u)$, select one such node $v(u)$. Note that u is on the path between $v(u)$ and r. By Rule Inclusion, $v(u) \neq v(u')$ whenever $u \neq u'$. Let G be the graph with vertex set Z, and with edge set $\{(u,v(u)) | u \in Z\}$. This graph G is a disjoint union of paths. By Rule Disjoint Requests, paths in G have length at most k. Vertices u in G which have no R-neighbour in $A(u)$ must be adjacent in T to some node not in Z, by Rule Unique Direction. There are $O(k^4)$ nodes not in Z, each of which can have at most k R-neighbours in Z. Indeed, a vertex cannot have two different R-neighbours in Z in the same direction, by Rule Inclusion. Thus there are $O(k^5)$ vertices u without R-neighbour in $A(u)$ in G, so there are $O(k^6)$ vertices in G, which finally means that there are $O(k^6)$ inner nodes in T. □

This concludes the proof of Theorem 19. □

We have shown that the (unweighted) Multicut In Trees problem admits a polynomial kernel. This kernelization algorithm, or just some particular sequence using some of the reduction rules presented above, can be used as a preprocessing or in-processing step in a practical algorithm.

This analysis might not be tight, so one can hope to improve this $O(k^6)$ bound retaining the same set of reduction rules. But new reduction rules should be needed to decrease this bound even further, to cubic size for example.

Our technique does not seem to generalise to the weighted version of MULTICUT IN TREES. Thus deciding whether the weighted MULTICUT IN TREES problem admits a polynomial kernel is still open.

The general Multicut in Graphs problem is now known to be FPT with respect to the parameter k, the size of the desired solution. The existence of a polynomial kernel for this unrestricted MULTICUT problem would be very surprising, but no proof of a negative answer has been provided yet. The following natural question seems both interesting and hard:

Problem 31. *For which graph classes $\mathcal{G}$ does the* MULTICUT *problem restricted to $\mathcal{G}$ admit a polynomial kernel?*

CHAPTER

3 Finding Directed Trees with Many Leaves

This chapter is organized as follows. In Section 3.1 we exhibit combinatorial bounds on the problem of finding a directed tree with many leaves. We use the notion of $s-t$ *numbering* introduced in [94]. We next present our reduction rules, which are independent of the parameter, and in Section 3.3 we prove that these rules give an edge-quadratic kernel when the root is prescribed. We present a constant factor approximation algorithm in Section 3.4 for finding an out-tree with many leaves in a digraph. Finally, in Section 3.5, we present a linear kernel for the rooted problem restricted to acyclic digraphs.

3.1 Combinatorial Bounds

Let D be a directed graph. For an arc (u,v) in D, we say that u is an *in-neighbour* of v, that v is an *out-neighbour* of u, that (u,v) is an *in-arc* of v and an *out-arc of* u. The *out-degree* of a vertex is the number of its out-neighbours, and its *in-degree* is the number of its in-neighbours.

Recall that an *out-tree* is a directed tree where each edge is oriented from the root towards the leaves, and an *out-branching* of a digraph D is a spanning out-tree in D.

An out-branching with a maximum number of leaves is said to be *optimal.* Let us denote by maxleaf(D) the number of leaves in an optimal out-branching of D.

Without loss of generality, we restrict ourselves to the following. *We exclusively consider loopless digraphs with a distinguished vertex of in-degree 0, denoted by* r. *We assume that there is no arc* (x,y) *with* $x \neq r$ *and where* y *is an out-neighbour of* r, *and that* r *has out-degree at least 2.* Throughout this chapter, we call such a digraph a *rooted digraph.* Definitions will be made exclusively with respect to rooted digraphs, hence the notions we present, like connectivity and resulting concepts, do slightly differ from standard ones.

Let D be a rooted digraph with a specified vertex r. The rooted digraph D is *connected* if every vertex of D is reachable by a directed path starting at r in D. A *cut* of D is a set $S \subseteq V(D) - r$ such that there exists a vertex $z \notin S$ which is not the endpoint of a directed path from r in $D - S$. In other words, the cut S separates z from r. We say that D is *2-connected* if D has no cut of size at most 1. A cut of size 1 is called a *cutvertex*. Equivalently, a rooted digraph is 2-connected if, for every vertex x distinct from r and its out-neighbours, there are two internally vertex-disjoint paths from r to x.

We will show that the notion of $s - t$ numbering behaves well with respect to out-branchings with many leaves. This concept has been introduced in [94] for 2-connected undirected graphs, and generalised in [31] by Cheriyan and Reif for digraphs which are 2-connected in the usual sense. We adapt this notion in the context of rooted digraphs.

Let D be a 2-connected rooted digraph. An $r - r$ *numbering* of D is a linear ordering σ of $V(D) - r$ such that each vertex x, which is not r and not an out-neighbour of r, has two in-neighbours u and v such that $\sigma(u) < \sigma(x) < \sigma(v)$. An equivalent presentation of an $r - r$ numbering of D is an injective embedding f of $V(D)$ where r has been duplicated into two vertices r_1 and r_2, into the $[0,1]$-segment of the real line, such that $f(r_1) = 0$, $f(r_2) = 1$, and such that the image by f of every vertex besides r_1 and r_2 lies inside the convex hull of the images of its in-neighbours. Such *convex embeddings* have been defined and studied in general dimension by Lovász, Linial and Wigderson in [95] for undirected graphs, and in [31] for directed graphs.

Given a linear order σ on a finite set V, we denote by $\overline{\sigma}$ the linear order on V which is the reverse of σ. An arc (u,v) of D is a *forward* arc if $u = r$ or if u appears before v in σ; (u,v) is a *backward* arc if $u = r$ or if u appears after v in σ[1]. A spanning out-tree T is *forward* (resp. *backward*) if all its arcs are forward (resp. backward) with respect to σ (and we omit the reference to σ when the context is clear).

The following result and proof is just an adapted version of [31], given here for the sake of completeness.

Lemma 32. *Let D be a 2-connected rooted digraph. There exists an $r - r$ numbering of D.*

Proof. By induction over D. We first reduce to the case where the in-degree is at most 2 (thus the in-degree of every vertex besides r and its out-neighbours is exactly 2). Let x be a vertex of in-degree at least 3 in D. Let us show that there exists an in-neighbour y of x such that the rooted digraph $D - (y,x)$ is 2-connected. Indeed, there exist two internally vertex disjoint paths from r to x. Consider such two paths intersecting the in-neighbourhood $N^-(x)$ of x only once each, and denote by D' the rooted digraph obtained from D by removing one arc (y,x) not involved in these two paths. There are two internally disjoint paths from r to x in D'. Consider $z \in V(D) - r - x$. Assume by contradiction that there exists a vertex t which cuts z from r in D'. As t does not cut z from r in D and the arc (y,x) alone is missing in D', t must cut x and not y from r in D'. Which is a contradiction,

1. Yes, arcs going out of the root are considered both backward and forward.

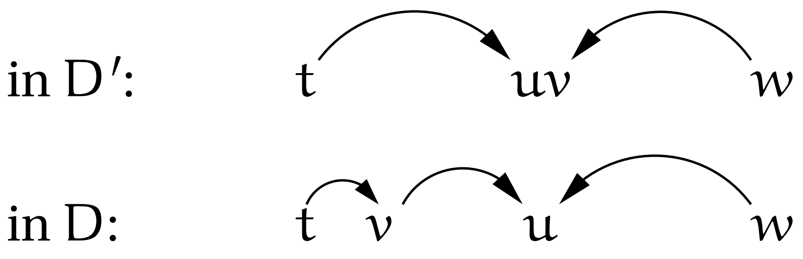


Figure 3.1: At the top, an $r-r$ numbering σ of D'. At the bottom, the resulting rr-ordering of D.

as there are two internally disjoint paths from r to x in D'. By induction, D' has an $r-r$ numbering, which is also an $r-r$ numbering for D.

Hence, let D be a 2-connected rooted digraph, where every vertex has in-degree at most 2. As r has in-degree 0, there exists a vertex v with out-degree at most 1 in D by a counting argument. If v has out-degree 0, then let σ be an $r-r$ numbering of $D-v$, let u_1 and u_2 be the two in-neighbours of v. Insert v between u_1 and u_2 in σ to obtain an $r-r$ numbering of D. Assume now that v has a single out-neighbour u. Let w be the second in-neighbour of u. Let D' be the graph obtained from D by contracting the arc (v,u) into a single vertex uv. As D' is 2-connected, consider by induction an $r-r$ numbering σ of D'. Replace uv by u in σ. It is now possible to insert v between its two in-neighbours so that u lies between v and w. Indeed, assume without loss of generality that w is after uv in σ. Consider the in-neighbour t of v smallest in σ. As σ is an $r-r$ numbering of D', t lies before uv in σ. We insert v just after t to obtain an $r-r$ numbering of D (see Figure 3.1). □

Note that an $r-r$ numbering σ of D naturally gives two acyclic covering subdigraphs of D, the rooted digraph $D_{|\sigma}$ consisting of the forward arcs of D, and the rooted digraph $D_{|\overline{\sigma}}$ consisting of the backward arcs of D. The intersection of these two acyclic digraphs is the set of out-arcs of r.

Corollary 33. *Let D be a 2-connected rooted digraph. There exists an acyclic connected spanning subdigraph A of D which contains at least half of the arcs of $D-r$.*

Let G be an undirected graph. A *vertex cover* of G is a set of vertices which hits all edges of G. We need the following folklore result:

Lemma 34. *Any undirected graph G on n vertices and m arcs has a vertex cover of size $\frac{n+m}{3}$.*

Proof. By induction on $n+m$. If there exists a vertex of degree at least 2 in G, choose it in the vertex cover, otherwise choose any non-isolated vertex. □

Let G be a bipartite graph with vertex bipartition (A,B). A set $S \subseteq B$ *dominates* A if $\bigcup_{v\in S} N(v) = A$.

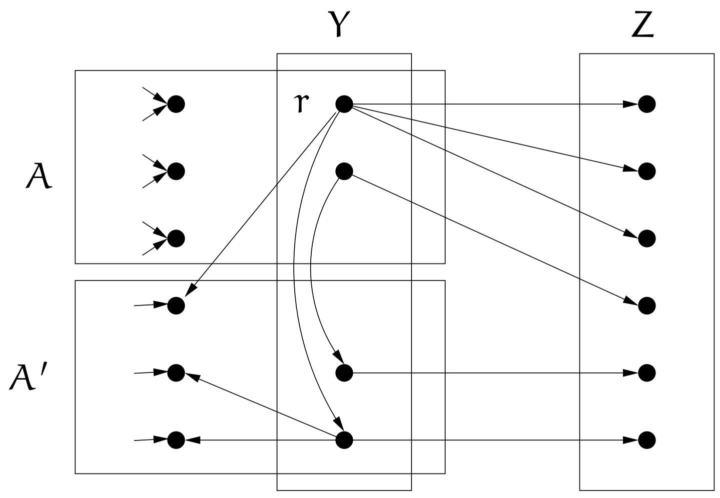


Figure 3.2: An illustration of the proof of Corollary 36.

Lemma 35. *Let* G *be a bipartite graph over* $A \cup B$*, with* $d(a) = 2$ *for every* $a \in A$*. There exists a subset of* B *of size at most* $\frac{|A|+|B|}{3}$ *which dominates* A*.*

Proof. Let G' be the graph whose vertex set is B, and where (b, b') is an arc if b and b' share a common neighbour in A. The result follows from Lemma 34 since G' has $|A|$ arcs and $|B|$ vertices. □

Let G be an undirected graph. A *dominating set* of G is a set $S \subseteq V$ such that for every vertex $x \notin S$, x has a neighbour in S. A *strongly dominating set* of G is a set $S \subseteq V$ such that every vertex has a neighbour in S.

Let D be a rooted digraph. A *strongly dominating set* of D is a set $S \subseteq V$ such that every vertex besides the root r has an in-neighbour in S. Denote by $d(r)$ the out-degree of the r.

Corollary 36. *Let* D *be an acyclic connected rooted digraph with* l *vertices of in-degree at least 2. Then* D *has an out-branching with at least* $\frac{l+d(r)-1}{3} + 1$ *leaves.*

Proof. Denote by n the number of vertices of D. For every vertex v of in-degree at least 3, delete incoming arcs until v has in-degree exactly 2. Since D is acyclic, it has a vertex s with out-degree 0.

Let Z be the set of vertices of in-degree 1 in D. Let Y be the set of in-neighbours of vertices of Z. Note that $|Y| \leq |Z| \leq n-1-l$. Let A' be the set of vertices of in-degree 2 which are out-neighbours of some vertex in Y. Let $B = V(D) - Y - s$. Let A be the set of vertices of in-degree 2 which are not in A'. See Figure 3.2.

Note that Y cannot have the same size as Z. Indeed, Z contains the out-neighbours of r, and hence Y contains r, which has out-degree at least 2. More precisely, $|Y| + d(r) - 1 \leq |Z \cup A'|$. As $B = V(D) - Y - s$ (and $s \notin Y$) and $A = V(D) - A' - Z - r$, we have that

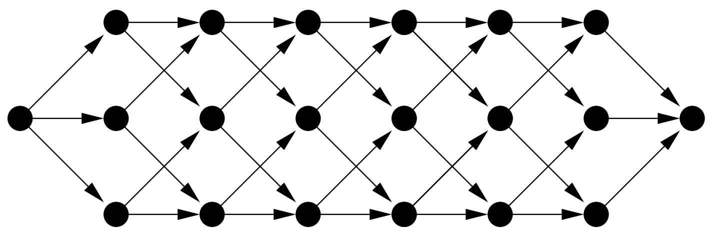


Figure 3.3: The "boloney" graph D_6

$|B| \geq |A| + d(r) - 1$. Moreover, as Y has size at most $n-1-l$, we have that $|B| \geq l$. Consider a copy A_1 of A and a copy B_1 of B. Let G be the bipartite graph with vertex bipartition (A_1, B_1), and where (b, a), with $a \in A_1$ and $b \in B_1$, is an edge if (b, a) is an arc in D. By Lemma 35 applied to G, there exists a set $X \subseteq B$ of size at most $\frac{|A|+|B|}{3} \leq \frac{2|B|-(d(r)-1)}{3}$ which dominates A in D. The set $C = X \cup Y$ strongly dominates $V(D) - r$ in D, and has size at most $|X| + |Y| \leq \frac{2|B|-(d(r)-1)}{3} + |Y| = |B| + |Y| - \frac{|B|+d(r)-1}{3}$. As $|Y| + |B| = n-1$ and $|B| \geq l$, this yields $|X \cup Y| \leq n-1-\frac{l+d(r)-1}{3}$. As D is acyclic, any set which strongly dominates $V - r$ contains r and is a connected dominating set. Hence there exists an out-branching T of D having a subset of C as internal vertices. T has at least $\frac{l+d(r)-1}{3} + 1$ leaves. □

This bound is tight up to one leaf. The rooted digraph D_k depicted in Figure 3.3 is 2-connected, has $3k-2$ vertices of in-degree at least 2, $d(r) = 3$ and maxleaf$(D_k) = k+2$.

Finally, the following combinatorial bound is obtained:

Theorem 37. *Let* D *be a 2-connected rooted digraph with* l *vertices of in-degree at least 3. Then maxleaf*$(D) \geq \frac{l}{6}$.

Proof. Apply Corollary 36 to the rooted digraph with the larger number of vertices of in-degree 2 among D_σ and $D_{\overline{\sigma}}$. □

A *2-circuit* is a pair of vertices u, v of D such that (u, v) and (v, u) are arcs in D. An arc is *simple* if does not belong to a 2-circuit. A vertex v is *nice* if it is incident to a simple in-arc.

The second combinatorial bound is the following:

Theorem 38. *Let* D *be 2-connected rooted digraph with* l *nice vertices (*i.e. *vertices incident to a simple in-arc). Then* D *has an out-branching with at least* $\frac{l}{24}$ *leaves.*

Proof. By Lemma 32, we consider an $r-r$ numbering σ of D. For every nice vertex v (incident to some in-arc a) with in-degree at least three, delete incoming arcs of v different from a until v has only one incoming forward arc and one incoming backward arc. For every other vertex of in-degree at least 3 in D, delete incoming arcs of v until v has only one incoming forward arc and one incoming backward arc. At the end of this process, σ is still an $r-r$ numbering of the digraph D, and the number of nice vertices has not decreased.

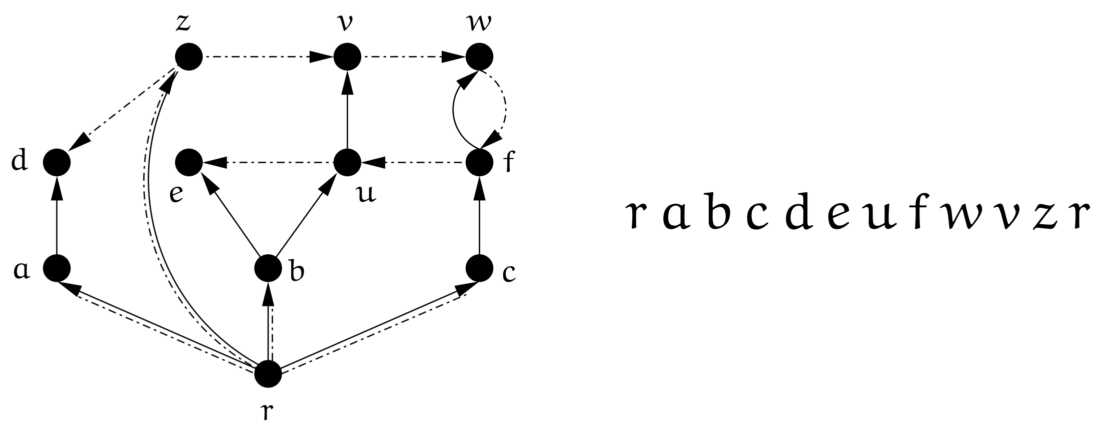


Figure 3.4: A 2-connected rooted digraph along with an $r-r$ numbering. Backward arcs are depicted with dashed lines and forward arcs are depicted with full lines. The nice vertex v is incident to the simple arc uv of T_f which is not transverse in T_b. Hence the arc vw of T_b is transverse in T_f, as per the proof of Theorem 38.

Denote by T_f the set of forward arcs of D, and by T_b the set of backward arcs of D. As σ is an $r-r$ numbering of D and the in-degree in D is at most 2, T_f and T_b are two spanning trees of D which partition the arcs of $D-r$.

The crucial definition is the following: say that an arc uv of T_f (resp. of T_b), with $u \neq r$, is *transverse* if u and v are *incomparable* in T_b (resp. in T_f), that is if v is not an ancestor of u in T_b (resp. in T_f). Observe that u cannot be an ancestor of v in T_b (resp. in T_f) since T_b is backward (resp. T_f is forward) while uv is forward (resp. backward) and $u \neq r$.

Assume without loss of generality that T_f contains at least as many transverse arcs as T_b. Consider now any planar drawing of the rooted tree T_b. We make use of this drawing to define the following: if two vertices u and v are incomparable in T_b, then one of these vertices is to the left of the other, with respect to our drawing. In other words, we fix the order of the branches in T_b.

We can now partition the transverse arcs of T_f into two subsets: the set S_l of transverse arcs uv for which v is to the left of u, and the set S_r of transverse arcs uv for which v is to the right of u. Assume without loss of generality that $|S_l| \geq |S_r|$.

The digraph $T_b \cup S_l$ is an acyclic digraph by definition of S_l. Moreover, it has $|S_l|$ vertices of in-degree two since the heads of the arcs of $|S_l|$ must be pairwise distinct. Hence, by Corollary 36, $T_b \cup S_l$ has an out-branching with at least $\frac{|S_l|+d(r)-1}{3}+1$ leaves, hence so does D.

We now give a lower bound on the number of transverse arcs in D to bound $|S_l|$. Consider a nice vertex v in D, which is not an out-neighbour of r, and with a simple in-arc uv belonging to, say, T_f. If uv is not a transverse arc, then v is an ancestor of u in T_b. Let w be the out-neighbour of v on the path from v to u in T_b. Since uv is simple, the vertex w

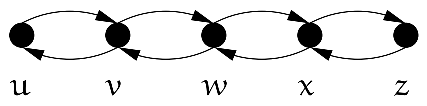


Figure 3.5: A bipath of length 4. Rule (2) states that v and w can be contracted (or equivalently w and x).

is distinct from u. No path in T_f goes from w to v since w is before v in σ, hence vw is a transverse arc (see Figure 3.4). Therefore, we proved that v (and hence every nice vertex which is not an out-neighbour of r) is incident to a transverse arc (either an in-arc, or an out-arc). Thus there are at least $\frac{l-d(r)}{2}$ transverse arcs in D.

Finally, there are at least $\frac{l-d(r)}{4}$ transverse arcs in T_f, and thus $|S_l| \geq \frac{l-d(r)}{8}$. In all, D has an out-branching with at least $\frac{l}{24}$ leaves. □

As a corollary, the following result holds for oriented graphs (digraphs with no 2-circuit):

Corollary 39. *Every 2-connected rooted oriented graph on n vertices has an out-branching with at least $\frac{n-1}{24}$ leaves.*

3.2 Reduction Rules

We say that $P = \{x_1, \dots, x_l\}$, with $l \geq 3$, is a *bipath of length* $l-1$ if the set of arcs adjacent to $\{x_2, \dots, x_{l-1}\}$ in D is exactly $\{(x_i, x_{i+1}), (x_{i+1}, x_i) | i \in \{1, \dots, l-1\}\}$. See Figure 3.5.

To exhibit a quadratic kernel for ROOTED MAXIMUM LEAF OUT-BRANCHING, we use the following four reduction rules:

(0) If there exists a vertex not reachable from r in D, then reduce to a trivially FALSE instance.

(1) Let x be a cutvertex of D. Delete vertex x and add an arc (v, z) for every $v \in N^-(x)$ and $z \in N^+(x) - v$.

(2) Let P be a bipath of length 4. Contract two consecutive internal vertices of P.

(3) Let x be a vertex of D. If there exists $y \in N^-(x)$ such that $N^-(x) - y$ cuts y from r, then delete the arc (y, x).

Note that these reduction rules are not parameter dependent. Rule (0) only needs to be applied once.

Observation 40. *Let S be a cutset of a rooted digraph D. Let T be an out-branching of D. There exists a vertex in S which is not a leaf in T.*

Lemma 41. *The above reduction rules are safe and can be checked and applied in polynomial time.*

Proof. (0) Reachability can be tested in linear time.

(1) Let x be a cutvertex of D. Let D' be the graph obtained from D by deleting vertex x and adding an arc (v,z) for every $v \in N^-(x)$ and $z \in N^+(x) - v$. Let us show that maxleaf$(D) =$ maxleaf(D'). Assume that T is an out-branching of D rooted at r with k leaves. By Observation 40, x is not a leaf of T. Let $f(x)$ be the father of x in T. Let T' be the tree obtained from T by contracting x and $f(x)$. T' is an out-branching of D' rooted at r with k leaves.
Let T' be an out-branching of D' rooted at r with k leaves. $N^-(x)$ is a cut in D', hence by Observation 40 there is a non-empty collection of vertices $y_1,\dots,y_l \in N^-(x)$ which are not leaves in T'. Choose y_i such that y_j is not an ancestor of y_i in T' for every $j \in \{1,\dots,l\}-\{i\}$. Let T be the graph obtained from T' by adding x as an isolated vertex, adding the arc (y_i,x), and for every $j \in \{1,\dots,l\}$, for every arc $(y_j,z) \in T$ with $z \in N^+(x)$, deleting the arc (y_j,z) and adding the arc (x,z). As y_i is not reachable in T' from any vertex $y \in N^-(x)-y_i$, there is no cycle in T. Hence T is an out-branching of D rooted at r with at least k leaves. Moreover, deciding the existence of a cut vertex and finding one if such exists can be done in polynomial time.

(2) Let P be a bipath of length 4. Let u, v, w, x and z be the vertices of P in this consecutive order. Let T be an out-branching of D. Let D' be the rooted digraph obtained from D by contracting v and w. The rooted digraph obtained from T by contracting w with its father in T is an out-branching of D' with as many leaves as T.
Let T' be an out-branching of D'. If the father of vw in T' is x, then $T' - (x,vw) \cup (x,w) \cup (w,v)$ is an out-branching of D with at least as many leaves as T'. If the father of vw in T' is u, then $T' - (u,vw) \cup (u,v) \cup (v,w)$ is an out-branching of D with as many leaves as T'.

(3) Let x be a vertex of D. Let $y \in N^-(x)$ be a vertex such that $N^-(x) - y$ cuts y from r. Let D' be the rooted digraph obtained from T by deleting the arc (y,x). Every out-branching of D' is an out-branching of D. Let T be an out-branching of D containing (y,x). There exists a vertex $z \in N^-(x) - y$ which is an ancestor of x. Thus $T - (y,x) \cup (z,x)$ is an out-branching of D' with as many leaves as T.

□

Observe that Rule (1) could be generalised as follows:

(1') Let x be a vertex of D which is not a leaf in some optimal out-branching of D. Delete vertex x and add an arc (v,z) for every $v \in N^-(x)$ and $z \in N^+(x) - v$.

But we will not use this more general rule.

Observation 42. *If a digraph has an out-branching rooted at* r*, then an out-tree rooted at* r *with* k *leaves can be extended to an out-branching rooted at* r *with at least* k *leaves in linear time.*

We apply rules (0), (1), (2) and (3) iteratively until reaching a *reduced instance*, on which none can be applied.

Lemma 43. *Let* D *be a reduced rooted digraph with a vertex of in-degree at least* k*. Then* D *is a TRUE instance.*

Proof. Assume that x is a vertex of D with in-neighbourhood $N^-(x) = \{u_1, \dots, u_l\}$, with $l \geq k$. For every $i \in \{1, \dots, l\}$, $N^-(x) - u_i$ does not cut u_i from r. Thus there exists a path P_i from r to u_i outside $N^-(x) - u_i$. The rooted digraph $D' = \cup_{i \in \{1,\dots,l\}} P_i$ is connected, and for every $i \in \{1, \dots, l\}$, u_i has out-degree 0 in D'. Thus D' has an out-branching with at least k leaves, and such an out-branching can be extended into an out-branching of D with at least as many leaves, by Observation 42. □

3.3 Quadratic Kernel

In this section and in Section 3.4, a vertex of a 2-connected rooted digraph D is said to be *special* if it has in-degree at least 3 or if one of its incoming arcs is simple. A non-special vertex is a vertex u which has exactly two in-neighbours, which are also out-neighbours of u. A *weak bipath* is a maximal connected set of non-special vertices. If $P = \{x_1, \dots, x_l\}$ is a weak bipath of length $l - 1$, with $l \geq 1$, then the in-neighbours of x_i, for $i = 2, \dots, l-1$ in D are exactly x_{i-1} and x_{i+1}. Moreover, x_1 and x_l each forms a 2-circuit with a special vertex. Indeed, each must have a second in-neighbour, and non-special vertices are not incident to simple in-arcs. If $l = 1$, then x_1 forms two 2-circuits with two special vertices. Denote by $s(P)$ the special in-neighbour of x_1. If $l = 1$, we distinguish one of the two special in-neighbours as $s(P)$.

See Figure 3.7 for an example.

This section is dedicated to the proof of the following statement:

Theorem 44. *A digraph* D *with at least* $90k^2$ *vertices and reduced under the reduction rules of the previous section has an out-branching with at least* k *leaves.*

Proof. By Theorem 37 and Theorem 38, if there are at least $6k + 24k - 1$ special vertices, then D has an out-branching with at least k leaves, as a vertex is special if it is nice or if it has in-degree at least 3. Assume that there are at most $30k - 2$ special vertices in D.

As D is reduced under Rule (2), there is no bipath of length 4. We can associate to every weak bipath B of D on l vertices a set of $\lceil \frac{l}{3} \rceil$ out-arcs toward special vertices. Indeed, let $P = (x_1, \dots, x_l)$ be a weak bipath of D. For every three consecutive vertices x_i, x_{i+1}, x_{i+2} of P, with $2 \leq i \leq l - 3$, $(x_{i-1}, x_i, x_{i+1}, x_{i+2}, x_{i+3})$ is not a bipath by Rule (2), hence there exists an arc (x_j, z) with $j = i, i+1$ or $i+2$ and $z \notin P$. Moreover, z must be a special vertex as arcs between non-special vertices lie within their own weak bipath. Finally, x_1 and x_l have an out-arc toward a special vertex, so each bipath on l vertices contributes for at least $2 + \lfloor \frac{l-2}{3} \rfloor \geq \lfloor \frac{l}{3} \rfloor + 1 \geq \lceil \frac{l}{3} \rceil$ out-arcs toward special vertices.

By Lemma 43, a special vertex in D has in-degree at most $k - 1$ as D is reduced under Rule (3). Thus, there are at most $3(k-1)(30k-2)$ non-special vertices in D, so the total number of vertices in D sums to less than $90k^2$ as claimed. □

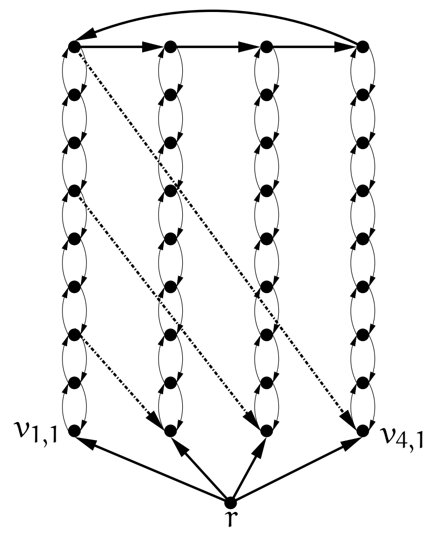


Figure 3.6: The lower bound graph T_l, with $l = 4$. Arcs of the type $(v_{i,3t}, v_{i+t,1})$ are only depicted for $i = 1$ for clarity.

To sum up, the kernelization algorithm is as follows: starting from a rooted digraph D, apply the reduction rules. Let D' be the obtained reduced rooted digraph. If D has size more than $90k^2$, then reduce to a trivially TRUE instance. Otherwise, D' is an instance equivalent to D with $O(k^2)$ vertices.

Observation 45. *A reduced instance has actually* $O(k^2)$ *arcs.*

Proof. There is a linear number of special vertices, so there is at most a quadratic number of arcs between special vertices. The number of arcs between non-special vertices is linear in the number of non-special vertices (and hence quadratic in k) since non-special vertices form weak bipaths. The are only two arcs from special vertices to all non-special vertices in a given weak bipath, thus the number of arcs from special vertices to non-special vertices is at most quadratic as well. Finally, special vertices have in-degree less than k, thus the number of arcs from non-special vertices to special vertices is at most quadratic as well. □

This quadratic bound is tight up to a constant factor with respect to our reduction rules. Indeed, the graph T_l depicted in Figure 3.6 and described below is reduced under the reduction rules stated in Section 3.2 and has a number of vertices quadratic in its maximal number of leaves.

Let $V = \{v_{i,j} | i = 1, \dots, l, j = 1, \dots, 3(l-1)\}$. For every $i = 1, \dots, l$, the pair $(r, v_{i,1})$ is an arc of T. For every $j = 1, \dots, 3l-2$ and $i = 1, \dots, l$, the pair $(v_{i,j}, v_{i,j+1})$ is a 2-circuit of T_l. For every $i = 1, \dots, l$, the pair $(v_{i,3l-1}, v_{i+1[l],3l-1})$ is an arc of T_l. For every $t = 1, \dots, l-1$ and $i = 1, \dots, l$, the pair $(v_{i,3t}, v_{i+t,1})$ is an arc of T_l, where $i+t$ takes values in $\{1, \dots, l\}$

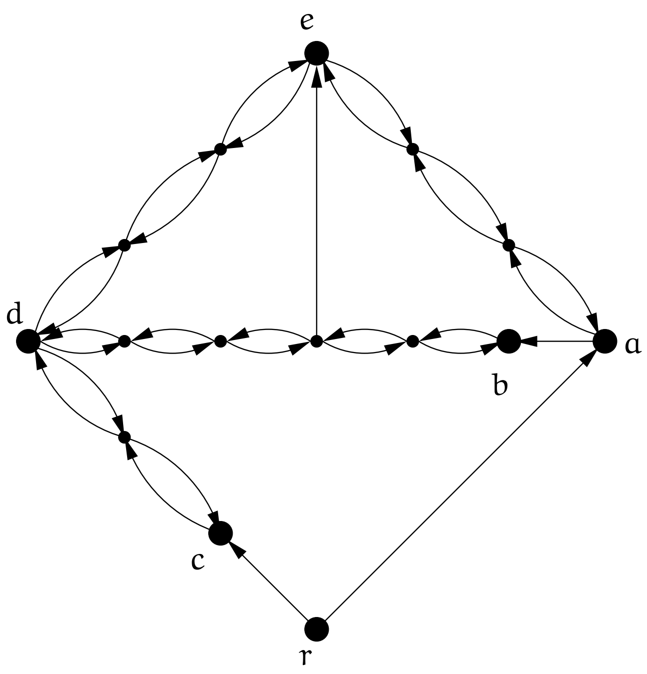


Figure 3.7: This graph is 2-connected and reduced under the reduction rules of Section 3.2. Indeed, the horizontal weak bipath (between d and b) contains no bipath of length 4 due to the simple out-arc towards e, so Rule (2) does not apply. The root r and the special vertices a, b, c, d, e are depicted with fat dots. Vertex d has in-degree 3, vertices a, b, c are incident to simple in-arcs, and vertex e verifies both. Other vertices are non-special, and thus are organized into weak bipaths between two special vertices. For example, there is a weak bipath (of length 3) between b and d, and a weak bipath (of length 0) between c and d.

depending on its value modulo l. This digraph T_l is reduced under the reduction rules of Section 3.2, and $\text{maxleaf}(T_l) = 2(l-1)$. Finally, T_l has $3l(l-1)+1$ vertices.

Note that this graph has many 2-circuits. We are not able to deal with them with respect to kernelization. For the approximation on the contrary, we are able to deal with the 2-circuits to produce a constant factor approximation algorithm.

3.4 Constant-Factor Approximation

This section is devoted to a constant factor approximation algorithm for ROOTED MAXIMUM LEAF OUT-BRANCHING, being understood that this also gives an approximation algorithm of the same factor for MAXIMUM LEAF OUT-BRANCHING as well as for finding an out-tree (not necessarily spanning) with many leaves in a digraph.

The reduction rules described in Section 3.2 directly give an approximation algorithm asymptotically as good as the best known approximation algorithm [50]. Indeed, as these rules are independent of the parameter, and as our proof of the existence of a solution of size k when the reduced graph has size more than $90k^2$ is constructive, this yields a $O(\sqrt{OPT})$ approximation algorithm. Let us sketch this approximation algorithm. Start by applying the reduction rules described in Section 3.2 to the input rooted digraph. This does not change the maximum number of leaves in an out-branching. Let m be the size of the reduced digraph. We exhibit an out-branching with at least $\sqrt{\frac{m}{90}}$ leaves as in the proof of Theorem 44. Finally, undo the sequence of contractions given by the application of the reduction rules at the start of the algorithm, repairing the tree as in the proof of Lemma 41. The tree thus obtained has at least $\sqrt{\frac{m}{90}}$ leaves, while the tree with a maximum number of leaves in the input graph has at most $m-1$ leaves. Thus this algorithm is an $O(\sqrt{OPT})$ approximation algorithm.

Let us now describe our constant factor approximation algorithm. Given a rooted digraph D'', apply exhaustively Rule (1) of Section 3.2. The resulting rooted digraph D is 2-connected. By Lemma 41, maxleaf(D'') = maxleaf(D).

Let us denote by D_{ns} the digraph D restricted to non-special vertices. Recall that D_{ns} is a disjoint union of bipaths, which we call *non-special components*. A vertex of out-degree 1 in D_{ns} is called an *end*. Each end has exactly one special vertex as an in-neighbour and out-neighbour in D. See Figure 3.7 for an example, where D_{ns} is the graph induced by the vertices depicted with small dots.

Theorem 46. *Let D be a 2-connected rooted digraph with l special vertices and h non-special components. Then max*$(\frac{l}{30}, h-l)$ ≤*maxleaf*$(D) \leq l+2h$.

Proof. The upper bound is clear, as at most two vertices in a given non-special component can be leaves of a given out-branching. The first term of the lower bound comes from Theorem 37 and Theorem 38. To establish the second term, consider the digraph D' which is to restriction of D to r and the special vertices of D. For every non-special component of D, add a 2-circuit in D' between the special in-neighbours of its two ends. Consider an out-branching of D' rooted at r. This out-branching uses l edges in D', and directly corresponds to an out-tree T in D. Extend T into an out-branching $\tilde{T}$ of D. Every non-special component which is not used in T contributes to at least a leaf in $\tilde{T}$, which concludes the proof. □

Consider the best of the three out-branchings of D obtained in polynomial time by Theorem 37, Theorem 38 and Theorem 46. This out-branching has at least max$(\frac{l}{30}, h-l)$ leaves. The worst case is when $\frac{l}{30} = h-l$. In this case, the upper bound becomes: $l+2h = \frac{92l}{30}$, so we have a factor 92 approximation algorithm for ROOTED MAXIMUM LEAF OUT-BRANCHING.

Note that the above analysis is probably not tight, and this constant factor approximation algorithm to find directed trees with many leaves should actually have a better ratio than 92.

3.5 Linear Kernel for DIRECTED k-LEAF restricted to Acyclic Digraphs

Note that the results in Section 3.3 do not give a vertex-linear kernel for oriented graphs. Indeed, the reduction rules applied to an oriented graph can create 2-circuits. The existence of a vertex-linear kernel for ROOTED MAXIMUM LEAF OUT-BRANCHING is still open even in the case of oriented graphs. In this section, we prove that the simpler acyclic case admits a vertex-linear kernel.

An acyclic digraph D has an out-branching if and only if D has a single vertex of in-degree zero. This follows from Observation 47 which will also be used in Chapter 8.

Observation 47. *[[7]] A digraph D has an out-branching if and only if D has a single strongly connected component without incoming arcs. One can decide whether a digraph has an out-branching in time $O(n+m)$.*

Hence, we assume that the acyclic digraph D under consideration has a single vertex r of in-degree zero.

We start from the following simple lemma.

Lemma 48. *In an acyclic digraph D with a single source r, every spanning subgraph of D, where each vertex different from r has in-degree 1, is an out-branching.*

Let B be an undirected bipartite graph with vertex bipartition (V', V''). A subset S of V' is called a *bidominating set* if for each $y \in V''$ there is an $x \in S$ such that $xy \in E(B)$. The so-called *greedy covering algorithm* [6] finds a bidominating set as follows. Start from the empty set C. While $V'' \neq \emptyset$, choose a vertex v of V' of maximum degree, add v to C, and delete v from V' and the neighbours of v from V''.

The following lemma has been obtained independently by several authors, see Proposition 10.1.1 in [6].

Lemma 49. *If the minimum degree of a vertex in V'' is d, then the greedy covering algorithm finds a bidominating set of size at most $1 + \frac{|V'|}{d}(1 + \ln \frac{d|V''|}{|V'|})$.*

Let D be an acyclic digraph with a single source. We use the following reduction rules to get rid of some vertices of in-degree 1.

(A) If D has an arc $a = xy$ with $d^+(x) = d^-(y) = 1$, then contract a.

(B) If D has an arc $a = xy$ with $d^+(x) \geq 2$, $d^-(y) = 1$ and $x \neq r$, then delete x and add an arc uv for each $u \in N^-(x)$ and $v \in N^+(x)$.

Note that applying Rules (A) and (B) preserves the acyclicity. These reduction rules are safe, essentially because the arc a must belong to an out-branching in both cases:

Lemma 50. *Let D^* be the digraph obtained from an acyclic digraph D with a single source by applying exhaustively Rule A and Rule B. Then D^* has a k-out-branching if and only if D has one.*

Proof. Consider an acyclic digraph D with an arc $a = xy$ with $d^+(x) = d^-(y) = 1$ and let D' be the digraph obtained from D by contracting a. Let T be a k-out-branching of D. Clearly, T contains a and let T' be the out-branching obtained from T by contracting a. Then T' is also a k-out-branching. Similarly, if D' has a k-out-branching T', then D has a k-out-branching T obtained from T' by adding the arc a. Hence Rule (A) is safe.

Consider now an acyclic digraph D with an arc $a = xy$ with $d^+(x) \geq 2$, $d^-(y) = 1$ and $x \neq r$ and let D' be obtained from D by applying Rule (B). Let T be a k-out-branching in D. Then T contains the arc xy and x is not a leaf of T. Let U be the subset of $N^+(x)$ such that $xu \in A(T)$ for each $u \in U$ and let v be the vertex such that $vx \in A(T)$. Then the out-branching T' of D' obtained from T by deleting x and adding arcs vu for every $u \in U$ has at least k leaves. Note that T' is indeed an out-branching of D' by Lemma 48. Similarly, a k-out-branching of D can easily be constructed from a k-out-branching of D' by appropriately inserting the vertex x. □

Consider the resulting digraph D^*. Let B be an undirected bipartite graph, with vertex bipartition (V', V''), where V' is a copy of $V(D^*)$ and V'' is a copy of $V(D^*) - \{r\}$. The set of edges of B is $E(B) = \{u'v'' \mid u' \in V', v'' \in V'', uv \in A(D^*)\}$.

Lemma 51. *Let R be a bidominating set of B. Then D^* has an out-branching T such that the copies of the leaves of T in V' form a superset of $V' - R$.*

Proof. Consider a subgraph Q of B obtained from B by deleting all edges apart from one edge between every vertex in V'' and its neighbour in R. By Lemma 48, Q corresponds to an out-branching T of D^* such that the copies of the leaves of T in V' form a superset of $V' - R$. □

Theorem 52. *If D^* has no k-out-branching, then the number n^* of vertices of D^* is less than $6.6(k+2)$.*

Proof. Suppose that $n^* \geq 6.6(k+2)$; we will prove that D^* has a k-out-branching. Observe that by Rules A and B, all vertices of D^* are of in-degree at least 2, besides r and possibly some of its out-neighbours. Let X denote the set of out-neighbours of r of in-degree 1 and let X'' be the set of copies of X in V''. Observe that the vertices of $V'' - X''$ are all of degree at least 2 in the bipartite graph $B - X''$. Thus, by Lemma 49, $B - X''$ has a bidominating set S of size at most $\frac{n^*}{2}(1 + \ln 2) + 1$. Hence, $S \cup \{r\}$ is a bidominating set of B and, by Lemma 51, D^* has a b-out-branching with $b \geq n^* - \frac{n^*}{2}(1 + \ln 2) - 2 \geq 0.153n^* - 2 \geq k$. □

3.6 Conclusion

In this chapter, we have given an edge-quadratic kernel and a constant factor approximation algorithm for Rooted Maximum Leaf Out-Branching, reducing the gap between the problem of finding trees with many leaves in undirected and directed graphs.

MAXIMUM LEAF SPANNING TREE has a factor 2 approximation algorithm, and ROOTED MAXIMUM LEAF OUT-BRANCHING now has a factor 92 approximation algorithm. Reducing this 92 factor into a small constant is one challenge.

The gap now essentially lies in the fact that MAXIMUM LEAF SPANNING TREE has a vertex-linear kernel while ROOTED MAXIMUM LEAF OUT-BRANCHING has a quadratic kernel. Deciding whether ROOTED MAXIMUM LEAF OUT-BRANCHING has a vertex-linear kernel is a challenging question. We have given a positive answer in the simpler acyclic case. Whether long paths made of 2-circuits can be dealt with or not through reduction rules might be key for the existence of a vertex-linear kernel in the general case.

Part II: Graph Structure and Decompositions

We give a brief introduction to treewidth and graph minors, two important structural concepts used in this part. We refer the reader to Robertson and Seymour's Graph Minors series and to Bodlaender's survey [10] for a more complete presentation of treewidth.

Introduction to Treewidth

Informally, treewidth mesures the closeness of a graph to a tree. The definition of treewidth is usually credited to Robertson and Seymour in their Graph Minors series. Halin had apparently introduced this concept earlier, and several different formulations have been independently introduced by other authors later on. The following definition is the standard presentation of treewidth.

A *tree decomposition* of a graph $G = (V, E)$ is a tree T and a set $\{T_v | v \in V\}$ of subtrees of T such that $T_u \cap T_v \neq \emptyset$ for every edge $(u, v) \in E$. The *width* of a decomposition is the cardinal of the largest pairwise intersecting set of subtrees *minus one*. The treewidth of a graph is the minimum width of its tree decompositions.

Alternatively, a graph has treewidth at most k if it is a subgraph of a chordal graph whose cliques have size at most $k+1$. Another way to express this property is that a graph has treewidth at most k if there exists a vertex elimination ordering such that each vertex has at most k non-eleminated neighbours when eliminated.

Treewidth can also be defined in terms of cops and robber games. A *haven of order* $k+1$ of a graph G is a function f mapping each set X of at most k vertices of G to one of the connected components of $G - X$ and such that $f(Y) \subset f(X)$ when $X \subset Y$. The treewidth of G is the largest index k such that G admits a haven of order $k+1$.A haven of order $k+1$ can be used by an infinitely fast visible robber to avoid k cops trying to catch him in G [116], and conversely a tree-decomposition of width at most $k-1$ gives a search strategy for k cops to catch the robber.

The $+1$ or -1 is added in these definitions so that the graphs of treewidth 1 are the forests.

Treewidth admits a negative certificate: brambles. A *bramble* of a graph is a collection of connected sets of vertices such that the union of every two sets is connected. The *order* of a bramble is the minimum size of a transversal, *i.e.* of a vertex set which intersects each set in the bramble.

Theorem 53 ([116])**.** *A graph has treewidth at most* k *if and only if it admits no bramble of order* $k+1$.

The crucial importance of treewidth lies in the fact that many difficult and important graph problems become tractable on graphs of bounded treewidth. Among others: CLIQUE, COLOURING, FEEDBACK VERTEX SET, HAMILTONIAN CYCLE, GRAPH ISOMORPHISM. Algorithmic techniques relying on bounded treewidth are generally called Dynamic Programming. The majot tractability meta-result for problems on graphs of bounded treewidth is Courcelle's Theorem [33]: every problem definable in monadic second order logic is FPT when parameterized by the treewidth of the input graph.

Computing the treewidth of a graph is NP-hard, and FPT when parameterized by the treewidth [9]. Treewidth can be approximated within a factor $O(\sqrt{\log(n)})$ [58], and the existence of a constant factor approximation algorithm for treewidth is open. On planar graphs, is it not known whether computing treewidth is NP-hard.

Minors, treewidth and order theory

Minor is a richer relation than subgraph. A graph H is a *minor* of a graph G if H can be obtained from G by a sequence of edge-contractions and edge- and vertex-deletions.

Many natural graph problems are closed under minors. For example planarity is preserved by contractions and deletions. K_5 and $K_{3,3}$ are two non-planar graphs, and Kuratowski showed that planarity is actually characterised by these two graphs through the minor relation:

Theorem 54 (Kuratowski)**.** *A graph is planar if and only if it does not have* K_5 *and* $K_{3,3}$ *as a minor.*

Every class of graph closed under a *quasi-order* (a reflexive and transitive relation) $\preceq$ is characterised by a (possibly infinite) set of obstructions under $\preceq$, namely the set of minimal (by $\preceq$) graphs outside the class. This set of obstructions is an *antichain* for $\preceq$ (a set of pairwise incomparable elements). The key point of Kuratowski's result is that this set is actually finite for planar graphs ordered by the minor relation. More generally, a quasi-order is called a *well-quasi-order* when every antichain is finite. Wagner reportedly conjectured [2] that the minor relation is a well-quasi-order, which implies that every class of graphs closed under minor is characterised by a finite set of obstructions. This conjecture was proved by Robertson and Seymour and is now known as the graph minor theorem.

Theorem 55 (Robertson and Seymour's graph minor theorem)**.** *The minor relation is a well-quasi-order on the set of finite graphs.*

On top of its fundamental importance, the graph minor theorem has a also great algorithmic significance, as minor testing can be done in polynomial time.

Corollary 56. *Graph problems closed under minor are polynomial time decidable.*

Treewidth is a key tool in Robertson and Seymour's approach. Graphs of bounded treewidth have a sufficiently simple structure for Robertson and Seymour to prove that minor is a well-quasi-order on a set of graphs of bounded treewidth [111, 112]. Hence we can consider only graphs of large treewidth. This is essentially enough to prove the graph minor theorem on planar graphs. Indeed, a planar graph is a minor of a large enough grid (imagine a planar drawing of the graph approximated through a very fine crosshatch). The following result allows to conclude in the planar case:

Theorem 57 ([115])**.** *A graph with treewidth at least* 2^{2m^5} *contains an* $m * m$ *grid minor.*

Putting it all together, an infinite antichain of planar graphs must contain graphs of arbitrarily large treewidth, which admit arbitrarily large grid minors. This is a contradiction, as each of these planar graphs is a minor of a large enough grid.

Things are more complicated when considering non-planar graphs. In an infinite antichain S of graphs, consider a graph H. Each graph in S is H-minor free by definition. Robertson and Seymour exhibited a structure theorem for H-minor free graphs: these graphs can be constructed with a tree-like stucture from graphs "nearly-embeddable" in some surface. This structure theorem is a big part of Robertson and Seymour's proof of the graph minor theorem, and a reasoning similar to the proof sketched above for planar graphs can be used to conclude.

Reducing Multicut to Bounded Treewidth

In Chapter 4, we show that MULTICUT parameterized by the solution size k can be reduced to graphs of treewidth bounded by a function of k. This is a joint work with Christophe Paul, Anthony Perez and Stéphan Thomassé.

This chapter is highly concerned with minors. Indeed, graphs of large treewidth admit large grid minors, and we use different types of minors to structure the instance and obtain more information on the behavior of multicuts. Chapter 4 can be seen as an first step

2. Although he actually denied having made this conjecture.

towards the proof that MULTICUT is FPT given in Chapter 7. The connectivity tools given in Chapter 4 are the foundations of the approach in Chapter 7.

In Section 4.2, we study constrained separation problems of the type: separate vertex x from vertex y without seperating y from vertex z. This allows us to design powerful reduction rules for MULTICUT. For example, Rule 2 in Section 4.2 allows each vertex to participate only to a number of requests bounded in k. Rule 3 is yet more powerful. If a large set T of terminals is gathered, then a request can be safely removed from the instance. Informally, *gathered* means that a k-edge cut cannot separate two vertices of T from the main part of T in the same connected component. In Section 4.3 and Section 4.4 we work on the structure of the instance given by a large treewidth in order to find such a large gathered set of terminals.

Well-Quasi-Ordering Induced Subgraphs

Minor-closed graph problems are automatically polynomial-time decidable by Corollary 56, as the minor relation is a well-quasi-order. It is natural to consider more restrictive relations, but the subgraph and (hence) induced subgraph relations are not well-quasi-orders. Indeed, the set of cycles is an infinite antichain for these relations. An algorithmic meta-result as powerful as the Graph Minor Theorem will not hold in the setting of induced subgraphs. Even if the induced subgraph relation is a well-quasi-order on a restricted class of graphs S, this does not imply that S is recognizable in polynomial time. Indeed, the obstructions to S lie outside the class S, so there could be infinitely many obstructions nonetheless.

In Chapter 5, we are interested in an attempt towards a weaker meta-result. The following definition is a strengthening of the notion of well-quasi-order. A set S of graphs is n-*well-quasi-ordered* if the set $\hat{S}$, consisting of all vertex n-colourings of graphs in S, is well-quasi-ordered by the coloured induced subgraph relation $\leq_c$. More precisely, $G \leq_c G'$ if there is an injection from $V(G)$ to $V(G')$ preserving adjacency and colour. The notion of 1-well-quasi-order is exactly that of well-quasi-order, and the notion of 2-well-quasi-order is more constrained, as the set of paths is not 2-well-quasi-ordered by the induced subgraph relation. Indeed, the set of paths with their extremities of colour 1 and their internal vertices of colour 2 forms an infinite antichain.

This notion allows an algorithmic meta-result (see Proposition 112):

Proposition 58. *If the induced subgraph relation is a 2-well-quasi-order on a class of graphs S, then S is recognizable in polynomial time.*

In Chapter 5 we try to characterise the 2-well-quasi-ordered classes of graphs, with the following conjecture of Pouzet in mind, which states that using more than two colours would actually be irrelevant:

Conjecture 59. *A class of graphs closed under induced subgraph is 2-well-quasi-ordered if and only if it is n-well-quasi-ordered, where $n \geq 3$.*

We give a partial characterisation (which we conjecture to be complete) of the 2-well-quasi-ordered classes of graphs, using NLC-decomposition, a tree-like decomposition of graphs absolutely similar to clique-decomposition. We use NLC-width and NLC-decomposition for the sake of simplicity, but all results could be stated in terms of clique-width and clique-decomposition. We refer the reader looking for the definition of NLC-width to Chapter 5.

More technically, we will define classes $\mathrm{NLC}_k^{\mathcal{F}}$ to be the restriction of the class of graphs NLC_k, where relabelling functions are exclusively taken from a set $\mathcal{F}$ of functions from $\{1,\dots,k\}$ into $\{1,\dots,k\}$. We characterise the sets of functions $\mathcal{F}$ for which $\mathrm{NLC}_k^{\mathcal{F}}$ is well-quasi-ordered by the induced subgraph relation $\leq_i$. Precisely, these sets $\mathcal{F}$ are those which satisfy that for every $f, g \in \mathcal{F}$, we have $Im(f \circ g) = Im(f)$ or $Im(g \circ f) = Im(g)$. To obtain this, we show that words (or trees) on $\mathcal{F}$ are well-quasi-ordered by a relation slightly more constrained than the usual subword (or subtree) relation.

To characterise the 2-well-quasi-ordered classes of graphs, we conjecture that such a class is always included in some well-quasi-ordered $\mathrm{NLC}_k^{\mathcal{F}}$ for some family $\mathcal{F}$. This would imply Pouzet's conjecture.

Chapter 5 is based on [41], a joint work with Stéphan Thomassé and Michael Rao.

Diamond-free Circle Graphs are Helly Circle

In Chapter 6, we describe a characterisation of a geometric class of graphs, namely the Helly Circle graphs. This chapter follows [38], a joint work with Daniel Gonçalves and Michael Rao.

A *circle graph* is the intersection graph of a set of chords in a circle. Such a circle model has the *Helly property* if every three pairwise intersecting chords intersect in a single point, and a graph is *Helly circle* if it has a circle model with the Helly property.

Not all circle graphs admit a Helly circle model. For example, the *diamond*, the graph obtained from K_4 by deleting an edge, has no Helly circle model. Indeed, the two induced triangles would need to correspond to the same intersection point in the circle model to respect the Helly property. We show that the diamond is essentially the only such counter-example, *i.e.* that the Helly circle graphs are exactly the diamond-free circle graphs, as conjectured by Durán [51]. This characterisation gives an efficient recognition algorithm for Helly circle graphs.

Our proof is constructive. We maintain a circle model of the graph G while growing a subgraph $H \subseteq G$ on which the circle model is Helly. Informally, we grow H by finding a vertex $u \notin H$ such that the circle model of $H \cup \{u\}$ is convex, and we prove that the neighbourhood of such a vertex u possesses nice linear ordering properties. This allows us to define a sequence S of vertices which contains u, and such that we can propagate small endpoint movements over the chords representing the vertices of S. We do not change the circular order of the endpoints around the circle, and we finally obtain a new circle model of G which is Helly on $H \cup S$. Growing the subgraph H layer by layer until $H = G$ completes the proof.

CHAPTER

4 Reducing Multicut to Bounded Treewidth

In this chapter, we show that MULTICUT parameterized by the solution size k can be reduced to graphs of treewidth bounded by a function of k.

The key idea is to establish conditions under which we can identify an *irrelevant* request, *i.e.* a request whose removal yields an equivalent instance. For example, we prove that if a vertex is an endpoint of too many requests, then one of these requests is irrelevant. Hence, we will be able to assume that the request graph of an instance has maximum degree bounded by a function of k. Likewise, if a large *gathered* set of terminals exists (*i.e.* a set of terminals which satisfies some separability properties), then one of these terminals is incident to an irrelevant request. Both cases yield reduction rules as the irrelevant request can be identified in FPT time.

Given an input graph of large treewidth (with respect to the solution size k), we establish a win / win situation. Either one of the reduction rules applies or we can identify an edge whose contraction yields an equivalent instance. Informally, such a useless edge can be found in a zone of the graph sufficiently far away from the terminals and sufficiently connected, so that this edge cannot help disconnect a meaningful part of the graph.

The cases where the input graph has a large grid minor and a large clique minor are treated separately. The proofs of these two cases share some similarities, but we could not unify them.

Therefore we prove that MULTICUT is FPT when parameterized by the solution size k if and only if MULTICUT is FPT when restricted to graphs of treewidth bounded in k. Besides, we give an $O^*((2k+1)^k)$ algorithm for the INTEGER WEIGHTED MULTICUT IN TREES problem. This last problem, which was left open in Chapter 2 and in [80], is one of the simplest sub-cases of the MULTICUT problem on bounded treewidth graphs.

The work in this chapter was completed prior to finding the "emulate requests with 2-

SAT clauses" idea, which eventually led to the self-contained complete proof in Chapter 7. Although results from this chapter will not be explicitly used in Chapter 7, the treewidth reduction exposed below paves the way for the complete proof that MULTICUT is FPT in two ways. Most importantly, the connectivity framework established in this chapter is the core of the complete proof. Less importantly, working on seemingly hard instances of small treewidth gave birth to the starting point of Chapter 7, emulating requests with 2-SAT clauses in simplified instances.

4.1 Preliminaries

Given two vertices $x, y \in V(G)$, *contracting* x and y in G means deleting x and y and adding a new vertex with neighbourhood $N(x) \cup N(y) \setminus \{x,y\}$. When $X \subseteq V(G)$ we denote by $G[X]$ the subgraph of G induced by X.

An (x,y)-*cut* is a set of edges of the graph G whose removal disconnects x and y. We define similarly an (x,Y)-cut and an (X,Y)-cut for two sets X,Y of vertices in $V(G)$. We say that an edge xy *belongs* to a set $X \subseteq V(G)$ if $x, y \in X$. A set of edges F *touches* an induced subgraph G' of G if an edge in F belongs to $V(G')$.

Given a graph J, let us denote by L_i and call the *level* i *of* J *starting from* $L_0 \subseteq V(J)$ the subset of $V(J)$ containing vertices lying at distance exactly i from L_0 in J. The *level decomposition* of J starting with $L_0 \subseteq V(J)$ is the partition of $V(J)$ into levels L_i, $i \geq 0$.

Given an instance (G,R,k) of the parameterized MULTICUT problem, a k-*multicut* is a multicut of size at most k. We say that a k-multicut is *optimal* when its size is minimum among all k-multicuts. We say that a request (s_i,t_i) is *irrelevant* whenever the instance $(G, R \setminus \{(s_i,t_i)\}, k)$ is equivalent to the instance (G,R,k).

We use the following notation with respect to minors. Let $\mathcal{P}$ be a partition of $V(G)$, such that every part h of $\mathcal{P}$ induces a connected subgraph of G. Let H be the graph G quotiented by $\mathcal{P}$, *i.e.* the graph G where each part of $\mathcal{P}$ is contracted into a single vertex. If J is a subgraph of H, then J is said to be a *minor* of G. We say that the pair $(\mathcal{P},H)$ is a J-*model* of G. In a slight abuse of notation, we write that H is a J-*model* of G (or simply a *model* of G), leaving partition $\mathcal{P}$ implicit. We also abusively write that a part h of $\mathcal{P}$ is a *part* of H. We want to emphasize that edges of H are *all* pairs of parts of $\mathcal{P}$ between which G induces an edge. When H' is a subgraph of H, $V(H')$ naturally denotes the set of vertices of H', which are parts, and we write $V_G(H')$ to denote the set of vertices $\bigcup_{h \in H'} x \in h \subseteq V(G)$.

These definitions are quite cumbersome, but we refer the reader to the above paragraph in case of a doubt about the minor-related notations and terminology.

4.2 General Reduction Rules for MULTICUT

Let (G,R,k) be an instance of MULTICUT. This section is devoted to show that the following three reduction rules are correct and can be applied in FPT time.

Rule 1. *If there exist* $k+1$ *edge-disjoint paths between two vertices* x *and* y*, then contract* x *and* y*.*

Ihere exists an integer $\mathtt{req}(k) = k^{O(k)}$ such that the following holds [1].

Claim 60. *If a vertex* t *is an endpoint of at least* $\mathtt{req}(k)$ *requests, then we can find in FPT time an irrelevant request incident to* t*.*

Rule 2. *If a vertex* t *is an endpoint of at least* $\mathtt{req}(k)$ *requests, then we remove the irrelevant request found with Claim 60.*

By Rule 2 we may assume that the degree of the request graph is bounded by $\mathtt{req}(k)$. We say that a set $T \subseteq V$ is *gathered* if for every $F \subseteq E$, $|F| \le k$, there exists at most one connected component in $G\setminus F$ containing more than one vertex of T. We denote this connected component by C_F. Note that a subset of a gathered set is gathered.

Claim 61. *If the instance is reduced under Rule 2 and there exists a gathered set of terminals* T *of size at least* $\mathtt{gath}(k) = 4 \cdot \mathtt{req}(k)^3 = k^{O(k)}$*, then we can find in FPT time an irrelevant request incident to one of these terminals.*

Rule 3. *If the instance is reduced under Rule 2 and there exists a gathered set of terminals* T *of size at least* $\mathtt{gath}(k) = 4 \cdot \mathtt{req}(k)^3 = k^{O(k)}$*, then we remove the irrelevant request found with Claim 61.*

Lemma 62. *Rule 1 is safe and can be applied in polynomial time.*

Proof. If there exist $k+1$ edge-disjoint paths between two vertices x and y, then x and y lie in the same connected component of $G \setminus F$ for every set F of k edges. Hence contracting x and y yields an equivalent instance. Testing whether two vertices are $(k+1)$-edge-connected is polynomial by usual flow techniques. $\square$

To prove the soundness of Rule 2 and Rule 3, we study two edge-connectivity problems of independent interest. We define the central notions of this section:

– A $(zy|x)$-cut is a (z,x)-cut which is not a (z,y)-cut. Similarly, given a subset of vertices T, a $(zy|T)$-cut is a (z,T)-cut which is not a (z,y)-cut.
– A vertex $y \notin T$ is $(z|T)$-k-*linked* if there is no $(zy|T)$-cut of size at most k, *i.e.* if every (z,T)-cut of size at most k is a (z,y)-cut.

We call TRIPLE SEPARATION the problem of finding a $(zy|x)$-cut of size at most k.

1. More precisely, $\mathtt{req}(k) = f(k,k)$, where f is defined at the beginning of the proof of Theorem 66. Some of the bounds proved in this chapter are rather heavy, and we do not wish to write them explicitly in the main statements.

TRIPLE SEPARATION:
Input: A graph $G = (V, E)$, three vertices $x, y, z \in V(G)$ and a positive integer k.
Parameter: k.
Output: A $(zy|x)$-cut of size at most k (if one exists).

Theorem 63. *The* TRIPLE SEPARATION *problem is* FPT *with respect to the solution size* k.

A stronger statement has recently been proved by Marx *et al.* [98]. Their Theorem 3.4 states that deciding whether there exists a set of k vertices separating prescribed pairs and not separating other prescribed pairs is FPT with respect to k and the number of prescribed pairs. We give a different proof of our statement for the sake of completeness.

Proof. Consider an input of the TRIPLE SEPARATION problem, with a graph $G = (V, E)$, three vertices x, y, z and an integer k. Given a set of vertices X, we denote by $\delta_G(X)$, or more simply by $\delta(X)$, the *border* of the set of vertices X, *i.e.* the set of edges having exactly one endpoint in X. Let c be the size of a minimum cut between vertices z and x, and let X be a minimal set of vertices of border of size c containing x. Observe first that if $c > k$ then the answer is NO. So assume that $c \leqslant k$. Notice that a minimal edge-cut separates the graph into exactly 2 connected components. Hence if $y \notin X$ then $\delta(X)$ is a $(zy|x)$-cut and the answer is YES. Otherwise, let G' be the multigraph obtained from G by contracting $V(G) \setminus X$ into a single vertex z', keeping multiple edges.

Claim 64. *There exists a* $(zy|x)$*-cut of size at most* k *in* G *if and only if there exists a* $(z'y|x)$*-cut of size at most* k *in* G'.

Proof. Assume first that there exists a $(z'y|x)$-cut F' of size at most k in G'. Let W be the connected component of $G' \setminus F'$ containing x. We have that $|\delta_G(W)| = |\delta_{G'}(W)|$, hence $\delta_G(W)$ is a $(zy|x)$-cut of size at most k in G.

Conversely, let F be a $(zy|x)$-cut of size at most k in G, and let Z be the connected component of $G \setminus F$ containing z and y. Let $Z' \subseteq V(G')$ be the set $Z \cap X \cup \{z'\}$. The set $\delta_{G'}(Z')$ is a $(z'y|x)$-cut in G'. By definition of G' and Z', we have $|\delta_{G'}(Z')| = |\delta_{G'}(Z \cap X \cup \{z'\})| = |\delta_G(Z \cup \overline{X})|$. By submodularity of the border, we know that $|\delta_G(Z)| + |\delta_G(\overline{X})| \geq |\delta_G(Z \cup \overline{X})| + |\delta_G(Z \cap \overline{X})|$. Thus, $|\delta_{G'}(Z')| \leq |\delta_G(Z)| + |\delta_G(X)| - |\delta_G(Z \cap \overline{X})|$. As $\delta_G(X)$ is minimum zx-cut in G, we know that $|\delta_G(X)| \leq |\delta_G(Z \cap \overline{X})|$, which gives $|\delta_{G'}(Z')| \leq |\delta_G(Z)| \leq k$. ◇

We are now looking for a $(z'y|x)$-cut of size at most k in G'. By minimality of X, we know that $\delta_{G'}(X)$ is the only minimum (z', x)-cut in G' (otherwise we would find a smaller set $X' \subsetneq X$ containing x of border of size c in G). We have to look the a $(zy|x)$-cut of size l with $c + 1 \leq l \leq k$. By definition, such a cut does not contain all edges of $\delta_{G'}(X)$, because otherwise it would be a (z, y)-cut. We thus branch over the c edges adjacent to z', obtaining c new instances where the considered edge has been contracted to a single vertex $\tilde{z}$. Doing so, we strictly increase the connectivity between z' and x by minimality of X. We now have to decide if there exists a $(\tilde{z}y|x)$-cut in a graph where the connectivity between $\tilde{z}$ and x is at least $c+1$. Note that if the contracted edge was $z'y$, we just need to decide whether there exists a cut of size at most k between $\tilde{z}$ and x. Since $c \leq k$ and since the connectivity between $\tilde{z}$ and x strictly increases at each step, the whole branching algorithm runs in time $O(k! \times \mathrm{poly}(n))$. □

We say that a vertex x is k'*-strongly* $(z|T)$*-*k*-linked* if and only if for every $S \subseteq T$ such that $|S| \geq |T| - k'$, the vertex x is $(z|S)$-k-linked (*i.e.* if there is no $(zx|S)$-cut of size at most k). Note that when x is k'-strongly $(z|T')$-k-linked, x is a fortiori k'-strongly $(z|T)$-k-linked when $T' \subseteq T$. In particular, we get the following corollary by using Theorem 63 on every subset $S \subseteq T$ such that $|S| \geq |T| - k'$, contracting the set S into a single vertex:

Corollary 65. *Deciding whether a vertex* x *is* k'*-strongly* $(z|T)$*-*k*-linked and producing a witness of non-strong-linkage,* i.e. *a* $(zx|S)$*-cut for* S *a subset of* T *of size at least* $|T| - k'$*, is FPT in* k*,* k' *and* $|T|$*.*

The following Theorem is a key result of this chapter:

Theorem 66. *Let* $G = (V, E)$ *be a graph, z a vertex of* $V(G)$ *and* T *a subset of* $V(G) \setminus \{z\}$ *of size at least* $f(k, k') = k^{O(k+k')}$*. There exists a vertex* $x \in T$ *which is* k'*-strongly* $(z|T \setminus \{x\})$*-*k*-linked and which can be found in FPT time in* k*,* k' *and* $|T|$*.*

Proof. Assume that T has size at least $f(k, k') = k^{k+k'+1}(\frac{k'+1+k^2}{k-1} + 1) - \frac{k'+1+k^2}{k-1}$.

We initiate our algorithm with the graph $G_0 := G$ and let $T_0 := T$. We repeat the following process $k + k' + 2$ times, selecting some vertex x_i in T_i for $i = 0, \ldots, k + k' + 1$. We first test whether x_i is k'-strongly $(z|T_i \setminus \{x_i\})$-$k$-linked in G_i as in Corollary 65. If this is the case, then we are done. If this is not the case, we obtain a subset S_i of T_i of size at least $|T_i| - k' - 1$ and a $(z|S_i)$-cut C_i of size at most k which is not a (z, x_i)-cut. Denote by L_i the connected component of $G_i \setminus C_i$ containing z, and consider the connected component V_{i+1} of $G_i \setminus C_i$ not containing z and containing the largest number of vertices of S. Let G_{i+1} be the graph obtained from $G[V_{i+1} \cup L_i]$ by contracting $L_i \cup V(C_i)$ into z, where $V(C_i)$ denotes the set of vertices incident to C_i. Note that $V(G_{i+1})$ contains a large subset $T_{i+1} = T_i \cap (V_{i+1} \setminus V(C_i))$ of vertices of T, and that x_i is well defined for every $i = 1, \ldots, k + k' + 1$. Indeed, by definition of S_i, we have that $|T_{i+1}| \geq \frac{|S_i|}{k} - k$, since at most k vertices of S_i belong to $V(C_i)$, and V_{i+1} is chosen among at most k components. As $|S_i| \geq |T_i| - k' - 1$ by definition, we obtain $|T_{i+1}| \geq \frac{|T_i| - k' - 1}{k} - k$. Equivalently, this means that $|T_{i+1}| + \frac{k'+1+k^2}{k-1} \geq \frac{1}{k}(|T_i| + \frac{k'+1+k^2}{k-1})$. Thus the sequence $(|T_i| + \frac{k'+1+k^2}{k-1})$ follows a geometric progression of factor $\frac{1}{k}$. So that $|T_{k+k'+1}| \geq 1$ we need that $|T| + \frac{k'+1+k^2}{k-1} = |T_0| + \frac{k'+1+k^2}{k-1} \geq k^{k+k'+1}(\frac{k'+1+k^2}{k-1} + |T_{k+k'+1}|) \geq k^{k+k'+1}(\frac{k'+1+k^2}{k-1} + 1)$, that is $|T| \geq k^{k+k'+1}(\frac{k'+1+k^2}{k-1} + 1) - \frac{k'+1+k^2}{k-1}$, which is the assumption of Theorem 66.

If the algorithm did not stop after step $k + k' + 1$, this means that for every $i = 1, \ldots, k + k' + 1$, x_i is not k'-strongly $(z|T \setminus \{x_i\})$-k-linked in G_i.

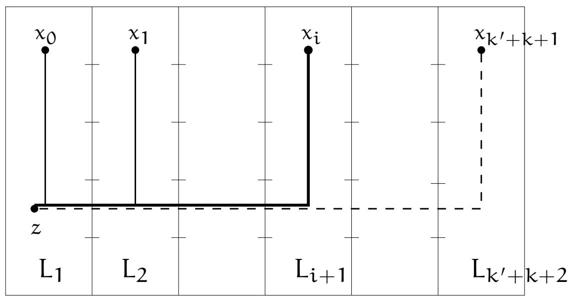


Figure 4.1: On the left, an illustration of the sets L_i, C_i, S_i, V_{i+1} and T_{i+1} in the proof of Theorem 66. On the right, the sets L_i are depicted in the original graph G, and vertex $x_{k'+k+1}$ is k'-strongly $(z|T\setminus\{x\})$-k-linked in G.

Let us denote $T_e := \{x_i : i = 0,\dots,k+k'\}$. We will show that $x_{k+k'+1}$ is k'-strongly $(z|T_e)$-k-linked in G, which implies that $x_{k+k'+1}$ is k'-strongly $(z|T\setminus\{x_{k+k'+1}\})$-k-linked in G. Consider a path P between $x_{k+k'+1}$ and z. Note that $C_i \cap C_j = \emptyset$ for $i \neq j$, which implies that each edge of C_i has one endpoint in L_i and one endpoint in L_{i+1}. For every $i = 0,\dots,k+k'$, P intersects L_i since P intersects C_i. For every $i = 0,\dots,k+k'$, let P_i be a path between x_i and z in G which is included in P outside L_i. Such a path exists because L_{i+1} is connected. This implies that, for every set $S_e \subseteq T_e$ of size at least $k+k'+1-k' = k+1$, and for every path P between $x_{k+k'+1}$ and z, there exist $k+1$ paths P_j for $j = 1,\dots,k+1$ between vertices of S_e and z, such that $P_j \cap P_l \subseteq P$ for $j \neq l$. Every cut of size at most k between z and S_e contains an edge which intersects two distinct paths P_i, P_j, and hence cuts P. This holds for every path P between $x_{k+k'+1}$ and z, so every cut of size at most k between z and S_e separates $x_{k+k'+1}$ and z.

Note that, if the algorithm did stop before step $k+k'+1$, *i.e.* if x_i is k'-strongly $(z|T_i\setminus\{x_i\})$-k-linked in G_i, then x_i is k'-strongly $(z|T\setminus\{x_i\})$-k-linked in G by monotonicity of this linkedness notion and by construction of G_i.

Finally, this algorithm runs in FPT time thanks to Corollary 65. □

Lemma 67. *Claim 60 is correct so Rule 2 can be applied in FPT time.*

Proof. Let t be a vertex which is incident to at least $\mathtt{req}(k) = f(k,k)$ requests, and $\tilde{T}$ be the set consisting of the corresponding second endpoints of these requests. Since $|\tilde{T}| \geqslant f(k,k)$, Theorem 66 implies that there exists a vertex $\tilde{t} \in \tilde{T}$ which is k-strongly $(t|\tilde{T}\setminus\{\tilde{t}\})$-k-linked in G, so $\tilde{t}$ is in particular $(t|\tilde{T}\setminus\{\tilde{t}\})$-k linked in G. Let F be a k-multicut of $(G, R\setminus(t,\tilde{t}), k)$. By definition, F is a $(t,\tilde{T}\setminus\{\tilde{t}\})$-cut in G, and hence a $(t,\tilde{t})$-cut in G as $(t|\tilde{T}\setminus\{\tilde{t}\})$-k-linked. It follows that F is actually a k-multicut for (G,R,k) and thus that the request $(t,\tilde{t})$ is irrelevant. □

Lemma 68. *Claim 61 is correct so Rule 3 can be applied in FPT time.*

Proof. Let T be a gathered set of terminals of size at least $4f(k,k)^3$, and $S = \{s|(s,t) \in R, t \in T\}$. By Rule 2 each vertex $t \in T$ is the endpoint of at most $f(k,k)$ requests. Hence there exist two sets $T' = \{t'_1,\dots,t'_p\} \subseteq T$ and $S' = \{s',\dots,s'_p\} \subseteq S$ with $p = 4f(k,k)^2$ such that (t'_i, s'_i) is a request for every $i = 1,\dots,p$.

We now define an auxiliary bipartite graph $B = (T' \cup S', E')$ where $t'_i s'_j \in E'$ if $i \neq j$ and there exists a cut F of size at most k in G such that t'_i and s'_j lie in a same connected component different from C_F in $G \setminus F$. As T' is gathered, in this case, no other vertex of T' lies in the same component as t'_i and s'_j. For every $i \neq j$ we use Theorem 63 on the graph obtained from G by contracting $T' \setminus \{t'_i\}$ into a single vertex x to test whether there exists a $(t'_i s'_j|x)$-cut of size at most k. If so, then t'_i and s'_j are in the same connected component in $G \setminus F$, which is not C_F since t'_i is cut from any vertex of $T' \setminus \{t'_i\}$, and t'_i and s'_j are adjacent in B. Hence B can be constructed in FPT time in k.

Claim 69. *Every vertex $s' \in S'$ has degree at most $f(k,k)$ in B.*

Proof. Assume that there exists a vertex $s' \in S'$ with degree at least $f(k,k)$. By Theorem 66 there exists a vertex x in $N_B(s')$, the set of neighbours of s' in D, such that x is k-strongly $(s'|N_B(s') \setminus \{x\})$-$k$-linked. Since x and s' are adjacent in B, there exists a set $F \subseteq E(G)$, $|F| \le k$ such that s' and x belongs to a same component different from C_F in $G \setminus F$. Since T is a gathered set, such a cut F must cut x from $N_B(s) \setminus \{x\}$ and is thus a cut between x and s', a contradiction. ◇

Claim 70. *There exists $T'' \subseteq T'$ and $S'' \subseteq S'$ of size $f(k,k)$ such that for every set F of at most k edges, if $t'_i \in T''$ and $s'_j \in S''$ both lie in a component $C \neq C_F$ of $G \setminus F$, then $i = j$.*

Proof. By Claim 69, we have $|E'| \le |S'| \cdot f(k,k)$. Hence there exists a set J of at least $\frac{|T'|}{2} = 2f(k,k)^2$ vertices of T' with degree at most $2f(k,k)$ in B. Let $I = \{s|(t,s) \in R, t \in J\}$. The degree of $B[J \cup I]$ is at most $2f(k,k)$ implying that we can greedily find two sets $T'' = \{t''_1,\dots,t''_{f(k,k)}\} \subseteq J$ and $S'' = \{s''_1,\dots,s''_{f(k,k)}\} \subseteq I$ such that (t''_i, s''_i) is a request for every i and $B[T'' \cup S'']$ does not contain any edge. By construction, this means that for every set F of at most k edges if t''_i and s''_j both lie in a component $C \neq C_F$ of $G \setminus F$ then $i = j$. ◇

By Theorem 66, there exists a vertex $s''_i \in S''$ which is k-strongly $(z|S'' \setminus \{s''\})$-$k$-linked in $\tilde{G}$, where $\tilde{G}$ is the graph obtained from G by contracting T'' to a single vertex z. We conclude the proof of Lemma 68 by showing the following:

Claim 71. *The request (s''_i, t''_i) is irrelevant.*

Proof. Let F be a k-multicut for the instance $(G, R \setminus (t'', s''), k)$. In particular, F is a (T''_F, S''_F)-cut where $T''_F = T'' \cap C_F$ and S''_F denotes the set of terminals corresponding to $T'' \cap C_F$. Observe that F is a (z, S''_F)-cut in $\tilde{G}$: otherwise this would imply that there exists a vertex of $T'' \setminus T''_F$ lying in a component $C \neq C_F$ of $G \setminus F$ which also contains a vertex of S''_F, which cannot be by Claim 70. Hence F is a (s'', z)-cut in $\tilde{G}$ and thus a (s'', t'')-cut in G, implying that F is actually a k-multicut for (G, R, k). ◇

This concludes the proof of Lemma 68. □

The rest of this chapter is essentially devoted to finding a situation where Rule 3 can be applied.

4.3 Clique Minor

In this section, we assume that G has a large clique minor, more precisely a $CM(k)$-clique minor, with $CM(k) = O(k^{k^{O(k)}})$ [2]. Let H be a $CM(k)$-clique model of G. Such a model can be computed in FPT time in k. Let a be a vertex of G lying in a part h_a of H. The vertex a is *special* if there exist $k+1$ vertex-disjoint paths internally in h_a between a and distinct parts of H different from h_a.

4.3.1 Finding a Nice Model

We say that a model H of a graph G is *nice* whenever it satisfies the following properties:

(1) $|V(H)| \geq CM(k)$ and H is $(k+1)$-vertex connected.

(2) There exists at most one special vertex a in G (and we denote its part by h_a).

(3) All parts but possibly h_a have degree at most $(k+1)^2$ in H.

(4) The special vertex a separates $h_a \setminus \{a\}$ from $V(G) \setminus h_a$ in G.

Theorem 72. *Let* (G, R, k) *be an instance of* MULTICUT *reduced under Rule 1 which admits a* $CM(k)$*-clique model* H*. There exists a nice model of* G *which can be computed in FPT time in* k*.*

Proof. Let us first show that H respects the second property of the nice model definition. Observe that any $CM(k)$-clique model for G is in particular $(k+2)$-vertex connected since $CM(k) \geqslant k+2$.

Claim 73. *Let* (G, R, k) *be an instance of* MULTICUT *reduced under Rule 1 which admits a* $(k+1)$*-vertex connected model* H*. Then* G *has at most one special vertex.*

Proof. Assume by contradiction that there exist two vertices a and a' having at least $k+1$ vertex-disjoint paths $P_1, \ldots, P_{k+1}$ and $P'_1, \ldots, P'_{k+1}$ in their own parts h_a and $h_{a'}$ towards distinct parts different from h_a (resp. $h_{a'}$) in H. Since G is reduced under Rule 1, there are at most k vertex-disjoint paths between a and a' in G. Hence by Menger's theorem [101] there exists a set S of at most k vertices of G whose removal disconnects a and a'. Let $\mathcal{S}$ be the set of parts of H containing vertices of S. By hypothesis, there exists at least

2. More precisely, $CM(k) = ((k+1)^2)^{(2(k+2)+1)\cdot(gath(k)+1)}$ (where $gath(k)$ is the size of a gathered set reduced by Rule 3).

one path P_i towards a neighbour h_i of h_a and one path P'_i towards a neighbour h'_i of h'_a such that $P_i \cup h_i$ and $P'_i \cup h'_i$ are not intersected by $\mathcal{S}$. By $(k+1)$-vertex connectivity of H there exists a path between h_i and h'_i in $H \setminus \mathcal{S}$. Since every part of $H \setminus \mathcal{S}$ is connected, such a path can be extended to a path between a and a' in $G \setminus S$, leading to a contradiction. $\diamond$

We now show a technical result:

Lemma 74. *Let* $G = (V, E)$ *be a graph which admits a* p*-vertex connected model* H *with* $p > k$ *and assume that* G *contains at most one special vertex* a*. If there is a part* $h \neq h_a$ *with degree greater than* $(p-1)(k+1)$ *in* H*, then there exists a bipartition* h_1, h_2 *of* h *such that* h_i *is connected in* G *for* $i = 1, 2$ *and the partition* $\mathcal{P} \setminus h \cup \{h_1, h_2\}$ *of* $V(G)$ *is a* p*-vertex connected model of* G*, where* $\mathcal{P}$ *is the partition of* $V(G)$ *associated to* H.

Proof. We need the two following claims:

Claim 75. *Let* T *be a tree of maximum degree* d*, with a weight function* ω *from the nodes of* T *into* $\{0, \ldots, M\}$*, such that* $\omega(T) > q(d+1)$ *with* $q \geq M$*. There exists an edge of* T *separating* T *into two subtrees* T_1 *and* T_2 *such that* $\omega(T_i) \geqslant q+1$ *for* $i = 1, 2$.

Proof. Assume that T contradicts the lemma. For each edge e in T, let T_1^e and T_2^e be the connected components of $T \setminus e$. We have $\omega(T_1^e) \leq q$ or $\omega(T_2^e) \leq q$ (and these cases are mutually exclusive). Let us orient T: e gets the orientation $T_1^e \rightarrow T_2^e$ if $\omega(T_1^e) \leq q$. Note that an edge uv gets the orientation $u \rightarrow v$ whenever u is a leaf. Since this orientation is acyclic, there exists an internal node x of T such that all edges incident to x are oriented towards x. Let $T_1^x, \ldots, T_l^x$ be the connected components of $T \setminus x$. We have $l \leq d$. Thus $w(T) = \omega(T_1^x) + \ldots + \omega(T_l^x) + \omega(x) \leq l \cdot q + M \leq q(d+1)$, a contradiction. This concludes the proof of Claim 75. $\diamond$

Claim 76. *Let* H *be a* c*-vertex-connected graph and* $h \in V(H)$ *be a vertex of degree at least* $2c$*. Let* N_1, N_2 *be a bipartition of* $N(h)$*, with* $|N_i| \geq c$ *for* $i = 1, 2$*. Let* H' *be the graph obtained from* H *by deleting* h *and adding two vertices* h_1 *and* h_2*, of respective neighbourhoods* $N_1 \cup \{h_2\}$ *and* $N_2 \cup \{h_1\}$*. Then* H' *is* c*-connected.*

Proof. Assume that there exists a set S of $c-1$ vertices whose removal disconnects H'. If $h_i \notin S$ for $i = 1, 2$, then S disconnects H, a contradiction. If h_1 and h_2 both lie in S, then $(S \setminus \{h_1, h_2\}) \cup \{h\}$ disconnects H, a contradiction. If h_i lies in S, for $i = 1$ or $i = 2$, then $\{h_{3-i}\}$ is not a connected component of $H \setminus S$ since it has at least c neighbours besides h_i in H'. Thus $S \setminus \{h_i\} \cup \{h\}$ is a $(c-1)$-cut in H, a contradiction. This concludes the proof of Claim 76. $\diamond$

We use these two results to prove Lemma 74. Consider a part $h \neq h_a$ of degree greater than $(p-1)(k+1)$ in H. For each part h' adjacent to h in H, we distinguish one vertex $x \in h$ such that there exists an edge xx' in G with $x' \in h'$. Denote this vertex x by $v(h')$. Let T

be a tree of $G[h]$ spanning all vertices x such that $\nu^{-1}(x) \neq \emptyset$, with T minimal by inclusion. The leaves of T are exactly the vertices such that $\nu^{-1}(x) \neq \emptyset$. Let ω be a weight function on h, such that $\omega(x) = |\nu^{-1}(x)|$.

Since G contains at most one special vertex, vertices distinct from a can have at most k neighbours in distinct other parts, implying that $\omega(x) \leqslant k$ for every vertex $x \neq a$. Moreover, the degree of any vertex in T is at most k, since a vertex with degree at least $k+1$ would have $k+1$ vertex-disjoint paths towards leaves of T and hence towards $k+1$ distinct parts different from its own part, which cannot be since a is the only special vertex of G. By construction, $\omega(T)$ is equal to the degree of h in H, so $\omega(T) > (p-1)(k+1)$. By Claim 75 applied with $q = p-1$ and $d = k$, there exists a bipartition T_1, T_2 of T such that $\omega(T_i) \geqslant p$. Observe that this bipartition can be extended into a bipartition of h into two connected sets h_1, h_2 such that $|N_H(h_1)|, |N_H(h_2)| \geqslant p$, by definition of the weight function ω. By Claim 76 it follows that the partition $\mathcal{P} \setminus h \cup \{h_1, h_2\}$ defines a model H' which is p-connected. This concludes the proof of Lemma 74. □

Assume now that H does not contain a special vertex, and let h be a part of H with degree at least $(k+1)^2+1$ in H. Since H is $(k+2)$-vertex connected, it follows by Lemma 74 applied with $p = k+2$ that we can find in polynomial time a new model of G which is $(k+2)$-vertex connected, which contains at most one special vertex by Claim 73, and with size strictly greater than $V(H)$. We apply repeatedly Lemma 74 until either no part has degree at least $(k+1)^2+1$ in H_i or there exists a special vertex a in a part h_a. In the former case, since the model finally obtained is $(k+2)$-vertex connected and does not contain any special vertex, it is actually a nice model, thus we are done. In the latter case, we modify the finally obtained model to obtain a new model H' such that the special vertex a cuts $h_a \setminus \{a\}$ from $V(G) \setminus h_a$ in G as follows.

For every vertex $v \neq a \in h_a$ having a neighbour in a part $h_v \neq h_a$, we denote by A_v the set of vertices disconnected from a in h_a by the removal of v. We remove the set $A_v \cup \{v\}$ from h_a and add it to h_v, repeating this process until no vertex in h_a but a is adjacent to other parts. Observe that the model H' thus obtained may no longer be $(k+2)$-vertex connected after this process: however, $H' \setminus h_a$ remains $(k+1)$-vertex connected. Since h_a has degree at least $k+1$ in H_i, it follows that H' is a $(k+1)$-vertex connected model. Observe that H' contains exactly one special vertex a by Claim 73.

Since this process may increase the degree of some parts different from h_a in H', we apply repeatedly Lemma 74 to H' to reduce its degree while preserving $(k+1)$-vertex connectivity. Once this process is over, we obtain a $(k+1)$-vertex connected model H' such that $|V(H')| \geqslant CM(k)$, whose special vertex a separates $h_a \setminus \{a\}$ from $V(G) \setminus h_a$ in G, and such that every part but possibly h_a has degree at most $(k+1)^2$. It follows that H' is a nice model, which concludes the proof of Theorem 72. □

4.3.2 Small and Giant Components

In the following we consider a nice model H of the instance (G, R, k) of the MULTICUT problem.

Lemma 77. *Let H be a nice model of G. Two parts not touched by a set $F \subseteq E(G)$ of at most k edges are included in the same connected component of $G \setminus F$.*

Proof. Assume that a set F of at most k edges does not touch parts h_1 and h_2. It follows that part h_1 (resp h_2) is completely included in a connected component of $G \setminus F$. By $(k+1)$-connectivity in H, there are $k+1$ disjoint paths between h_1 and h_2 in H, which cannot all be cut by a set of at most k edges. □

Corollary 78. *Let H be a nice model of G, h_1, h_2 be two parts of H and (t_1, t_2) a request with $t_1 \in h_1$ and $t_2 \in h_2$. Every k-multicut contains an edge that belongs to h_1 or an edge that belongs to h_2.*

Observe that any set of at most k edges cuts the graph G into at most $k+1$ connected components. Given a set F of edges, we call *giant component* of $G \setminus F$ and denote by C_F a connected component of $G \setminus F$ containing entirely at least $|V(H)| - k$ parts.

Lemma 79. *Let F be a set of at most k edges of a graph G admitting a nice model H. Then $G \setminus F$ has a giant component, which contains all parts not touched by F.*

Proof. Let h_1 and h_2 be two parts not touched by a set F of at most k edges. Then h_1 and h_2 belong to the same connected component in $G \setminus F$ by Lemma 77. Since there are at most k parts touched by F, all other parts are entirely included in the same connected component of $G \setminus F$. □

The connected components distinct from the giant component are called *small components.* Altogether, they intersect vertex-wise at most k parts.

Lemma 80. *Assume that G admits a nice model. For every set F of at most k edges, the special vertex belongs to the giant component of $G \setminus F$.*

Proof. Since a is adjacent to at least $k+1$ parts, a is adjacent to at least one vertex lying in a part not touched by F in $G \setminus F$. By Lemma 79, a cannot belong to a small component. □

Lemma 81. *Let F be an optimal k-multicut of a graph G admitting a nice model. Every small component C of $G \setminus F$ contains a vertex which is a terminal.*

Proof. Let F be an optimal k-multicut of G and e be any edge of F which touches C. If C contains no terminal then $F \setminus \{e\}$ is still a multicut, contradicting the optimality of F. □

Let $H_a = H \setminus h_a$. Adjacent vertices x and y which are far away in H_a from any terminal can be contracted as stated in the following Lemma:

Lemma 82. *Let* G *be a graph admitting a nice model* H. *Let* h *be a part of* H_a *not containing any terminal, such that every part at distance at most* $k+2$ *from* h *in* H_a *has no terminal. Let* e *be an edge of* G *with at least one endpoint in* h. *Then* e *does not belong to any optimal* k*-multicut.*

Proof. Consider an optimal k-multicut F of G, and let h' be a part adjacent to h in H. Every small component contains a terminal by Lemma 81 and intersects at most k parts by definition. Since part h (resp. h') is assumed to be too far from any terminal in H_a, part h (resp. h') can intersect a small component C of $G \setminus F$ only if $a \in C$, which contradicts Lemma 80. Hence neither part h nor part h' can intersect a small component. It follows that if e belongs to F then $F \setminus \{e\}$ is also a k-multicut, which contradicts the optimality of F. □

Hence the following reduction rule is correct:

Rule 4. *Let* G *be a graph admitting a nice model* H *and* h *be a part of* H_a *without any terminal. If every part at distance at most* $k+2$ *from* h *in* H_a *has no terminal, then contract an arbitrary edge* e *of* G *with at least one endpoint in* h.

When this reduction rule does no longer apply, every part is at distance at most $k+2$ in H_a from a part containing a terminal.

4.3.3 Reducing the Instance

Let us review the structure of the graph when none of Rule 1, Rule 2 and Rule 4 applies. Recall that we consider an instance (G, R, k) which contains a nice model H.

We consider the level decomposition of H_a starting from some part L_0 of H_a. Since H is a nice model, it follows that every part of H_a has degree at most $(k+1)^2$ in H_a. Hence every level L_i has size at most $(k+1)^2|L_{i-1}|$. Denote by d the number of non-empty levels in this decomposition. We have $CM(k) \leq |V(H_a)| + 1 \leq (\sum_{i=0}^{d}((k+1)^2)^i) + 1 \leq (k+1)^{2(d+1)}$, so $d \geq \log_{(k+1)^2}(CM(k) - 1) - 1$.

Let us show that we can find a large enough gathered set to reduce the instance with Rule 3. Let T be a set of maximum size of terminals in $V_G(H_a)$ belonging to different parts lying pairwise at distance at least $k+1$ in H_a. By Rule 4, every part is at distance at most $k+2$ from a terminal in H_a, so there exists at least one terminal in every $2(k+2)+1$ consecutive levels, implying that $|T| \geq \frac{d}{2(k+2)+1} \geq gath(k)$ as $CM(k) = ((k+1)^2)^{(2(k+2)+1)\cdot(gath(k)+1)}$.

Let us argue that T is a gathered set. Indeed, consider a set F of at most k edges. Since parts containing terminals of T are pairwise at distance at least $k+1$ in H_a, a small component of $G \setminus F$ can contain two terminals of T only if it contains the special vertex a (Lemma 79). Since a belongs to the giant component by Lemma 80, it follows that $G \setminus F$ contains at most one component containing more than one vertex of T (namely the giant component), and hence T is a gathered set. Rule 3 applies to T, so an irrelevant request can be found in FPT time.

We have shown in this section that whenever G has a large clique model, we can reduce the instance in FPT time either by safely contracting an edge or by finding an irrelevant request. This concludes the large clique minor part.

4.4 Grid Minor

In the previous section, we have shown that every graph with a large clique minor can be reduced in FPT time.

The aim of this section is to complete the treewidth reduction by showing that a graph with a large grid minor but no large clique minor can also be reduced in FPT time. Indeed, every graph with treewidth at least 2^{2m^5} contains an $(m \times m)$-grid minor [115], and a model for such a grid minor can be computed in FPT time in m.

Given a grid model $H = (V(H), E(H))$ of a graph G we denote by $\overline{H}$ a graph $(V(H), E(\overline{H}))$ where $E(\overline{H}) \subseteq E(H)$ is such that $\overline{H}$ is a grid.

We call the set of vertices lying on the outer-face in the unique planar drawing of a grid H' the *border* of the grid, and denote it by $B(H')$.

4.4.1 On Grid Minors without Clique Minors

In this section, we concentrate on structural properties of graphs with large grid minors but no large clique minors. Let G_k be the graph obtained from the $(k \times k)$-grid by adding the two diagonals to each internal face of the grid (see Figure 4.2). The proof of the following result is due to Gwenaël Joret.

Lemma 83. *The crossed grid* G_{2k} *has a* k*-clique minor.*

Proof. We are looking for a k-clique model inside G_{2k}, *i.e.* for k vertex-disjoint connected sets of vertices of G_{2k} such that any two of these sets are adjacent in G_{2k}. Roughly speaking, these subsets will be the union of the ith column and ith row of G_{2k} for odd i. These sets are not vertex disjoint, but this can easily be dealt with: for any crossing of two sets, we use two diagonals inside a face of the original grid to uncross them while preserving connectivity.

More formally, denote by (i, j) the vertex lying on row i and column j of the crossed grid G_{2k}. Let S_i be the set of vertices defined as follows, where $1 \leqslant i + l \leqslant 2k$:

$$S_i := \begin{cases} (i, i+l) \text{ and } (i+l, i) \text{ for } \textit{even } l,\ l \geqslant 0 \\ (i+1, i+l) \text{ and } (i+l, i+1) \text{ for } \textit{even } l,\ l < 0 \\ (i+1, i+l) \text{ and } (i+l, i+1) \text{ for odd } l,\ l \geqslant 0 \\ (i, i+l) \text{ and } (i+l, i) \text{ for odd } l,\ l < 0 \end{cases}$$

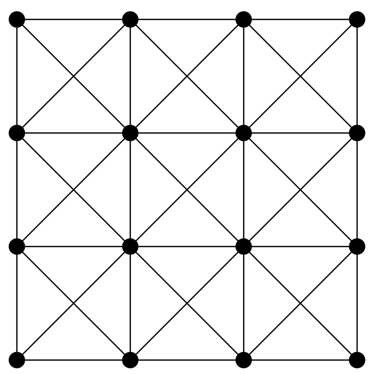
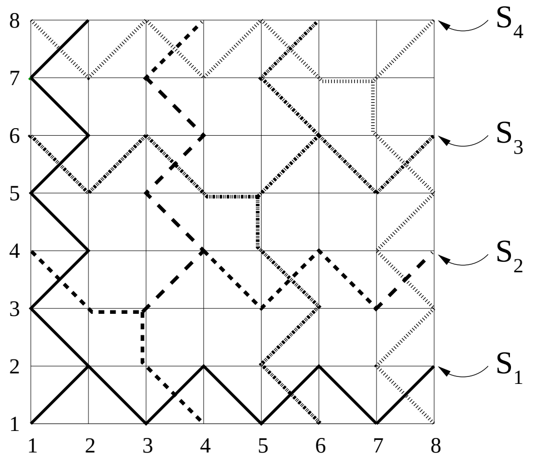



Figure 4.2: On the left, the crossed grid G_4. On the right, a clique model in the crossed grid G_8

The sets S_i for odd i are a model of a K_k in G_{2k}. Indeed they are disjoint, and they are pairwise adjacent since for every odd i, j, with $i < j$, we have $(i, j) \in S_i$ and $(i+1, j) \in S_j$ (see Figure 4.2). $\square$

Given a grid G, a non-edge $e \notin E(G)$ is said to be *non-planar* if the graph $G \cup \{e\}$ is not planar. We call a planar supergraph of a grid an *augmented grid.* We define the border of an augmented grid similarly to the border of a grid.

We now show that a grid together with many non-planar edges has a large clique minor. In [113], Robertson and Seymour give a result (7.4) similar to what we need. To reformulate it in a form closer to our terminology:

Lemma 84. *[113] For every integer m, there exists an integer $r(m)$ such that a graph obtained from an augmented grid H by adding $\binom{m}{2}$ non-planar edges in $(V(H) \times V(H)) \setminus E(H)$, whose endpoints lie at distance at least $r(m)$ from $B(H)$ in H and are pairwise lying at distance at least $r(m)$ in H, has a K_m minor.*

Note that result (7.4) from [113] concerns walls rather than grids and is stated with a different distance function, but Lemma 84 can be deduced from there.

It is important that the extra edges are non-planar in Lemma 84, and that they lie far away from the border of the grid. The fact that the extra edges are long is actually not necessary, as stated in the following extension of Lemma 84.

Lemma 85. *For every integer m, there exists two integers $r(m)$ and $\mathrm{jump}(m) = 2^{O(r(m))}$ such that a graph obtained from an augmented grid H by adding $\mathrm{jump}(m)$ non-planar*

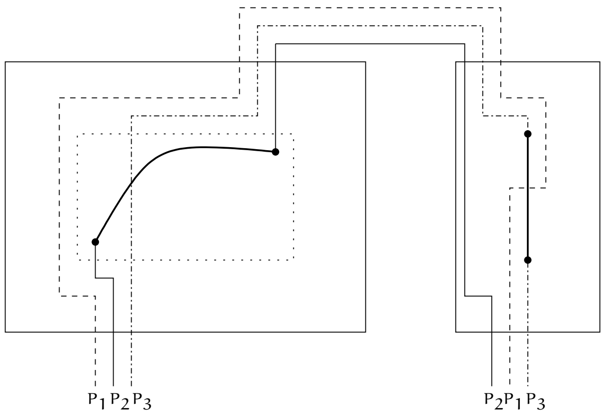


Figure 4.3: Illustration of the swaps made within two consecutive zones in the proof of Lemma 85.

endpoint disjoint edges in $(V(H) \times V(H)) \setminus E(H)$*, whose endpoints lie at distance in* H *at least* $r(m)$ *from the border* $B(H)$*, has a* K_m *minor.*

Proof. Let $\mathrm{jump}(m) = 2 \cdot 8^{2(r(m)+m)\binom{m-1}{2}} + 2 \cdot 8^{r(m)\binom{m}{2}}$. If $2 \cdot 8^{r(m)\binom{m}{2}}$ non-planar edges have length at least m in the grid, then there exists a subset of size $\binom{m}{2}$ of these non-planar edges whose endpoints lie pairwise at distance at least $r(m)$ in H. Indeed a vertex of an augmented subgrid has degree at most 8. By Lemma 84, it follows that H has a clique minor of size m. Otherwise, there exists a subset X of at least $\mathrm{jump}(m) - 2 \cdot 8^{r(m)\binom{m}{2}} \geq 2 \cdot 8^{2(r(m)+m)\binom{m-1}{2}}$ non-planar edges of length at most $r(m)$. Hence there exists a subset $S \subseteq X$ of non-planar edges whose endpoints lie pairwise at distance at least $2(r(m)+m)$ of size at least $\binom{m-1}{2}$. For every edge $e = xy \in S$, let S_e be the minimal subgrid containing both x and y. The *zone* Z_e of e is the subgrid obtained from S_e by adding m layers around S_e. The *layer* around a subgrid S of an augmented grid is the set of vertices outside S adjacent to the a vertex of the border of S.

Note that for any two edges $e, e' \in S$ we have $Z_e \cap Z_{e'} = \emptyset$. There exists a set of m vertex disjoint paths $P_1, \ldots, P_m$ which are parallel and adjacent outside the zones and enter all zones in the grid exactly once.

A zone can be used to swap two adjacent paths, as shown in Figure 4.3. We use the following swap strategy:

- use the $(m-1)$ first encountered zones to place P_m before P_1
- use the $(m-2)$ next zones to place P_{m-1} before P_1
- ...
- use the $(m-i)$ next zones to place P_{m-i} before P_1
- ...
- use the next zone to place P_2 before P_1

We have $\frac{m(m-1)}{2}$ zones, enough to do the $\sum_{i=1}^{m-1} i = \frac{m(m-1)}{2}$ above swaps, and a given zone is large enough to make two adjacent paths cross within it. We have reversed the original planar left-to-right order of paths P'_i, $i \in \{1, \ldots, m\}$ by making consecutive swaps, hence any two paths have crossed and hence are adjacent in the grid. Thus paths P_i, $i \in \{1, \ldots, m\}$ form a K_m minor model. □

Given a grid model H we say that $H' \subseteq H$ is a *proper subgrid* of H if it satisfies the following properties:

- $|V(H')| \geq prop(k) = 2 \cdot r(CM(k))$.
- H' is an induced augmented grid.
- There are at most $spec(k)^2$ vertices in $V(G) \setminus V(H')$ with neighbours in $V(H') \setminus B(H')$, where $spec(k) = jump(k) \cdot (deg(k) + 1) + 1$ and $deg(k) = k(k+2) \cdot jump(CM(k)+1) \cdot jump(CM(k)) + 6$. Such vertices are called *special vertices* and are denoted by W.
- Each special vertex has at least $k^2/4$ neighbours in $V(H') \setminus B(H')$.

Theorem 86. *Let* G *be a graph admitting a* $(GM(k) \times GM(k))$*-grid model* H*, with* $GM(k) = prop(k) \cdot spec(k)^{spec(k)}$*, but no* $K_{CM(k)}$ *minor. There exists a proper subgrid of* H *which can be computed in polynomial time.*

Proof. We first show a technical result which will be used in the proof of Theorem 86. Consider a grid model H of a graph G and an induced augmented grid H' which lies at distance at least $r(CM(k))$ from $B(\overline{H})$ in $\overline{H}$. An edge wz, where $z \in V_G(H') \setminus V_G(B(H'))$ lies at distance at least $r(CM(k))$ of the border of H' in H' and $w \in V_G(H \setminus H')$, is called an H'*-matched edge.*

Lemma 87. *Let* G *be a graph admitting a grid model* H *with an induced augmented grid* H' *of size at least* $prop(k)$ *which lies at distance at least* $r(CM(k)+1)$ *in* $\overline{H}$ *from the border of* $\overline{H}$*. If there exist at least* $spec(k)$ *endpoint disjoint* H'-matched edges *in* H*, then one of the following holds:*

(1) either G *admits a grid model* H_1 *and an induced augmented grid* $H'_1 \subseteq H_1$ *such that* $|H'_1| \geqslant prop(k)$ *and every set of endpoint disjoint* H'_1*-matched edges has size at most* $spec(k)$*,*

(2) or G *has a* $CM(k)$ *clique minor.*

Proof. Recall that $spec(k) = jump(k) \cdot (deg(k) + 1) + 1$. The following result follows by definition of a minor:

Observation 88. *Let* G *be a graph admitting a grid model* H *with an induced augmented grid* H' *such that there exist* p *paths internally in* $G \setminus V_G(H')$ *which are vertex-disjoint, between pairs of vertices* $z_i, z'_i \in V_G(H')$ *at distance at least* $r(CM(k))$ *from the border of* H' *in* H'*,* $1 \leq i \leq p$*. If edges* $z_i z'_i$ *are non-planar in* H' *for* $i = 1, \ldots, p$*, then* G *has as a minor a* *grid* $\check{H}$ *with* p *non-planar edges whose endpoints lie at distance at least* $r(CM(k))$ *from the border of* $\check{H}$ *in* $\check{H}$.

Let $W = \{w_1, \ldots, w_p\}$ be the endpoints of the H'-matched edges which belong to $V_G(H \setminus H')$, and assume that $|W| \geqslant spec(k)$. Since the graph $G \setminus V_G(H')$ is connected, let T be a tree spanning W in $G \setminus V_G(H')$. We can assume that this tree where parts have been contracted is a path, since $H \setminus V_G(H')$ has a Hamiltonian path. Moreover, we can assume that any vertex of a given part has at most one neighbour in T in each of its two adjacent parts. Indeed, assume that a vertex v in a part h has more than one neighbour in T in one of its adjacent part h'. We choose some vertex $u \in N_{h'}(v)$ and remove all edges between v and $N_{h'}(v) \setminus u$ from T. We now use the connectivity of h' to find a spanning tree of all vertices in $N_{h'}(v)$, which gives a new spanning tree T' having the claimed property. Finally, we choose such a tree T with minimum maximum degree. We again distinguish two cases. Recall that $deg(k) = k(k+2) \cdot jump(CM(k)+1) \cdot jump(CM(k)) + 6$.

Case 1: T **has degree bounded by** $deg(k)$**.**

We use the following Lemma to exhibit a large clique minor in G:

Lemma 89. *Let* T *be a tree of degree at most* d*. Let* W *be a set of nodes of* T*. There exist* $\frac{|W|-1}{d+1}$ *vertex-disjoint paths in* T *with distinct endpoints in* W*.*

Proof. We root T arbitrarily. The hypothesis trivially holds for $|W| = 1$, so assume that $|W| \geq 2$. Among all pairs of vertices in W, we choose one pair (x, y) whose least common ancestor u is minimum for the ancestor relation. Let T_u be the subtree of T rooted at u. As u is minimum, each subtree rooted in a son of u contains at most one vertex of W. Hence, as u has degree at most d, T_u contains at most $d+1$ vertices of W. By induction on $T \setminus T_u$, there exist $\frac{|W|-(d+1)-1}{d+1} = \frac{|W|-1}{d+1} - 1$ vertex-disjoint paths in $T \setminus T_u$ with distinct endpoints in W. We add to these paths the disjoint path $(x, y) \subseteq T_u$ to obtain the desired bound. □

Applying Lemma 89 to T and W, we find $\frac{spec(k)-1}{deg(k)+1} \geq jump(CM(k)))$ vertex-disjoint paths between distinct vertices $w_{i,j}$. Using Lemma 88, G has as a minor a grid with $jump(CM(k))$ endpoint disjoint non planar edges with endpoints at distance at least $r(CM(k))$ from the border of H'. Using Lemma 85, we deduce that G has a $K_{CM(k)}$ minor.

Case 2: T **has a node** u **of degree more than** $deg(k)$**.**

Assume that u has maximum degree in T. As T where each part has been contracted to a vertex is a path, u has at least $deg(k) - 2$ neighbours in T lying in its own part h_u. Among these vertices, we distinguish at most 4 of them, which are vertices connecting h_u to its four adjacent parts in $\overline{H}$. As T has minimum maximum degree, there must exist a set $W_u \subseteq W \cap h_u$ of size at least $deg(k) - 6$ such that there exist vertex disjoint paths between u and vertices in W_u. For any $w = w_i \in W_u$, we use z_w to denote the corresponding vertex

z_i, Z_w to denote the part of z_w, and let A_w be the set of vertices disconnected from u in T by the deletion of w. Note that $A_w \cup W_u = \emptyset$ since u has vertex disjoint paths to vertices in W_u. Let us denote by T_w the set of neighbours of $A_w \cup \{w\}$ distinct from z_w in H'.

Assume that for some vertex $w \in W_u$, there does not exist any part b_w containing a vertex in T_w such that $b_w Z_w$ would be a non-planar edge in H'. We change the partition H by removing $A_w \cup \{w\}$ from part h_u and adding it to part Z_w.

The resulting partition is still a grid model of G since all parts are still connected sets in G and the four distinguished vertices connecting h_u to its four adjacent parts in $\overline{H}$ are still in h_u. This also implies that H' still lies at distance at least $r(CM(k))$ from the border of $\overline{H}$ in $\overline{H}$. Moreover, H' is still an induced augmented grid of H since no non-planar edge has been created in H'. We go back to the beginning of the proof of Lemma 87 with this new grid model H and augmented subgrid H' of G. We cannot loop more than n times as the number of vertices of G contained in $V_G(H')$ strictly increases. When this process terminates, if we did not find either a clique minor of G or a grid model H with an induced augmented grid H' containing at most $spec(k)$ endpoint disjoint matched-edges towards vertices in $V_G(H \setminus H')$, then we are still in Case 2. Thus, for every vertex $w \in W_u$ there exists a vertex in $A_w \cup \{w\}$ with a neighbour z'_w in a part $Z'_w \in V(H')$ such that $Z_w Z'_w$ would be a non-planar edge in H'. Hence there exist paths P_w between z_w and z'_w for $w \in W_u$ which are vertex disjoint in h_u, as $A_w \cap A_{w'} = \emptyset$ for $w \neq w' \in W_u$.

Let $Z = \{Z_w | w \in W_u\}$ and $Z' = \{Z'_w | w \in W_u\}$. Note that by construction $Z_w \neq Z_{w'}$ for $w \neq w' \in W_u$. Consider the bipartite graph $B = (Z' \cup W_u, E)$, where $E := \{Z'_w w | w \in W_u\}$.

Claim 90. *Every part in Z' has degree at most* $\mathrm{bipdeg}(k) = k(k+2) \cdot \mathrm{jump}(CM(k)+1)$ *in* B.

Proof. Assume by contradiction that there exists a part h in Z' of degree at least $\mathrm{bipdeg}(k)$. Observe that a vertex $x \in h$ cannot have $k+1$ distinct neighbours in W_u, as there would exist $k+1$ edge-disjoint paths between u and x, which contradicts the fact that the instance is reduced under Rule 1. Let $Z'' = \{z'_w | w \in W_u\} \cap h$. Since h is connected, consider a tree T_h of minimum maximum degree spanning Z'' in h.

Assume first that T_h has minimum degree bounded by $k+1$. We apply Lemma 89 to Z'' and T_h to find $\frac{\mathrm{bipdeg}(k)}{k \cdot (k+1+1)} = \mathrm{jump}(CM(k)+1)$ vertex disjoint paths in T_h between vertices in Z''. This gives $\mathrm{jump}(CM(k)+1)$ vertex disjoint paths in $G' := (G \setminus V_G(H') \cup \{h\}$ between vertices in Z (see Figure 4.4). Consider $H'' := (H' \setminus \{h\}) \cup h'$, where h' is a new part having the same adjacency with H' than h. As H'' defines an induced augmented grid, we can apply Lemma 88 to H'' and the above $\mathrm{jump}(CM(k)+1)$ vertex-disjoint paths of G' to find a grid with $\mathrm{jump}(CM(k)+1)$ non-planar endpoint disjoint edges with endpoints at distance at least $r(CM(k)+1)$ from the border of H'. By Lemma 85, the graph H'' has a $K_{CM(k)+1}$-minor, implying that G has a $K_{CM(k)}$ minor.

Assume now that T_h has a vertex u_h of degree at least $k+1$. Assume that u_h has maximum degree in T_h. In particular, this means that there exists $k+1$ vertices of Z' with vertex

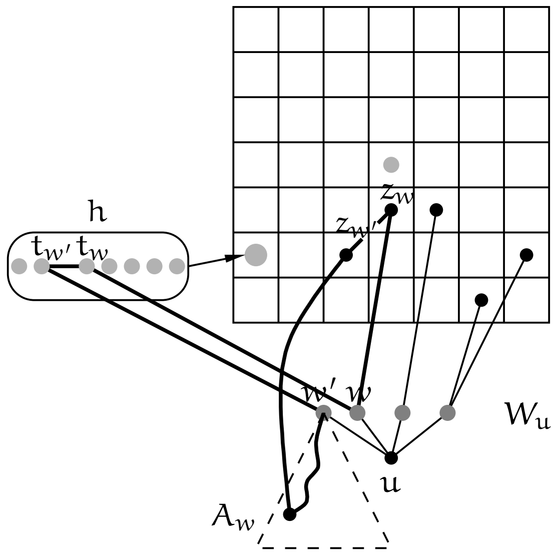


Figure 4.4: An illustration of the vertex disjoint paths found in the proof of Claim 90.

disjoint paths to u_h in T_h. As we have $\frac{\mathrm{bipdeg}(k)}{k} \geq k+1$, there exist $k+1$ edge-disjoint paths between u and u_h, which contradicts the fact that the instance is reduced under Rule 1.

This completes the proof of Claim 90. ◇

By Claim 90 we can greedily find $\frac{|W_u|}{\mathrm{bipdeg}(k)}$ vertex-disjoint paths in $V(G) \setminus H'$ between distinct vertices of H' of the form $z_i w_i z_i'$. By Observation 88, G admits as a minor a grid $\overline{H}$ with at least $\frac{\deg(k)-6}{\mathrm{bipdeg}(k)} = \mathrm{jump}(CM(k))$ endpoint disjoint non-planar edges lying at distance at least $r(CM(k))$ from the border of $\overline{H}$ in $\overline{H}$. Together with Lemma 85, this implies that G admits a $K_{CM(k)}$-minor. This completes the proof of Lemma 87.

□

Proof of Theorem 86. We need the following technical result:

Lemma 91. *Let H be an $(m \times m)$-grid and Z be a set of i^2 vertices of H. Then H contains an $(\lfloor\frac{m}{i+1}\rfloor \times \lfloor\frac{m}{i+1}\rfloor)$-subgrid which does not contain any vertex from Z.*

Proof. We refine H by considering $(i+1)^2$ vertex-disjoint subgrids of size $\lfloor\frac{m}{i+1}\rfloor \cdot \lfloor\frac{m}{i+1}\rfloor$ each. As there are more grids than elements of Z there exists one such subgrid containing no vertex from Z, which is of size $\lfloor\frac{m}{i+1}\rfloor \cdot \lfloor\frac{m}{i+1}\rfloor$. □

Consider the refinement of the subgrid obtained from H by removing its $r(CM(k))$ first layers in $\overline{H}$, in vertex disjoint subgrids $H_{i,j}$, for $1 \leq i, j \leq 2CM(k)$ of size $\frac{GM(k)}{2CM(k)} \cdot \frac{GM(k)}{2CM(k)}$ each.

Claim 92. *If every subgrid* $H_{i,j}$ *is a non-planar graph, then* G *has the* $(2CM(k) \times 2CM(k))$*-crossed grid as a minor.*

Together with Lemma 83, Claim 92 implies that there exist $1 \le i, j \le 2CM(k)$ such that $H_{i,j}$ is an induced augmented grid. In the following, we use H' to denote $H_{i,j}$.

Claim 93. *If every set of* H'*-matched edges has size at most* $\mathrm{spec}(k)$ *then* G *admits a proper subgrid model.*

Proof. Recall that an H'-matched edge is an edge wz where $z \in V_G(H') \setminus V_G(B(H'))$ lies at distance at least $r(CM(k))$ of the border of H' in H' and $w \in V_G(H \setminus H')$. Let $\mathrm{Out} = V(G) \setminus V_G(H')$ denote the set of vertices of G outside the grid H', and In the set of parts of $V(H')$ lying at distance at least $r(CM(k))$ from the border of H' in H'. Let B be the bipartite graph with vertex bipartition $(\mathrm{Out}, \mathrm{In})$, and where a part $h \in \mathrm{In}$ is adjacent to a vertex $x \in \mathrm{Out}$ if there exists a vertex $y \in h$ such that xy is an edge of G. Denote by A the set of parts in In with degree at least $\mathrm{spec}(k)$ in B. Since every set of H'-matched edges has size at most $\mathrm{spec}(k)$, we have that $|A| < \mathrm{spec}(k)$. Hence there exists a subgrid $H'' \subseteq \mathrm{In}$ of size at least $\frac{GM(k)}{\sqrt{|\mathrm{spec}(k)|}}$ which contains no part of A by Lemma 91. Let C be the set of parts in H'' with degree at least 1 in B. If $|C| \le \mathrm{spec}(k)$, Lemma 91 gives a subgrid H''' of H'' of size at least $\frac{GM(k)}{\mathrm{spec}(k)} > \mathrm{prop}(k)$ which contains no part in C. Parts in H''' have no neighbours in $V(G) \setminus V_G(H')$, hence H''' is a proper subgrid. Otherwise if $|C| > \mathrm{spec}(k)$, let D be the set of vertices in Out with at least one neighbour in H''. We have by hypothesis that $|D| \leqslant \mathrm{spec}(k)^2$ (otherwise we would find a large set of H'-matched edges).

Hence there exists a grid model with a subgrid H'' which respects all properties of the definition of a proper subgrid but the last one. We now show how to deal with the special vertices. Let $H_0 = H''$. For $i \ge 0$, if there exists a vertex in Out adjacent in B to a set P_i of parts in H_i of size less than $k^2/4$, then we let H_{i+1} be the subgrid of H_i obtained by Lemma 91 which contains no vertex in P_i. Otherwise H_i is a proper subgrid and we stop. The above process can be repeated at most $D < \mathrm{spec}(k)^2$ times. Hence the proper subgrid found by this process has size at least $\frac{GM(k)}{\mathrm{spec}(k)^{\mathrm{spec}(k)}}$ which is at least $\mathrm{prop}(k)$ by definition of $GM(k)$. ◇

To complete the proof of Theorem 86, it remains to show that we can find a subgrid H_m of sufficiently large size with at most $\mathrm{spec}(k)$ H_m-matched edges. Assume that there are at least $\mathrm{spec}(k)$ H'-matched edges. Using Lemma 87, this implies either that G has a large clique minor or that G admits a grid model H with an induced augmented grid H_m having at most $\mathrm{spec}(k)$ H_m-matched edges. Since the former case is impossible by assumption, it follows by Claim 93 that G admits a proper subgrid model. This completes the proof of Theorem 86.

□

4.4.2 Reducing the Instance

Assume now that G has a $(GM(k) \times GM(k))$ grid model H but no $CM(k)$-clique model. By Theorem 86 we can moreover assume that G has a proper subgrid H'. Given a set F of edges, a *giant component* denotes in this case a connected component of $G \setminus F$ containing entirely at least $|V(H)| - k^2/4$ parts. We do not give detailed proofs of our claim when they are similar to claims in the previous section.

Lemma 94. *Let F be a set of at most k edges of a graph G admitting a grid model. Then $G \setminus F$ has a* giant component, *denoted by C_F.*

This is analogous to Lemma 79. The way to cut the most vertices in a grid with k edges is to cut off a corner of size $(k/2 \times k/2)$. We call the other connected components the *small components* (and they intersect altogether at most $k^2/4$ parts).

Lemma 95. *For every set F of at most k edges the special vertices belong to the giant component of $G \setminus F$.*

This is analogous to Lemma 80.

Lemma 96. *Let F be an optimal k-multicut of a graph G admitting a grid model H. Every small component C of $G \setminus F$ contains a vertex which is a terminal.*

This is analogous to Lemma 81.

Let us now consider the refinement of the subgrid H' into vertex-disjoint subgrids $H'_{i,j}$ of size $f_7(k) \cdot f_7(k)$ each, where $f_7(k) = \frac{prop(k)}{6 \cdot gath(k)} - \frac{k^2}{12}$. In other words, we will have $\frac{prop(k)}{2 \cdot (3f_7(k) + (k^2/4))} = gath(k)$.

Case 1: There exists a subgrid without terminal

Consider integers i, j be such that $H'_{i,j}$ does not contain any terminal, and let $h_{i,j}$ be a part of $H'_{i,j}$ at maximum distance in $H'_{i,j}$ of the border of the grid $H'_{i,j}$. We define $x_{i,j}$ to be a vertex of $h_{i,j}$ and $y_{i,j}$ to be some neighbour of $x_{i,j}$ in G.

Claim 97. *For every k-multicut F, $x_{i,j}$ and $y_{i,j}$ both belong to the giant component of $G \setminus F$.*

Proof. Recall that every small component contains a terminal and intersects at most $k^2/4$ parts, by Lemma 94 and Lemma 96. If part $h_{i,j}$ lies at distance more than $k^2/4 + 1$ in H of a part containing a terminal, then $x_{i,j}$ has no path in G to a terminal intersecting at most $k^2/4 + 1$ parts, and so $x_{i,j}$ and $y_{i,j}$ must both belong to the giant component of any k-multicut. Assume that there exists a path in G between $x_{i,j}$ and a terminal intersecting at most $k^2/4 + 1$ parts. This path uses an edge $z_{i,j} w_{i,j}$ with $z_{i,j} \in H'_{i,j}$ for some special vertex $w_{i,j}$. Indeed $H'_{i,j}$ has size larger than $2(k^2/4 + 2)$ by definition. So $h_{i,j}$ is at distance more than $k^2/4 + 1$ from the border of $H'_{i,j}$, and $H'_{i,j}$ contains no terminal. Since the special vertices belong to the giant component of $G \setminus F$ for every set F of at most k edges by Lemma 95, it follows that $x_{i,j}$ and $y_{i,j}$ belong to the giant component of $G \setminus F$. $\diamond$

Since $x_{i,j}$ and $y_{i,j}$ belong to the giant component of $G \setminus F$ for every k-multicut F, we can safely contract them.

Case 2: All subgrids contain a terminal

We find a gathered set using arguments similar to the ones used in the previous section. Consider the level decomposition of $G \setminus W$ starting from $L_0 := (H \setminus (H' \cup W)) \cup B(H')$. Since H' is a proper subgrid and hence contains no non-planar edges, if follows that there are at least $\frac{\text{prop}(k)}{2}$ levels in the level decomposition of $G \setminus W$ starting from L_0. Let T be a set of maximum size of terminals in $V_G(L_1 \cup \ldots \cup L_d)$ belonging to different parts lying at distance at least $k^2/4$ in $H_{>1} := V_H(L_1 \cup \ldots \cup L_d)$. Since every subgrid $H'_{i,j}$ in H' contains a terminal and has size $f_7(k) \cdot f_7(k)$ there is a terminal in every $3f_7(k)$ consecutive levels, implying that $|T| \geq \frac{d}{3f_7(k)+(k^2/4)} \geq \frac{\text{prop}(k)}{2\cdot(3f_7(k)+(k^2/4))} = \text{gath}(k)$.

We now show that T is a gathered set. Consider any set F of at most k edges. Since parts containing terminals of T are pairwise at distance at least $k^2/4$ in $H_{>1}$ and since small components intersect at most $k^2/4$ parts by Lemma 94, a small component of $G \setminus F$ can contain two terminals of T only if it contains a special vertex $w \in W$. Since w belongs to the giant component by Lemma 95, it follows that $G \setminus F$ contains at most one component (the giant component) containing more than one vertex of T and hence T is a gathered set. Applying Rule 3 to T, there exists an irrelevant request adjacent to a terminal in T which can be found in FPT time.

This completes the reduction of this section, showing that we can reduce in FPT time in k a graph with a large grid minor but with no large clique minor.

4.5 Conclusion

Deciding whether the MULTICUT problem parameterized by the solution size k is fixed parameter tractable was one of the most important open question in parameterized complexity [46]. In this Chapter we have shown that this problem can be reduced in FPT time to graphs of treewidth bounded by a function of k. Remains to prove that the MULTICUT problem is fixed parameter tractable on graphs of treewidth bounded by a function of k. This will be the purpose of Chapter 7, which will actually give a direct proof, not using a bounded treewidth assumption. Nonetheless, the key ideas of Chapter 7 will be the connectivity tools presented in this chapter.

The other idea which gave birth to Chapter 7 is the following. Let us say that an input graph G is a *meridian graph* if G consists in internally vertex disjoint paths between two given vertices u and v. The graph G has treewidth two since $G \setminus v$ is a tree. Although MULTICUT restricted to meridian graphs (or equivalently restricted to flowers) appears to be a very easy subcase of MULTICUT, it took us a rather long time to realize that it could efficiently be dealt with.

A meridian graph admits a vertex-multicut of small size: the two endpoints u and v, along with a bounded number (in k) of other vertices (unless the instance admits more than k disjoint paths realizing different requests, and thus is trivially false). Additionally, the components obtained after removing this vertex-multicut are very simple (just paths, here). We will see in Chapter 7 how MULTICUT can be solved efficiently on graphs sharing these properties of the meridian graphs (with rather easy branching and simulation of the requests with 2-SAT clauses), and how MULTICUT can be reduced to such simple graphs (and this will be rather involved).

CHAPTER

5 Well-Quasi-Ordering Induced Subgraphs

5.1 Introduction

Let S be a set and $\leq$ be a *quasi-order* on S, *i.e.* a reflexive and transitive relation. Given an infinite sequence $(x_i)_{i\in\omega}$ of elements of S, a *good pair* consists of two elements $x_i \leq x_j$, with $i < j$. An infinite sequence with no good pair is called a *bad sequence* of $(S, \leq)$. A quasi-order with no bad sequence is a *well-quasi-order.*

The following are other equivalent presentations of the notion of well-quasi-ordering (see for instance [93]). A quasi-order is a well-quasi-order if and only if:

- it has no infinite *antichain* (*i.e.* no infinite set of pairwise uncomparable elements) and no infinite strictly decreasing sequence.
- every infinite sequence has an infinite increasing subsequence.
- every nonempty subset of S has a nonempty finite set of minimal elements (up to equivalence).

The theory of well-quasi-ordering has been flourishing. Higman's Theorem states that the set of words over a well-quasi-ordered set is well-quasi-ordered by the subword relation [83], and this has been extended by Kruskal to trees [92]. Robertson and Seymour's celebrated graph minor theorem [114] (briefly discussed on pages 64-65) asserts that the minor relation is a well-quasi-order on the set of finite graphs. It implies that every graph class closed under minor (also called minor *ideal*) can be characterized by a finite list of excluded minors. This in turn implies that every minor ideal can be recognized in polynomial time.

The class of finite graphs is not well-quasi-ordered by the induced subgraph relation since the cycles form an infinite antichain. The good properties of the minor ideals ensured by the minor theorem do not hold for induced subgraph ideals (for instance, the set of paths, which is well-quasi-ordered, does not have a finite set of forbidden induced subgraphs, since it contains all cycles). This is a motivation for the stronger notion of 2-well-quasi-ordering.

In the following, we will be exclusively interested in the induced subgraph relation. Throughout this chapter, we will abbreviate "well-quasi-ordered by the induced subgraph relation" by *well-quasi-ordered*, with the understanding that we are dealing with the induced subgraph relation.

An extension of the notion of well-quasi-order is the notion of n-well-quasi-order. Given a class S of graph, the class $\hat{S}$ consists of all vertex n-colourings of graphs in S (*i.e.* each vertex is labelled by an integer in $\{1, \dots, n\}$) [1]. Let us consider the the coloured induced subgraph relation $\leq_c$ on vertex-coloured graphs: $G \leq_c G'$ if there is an injection from $V(G)$ to $V(G')$ preserving adjacency and colour. A set S of graphs is n-*well-quasi-ordered* if $\hat{S}$ is well-quasi-ordered by the coloured induced subgraph relation $\leq_c$.

Finally, the set S is ∞-*well-quasi-ordered* if S is n-well-quasi-ordered, for all $n \geq 1$.

The notion of 1-well-quasi-order is exactly the notion of well-quasi-order. Being 2-well-quasi-ordered is a strictly stronger property than being (1-)well-quasi-ordered, for instance the set of paths is not 2-well-quasi-ordered. Indeed, the set of paths where the endpoints have colour 1 and the internal vertices have colour 2 is an infinite antichain for $\leq_c$.

The notion of 2-well-quasi-ordering is especially interesting in view of algorithmic properties, as induced subgraph ideals which are 2-well-quasi-ordered can be characterized by a finite list of forbidden induced subgraphs, and thus are recognizable in polynomial time. Our ultimate aim would be to characterize the 2-well-quasi-ordered ideals of graphs, hopefully proving the following conjecture of Pouzet [107], also appearing in Fraïssé [68]:

Conjecture 98. *An induced subgraph ideal is 2-well-quasi-ordered if and only if it is ∞-well-quasi-ordered.*

In the more general framework of categories, Pouzet's question has a negative answer, as shown by Kriz and Sgall [91].

We will come back to this topic in Section 5. Our main purpose here is to study a restriction of the hierarchy of graph classes NLC, or equivalently of the cliquewidth hierarchy. [2]

The class NLC_k of k-*node labelled controlled graphs* was introduced in [123]. A *k-labelled graph* is a graph in which every vertex has a label in $\{1, \dots, k\}$. Let $\mathcal{F}$ be a set of functions from $\{1, \dots, k\}$ into $\{1, \dots, k\}$. The class $\mathrm{NLC}_k^{\mathcal{F}}$ is defined recursively on k-labelled graphs using three operators: $\bullet_i$, $\circ_f$ and χ_S. For $i \in \{1, \dots, k\}$, the operator $\bullet_i$ creates a single vertex labelled by i. The operator $\circ_f$, with $f \in \mathcal{F}$, applied to a k-labelled graph, replaces

1. In this chapter, "colouring" means "labelling", and has nothing to do with the graph theoretic concept of "proper colouring".

2. We do not define cliquewidth here. It is equivalent to NLC-width, in the sense that one is bounded if and only if the other is bounded. Although cliquewidth is much more common, we state our results in terms of NLC-decomposition for technical convenience.

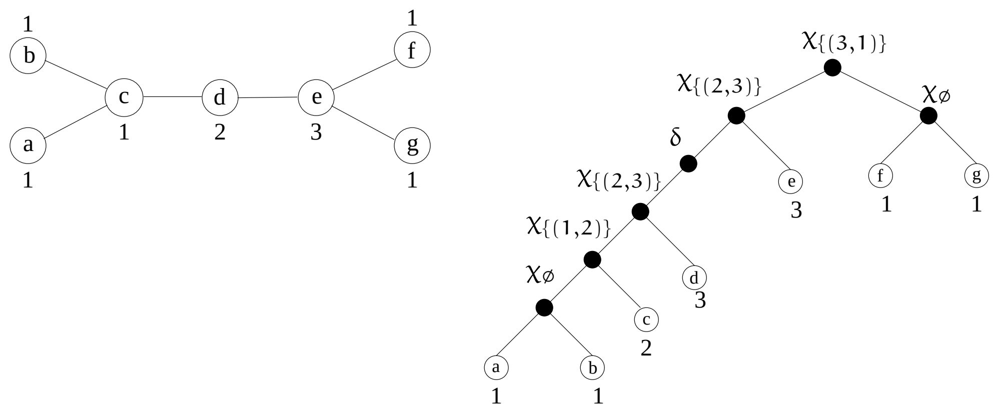


Figure 5.1: The decorated path G_3 and an associated NLC-expression, using a singe relabelling function, δ: $3 \rightarrow 2 \rightarrow 1$.

every label i with $f(i)$. The operator χ_S, with $S \subseteq \{1,\dots,k\} \times \{1,\dots,k\}$, applied to two k-labelled graphs G and H in this order, creates the disjoint union of graphs G and H, and for every pair $(i,j) \in S$ adds edges between every vertex of label i in G and every vertex of label j in H. The class NLC_k is defined to be NLC_k^{Φ} where Φ is the set of all functions from $\{1,\dots,k\}$ into $\{1,\dots,k\}$. The *NLC-width* of G is the minimum k for which some labelling of G is in NLC_k.

See Figure 5.1 for an example of an NLC_3 expression of a small graph.

When k is fixed, it is not known whether there exists a polynomial time algorithm computing a NLC decomposition using k colours for graphs in NLC_k. Only the cases $k = 1$ (which corresponds to cographs) and $k = 2$ [84] have been solved so far. Computing the NLC-width is NP-hard [82].

The NLC-width has a strong link with another well-known parameter: the clique-width, introduced by Courcelle *et al.* [35]. NLC-width and clique-width indeed differ by a factor at most 2. More precisely, the clique-width of a graph is bounded below by its NLC-width, and above by twice its NLC-width. Moreover, transformations respecting these bounds between decompositions of the two types can be done in linear time.

The class of graphs NLC_1 (cographs) is well-quasi-ordered, see [43] and [121] for the countable case. The class of graphs NLC_2 is well-quasi-ordered (and even ∞-well-quasi-ordered), this easily follows from the results in [84]. Indeed, the NLC_2 prime graphs for the modular decomposition are constructible in NLC_2 without relabelling, and thus form a well-quasi-ordered family by Kruskal's Tree Theorem [92]. However, the class NLC_3 is not well-quasi-ordered, as it contains for every i the *decorated path of length* i depicted in Figure 5.1, whichs consists of a path of length i with two pendant vertices attached to each extremity, and which we denote by G_i. These graphs indeed do not form a well-quasi-ordered family. Allowing all relabelling operators $\circ_f$ is too much to construct a well-quasi-ordered class of graphs if we have at least 3 colours. This is why we define a restriction of NLC, using only relabelling operators from a specified set of functions $\mathcal{F}$. Our main purpose is to characterize the sets $\mathcal{F}$ such that $\mathrm{NLC}_k^{\mathcal{F}}$ is well-quasi-ordered. We will see that $\mathrm{NLC}_k^{\mathcal{F}}$ is well-quasi-ordered (equivalently ∞-well-quasi-ordered) if and only if it does not contain arbitrarily large paths.

In Section 5.2, we introduce a binary relation $\preceq$ on a set of functions. In Section 5.3 we introduce a subword order $\leq$ on words labelled with a set of functions which is more constrained than Higman's subword order. In Section 5.4 we extend $\leq$ to trees, with the purpose of applications to $\mathrm{NLC}_k^{\mathcal{F}}$ expressions. In Section 5.5, we characterize the sets $\mathcal{F}$ for which $\mathrm{NLC}_k^{\mathcal{F}}$ is well-quasi-ordered. In the final section, we discuss Pouzet's conjecture on n-well-quasi-ordering, and further well-quasi-ordering problems.

Throughout this chapter, we will obtain the following equivalent characterizations for a set of functions $\mathcal{F}$ (with Theorem 104, Theorem 106, Theorem 110, Corollary 109 and Corollary 111):

- The set $\mathcal{F}$ is totally ordered by $\preceq$.
- The set of words on $\mathcal{F}$ is well-quasi-ordered by $\leq$.
- The set of trees on $\mathcal{F}$ is well-quasi-ordered by $\leq$.
- The set of graphs $\mathrm{NLC}_k^{\mathcal{F}}$ is well-quasi-ordered.
- The set of graphs $\mathrm{NLC}_k^{\mathcal{F}}$ is ∞-well-quasi-ordered.
- The set of graphs $\mathrm{NLC}_k^{\mathcal{F}}$ does not contain arbitrarily large paths.

5.2 Totally Ordered Sets of Functions

Let $\mathcal{F}$ be a set of functions from $\{1,\dots,k\}$ into $\{1,\dots,k\}$ closed under composition (with the convention that the identity function ε belongs to $\mathcal{F}$). We can make these assumptions without loss of generality with respect to classes $\mathrm{NLC}_k^{\mathcal{F}}$, in the sense that such a class can always be defined by using a set $\mathcal{F}$ of relabel functions as above. The key definition of this section is the following. Let us say that $f \preceq g$ whenever $Im(f \circ g) = Im(f)$.

Assume that $\preceq$ is total on $\mathcal{F}$, *i.e.* for every f, g in $\mathcal{F}$, at least one of $f \preceq g$ and $g \preceq f$ holds. This implies in particular that $Im(f^2) = Im(f)$ for all $f \in \mathcal{F}$. Observe that $f \preceq g$ implies that $|Im(f)| \leq |Im(g)|$.

Lemma 99. *If $\preceq$ is total on $\mathcal{F}$, then $\preceq$ is transitive.*

Proof. Assume by contradiction that $Im(f \circ g) = Im(f)$, $Im(g \circ h) = Im(g)$ and $f \not\preceq h$, *i.e.* $|Im(f \circ h)| < |Im(f)|$. Since $\preceq$ is total, we must have $h \preceq f$, and then $Im(h \circ f) = Im(h)$. Hence $f \preceq g \preceq h \preceq f$, and thus $|Im(f)| = |Im(h)|$. As $h \circ f \preceq h \circ f$ holds, we have $|Im(h \circ f \circ h \circ f)| = |Im(h \circ f)| = |Im(h)|$. Moreover $|Im(h \circ (f \circ h) \circ f)| \leq |Im(f \circ h)| < |Im(f)| = |Im(h)|$, which is a contradiction. $\square$

Thus $\preceq$ is a reflexive and transitive relation, in other words $\preceq$ is a total quasi-order on $\mathcal{F}$. This is equivalent to the existence of a partition of $\mathcal{F}$ into t equivalence classes $F_1,...,F_t$ such that $f \in F_i$ and $g \in F_j$ satisfy $f \preceq g$ if and only if $i \leq j$.

Lemma 100. *When $\mathcal{F}$ is totally quasi-ordered by $\preceq$, the equivalence classes $F_1,...,F_t$ are exactly the classes of functions having an image of the same size, in increasing order of the image size.*

Proof. If $|Im(f)| < |Im(g)|$ then $g \not\preceq f$, so we must have $f \preceq g$. Suppose now by contradiction that $|Im(f)| = |Im(g)|$, $f \preceq g$ and $g \not\preceq f$. This means that $|Im(f\circ(g\circ f)\circ g)| \leq |Im(g\circ f)| < |Im(g)|$. We also have that $|Im(f \circ g)| = |Im(f)| = |Im(g)|$, thus $|Im(f\circ g\circ f\circ g)| < |Im(f\circ g)|$. This contradicts $f \circ g \preceq f \circ g$. □

Observe that the top class F_t contains ε, and contains only permutations, since other functions do not have the whole set $\{1,\dots,k\}$ as an image.

Lemma 101. *For all i, F_i and $\cup_{k\geq i}F_k$ are closed under composition.*

Proof. The first part of the statement follows by Lemma 100 and by definition of $\preceq$. To prove that $\cup_{k\geq i}F_k$ is closed under composition, consider $f \in F_i$ and $g \in F_j$ with $i < j$. Since $f \preceq g$, $f \circ g$ is in F_i, as its image is the same as the image of f. Assume now by contradiction that $g \circ f \in F_p$, with $p < i$. Then $|Im(f \circ g)| = |Im(f)| > |Im(g \circ f)| \geq |Im(f \circ (g \circ f) \circ g)|$. Thus $|Im(f \circ g)| \neq |Im((f \circ g)^2)|$, contradicting $f \circ g \preceq f \circ g$. □

Lemma 102. *The functions of the bottom class F_1 verify a "left-cancellation" identity:*

$$\forall f \in F_1, \forall h, h' \in \mathcal{F}, \textit{ if } h \circ f \circ h' = h \circ f \textit{ then } f \circ h' = f$$

Proof. This identity actually holds whenever $f \preceq h$. We will use Lemma 102 only with $f \in F_1$, but we prove the more general statement where the assumption $f \in F_1$ has been replaced with the weaker $f \preceq h$.

By Lemma 101, $f \preceq h \circ f$. Assume that $h \circ f \circ h' = h \circ f$. If there is an $x \in \{1,\dots,k\}$ such that $f \circ h'(x) \neq f(x)$, then these two distinct elements belong to the image of f and have the same image under h. This means that $|Im(h \circ f)| < |Im(f)|$, and contradicts the fact that $f \preceq h \circ f$. □

Here is an example of a set of functions which is totally ordered by $\preceq$. An $(i,j)-$*cast*, with $i \leq j$, is a function f from $\{1,\dots,k\}$ into itself such that $f(l) = l$ for all $l < i$ and $f(l) = j$ whenever $l \geq i$. It is routine to check that the set of all such casts is indeed totally ordered by $\preceq$. We feel that the following problem would give some insight on the well-quasi-ordered $NLC_k^{\mathcal{F}}$ classes:

Problem 103. *Find a generic class of functions $\mathcal{G}$ (like casts for instance) such that for every totally ordered $\mathcal{F}$ and k, there exists some k' for which $NLC_k^{\mathcal{F}}$ is included in $NLC_{k'}^{\mathcal{G}}$.*

Such a class of function would describe much more precisely how to construct the well-quasi-ordered classes $NLC_k^{\mathcal{F}}$.

5.3 Words on Functions

An $\mathcal{F}$-*word* is a finite word on the alphabet $\mathcal{F}$, *i.e.* a finite sequence $f_1,\dots,f_l$ of elements of $\mathcal{F}$, with $l \geq 0$. Let $\mathcal{W}^{\mathcal{F}}$ be the set of $\mathcal{F}$-words. Let $M = f_1,\dots,f_l$ and $M' = f'_1,\dots,f'_{l'}$ be two $\mathcal{F}$-words. We denote by $|M| = l$ the length of M. The word M is a *subword* of M' if there exists an increasing injection ϕ from $\{1,\dots,l\}$ into $\{1,\dots,l'\}$ such that $f_i = f'_{\phi(i)}$. In other words, a subword respects the order of the letters but does not preserve consecutiveness. Higman's Theorem [83] asserts that the subword partial order is a well-quasi-order when the alphabet is finite. In our more constrained partial order on $\mathcal{F}$-words, we write $M \leq M'$ if two conditions are satisfied:

- There is a function ϕ for which M is a subword of M'.
- For all $1 \leq i < l$, we have $f_i = f'_{\phi(i)} \circ f'_{\phi(i)+1} \circ \cdots \circ f'_{\phi(i+1)-1}$.

Thus, when $M \leq M'$ and $i < j$, the composition of functions $f_i \circ f_{i+1} \circ \cdots \circ f_{j-1}$ is equal to the function $f'_{\phi(i)} \circ f'_{\phi(i)+1} \circ \cdots \circ f'_{\phi(j)-1}$. And since $f_j = f'_{\phi(j)}$, we also have $f_i \circ f_{i+1} \circ \cdots \circ f_j = f'_{\phi(i)} \circ f'_{\phi(i)+1} \circ \cdots \circ f'_{\phi(j)}$.

Our goal is to prove here that $\mathcal{W}^{\mathcal{F}}$ is well-quasi-ordered by $\leq$ if and only if $\preceq$ is total on $\mathcal{F}$. For this, we have to be more general and we need to consider $\mathcal{W}^{\mathcal{F}}_Q$, the set of words on the set $\mathcal{F} \times Q$, where Q is a set equipped with a well-quasi-order $\leq_Q$.

We naturally extend the partial order $\leq$ on $\mathcal{W}^{\mathcal{F}}_Q$. For w in $\mathcal{W}^{\mathcal{F}}_Q$ and $1 \leq x \leq |w|$ we denote by (f^w_x, q^w_x) the xth letter of w. For every pair of indices a, b, with $1 \leq a < b \leq |w|$, we define $L^w(a,b)$ to be the composition $f^w_a \circ f^w_{a+1} \circ \ldots \circ f^w_{b-1}$. When $a = b$, we set $L^w(a,b) = \epsilon$. Let ϕ be an increasing injection from $\{1,\dots,|w|\}$ into $\{1,\dots,|w'|\}$. We say that ϕ *is compatible with labels* if $f^w_x = f^{w'}_{\phi(x)}$ and $q^w_x \leq_Q q^{w'}_{\phi(x)}$ for every $x \in 1,\dots,|w|$. We say that ϕ *preserves path-composition* if for every $x < |w|$, we have that $L^w(x,x+1) = L^{w'}(\phi(x),\phi(x+1))$ (recall that by definition $L^w(x,x+1) = f^w_x$). We write $w \leq w'$ if there exists an increasing injection ϕ from $\{1,\dots,|w|\}$ into $\{1,\dots,|w'|\}$ which is compatible with labels and preserves path-composition. When ϕ is only compatible with labels, we simply say that w is a *subword* of w' and write $w \leq_0 w'$.

Theorem 104. *The set of words* $\mathcal{W}^{\mathcal{F}}_Q$*, where* Q *is a well-quasi-order, is well-quasi-ordered by* $\leq$ *if and only if* $\preceq$ *is a total quasi-order on* $\mathcal{F}$*.*

Proof. Assume first that $\preceq$ is not a total quasi-order on $\mathcal{F}$, *i.e.* there exist two incomparable functions $f, g \in \mathcal{F}$. Depending if $f = g$ or not, let us show in both cases that $\mathcal{W}^{\mathcal{F}}$ is not well-quasi-ordered by $\leq$.

- If $f = g$, we claim that $S = (w_k)_{k \geq 0}$, where $w_k = \epsilon f^k \epsilon$, is a bad sequence[3] Indeed, if $w_i \leq w_j$ with $i < j$, the identity functions must be mapped to identity functions, as f must be different from ϵ. Moreover, to preserve path-composition, the first and

3. Note the similarity with the basic counter-example to 2-well-quasi-order, the decorated paths. The word w is essentially a path of f where the extremities are marked by an ϵ.

the last f of w_i must be mapped to the first and the last f of w_j. Hence, two consecutive f's in w_i must be mapped to non-consecutive f's in w_j, which contradicts path-composition since $f \neq f^l$ for every $l \geq 2$. Indeed, f being incomparable with itself means that $|\mathrm{Im}(f^2)| < |\mathrm{Im}(f)|$.

– If $f \neq g$, we claim that $S = (w_k)_{k\geq 0}$, where $w_k = \epsilon(fg)^k\epsilon$, is a bad sequence. Again, if $w_i \leq w_j$ with $i < j$, the identity functions must be mapped to identity functions. Moreover, to preserve path-composition, the first f and the last g of w_i must respectively be mapped to the first f and to the last g of w_j. Hence, two consecutive f and g in w_i must be mapped to non-consecutive f and g in w_j, which contradicts path-composition.

Assume now that $\preceq$ is a total quasi-order on $\mathcal{F}$. Our goal is to prove that $\mathcal{W}_Q^{\mathcal{F}}$ is well-quasi-ordered.

To achieve this, we need to enrich the structure, and consider words on $\mathcal{F}\times\mathcal{F}\times Q$ instead of $\mathcal{F}\times Q$. We will label every letter of a word by another function in $\mathcal{F}$. Precisely, we add, to every letter x of a word w, the function consisting of the composition of all functions of the prefix of w before x. The reason for this is to keep track, in every letter of the word, of some information preceding this letter. Technically, this bit of information allows us to use a Nash-Williams' minimum bad sequence argument, cutting every word in a prefix-suffix way and applying the minimality to prove the well-quasi-order. The key idea is to cut the words on some letter which belongs to the bottom class of $\mathcal{F}$, since left-cancellation (thanks to the extra label) enables to glue back the prefix and the suffix.

Let us turn to technicalities. Instead of dealing with words $w = ((f_x^w, q_x^w))_{x=1\ldots|w|}$ on $\mathcal{F}\times Q$, we transform w into the word $\tilde{w} = ((f_x^w, L^w(1,x), q_x^w))_{x=1\ldots|w|}$ on $\mathcal{F}\times\mathcal{F}\times Q$. To simplify notation, we still call this word w.

A word $((f_x, h_x, q_x))_{x=1\ldots k}$ on $\mathcal{F}\times\mathcal{F}\times Q$ is *coherent* if for every $1 \leq x < k$, we have $h_{x+1} = h_x \circ f_x$. Observe that w is coherent by construction, and so is every factor of w. We denote by $\widetilde{\mathcal{W}}_{\mathcal{F}\times Q}^{\mathcal{F}}$ the set of coherent words on $\mathcal{F}\times\mathcal{F}\times Q$. The set $\mathcal{F}\times Q$ is equipped with the well-quasi-order $(f,q) \leq_{\mathcal{F}\times Q} (f',q')$ if $f = f'$ and $q \leq_Q q'$. Theorem 104 follows from the following result.

Lemma 105. *For every well-quasi-ordered set* Q*, the set* $\widetilde{\mathcal{W}}_{\mathcal{F}\times Q}^{\mathcal{F}}$ *is well-quasi-ordered by* $\leq$*.*

Proof. By induction on t, the number of equivalence classes of $\mathcal{F}$.

Assume first that $t = 1$. Observe that $\{\varepsilon\} \subseteq \mathcal{F} \subseteq \mathfrak{S}_n$, where $\mathfrak{S}_n$ is the set of permutations of $\{1,\ldots,n\}$. By Higman's Theorem [83], $\widetilde{\mathcal{W}}_{\mathcal{F}\times Q}^{\mathcal{F}}$ is well-quasi-ordered by $\leq_0$. Let us prove that $\leq$ and $\leq_0$ coincide in this case. Suppose that $w = ((f_x^w, h_x^w, q_x^w))_{x=1\ldots|w|}$ and $w' = ((f_x^{w'}, h_x^{w'}, q_x^{w'}))_{x=1\ldots|w'|}$ belong to $\widetilde{\mathcal{W}}_{\mathcal{F}\times Q}^{\mathcal{F}}$ and satisfy $w \leq_0 w'$. There exists an increasing injection ϕ from $\{1,\ldots,|w|\}$ into $\{1,\ldots,|w'|\}$ which is compatible with labels. Consider a position $x < |w|$. Since ϕ preserves labels, we have $f_x^w = f_{\phi(x)}^{w'}$ (we call this function f) and $h_x^w = h_{\phi(x)}^{w'}$ (we call this function h). To prove that $\leq_0$ preserves path-composition, we

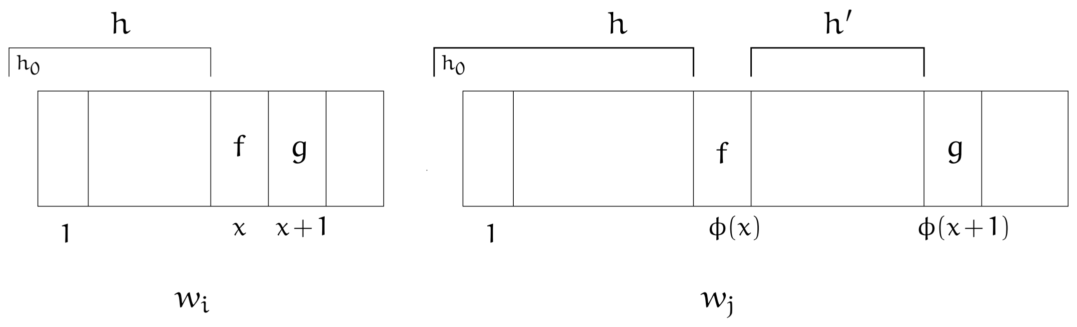


Figure 5.2: The words w and w' in the proof of Lemma 105.

just have to check that $f^{w}_{x} = f^{w'}_{\phi(x)} \circ f^{w'}_{\phi(x)+1} \circ \cdots \circ f^{w'}_{\phi(x+1)-1}$. Equivalently, by letting $h' = f^{w'}_{\phi(x)+1} \circ \cdots \circ f^{w'}_{\phi(x+1)-1}$ (the empty product being ϵ), we want to prove that $f = f \circ h'$. Since w is coherent, we have $h^{w}_{x+1} = h^{w}_{x} \circ f^{w}_{x} = h \circ f$. Since w' is coherent, we have $h^{w'}_{\phi(x+1)} = h^{w'}_{\phi(x)} \circ f^{w'}_{\phi(x)} \circ f^{w'}_{\phi(x)+1} \circ \cdots \circ f^{w'}_{\phi(x+1)-1}$, hence $h^{w'}_{\phi(x+1)} = h \circ f \circ h'$. Since ϕ preserves labels, we have $h \circ f = h^{w}_{x+1} = h^{w'}_{\phi(x+1)} = h \circ f \circ h'$. We can then deduce that $f = f \circ h'$ by left-cancellation of the permutation h. This concludes the proof of the base case.

Let us now assume that the induction hypothesis holds for $t-1$. Consider $\mathcal{F}$ with t equivalence classes. We will prove that $\widetilde{\mathcal{W}}^{\mathcal{F}}_{\mathcal{F}\times Q}$ is well-quasi-ordered by $\leq$ using Nash-Williams' minimal bad sequence argument [104]. By contradiction, let $S = (w_i)_{i\in\omega}$ be an infinite bad sequence of $\widetilde{\mathcal{W}}^{\mathcal{F}}_{\mathcal{F}\times Q}$, minimal in the sense that for every $i \geq 1$, the ith word w_i of this sequence is defined to be a word of minimal size such that there exists a bad sequence (with respect to $\leq$) starting with $w_1, \ldots, w_{i-1}, w_i$.

Set $w_i = ((f^{w_i}_{x}, (h^{w_i}_{x}, q^{w_i}_{x})))_{x=1\ldots|w_i|}$, and consider the minimal p_i such that $f^{w_i}_{p_i} \in F_1$, if such a p_i exists. The subword $w'_i = ((f^{w_i}_{x}, (h^{w_i}_{x}, q^{w_i}_{x})))_{x=1\ldots p_i}$, or $w'_i = w_i$ if p_i is undefined, is called the *beginning* of w_i. Likewise, the *end* of w_i is the subword $w''_i = ((f^{w_i}_{x}, (h^{w_i}_{x}, q^{w_i}_{x})))_{x=p_i+1\ldots|w_i|}$, or the empty word if p_i is undefined.

Let X be the set of all ends of words in S. We now prove that X is well-quasi-ordered by $\leq$. Let σ be an infinite sequence of X. Let $\alpha = (z_i)_{i\in\omega}$ be an infinite extracted subsequence of σ such that for all i, if z_i is the end of w_j and z_{i+1} is the end of w_k, then $j \leq k$. Let w_n be the word of which z_1 is the end. The sequence $(w_1, \ldots, w_{n-1}, z_1, z_2, \ldots)$ is good by the minimality of S. Since S is bad, one cannot have $w_i \leq z_j$, or $w_i \leq w_j$ with $i < j$. Hence there exists a good pair $z_i \leq z_j$ with $i < j$, thus X is well-quasi-ordered by $\leq$.

By Lemma 101, we know that $\mathcal{F} - F_1$ is closed under composition. Let $R = (F_1 \times X) \cup \{\triangle\}$ where $\triangle$ is a new marker. The set R is ordered by $\leq_R$ as follows: $\triangle \leq_R \triangle$, and $(f, x) \leq_R (g, x')$ if $f = g$ and $x \leq x'$. Since X is well-quasi-ordered, we have that R and, hence $Q' = Q \times R$ are well-quasi-ordered. By our induction hypothesis, $\widetilde{\mathcal{W}}^{\mathcal{F}-F_1}_{(\mathcal{F}-F_1)\times Q'}$ is well-quasi-ordered by $\leq$, where the order $\leq_{Q'}$ on Q' is defined by: $(q, r) \leq_{Q'} (q', r')$ if $q \leq_Q q'$ and $r \leq_R r'$.

Denote by w_i''' the word of $\widetilde{\mathcal{W}}^{\mathcal{F}-F_1}_{(\mathcal{F}-F_1)\times Q'}$ which is the concatenation of the word $((f_x^{w_i},(h_x^{w_i},(q_x^{w_i},\triangle))))_{x=1\dots p_i-1}$ with the extra letter $(\varepsilon,(h_{p_i}^{w_i},(q_{p_i}^{w_i},(f_{p_i}^{w_i},w_i''))))$ if p_i exists, or $((f_x^{w_i},(h_x^{w_i},(q_x^{w_i},\triangle))))_{x=1\dots p_i-1}$ otherwise. Note that w_i''' is coherent since w_i is coherent.

As $\widetilde{\mathcal{W}}^{\mathcal{F}-F_1}_{(\mathcal{F}-F_1)\times Q'}$ is well-quasi-ordered by $\leq$, there exist $i < j$ such that $w_i''' \leq w_j'''$. We denote by ϕ the mapping from w_i''' into w_j'''. Observe that if w_i''' does not have an extra letter, we would directly have that $w_i \leq w_j$ which is impossible since S is a bad sequence. Hence p_i and p_j exist. Let us now exhibit a mapping Φ from w_i into w_j which preserves labels and path-composition.

First, we set Φ to be the restriction of the function ϕ from $\{1,\dots,p_i\}$ into $\{1,\dots,p_j\}$. Observe that $\phi(p_i) = p_j$ since extra letters do not carry a marker δ. Moreover, the extra letter $(\varepsilon,(h_{p_i}^{w_i},(q_{p_i}^{w_i},(f_{p_i}^{w_i},w_i''))))$ is mapped to the extra letter $(\varepsilon,(h_{p_j}^{w_j},(q_{p_j}^{w_j},(f_{p_j}^{w_j},w_j''))))$. In particular, we have $w_i'' \leq w_j''$. The mapping ϕ'' from $w_i'' \leq w_j''$ which realizes $w_i'' \leq w_j''$ is our extension of Φ from $\{p_i+1,\dots,|w_i|\}$ into $\{p_j+1,\dots,|w_j|\}$.

Now Φ is completely defined. Moreover, it preserves labels by definition. Thus, we only need to show that Φ also preserves path-composition, and more precisely path-composition exactly after the letter p_i, since the other cases are already taken into account by ϕ or ϕ'.

We have to show that $f_{p_i}^{w_i} = f_{\Phi(p_i)}^{w_j} \circ \cdots \circ f_{\Phi(p_i+1)-1}^{w_j}$ or equivalently that $f_{p_i}^{w_i} = f_{p_j}^{w_j} \circ f_{p_j+1}^{w_j} \cdots \circ f_{\phi''(p_i+1)-1}^{w_j}$. Observe that this condition holds vacuously when $p_i = |w_i|$. As the extra letter of w_i''' is mapped to the extra letter of w_j''', we have $f_{p_i}^{w_i} = f_{p_j}^{w_j}$ (we call this function f) and $h_{p_i}^{w_i} = h_{p_j}^{w_j}$ (we call this function h). We now let $h' = f_{p_j+1}^{w_j} \cdots \circ f_{\phi''(p_i+1)-1}^{w_j}$ (the empty product being ϵ). Since the word w_i is coherent, we have $h_{p_i+1}^{w_i} = h_{p_i}^{w_i} \circ f_{p_i}^{w_i} = h \circ f$. Since w_j is coherent, we have $h_{\phi''(p_i+1)}^{w_j} = h_{p_j}^{w_j} \circ f_{p_j}^{w_j} \circ f_{p_j+1}^{w_j} \circ \cdots \circ f_{\phi''(p_i+1)-1}^{w_j}$. Hence $h_{\phi''(p_i+1)}^{w_j} = h \circ f \circ h'$. Since ϕ'' preserves labels, we have $h \circ f = h_{p_i+1}^{w_i} = h_{\phi''(p_i+1)}^{w_j} = h \circ f \circ h'$. We now conclude by Lemma 102 that $f = f \circ h'$. This concludes the proof of Lemma 105 and Theorem 104. □

□

5.4 Trees on Functions

We extend in this section our results to trees. However, since the arguments are similar to the previous section, we will not give the same level of detail, especially concerning the verification of path-composition.

A *structured tree* is a finite rooted tree where the children of a node are ordered from left to right. We denote by $\mathcal{T}_Q^{\mathcal{F}}$ the set of structured trees with nodes labelled by $\mathcal{F} \times Q$, where $\mathcal{F}$

is as usual a set of functions, and Q is a well-quasi-ordered set. A node x is then labelled by a pair $l(x) = (f(x), q(x))$. We simply write $\mathcal{T}^{\mathcal{F}}$ when there is no additional label Q. The set of nodes of a tree T is denoted by $V(T)$. We write $x \wedge y$ to denote the least common ancestor of x and y. We say that (x, y) is an *arc* of T when x is the father of y. A sequence of nodes $z_0, z_1, ..., z_n$ is a *downward path* in T if (z_i, z_{i+1}) is an arc, for every $i = 0, \dots, n-1$, and we say that z_0 is an *ancestor* of z_n and that z_n is a *descendant* of z_0. For such a downward path $z_0, z_1, ..., z_n$, we denote by $L(z_0, z_n)$ the composition $f(z_0) \circ f(z_1) \circ ... \circ f(z_{n-1})$.

Let us define a partial order $\leq$ on $\mathcal{T}^{\mathcal{F}}_Q$ which extends the order $\leq$ on words. Precisely, let us write that $T \leq T'$ if there exists an injection ϕ from $V(T)$ into $V(T')$ such that:

- ϕ preserves descendants.
- ϕ preserves least common ancestors, *i.e.* $\phi(x \wedge y) = \phi(x) \wedge \phi(y)$.
- ϕ preserves the *left/right order*, *i.e.* if x and y are not in descendant relation, and the branch of $x \wedge y$ containing x is to the left of the one containing y, the same holds for the branches of $\phi(x) \wedge \phi(y)$ containing $\phi(x)$ and $\phi(y)$.
- ϕ preserves labels, *i.e.* $f(x) = f(\phi(x))$ and $q(x) \leq_Q q(\phi(x))$.
- ϕ preserves *path-composition*, *i.e.* for every arc (x, y) in T, we have $L(x, y) = L(\phi(x), \phi(y))$, *i.e.* $f(x) = L(\phi(x), \phi(y))$.

When ϕ satisfies all these properties except possibly path-composition, we simply write $T \leq_0 T'$. Kruskal's Tree Theorem [92] asserts that $\leq_0$ is a well-quasi-order on the set of trees.

Note that the relation $\leq$ does indeed extend the relation $\leq$ on words defined in the previous section.

This more constrained order relation $\leq$ presents some analogies with the so-called *gap-condition embedding* studied by Kriz in [90]. For instance, when the class of functions $\mathcal{F}$ is totally ordered, and hence partitioned into $F_1, \dots, F_t$, the path-composition property implies that if y is a child of x and $f(x)$ belongs to F_i, then every function of the product $L(\phi(x), \phi(y))$ belong to classes with height at least i. It could be interesting to state a common generalisation of these results, possibly involving ordinal functions.

Theorem 106. *The set* $\mathcal{T}^{\mathcal{F}}_Q$*, where* Q *is a well-quasi-order, is well-quasi-ordered by* $\leq$ *if and only if* $\preceq$ *is total on* $\mathcal{F}$.

Proof. If $\preceq$ is not total on $\mathcal{F}$, the set of words, hence of trees, is not well-quasi-ordered as we have seen in the previous section.

Assume now that $\preceq$ is total on $\mathcal{F}$. As for words, we transform a tree T in $\mathcal{T}^{\mathcal{F}}_Q$ into a tree in $\mathcal{T}^{\mathcal{F}}_{\mathcal{F}\times Q}$. To every vertex x of T distinct from the root r, we give the extra label $L(r, x)$, the extra label of the root being ϵ. More generally, we say that a rooted tree T with nodes labelled by $\mathcal{F} \times \mathcal{F} \times Q$ is *coherent* if for every arc $(x, y) \in T$, with $l(x) = (f, h, q)$ and $l(y) = (g, h', q')$, we have $h' = h \circ f$. We denote by $\widetilde{\mathcal{T}}^{\mathcal{F}}_{\mathcal{F}\times Q}$ the set of coherent structured rooted trees with nodes labelled by $\mathcal{F} \times \mathcal{F} \times Q$. Theorem 106 is a consequence of the following result.

Lemma 107. *If* Q *is well-quasi-ordered, the set* $\widetilde{\mathcal{T}}^{\mathcal{F}}_{\mathcal{F}\times Q}$ *is well-quasi-ordered by* $\leq$.

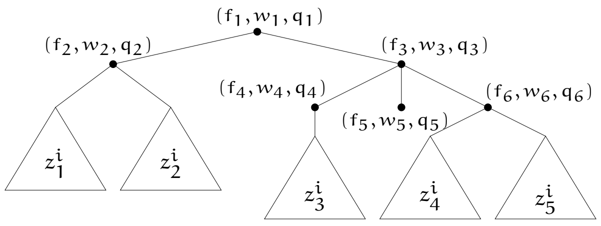
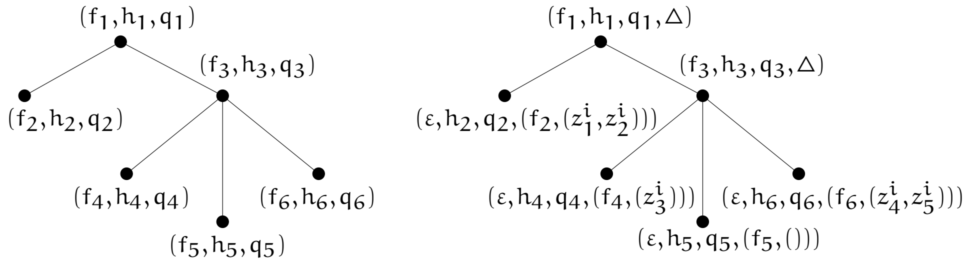


Figure 5.3: The tree T_i (f_2, f_4, f_5 and f_6 are in F_1, while f_1 and f_3 are not).



(f_1, h_1, q_1)
(f_3, h_3, q_3)
(f_2, h_2, q_2)
(f_4, h_4, q_4)
(f_6, h_6, q_6)
(f_5, h_5, q_5)
$(f_1, h_1, q_1, \triangle)$
$(f_3, h_3, q_3, \triangle)$
$(\varepsilon, h_2, q_2, (f_2, (z^i_1, z^i_2)))$
$(\varepsilon, h_4, q_4, (f_4, (z^i_3)))$
$(\varepsilon, h_6, q_6, (f_6, (z^i_4, z^i_5)))$
$(\varepsilon, h_5, q_5, (f_5, ()))$

Figure 5.4: The trees T'_i (left) and T''_i (right)

Proof. Let us prove this by induction on t, the number of equivalence classes of $\mathcal{F}$.

When $t = 1$, the proof goes according to the case of words: by Kruskal's Theorem, $\widetilde{\mathcal{T}}^{\mathcal{F}}_{\mathcal{F}\times Q}$ is well-quasi-ordered by $\leq_0$, which again coincides with $\leq$.

Let us assume that the induction hypothesis holds for $t-1$. Consider $\mathcal{F}$ with t equivalence classes. We prove that $\widetilde{\mathcal{T}}^{\mathcal{F}}_{\mathcal{F}\times Q}$ is well-quasi-ordered by $\leq$ using Nash-Williams' minimal bad sequence argument. By contradiction, let $S = (T_i)_{i\in\omega}$ be an infinite bad sequence of $\widetilde{\mathcal{T}}^{\mathcal{F}}_{\mathcal{F}\times Q}$, minimal in the sense that for $i \geq 1$, the i^{th} tree T_i of this sequence is defined to be a tree of minimal size for which there exists a bad sequence starting with $T_1, ..., T_{i-1}, T_i$.

A *branching vertex* is a node x labelled by (f, w, q), with $f \in F_1$, and with no ancestor having a label (g, w', q') with g in F_1. A *branch* is a subtree which is rooted in a child of a branching vertex. We denote by T'_i the subtree of T_i obtained by deleting all the branches of T (see Figure 5.3 and Figure 5.4). Let X be the set of all branches of the trees in S. As in the previous section, the minimality of S ensures that X is well-quasi-ordered by $\leq$.

Recall that $\mathcal{F} - F_1$ is closed under composition. We denote by $Seq(X)$ the set of sequences of X. The usual order $\leq_{seq}$ on $Seq(X)$ is defined as follows: for every two sequences $(x_1, ..., x_m)$ and $(x'_1, ..., x'_n)$ of X, we have $(x_1, ..., x_m) \leq_{seq} (x'_1, ..., x'_n)$ if there exists an increasing injection σ from $\{1, \ldots, m\}$ to $\{1, \ldots, n\}$ such that $\forall i \in \{1, \ldots, m\}\, x_i \leq x'_{\sigma(i)}$. Let $R = ((F_1 \times Seq(X)) \cup \{\triangle\})$ with $\triangle$ a new marker. Let $\leq_R$ be the following order on R: $\triangle \leq_R \triangle$, and $(f, x) \leq_R (g, x')$ if $f = g$ and $x \leq_{seq} x'$. By Higman's Theorem, $\leq_{seq}$ is a well-quasi-order on $Seq(X)$, hence $\leq_R$ is a well-quasi-order on R. By our induction hypothesis $\widetilde{\mathcal{T}}^{\mathcal{F}-F_1}_{(\mathcal{F}-F_1)\times Q'}$, with $Q' = Q \times R$, is well-quasi-ordered by $\leq$.

Denote by T_i'' the tree obtained from T_i' by modifying the labels of its nodes as follows:
– Labels of internal nodes are modified from (f, h, q) into $(f, h, (q, \triangle))$.
– Labels of leaves x are modified from (f, h, q) into $(\epsilon, h, (q, s))$, where s is the sequence of branches of the branching vertex x (see Figure 5.4).

Observe that the trees T_i'' are coherent, and since $\widetilde{\mathcal{T}}^{\mathcal{F}-F_1}_{(\mathcal{F}-F_1)\times Q'}$ is well-quasi-ordered by $\leq$, there exist $i < j$ such that $T_i'' \leq T_j''$. The node injection ϕ from T_i'' into T_j'' can be extended, via the labels s, to the vertices of the branches of T_i and T_j. By construction, ϕ directly gives $T_i \leq_0 T_j$, and coherence property, as for words, gives path-composition. Thus $T_i \leq T_j$, contradicting the badness of S. This completes the proof of Lemma 107 and Theorem 106. □

□

5.5 NLC with Restricted Relabelling Functions

We can see $NLC_k^{\mathcal{F}}$ expressions as binary trees, where the leaves are labelled by $\bullet_i$, the nodes of degree 1 by $\circ_f$, and the nodes of degree 2 by χ_S. To fit in the framework of the previous section, we add an extra label to every node of such an $NLC_k^{\mathcal{F}}$ tree, to see χ_S and $\bullet_i$ as identity relabelling functions. More formally, we replace $\bullet_i$ with $(\varepsilon, \bullet_i)$, $\circ_f$ with $(f, \circ_f)$ and χ_S with (ε, χ_S). We call such a tree a *construction tree* for the graph corresponding to this $NLC_k^{\mathcal{F}}$ expression. Let T_G be a construction tree for a graph G. Let x be a vertex of G which corresponds to the leaf x' of T_G of label $\bullet_i$ and let y be an ancestor of x' in T_G. When we apply the operation corresponding to the node y of T_G to the vertex x, the colour of x, denoted by $c_x(y)$ is exactly $L(y, x')(i)$.

Lemma 108. *Let G and H be two NLC_k graphs and let T_G and T_H be $NLC_k^{\mathcal{F}}$ construction trees of G and H. If $T_G \leq T_H$, then $G \leq_i H$.*

Proof. Let ϕ be an injection from $V(T_G)$ into $V(T_H)$ which realizes $T_G \leq T_H$. The restriction of ϕ on the leaves of T_G can be seen as an injection from $V(G)$ into $V(H)$. Let x, y be two vertices of G, with x on the left of y in T. Then x and y are neighbours in G if and only if their least common ancestor in $V(T_G)$ is a node labelled by χ_S with $(c_x(x \wedge y), c_y(x \wedge y)) \in S$. As ϕ preserves labels, path composition and right/left order, this is the case if and only if $\phi(x)$ and $\phi(y)$ are neighbours in H. So $G \leq_i H$. □

Theorem 106 and Lemma 108 immediately give that if $\preceq$ is total on $\mathcal{F}$, then $NLC_k^{\mathcal{F}}$ is well-quasi-ordered by $\leq_i$. Moreover, since we can always add some extra vertex-labels, we obtain that:

Corollary 109. *$NLC_k^{\mathcal{F}}$ is ∞-well-quasi-ordered when $\preceq$ is total on $\mathcal{F}$.*

Let us now characterize the sets $NLC_k^{\mathcal{F}}$ which are well-quasi-ordered:

Theorem 110. *$\mathrm{NLC}_k^{\mathcal{F}}$ is well-quasi-ordered by $\leq_i$ if and only if $\preceq$ is total on $\mathcal{F}$.*

Proof. Assume that $\preceq$ is not total on $\mathcal{F}$, and let (f, g) be an incomparable pair for the relation $\preceq$. Let us show that for every $n \geq 1$, the decorated path G_n consisting of a path of length n with two pending vertices attached to each extremity is in $\mathrm{NLC}_k^{\mathcal{F}}$. The set $\{G_n | n \in \omega\}$ is clearly not well-quasi-ordered.

Assume first that $f = g$, that is $|\mathrm{Im}(f^2)| < |\mathrm{Im}(f)|$. Hence there exist $x, y \notin \mathrm{Im}(f^2)$, such that $f(x) = y$. To construct the decorated path G_n, start from two vertices labelled by y and one vertex labelled by x, and apply $\chi_{\{(x,y)\}}$ to form a path of length 2. Relabel by f. Observe that the two extremities of this path will never again be labelled by x or y since their labels will stay within $\mathrm{Im}(f^2)$. Add a vertex labelled by x, apply again $\chi_{\{(x,y)\}}$. This adds an edge between the middle vertex of the path and the new one. Then relabel by f, and keep on building the path up to the desired length. The point is that after any step, the extremity of the path is distinguished by its label from the other vertices. When the last vertex of the path has been added (with label x as usual), add two isolated vertices with label x for instance, and apply $\chi_{\{(x,x)\}}$, completing the graph G_n.

We can generalise this when f and g are distinct. An f-*class* is a subset S of $\{1, \ldots, k\}$ such that $|f(S)| = 1$ and which is maximum with respect to inclusion. Since $f \not\preceq g$, there exists an f-class disjoint from $\mathrm{Im}(g)$. Let x be one of its elements. Similarly, let y be in a g-class disjoint from $\mathrm{Im}(f)$. Let us prove by induction that for every n, we can build paths of length $2n$ where the last vertex is labelled by y and the other vertices are labelled in the set $\mathrm{Im}(f)$. We will therefore be able to build the decorated path G_n for arbitrarily large n, by adding two pending nodes on each extremity as in the previous case.

To start with, take a vertex $z \in \mathrm{Im}(f)$, add a vertex y, and apply $\chi_{\{(z,y)\}}$. Now assume that we have a path of length $2n$ where the last vertex is labelled by y and the other vertices by some elements of $\mathrm{Im}(f)$. Relabel by g. Observe that the last vertex is still distinguished from the rest. Add a vertex x. At this point, no other vertex has label x, since x is not in $\mathrm{Im}(g)$. Apply $\chi_{\{(x,y)\}}$. This constructs a path of length $2n+1$. Now relabel by f, add a vertex y and apply $\chi_{\{(y,x)\}}$ in order to get a path of length $2(n+1)$ which satisfies the induction hypothesis. □

This proof actually gives the following result:

Corollary 111. *If $\preceq$ is not total on $\mathcal{F}$, then $\mathrm{NLC}_k^{\mathcal{F}}$ contains arbitrarily large paths.*

Also, if $\mathrm{NLC}_k^{\mathcal{F}}$ contains arbitrarily large paths then $\mathrm{NLC}_k^{\mathcal{F}}$ is not 2-well-quasi-ordered, which completes the proof of the equivalences claimed in the introduction.

To sum-up the different results obtained in this section, we have proved that the following statements are equivalent:

– $\mathrm{NLC}_k^{\mathcal{F}}$ is well-quasi-ordered by $\leq_i$
– $\mathrm{NLC}_k^{\mathcal{F}}$ is ∞-well-quasi-ordered by $\leq_i$
– $\preceq$ is total on $\mathcal{F}$

– $\mathrm{NLC}_k^{\mathcal{F}}$ does not contain arbitrarily large paths.

5.6 Further Well-Quasi-Ordering Problems

As we have mentioned before, one important motivation for the notion of a 2-well-quasi-ordered class is that it can be described by a finite set of bounds [107].

Proposition 112. *Let I be a 2-well-quasi-ordered induced subgraph ideal. There are finitely many graphs in the complement $\overline{I}$ of I which are minimal with respect to the induced subgraph relation.*

Proof. By contradiction, we assume that the set B of minimal graphs in $\bar{I}$ is infinite. For every graph G in B, choose a vertex x, colour the neighbours of x red and the non-neighbours of x black, and delete x. Call the resulting bicoloured graph G'. The set $B' = \{G' | G \in B\}$ is infinite, and consists of graphs whose underlying graphs are in I, by minimality of the graphs in B. As I is 2-well-quasi-ordered, there exist two graphs G_1' and G_2' in B', such that G_1' is a coloured induced subgraph of G_2'. Hence G_1 is an induced subgraph of G_2, contradicting the fact that G_2 is in B. □

We call the minimal graphs in $\overline{I}$ the *forbidden graphs of* I, and denote by $F(I)$ the set of these graphs. The same argument shows that $I \cup F(I)$ is well-quasi-ordered when I is 2-well-quasi-ordered.

Proposition 112 implies that any 2-well-quasi-ordered induced subgraph ideal is polynomial time recognizable. This means in particular that for a set $\mathcal{F}$ totally quasi-ordered by $\leq$, the class $\mathrm{NLC}_k^{\mathcal{F}}$ is recognizable in polynomial time, by simply brute-force testing the existence of each forbidden subgraph.

The following question would give an answer to Pouzet's conjecture.

Conjecture 113. *If $\mathcal{G}$ is a 2-well-quasi-ordered induced subgraph ideal, there exists an ∞-well-quasi-ordered set $\mathrm{NLC}_k^{\mathcal{F}}$ which contains $\mathcal{G}$.*

This problem seems hard. One first step would be to show the following:

Conjecture 114. *No class of graphs closed under induced subgraph, and of unbounded clique-width, is 2-well-quasi-ordered.*

The next step would be to show that if indeed a subclass of NLC_k is 2-well-quasi-ordered, then it is contained in some ∞-well-quasi-ordered set $\mathrm{NLC}_{k'}^{\mathcal{F}}$, with k' possbibly larger than k.

Recall that clique-width and NLC-width are equivalent from the point of view of boundedness. We present the conjectures of this final section in terms of clique-width, the more familiar of the two, while we used NLC-width in our proofs for technical convenience.

Concerning classes of unbounded clique-width, the following problem (which generalises Conjecture 114) is of independent interest:

Problem 115. *Is it true that every class of graphs closed under induced subgraph, and of unbounded clique-width, is not well-quasi-ordered by the induced subgraph relation?*

Let us say that an ideal $\mathrm{I}' \subseteq \mathrm{I}$ is a *sub-ideal* of an ideal I, and a *strict sub-ideal* if $\mathrm{I}' \subsetneq \mathrm{I}$. To prove Conjecture 114, one can actually restrict to minimal such classes:

Proposition 116. *Let I be a 2-well-quasi-ordered induced subgraph ideal of unbounded clique-width. There exists a sub-ideal I' of I of unbounded clique-width such that every strict sub-ideal of I' is of bounded clique-width.*

Proof. Since I is well-quasi-ordered, every non empty collection of sub-ideals of I has a minimal element by Higman's Theorem. This fact applied to the collection of sub-ideals of I with unbounded clique-width yields the result. ☐

Thus the following conjecture implies Conjecture 114:

Conjecture 117. *No ideal $\mathcal{G}$ verifies all of the following properties:*
– *$\mathcal{G}$ has unbounded clique-width*
– *$\mathcal{G}$ has a finite number of forbidden graphs*
– *Every strict sub-ideal of $\mathcal{G}$ has bounded clique-width*

For further discussion on minimal ideals of unbounded clique-width, see [85] for instance.

Finally, let us mention a question which would extend further Pouzet's conjecture. The answer for cographs can be found in [121].

A quasi-order Q is a *better-quasi-order* if the class of countable ordinals labelled by Q is a well-quasi-order [104].

Conjecture 118. *Let $\mathcal{G}$ be a class of countable graphs. If the class of finite induced subgraphs $\mathcal{G}_{\mathrm{F}}$ of $\mathcal{G}$ is 2-well-quasi-ordered, then $\mathcal{G}$ is better-quasi-ordered for every better-quasi-ordered vertex-labelling.*

CHAPTER

6 Helly Circle Graphs

In this chapter, we prove that the Helly Circle Graphs are exactly the Diamond-free Circle Graphs. This characterisation yields an efficient recognition algorithm.

6.1 Introduction

A *circle graph* is a graph whose vertices can be associated to chords of a circle such that two vertices are adjacent if and only if the corresponding chords intersect.

Recently, circle graphs have received renewed attention in relation to the vertex-minor relation, pivot-minor relation, and rankwidth (see for instance [34] and [106]). Circle graphs indeed play a similar role with respect to vertex minor and rankwidth as planar graphs do with respect to minor and treewidth. Circle graphs were characterized by Bouchet [21] by three excluded vertex-minor, and by Geelen and Oum [73] by a finite list of forbidden pivot-minors. Another characterization of circle graphs was given by de Fraysseix [44].

In the following, we prove that a subclass of the circle graphs, namely the Helly circle graphs, are characterized with respect to circle graphs by one excluded induced subgraph: the diamond.

A *circle model* of a circle graph G is a function which associates to every $v \in V(G)$ the two endpoints of a chord in the unit circle $\mathcal{C}$. For convenience we only consider models where endpoints are pairwise disjoint.

A *sequence model* σ_G of a circle graph G is a circular sequence in which every element of $V(G)$ appears exactly twice according to the order in which we meet the chord endpoints on a clockwise walk around $\mathcal{C}$. Note that many circle models correspond to a given sequence model and that a circle graph G may have several sequence models, for example if G is disconnected.

A *subsequence* σ of σ_G, which we will denote by $\sigma \subseteq \sigma_G$, is a circular sequence obtained by deleting from σ_G the two occurrences of every $v \in X$, for some subset $X \subseteq V(G)$.

A family of geometric objects is said to have the *Helly property* if every pairwise intersecting subfamily shares a common point. Thus a circle model is *Helly* if every three pairwise intersecting chords intersect in a single point. A circle graph G is *Helly* if it has a Helly circle model.

The *diamond* is the graph obtained from K_4 by deleting an edge. The diamond clearly admits no Helly circle model, so every Helly circle graph is diamond-free. Our main result is that the converse holds, as conjectured by Durán [51] (see also [8, 52]).

Theorem 119. *Every diamond-free circle graph* G *is a Helly circle graph.*

The rest of this chapter is devoted to the proof of Theorem 119.

This characterization ensures that the complexity of Helly circle graphs recognition is at most that of circle graphs recognition. Using the $O(n^2)$ recognition algorithm of circle graphs by Spinrad [119] yields a $O(n^2)$ recognition algorithm for Helly circle graphs (see the concluding remarks for a short discussion on algorithmic issues). Using a recent improvement of Gioan *et al.* [54], Helly circle graphs can even be recognised in quasi-linear time.

6.2 Computing a Helly Circle Model

Consider a diamond-free circle graph G and one of its sequence models σ_G. In the following, we make a slight abuse of notation in denoting (G, σ_G) by G. We prove Theorem 119 by showing that G admits a Helly circle model. We basically grow a Helly Circle model step by step. Formally, we need the following definitions.

An induced subgraph H of G is *convex* if for every subsequence (a, b, c, c, b, a) of σ_G, $\{a, c\} \subseteq V(H)$ implies that $b \in V(H)$. This corresponds exactly to the geometric notion of convexity, as in a subsequence (a, b, c, c, b, a) the chord b lies between the chords a and c.

A clique K_t is *non-trivial* if and only if $t \geq 2$. An induced subgraph H of G is *clique maximal* if every non-trivial maximal clique of H is a maximal clique of G. An induced subgraph H of G is *almost component maximal* if at most one (connected) component[1] of H is not a maximal component of G. An induced subgraph H of G is *convenient* if it is convex, clique maximal, and almost component maximal. Given an induced subgraph H of G, we denote by σ_H the sequence model in σ_G induced by $V(H)$. Also, a *mixed Helly model* of (G, H) is a circle model of G where the induced circle model of H is Helly.

Lemma 120. *Given a convenient subgraph* H *of* G, *for every vertex* $u \in V(G) \setminus V(H)$ *the neighbours of* u *in* H *are pairwise non-adjacent. Equivalently, for every* x *and* $y \in V(H)$, $(u, x, y, u, x, y) \not\subseteq \sigma_G$.

1. In the following, we write "component" instead of "connected component".

Proof. Since H is clique maximal, the neighbourhood $N_H(u)$ of u in H does not contain a non-trivial maximal clique of H. Thus if two vertices of $N_H(u)$ were adjacent (say x and y), then these vertices would belong to a maximal clique of H with at least one vertex z not adjacent to u. This is impossible since u, x, y, and z would induce a diamond. □

Lemma 121. *Consider a convenient subgraph* H *of* G*, and a vertex* $u \in V(G) \setminus V(H)$*. It is possible to denote the two occurrences of* u *in* σ_G *by* u^- *and* u^+ *so that for every* $x \in V(H) \setminus N_G(u)$ *we have* $(u^-, x, x, u^+) \subseteq \sigma_G$.

Proof. If there exists a vertex $x \in V(H) \setminus N_G(u)$, choose u^- and u^+ in such a way that $(u^-, x, x, u^+) \subseteq \sigma_G$, otherwise choose them randomly. For every $y \in V(H) \setminus (\{x\} \cup N_G(u))$, we observe that we have $(u^-, y, y, u^+) \subseteq \sigma_G$ and not $(y, u^-, u^+, y) \subseteq \sigma_G$. Indeed, in the second case we would have $(y, u^-, x, x, u^+, y) \subseteq \sigma_G$, contradicting the convexity of H. □

Theorem 119 follows from the fact that (G, G) admits a mixed Helly model. To show that (G, G) admits a mixed Helly model, we first note that the empty graph $G[\emptyset]$ is a convenient subgraph of G and that $(G, G[\emptyset])$ admits a mixed Helly model. Then we prove that given a convenient subgraph $H \subsetneq G$ such that (G, H) admits a mixed Helly model, there exists a convenient subgraph H' of G verifying $H \subsetneq H'$ and such that (G, H') admits a mixed Helly model.

To construct such a subgraph H' we need the following result.

Lemma 122. *Given any proper convenient subgraph* H *of* G*, there exists a vertex* $u \in V(G) \setminus V(H)$ *such that* $G[V(H) \cup \{u\}]$ *remains convex. Furthermore, if* H *has a component which is a proper subgraph of a component* C *of* G*, then there exists such a vertex* u *in* C.

Proof. Let $\prec$ be the relation on $V(G) \setminus V(H)$ such that $u' \prec u$ if (x, u, u', u', u, x) is a subsequence of σ_G for some $x \in V(H)$. It is easy to see that $\prec$ is anti-symmetric and transitive. Clearly, for any maximal u for $\prec$, $G[V(H) \cup \{u\}]$ is convex. Moreover, by convexity, if $u' \prec u$ and u' is adjacent to some $y \in V(H)$, then u is adjacent to y. Thus if H has a component that is a proper subgraph of a component C of G, then $\prec$ has a maximal element in C. □

We now distinguish the case where every component of H is a component of G and the case where one connected component of H is a proper subgraph of a component of G. In the first case we consider a vertex $u \in V(G) \setminus V(H)$ such that $G[V(H) \cup \{u\}]$ is convex. Such a vertex exists by Lemma 122. It is clear that $H' = G[V(H) \cup \{u\}]$ is almost component maximal. Since the non-trivial cliques of H' are exactly the non-trivial cliques of H, we have that H' is clique maximal and thus convenient. The non-trivial cliques of H' are exactly the non-trivial cliques of H, thus the mixed Helly model of (G, H) is a mixed Helly model of (G, H').

Thus we can assume that H has a component which is a proper subgraph of a component of G.

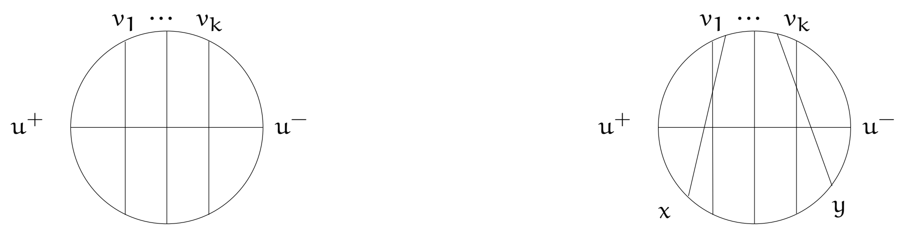


Figure 6.1: Left: A vertex $u \in V(G) \setminus V(H)$ and its neighbourhood in H. Right: x and y are respectively a predecessor and a successor of u.

Lemma 122 ensures that there exists at least one vertex $u \in V(G) \setminus V(H)$ such that $G[V(H) \cup \{u\}]$ is convex and almost component maximal (i.e. u has neighbours in H). The following result states that the neighbours of such a vertex u are linearly ordered.

Lemma 123. *Given a vertex $u \in V(G) \setminus V(H)$ adjacent to some vertex in H and such that $G[V(H) \cup \{u\}]$ is convex, Lemma 120 allows us to index the neighbours of u in H as $v_1, \ldots, v_k$, with $k \geq 1$, in such a way that $(u^+, v_1, \ldots, v_k, u^-, v_k, \ldots, v_1) \subseteq \sigma_G$ (see Figure 6.1). Furthermore, every common neighbour x of u and some vertex v_i, with $1 \leq i \leq k$, is adjacent to exactly one vertex in $N_H(u)$, namely v_i, and x satisfies either:*

(P) $v_i = v_1$ and $(u^+, v_1, x, u^-, v_1, x) \subseteq \sigma_G$, or

(S) $v_i = v_k$ and $(u^+, x, v_k, u^-, x, v_k) \subseteq \sigma_G$.

It is clear that $x \in V(G) \setminus V(H)$. In the first case (P), we call x a predecessor *of u, while in the second case (S) we say that x is a* successor *of u.*

Proof. Consider a vertex x adjacent to u and to some vertex v_i. Lemma 120 implies that $x \in V(G) \setminus V(H)$, and since G is diamond-free the vertex x is not adjacent to any v_j with $j \neq i$, otherwise x, u, v_i and v_j would induce a diamond. We now prove that $(u^+, x, v_i, u^-, x, v_i) \not\subseteq \sigma_G$, for $1 \leq i < k$. Since the subgraph H is almost component maximal and since u lies in the same component of G as v_i and v_{i+1}, the vertices v_i and v_{i+1} are in the same component of H. Thus v_i has at least one neighbour $z \neq v_{i+1}$ in H. As the vertex x is adjacent to v_i, Lemma 120 implies that x is not adjacent to z. So if $(u^+, x, v_i, u^-, x, v_i) \subseteq \sigma_G$, then $(z, x, v_{i+1}, v_{i+1}, x, z) \subseteq \sigma_G$, which contradicts the convexity of H. We could similarly prove that $(u, v_i, x, u, v_i, x) \not\subseteq \sigma_G$, for $1 < i \leq k$. This concludes the proof of Lemma 123. □

Lemma 124. *Consider a vertex $u \in V(G) \setminus V(H)$ adjacent to some vertex in H, such that $G[V(H) \cup \{u\}]$ is convex. For every predecessor (resp. successor) x of u, $G[V(H) \cup \{x\}]$ is convex and u is a successor (resp. a predecessor) of x.*

Proof. Assume that $G[V(H) \cup \{x\}]$ is not convex. There exist two vertices $y \in V(G) \setminus V(H)$ and $z \in V(H)$ such that $(x^+, x^-, y, z, z, y) \subseteq \sigma_G$. By convexity of $G[V(H) \cup \{u\}]$, the vertex u is adjacent to y, as otherwise we would have $(u^+, u^-, y, z, z, y) \subseteq \sigma_G$.

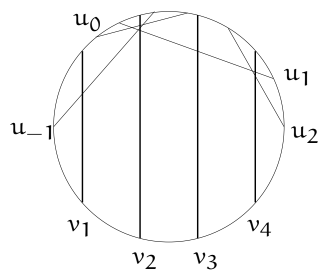


Figure 6.2: A sequence $S = (u_{-1}, u_0, u_1, u_2)$ allowing to extend H

Let us index the neighbours of u in H as $v_1,\dots,v_k$, with $k \geq 1$, in such a way that $(u^+, v_1,\dots,v_k, u^-, v_k,\dots,v_1) \subseteq \sigma_G$. Recall that by the definition of x, the vertices u and x have a common neighbour, that is v_1 (resp. v_k) by Lemma 123. Since u, x, y, and v_1 (resp. v_k) cannot induce a diamond, y and v_1 (resp. v_k) are not adjacent. This contradicts the convexity of H, as we would have (v_1, v_1, y, z, z, y) or $(v_k, v_k, y, z, z, y) \subseteq \sigma_G$. Finally it is clear by Lemma 123 that u is a successor (resp. a predecessor) of x. $\square$

By Lemma 122, there exists a vertex u_0 in $V(G) \setminus V(H)$ such that $G[V(H) \cup \{u_0\}]$ is convex and almost component maximal (*i.e.* u_0 has neighbours in H). We now define the vertices $u_i \in V(G) \setminus V(H)$ with $-p \leq i \leq q$, in such a way that $(u_i)_{-p \leq i \leq q}$ is the longest sequence containing u_0 with the property that u_i is a successor of u_{i-1} for every $i \in \{-p+1,\dots,q\}$. Given the definition of u_0, Lemma 124 implies that all the vertices u_i have a neighbour in H and have the property that $G[V(H) \cup \{u_i\}]$ is convex.

Informally, we are going to prove that the vertices which are neighbours of at least one vertex u_i have the good ordering property ensured by Lemma 123.

Lemma 123 allows to define an increasing sequence $(n_i)_{-p-1 \leq i \leq q}$ and to index the neighbours v_j of vertices in $(u_i)_{-p \leq i \leq q}$, with $n_{-p-1} \leq j \leq n_q$, in such way that for every $i \in \{-p,\dots,q\}$, the neighbours of u_i in H are exactly the vertices v_j with $n_{i-1} \leq j \leq n_i$. Lemma 123 also implies that $(u_i^+, v_{n_{i-1}},\dots,v_{n_i}, u_i^-, v_{n_i},\dots,v_{n_{i-1}}) \subseteq \sigma_G$, and thus the vertices v_j are such that $(v_{n_{-p-1}}, v_{1+n_{-p-1}},\dots,v_{n_q}, v_{n_q},\dots, v_{1+n_{-p-1}}, v_{n_{-p-1}}) \subseteq \sigma_G$. In other words, the vertices v_j are represented by parallel chords, as in Figure 6.2).

Finally, we define the subgraph H' to which we are going to extend the mixed Helly model:

$$H' = G[V(H) \cup \{u_i \mid -p \leq i \leq q\}]$$

Let us prove that H' is convenient and admits a Helly circle model.

Lemma 125. *For every* $i \in \{-p,\dots,q\}$, *the successors of* u_i *are the vertices* u_k *such that* $k > i$ *and* $n_{k-1} = n_i$. *For every* $i \in \{-p,\dots,q\}$, *the predecessors of* u_i *are the vertices* u_k *such that* $k < i$ *and* $n_k = n_{i-1}$.

Proof. Lemma 123 implies that the relation "successor" on the set of vertices intersecting both u_i and n_{n_i} is a total order. Thus, if a successor of u_i is missing in the sequence $(u_i)_{-p\leq i\leq q}$, then one can easily insert it and increase the length of the sequence, which contradicts the maximality of $(u_i)_{-p\leq i\leq q}$. □

Lemma 126. H' *is convenient.*

Proof. The graph H' has as many components as H; thus it is almost component maximal. Let us show that H' is clique maximal. Assume by contradiction that there exists a vertex $x \in V(G)\setminus V(H')$ adjacent to both vertices of an edge $ab \in E(H')$. If $ab \in E(H)$, then this would contradict Lemma 120. If $a = u_i$ and $b = v_j$, with $-p \leq i \leq q$ and $n_{i-1} \leq j \leq n_i$, then the vertex x is a successor or a predecessor of u_i (by Lemma 123) and it thus should belong to H' (by Lemma 125). Hence $a = u_i$ and $b = u_j$, with $-p \leq i < j \leq q$. As u_i and u_j are adjacent, these two vertices are also adjacent to v_{n_i}. If x is adjacent to v_{n_i}, then x should be a vertex u_k, and thus belong to H', by Lemma 125. Conversely, if x is not adjacent to v_{n_i}, then x, a, b and v_{n_i} would induce a diamond. In both cases, we obtain a contradiction.

Finally let us show that H' is convex. By contradiction, assume that there exist $x \in V(G)\setminus V(H')$ and $a,b \in V(H')$ such that $(a,a,x,b,b,x) \subseteq \sigma_G$. By convexity of H and $H\cup\{u_i\}$ for every $i \in \{-p,\dots,q\}$, both a and b belong to $V(H')\setminus V(H)$, say $a = u_i$ and $b = u_j$, with $-p \leq i < j \leq q$. Since u_i and u_j are not adjacent, we have that $(v_{n_i},u_i,v_{n_i},u_i,u_j,v_{n_j},u_j,v_{n_j}) \subseteq \sigma_G$. Thus we have either

- $(x,v_{n_i},u_i,v_{n_i},u_i,x,u_j,v_{n_j},u_j,v_{n_j}) \subseteq \sigma_G$ or
- $(v_{n_i},x,u_i,v_{n_i},u_i,x,u_j,v_{n_j},u_j,v_{n_j}) \subseteq \sigma_G$ or
- $(v_{n_i},u_i,v_{n_i},u_i,x,u_j,v_{n_j},u_j,x,v_{n_j}) \subseteq \sigma_G$

which respectively contradicts the convexity of H, $H\cup\{u_i\}$, and $H\cup\{u_j\}$. □

Lemma 127. *There is a mixed Helly model of* (G,H').

Proof. We consider the Helly circle model of H and extend it to H'. Lemma 123 allows us to distinguish one extremity of v_j, for every j; the distinguished extremity v_j^* being such that $(u_i^+,v_j^*,u_i^-,v_j) \subseteq \sigma_G$ for every vertex u_i crossing v_j. We extend the Helly circle model of H by successively processing the chords $u_{-p},\dots,u_q$ in this order. For every $i \in \{-p-1,\dots,q\}$ let $H_i = G[V(H)\cup\{u_k \mid -p \leq k \leq i\}]$. At each step we extend a Helly circle model of H_{i-1} to a Helly circle model of H_i. Additionally, xe construct these Helly circle models in such a way that for every i, the following property holds:

(*) the intersection point of u_i and v_{n_i} lies strictly between the point $v_{n_i}^*$ and the intersection of the chord v_{n_i} with the abstract chord $[v_{n_i-1}^*,v_{n_i+1}^*]$ (see Figure 6.3).

Assume that we have already processed the chords up to u_{i-1} (see Figure 6.3 Step 1). Since $\sigma_{H_{i-1}} \subseteq \sigma_{H_i}$, it is easy to draw the chord u_i in order to intersect the desired chords. We slightly move this chord u_i in order to fulfil both (*) and the Helly property, as follows. If $u_i = u_{-p}$, then the neighbours of u_{-p} in H_{-p} are pairwise non-adjacent (by Lemma 120),

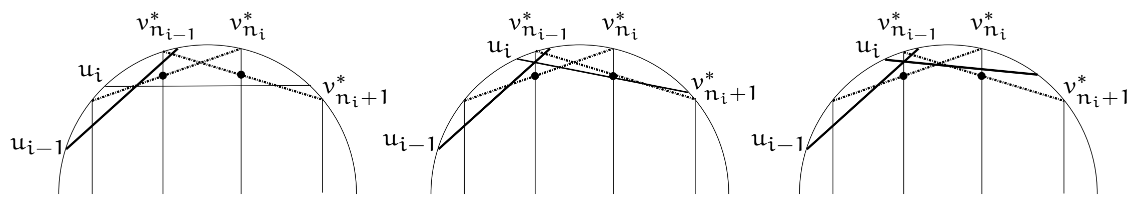


Figure 6.3: Processing the chord u_i. The dashed lines represent the abstract constraints.

so the Helly property follows immediately. Thus, we just have to move u^-_{-p} close enough to v^*_{n-p} in order to fulfil (*). This is possible since there is no chord extremity in between v^*_{n-p} and u^-_{-p} in $\sigma_{H_{-p}}$. If $u_i \neq u_{-p}$, the Helly circle model of H_{i-1} fulfils (*). This ensures that we can move the chord u_i in order to intersect u_{i-1} and $v_{n_{i-1}}$ exactly at their intersection point (see Figure 6.3 Step 2). The neighbourhood of u_i in H_i induces a graph with a unique non-trivial maximal clique (the clique with vertex set $\{v_{n_{i-1}}\} \cup \{u_k \mid k < i \text{ and } n_k = n_{i-1}\}$), so the circle model of H_i fulfils the Helly property. Finally if (*) is not satisfied, then we just move u^-_i close enough to $v^*_{n_i}$ by rotating u_i around the intersection point of the clique. (see Figure 6.3 Step 3). This concludes the proof of Lemma 127 and Theorem 119. □

6.3 Concluding Remarks

The first polynomial-time algorithms for circle graph recognition were independently given by Bouchet [20], Naji [103], and Gabor *et al.*. [69]. The latter was improved by Spinrad [119], who showed that the recognition of circle graphs can be done in $O(n^2)$ time and that a circle model can be computed within the same time bound. Given a circle model of a circle graph G, the graph induced by the neighbourhood of a vertex v in G is a permutation graph. Moreover, a permutation model of $G[N(v)]$ can be computed in $O(n)$ when a circle model is known. One can easily check in $O(n)$ time whether a permutation graph is P_3-free (i.e. is a disjoint union of complete graphs) using its permutation model. Thus one can check whether a circle graph G given by a circle model is diamond-free in time $O(n^2)$, by checking for every vertex v if the permutation graph $G[N(v)]$ is P_3-free. In consequence we have:

Proposition 128. *Helly circle graphs can be recognized in time* $O(n^2)$.

Actually, the test for an induced diamond can even be carried out in linear time $O(n+m)$ with an adequate data structure. This means that the complexity of Helly circle graph recognition is at most the complexity of circle graph recognition. Recently, Gioan *et al.* [53] produced an almost linear-time algorithm for computing Cunningham's split decomposition (see [36]), running in $O((n+m)\alpha(n+m))$ time, where α is the inverse

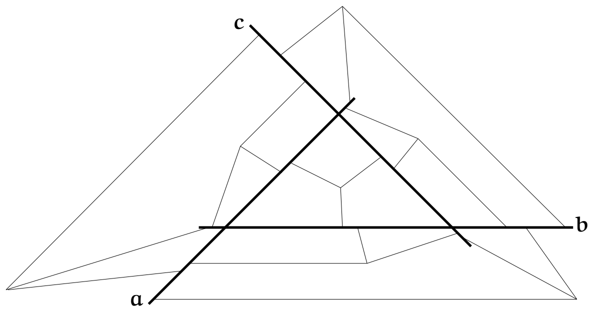


Figure 6.4: A set of segments inducing a diamond-free graph without a Helly model of intersecting segments.

of Ackerman's function. They show that this algorithm gives a $O((n+m)\alpha(n+m))$ algorithm for circle graph recognition [54], which provides a circle model within the same time bound. With the induced diamond test carried out in linear time on the output of their algorithm, this yields a $O((n+m)\alpha(n+m))$ recognition algorithm for Helly circle graphs.

A circle graph is the intersection graph of chords in a circle. One may wonder whether Theorem 119 extends to intersection graphs of segments. This is not the case. The intersection graph of the segments in Figure 6.4 is diamond-free, but one can check that this graph has no Helly intersection model (here the segments $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ are not concurrent).

Another question is whether such a characterisation can be generalised to higher dimensions. A graph is a d-*dimensional sphere graph* if it is an intersection graph of $(d-1)$-dimensional disks which borders lie on a d-dimensional sphere. We wonder whether Theorem 119 can be generalised in the following way (Theorem 119 corresponds to the case $d=2$):

Question 129. *Does every* d*-dimensional sphere graph which is* $(K_{d+2}-e)$*-free admit a Helly embedding ?*

The reverse is not true unless we forbid that d disks intersect on a segment. Indeed, with d distinct disks pairwise intersecting on a given segment, and two parallel disks orthogonal with these d disks, one obtains a $K_{d+2}-e$ Helly model. In the definition of a Helly circle model, one requires the chords to be distinct. The proper generalisation of this restriction is to forbid that d disks intersect on a segment.

Part III: Parameterized Algorithms

In Chapter 7 we prove that MULTICUT is FPT when parameterized by the solution size. MULTICUT was one the last important natural "FPT or not FPT" open problems. This chapter is based on [37] and [40], joint work with Nicolas Bousquet, Christophe Paul, Anthony Pérez and Stéphan Thomassé.

The proof is quite involved. The general idea is rather simple, but several steps are highly technical, so Section 7.2 will give the general outline. The actual proof spans Section 7.4 to Subsection 7.6.7. In Section 7.7 we claim that the same ideas work for the Vertex-Multicut problem, although we do not write down the details. In Subsection 7.8.1 we briefly discuss why the overall algorithm runs in single-exponential time. In Subection 7.8.2 we compare our proof with Marx and Razgon's independent proof in [100]. Finally, in Subsection 7.8.3 we give insights for a different approach of the MULTICUT problem.

Chapter 8 is mostly devoted to a fast parameterized algorithm for finding directed trees with many leaves. This chapter is based on [39], a joint work with Gregory Gutin, EunJung Kim and Anders Yeo.

Prior to this work, Kneis *et al.* had designed a simple and efficient $O^*(4^k)$ algorithm to decide the existence of an out-tree with at least k leaves in [88], improving over a long list of algorithms (even in the undirected case). Their branching algorithm was simple: grow a tree starting with just a root. As long as the number of leaves in the partial tree is less than k, pick a leaf, and decide whether it will be a leaf or an internal node in the solution. The "Leaf" step can occur at most k times. If we can ensure that, in the "Internal" step, the partial tree gains at least a leaf, then this "Internal" step can occur at most k times as well, giving a branching algorithm running in time $O^*(2^{k+k})$. The only case where the "Internal" step does not increase the number of leaves in the partial tree is when we branch over a current leaf which has only one out-neighbour outside the partial tree. In this case, we can actually assume that this single out-neighbour will be internal as well in the final

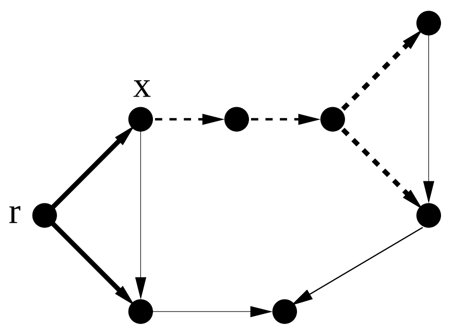


Figure 6.5: The bold arcs denote the current tree T. When branching over the current leaf x, if we decide that x will be internal, we can directly add the dashed subtree rooted at x to our partial tree. This ensures that we gain at least one leaf with each "Internal" step.

solution (Lemma 171), and proceed as depicted in Figure 6.5 by directly adding a whole subtree. This ensures a gain with each branching step.

This algorithm does not run quicker than 4^k only when almost all branching steps gain a single leaf during an "Internal" step, *i.e.* the leaf under consideration has out-degree two outside the partial tree. Our goal is to find a way to prune the branching tree when this happens too often. If we could ensure that a constant fraction of the branching steps are performed on current leaves with at least three out-neighbours outside the current tree, we would get an algorithm running in time $O(c^k)$ with $c < 4$. Likewise, if we could spare a constant fraction of the branching steps (*i.e.* safely prune branches from our branching tree), we would improve the running time. Section 8.2 is devoted to proving the correctness of such a pruning.

The key is step (4.2) in algorithm $\mathcal{B}(D, T, L)$, which essentially states that in the tight cases, when deciding that the current leaf x will be a leaf in our solution, we can assume that some other vertex p_0 will also be a leaf. Vertex p_0 does not belong to the current tree, so our algorithm decides to set as leaves vertices not necessarily explored yet. This somewhat counterintuitive technique allows us to break the $O^*(4^k)$ barrier, which in turns gives us the first algorithm to break the 2^n barrier for the unparameterized problem. To the best of our knowledge, this is still the only algorithm better than the straightforward brute-force algorithm for directed MaxLeaf problems.

CHAPTER

7 Multicut is FPT

7.1 Introduction

Bearing in mind the connectivity techniques from Chapter 4, we are ready to tackle the fixed-parameter tractability of the general multicut problem:

MULTICUT:
Input: A graph G, a set of requests R, an integer k.
Parameter: k.
Output: TRUE if there is an (edge)-multicut of size at most k, otherwise FALSE.

The status of this problem was one of the crucial open questions in parameterized complexity. The main result of this chapter is to provide an FPT algorithm for MULTICUT. We present our work in terms of edge-multicut rather than vertex-multicut, and Section 7.7 gives some hints to translate the algorithm to deal with vertex-multicuts.

Marx and Razgon independently found a proof of the fact that MULTICUT is FPT, with a rather different approach, see [100]. Subsection 7.8.2 provides a brief comparison between our work and theirs.

7.2 Detailed Outline of the Proof

Some parts of the proof are technically very involved. This section provides a detailed outline of the proof, underlying the structure of the main results and the reasons behind the main definitions. For formal definitions and statements, and for complete proofs, the reader is referred to the following sections.

A Vertex-Multicut. First of all, we can assume by iterative compression that a vertex-multicut Y of size $k+1$ is given, and that a solution must split Y. This is expressed in

Lemma 147. This vertex-multicut Y gives a first layer of structure to an instance: we can focus on the Y-components, *i.e.* the connected components of the graph where vertices of Y have been removed.

Setting the number of multicut-edges per component. The number of Y-components is bounded in k, considering that all connected components of $G \setminus Y$ which are adjacent to a single given vertex $y \in Y$ form a single Y-component. So, we can branch to decide how many edges of the solution lie in each Y-component.

Half-requests. No request is contained inside a Y-component with two or more attachment vertices, so we can simulate a request (u,v) with several half-requests (u,y,v), where $y \in Y$ is an attachment vertex of both the component $C(u)$ of u and of the component $C(v)$ of v. Cutting a half-request (u,y,v) means cutting all paths between u and v which go through y.

The goal: 2-SAT. Half-requests give a simpler structure to the multicut problem: cutting a half-request (u,y,v) is equivalent to either separating u from y in $C(u)$, or separating v from y in $C(v)$. We will express this "or" through 2-SAT clauses once we manage to express in a simple way whether the solution separates u from y. The remaining of the proof is devoted to simplifying the structure of the instance until we can express with 2-SAT variables whether the solution separates u from y.

Focus on 2-components. We first reduce Y-components with three or more attachment vertices in Lemma 150. Now, Y-components have either two attachment vertices (2-components) or one attachment vertex (cherries). Connectivity tools presented in Section 7.4 are essentially sufficient to deal with cherries. The complexity of the problem mostly lies in the existence of 2-components.

In order to give a better structure to 2-components we compute in Lemma 152 a particular path between its two attachment vertices: the backbone, which has the following property. The multicut must contain exactly one edge in the backbone. The set of multicuts is thus linearly partially ordered, according to the edge of the backbone they use. The goal is now to simplify the structure of the instance so that the multicuts that separate a vertex u from an attachment vertex y of $C(u)$ form an initial (or final) section of this linear order. Indeed, the fact that the solution belongs to an initial or final section of a linear order can be easily expressed with a 2-SAT variable.

BACKBONE MULTICUT. The instance as reduced up to this point fits the first intermediate variant, COMPONENT MULTICUT, defined in Section 7.5. We introduce in Section 7.6 the second intermediate problem BACKBONE MULTICUT. We need this more general problem than COMPONENT MULTICUT, so that we can enrich instances. Considering half-requests is of major importance in our proof, but the presence of 2-SAT clauses is necessary only for Lemma 158, and could possibly be avoided with a slightly different proof.

Lemonizing 2-components. Through Lemma 158 in Subsection 7.6.5 we reduce 2-components so that each vertex of the backbone becomes a cut-vertex of the 2-components. An example is drawn in Figure 7.4. This is the first highly technical part of the proof. The components now consist in a sequence of lemons with cherries attached to the backbone.

Linearly ordering the set of multicuts. In Subsection 7.6.6 we perform a complete linearisation of the set of multicuts. This can be seen as the core of our approach. We define a meaningful partial order $\preceq$ on multicuts, such that the set of multicuts can be partitioned into a number bounded in k of parts totally ordered by $\preceq$ by Dilworth's Theorem. This is Lemma 160.

Reducing 2-components to a backbone. The linear order $\preceq$ on the set of all multicuts allows us to move terminals of cherries to the backbone in Lemma 161. In an component with no cherries left, we manage to reduce the sequence of lemons to just the backbone in Theorem 163. This is the second highly technical part of the proof.

Reduction to 2-SAT. At this point, the 2-components are just paths. Only cherries attached to vertices of Y remain, and they have a bounded number of meaningful cuts, thanks to the connectivity tools from Section 7.4. These connectivity results are of the same flavour as those presented in Chapter 4. Thus, the cherries attached to vertices of Y are easily dealt with in Lemma 164. With an instance consisting of a subdivision of a graph with at most k edges, we easily express with 2-SAT variables in Theorem 165 whether a given vertex is separated from a given vertex of Y. We end up (after a heavy dose of branching throughout the proof) with a 2-SAT instance, which is polynomially solvable.

Complexity and Programmability. The overall algorithm is single exponential. It should be quite difficult to effectively implement the algorithm though, the whole proof being very involved. Finding a simpler proof leading to a simpler algorithm would be very interesting.

7.3 Preliminaries

7.3.1 Equivalence with Cluster Deletion variants

Before turning to the proof that MULTICUT is FPT, we prove in this subsection a claim mentioned in Chapter 1: MULTICUT is FPT-equivalent to CLUSTER EDITION WITH FREE EDGES, which consists in deciding whether a graph can be edited (through edge deletions and additions) into a cluster graph (a vertex-disjoint union of cliques), with cost at most k, where the edition of free edges has cost 0, while the edition of other edges has cost 1. Its equivalence with MULTICUT has been mentioned several times in the literature, we give a short proof in this subsection.

CLUSTER EDITION WITH FREE EDGES:
Input: A graph $G = (V, E)$, a set of free edges $E_0 \subseteq V * V \setminus E$, an integer k.
Parameter: k.
Output: TRUE if G can be edited into a cluster graph with cost at most k, otherwise FALSE.

Note that it does not matter whether free edges are edges or non-edges, we assume that $E_0 \subseteq V * V \setminus E$ for convenience. CLUSTER EDITION WITH FREE EDGES is actually FPT-equivalent to CLUSTER DELETION WITH FREE EDGES, which consists in deciding whether a graph can be edited into a cluster graph with cost at most k *using only deletions*:

> CLUSTER DELETION WITH FREE EDGES:
> **Input**:A graph $G = (V, E)$, a set of free edges $E_0 \subseteq E$, an integer k.
> **Parameter**: k.
> **Output**: TRUE if G can be edited, through edge-deletions only, into a cluster graph with cost at most k, otherwise FALSE.

Lemma 130. MULTICUT *is FPT-equivalent to* CLUSTER DELETION WITH FREE EDGES.

Proof. The clusters correspond to the connected components separated by the multicut, and free edges are edited to fill components into cliques. More formally: let $(G = (V, E)$, R, $k)$ be an instance of MULTICUT. We define $G' = (V, V * V \setminus R)$, and $E_0 = V * V \setminus R \setminus E$. In other words, requests of G are non-edges in G', edges of G are cost one edges in G', and the rest are free edges in G'.

A set of edges F is a k-multicut of G if and only if G' can be edited into a cluster graph by deleting F and the appropriate free edges (those whose endpoints lie in different connected components of $G \setminus F$). Indeed, the fact that a request of G is a non-edge in G' forces its endpoints to end up in different clusters in G' after edition.

Conversely, let $G' = (V, E')$, E_0, k be an instance of CLUSTER DELETION WITH FREE EDGES. Let $G = (V, E' \setminus E_0)$ and $R = V * V \setminus E'$. Then G' is editable into a cluster graph with cost at most k using only deletions if and only if the set F of cost 1 edges deleted from G' is a k-multicut of (G, R). □

Lemma 131. CLUSTER EDITION WITH FREE EDGES *is FPT-equivalent to* CLUSTER DELETION WITH FREE EDGES.

Proof. To transform an Edition instance into a Deletion instance, we add all non-edges to the graph, giving them cost 0, and add a gadget corresponding to each such edge to make sure that we account for the proper cost. More formally, let $(G = (V, E), E_0 \subseteq V * V \setminus E, k)$ be an instance of CLUSTER EDITION WITH FREE EDGES. Let G' be the graph whose vertices are: vertices of V, plus two new vertices x_u^{uv} and x_v^{uv} for each non-edge (u,v) of cost 1 in G. All elements of $V * V$ are edges of G', with the same cost as in G for edges of G, and with cost 0 for non-edges of G. For every non-edge (u,v) of cost 1 in G, u is adjacent to x_u^{uv} with cost 1, v is adjacent to x_v^{uv} with cost 1, x_u^{uv} and x_v^{uv} are not adjacent, and are each adjacent to all other vertices in $V(G')$ with edges of cost 0. The original instance (G, E_0, k) can be cluster-edited with cost at most k if and only if the new instance (G', E_0', k) can be cluster-deleted with cost at most k. The x_u^{uv} and x_v^{uv} gadget makes sure that we properly count the cost for the edge (u,v). Indeed, in G' we can delete the edge (u,v) at no cost, reverting to the original configuration in G, and vertex x_u^{uv} (resp x_v^{uv}) will end up in the cluster of u (resp. v) with no extra cost. If we keep the edge (u, v) though, we have to pay a price of one to delete either (u, x_u^{uv}) or (v, x_v^{uv}) as (x_u^{uv}, x_v^{uv}) is not an edge. This price was the correct price to add the edge (u, v) in G.

Conversely, let $(G' = (V', E'), E'_0 \subseteq E', k)$ be an instance of CLUSTER DELETION WITH FREE EDGES. We modify this graph so that only deletions can be useful. Let G be the graph obtained from G' by adding, for each non-edge $(u, v) \notin E$, $\frac{k}{2}$ new vertices $x_u^{uv}(i)$ for $i = 1, \ldots, \frac{k}{2}$ and $\frac{k}{2}$ new vertices $x_v^{uv}(i)$ for $i = 1, \ldots, \frac{k}{2}$. For every $i = 1, \ldots, \frac{k}{2}$, we add an edge of cost 1 between u and $x_u^{uv}(i)$ and between v and $x_v^{uv}(i)$, and non-edges of cost 1 between u and $x_v^{uv}(i)$ and between v and $x_u^{uv}(i)$. Finally, we add edges of cost 0 between $x_u^{uv}(i)$ (resp. $x_v^{uv}(i)$) and every vertex in $V(G) \setminus \{u, v\}$. The original instance (G', E'_0, k) satisfies CLUSTER DELETION WITH FREE EDGES if and only if the new instance (G, E_0, k) satisfies CLUSTER EDITION WITH FREE EDGES. Indeed, adding in G a non-edge between vertices in $V(G')$ would make it too costly to edit gadget vertices into clusters. Also, if we do not add in G a non-edge between vertices in $V(G')$, gadget vertices $x_u^{uv}(i)$ (resp. $x_v^{uv}(i)$) can be edited into the cluster of u (resp. v) at no cost. □

7.3.2 Reductions, branchings and invariants

Let us now turn to the proof of the fixed-parameter tractability of the general MULTICUT problem. In this chapter, we study MULTICUT variants with additional constraints on the deleted edges. In the original MULTICUT problem, we can delete a set of k edges without restrictions, but in some more constrained versions we must delete a prescribed number of edges on some particular paths. The total number of deleted edges is called *deletion allowance* of the multicut problem. We will make extensive use of the term *bounded* which always implicitly means bounded in terms of the deletion allowance. Also, when speaking of FPT time, we always mean $O(f(d)n^c)$ where c is a fixed constant and d is the deletion allowance. In the algorithm we will perform reductions and branchings, and we use invariants to bound the total running time.

Reductions. These are computations where the output is a new instance which is equivalent to the original instance with respect to the existence of a solution. One of the most natural reductions concerns irrelevant requests, *i.e.* a request xy such that every k-multicut of $R \setminus xy$ actually cuts x from y, where R is the set of requests. If one can certify that a request xy is irrelevant, the reduction consists in replacing R by $R \setminus xy$. Another reduction is obtained if we can certify that, if there exists a k-multicut, then there exists a k-multicut which does not separate two given vertices u and v. In this case, we simply contract u and v. Reductions are easy to control, and we can perform reductions liberally provided that some invariant polynomial in n decreases. For example, request deletions can be performed at most n^2 times, and vertex contractions at most n times.

Branchings. In our algorithm, we often have to decide if the multicut we are looking for is of a particular type, where the number of types is bounded. We will then say that we branch over all the possible cases. This means that, to compute the result of the current instance, we run our algorithm on each case, in which we force the solution to be of each given type. The output is TRUE if at least one of the outputs returns TRUE. To illustrate this, in the case of a graph G with two connected components G_1 and G_2, both containing requests, we would branch over $k-1$ instances, depending of the number of edges (between 1 and $k-1$) that we remove from G_1. This simple branching explains why we can focus on connected graphs.

Invariants. To prove that the total number of branches is bounded, we show that some invariant is modified at each branching step, and that the number of times that this invariant can be modified is bounded. We usually have several invariants ordered lexicographically. In other words, we have different invariants which we want to increase or decrease and each invariant can take a bounded number of values. These invariants are ordered, there is a primary invariant, a secondary invariant, etc. Each branching must *improve the invariant, i.e.* the first invariant (with respect to priority order) which is changed by the branching must be modified according to the preference, increase or decrease, that we specified for it. For instance, the primary invariant could be the number of edges in the multicut, which we want to decrease, and the secondary invariant could be the connectivity of G, which we want to increase. In this case, if we can decrease the number of edges in the solution we do so even if the connectivity of the graph decreases. Also, if a branching increases connectivity and leaves the number of multicut edges unchanged, we improve the invariant.

7.4 Connectivity in FPT time

Dealing with minimum cuts can be done in polynomial time with usual flow techniques. However, dealing with k-edge cuts when k is some fixed value larger than the optimum is more difficult. We develop here some tools to deal in FPT time with bounded-size cuts, inspired from some results from Chapter 4, and independently of the analogous ideas formulated in [29], [96], [100] and [102]. What we call *left cuts* is called *important cuts* in [100] and *extreme cuts* in [102], and is used implicitly in [29] and Chapter 4. The results from Subsection 7.4.1 can essentially be found in these references, but we give them here with simple proofs for the sake of completeness. Some results in Subsection 7.4.2 are similar to statements from Chapter 4, and the results from Subsection 7.4.3 are specific to this work.

7.4.1 Enumerating Cuts in FPT Time

Let G be a connected graph on n vertices with a particular vertex x called the *root.* A cut can be indifferently considered to be a set of edges, or a bipartition of the vertex set. As we want to focus on one side of the bipartition, we define a *cut* as a subset of vertices S containing x. The *border* of S is the set of edges of G with exactly one endpoint in S.

We denote the border of S by $\Delta(S)$, and the cardinality of $\Delta(S)$ is denoted by $\delta(S)$. The function δ is submodular, *i.e.* $\delta(A)+\delta(B) \geq \delta(A \cap B)+\delta(A \cup B)$.

The key definition of this part if the following. A cut S is a *left cut* if every cut $T \subsetneq S$ satisfies $\delta(T) > \delta(S)$. Note that every cut S contains a left cut S' with $\delta(S') \leq \delta(S)$.

The main interest of this notion is to deal with components of G separated by a single vertex x from the rest of the graph, and which contain no request (at most one endpoint per request). Inside such a component, the multicut edges might as well be the border of a left cut. Indeed, a left cut separates even more vertices from the rest of the graph, at the same (or even better) cost. The structure of the set of all multicuts is too complex to design good combinatorial results with algorithmic applications, but the set of left cuts has a better structure, as we shall see.

Lemma 132. *Left cuts are closed under union.*

Proof. Let $S_1 \cup S_2$ be the union of two left cuts. Let $S_3 \subsetneq S_1 \cup S_2$ be a cut with minimum border. Without loss of generality, we assume that S_1 is not included in S_3. Since S_1 is a left cut, $\delta(S_1 \cap S_3) > \delta(S_1)$. As $\delta(S_1 \cap S_3)+\delta(S_1 \cup S_3) \leq \delta(S_1)+\delta(S_3)$, we obtain $\delta(S_1 \cup S_3) < \delta(S_3)$. Furthermore $S_1 \cup S_3 \subseteq S_1 \cup S_2$. Also, S_3 has minimum border among strict subsets of $S_1 \cup S_2$, so we have $S_1 \cup S_3 = S_1 \cup S_2$. Finally $\delta(S_1 \cup S_2) < \delta(S_3)$, thus $S_1 \cup S_2$ is a left cut. □

Lemma 133. *If S_1, S_2 are distinct left cuts,* $\delta(S_1 \cup S_2) < \max(\delta(S_1), \delta(S_2))$.

Proof. As $S_1 \neq S_1 \cup S_2$ or $S_2 \neq S_1 \cup S_2$, we assume without loss of generality that $S_1 \subsetneq S_1 \cup S_2$. By Lemma 132, $\delta(S_1 \cup S_2) < \delta(S_1) \leq \max(\delta(S_1), \delta(S_2))$. □

A cut S is *indivisible* if $G \setminus S$ is connected, otherwise it is *divisible.*

Lemma 134. *If S is a divisible left cut and Y is a connected component of $G \setminus S$, the cut $\overline{Y}$ is an indivisible left cut with* $\delta(\overline{Y}) < \delta(S)$.

Proof. The cut $\overline{Y}$ is indivisible by construction and $\Delta(\overline{Y}) \subsetneq \Delta(S)$, so we just have to prove that $\overline{Y}$ is a left cut. Consider a left cut $T \subseteq \overline{Y}$ which minimizes $\delta(T)$. By Lemma 132, $S \cup T$ is a left cut. Moreover, $\delta(T) \leq \delta(S \cup T)$ by minimality of $\delta(T)$, hence $T = S \cup T$. In particular, $S \subseteq T$. Every edge of $\Delta(\overline{Y})$ has one endpoint in S and one endpoint in Y, so we have $\Delta(\overline{Y}) \subseteq \Delta(T)$. Therefore, by minimality of $\delta(T)$, we have $T = \overline{Y}$. Thus, $\overline{Y}$ is a left cut. □

Corollary 135. *Every indivisible cut S contains an indivisible left cut S' with* $\delta(S') \leq \delta(S)$.

Proof. Let S'' be a divisible left cut contained in S such that $\delta(S'') \leq \delta(S)$. Let Y be the component of $G \setminus S''$ which contains $\overline{S}$. By Lemma 134, $S' := \overline{Y}$ is an indivisible left cut with $\delta(S') < \delta(S'') \leq \delta(S)$. Moreover, $S' \subseteq S$ as $\overline{S} \subseteq Y$. □

Given a vertex y, an xy-*cut* is a cut S such that $y \notin S$. We denote by C^y_k the set of indivisible left xy-cuts with border k. We also denote by $C^y_{<k}$ (resp. $C^y_{\leq k}$) the union of the sets C^y_i for $i < k$ (resp. for $i \leq k$).

Theorem 136. *The set $C^y_{\leq k}$ has size at most $k!$ and can be computed in FPT time.*

Proof. We prove this result by induction on k. Let us start our induction with $k = \lambda$, the smallest value such that $C^y_{\leq\lambda}$ is non empty, *i.e.* the edge-connectivity between x and y.

Claim 137. *The set $C^y_{\leq\lambda}$ has only one element S_λ. It can be computed in polynomial time. Moreover S_λ contains all left xy-cuts.*

Proof. The function δ is modular on xy-cuts of size λ, *i.e.* $\delta(A)+\delta(B) \geq \delta(A\cap B)+\delta(A\cup B)$ when A and B are xy-cuts of size λ. In particular, $C^y_{\leq\lambda}$ is closed under intersection, and contains S_λ, the intersection of all xy-cuts with border λ. Indeed, the set S_λ is indivisible by minimality of λ (and can be computed in polynomial time). All other xy-cuts with border λ contain S_λ by definition, so $C^y_{\leq\lambda} = \{S_\lambda\}$.

Consider a left xy-cut T. We have $\delta(T\cap S_\lambda)+\delta(T\cup S_\lambda) \leq \delta(T)+\lambda$. By minimality of λ, $\delta(T\cup S_\lambda) \geq \lambda$. Thus $\delta(T\cap S_\lambda) \leq \delta(T)$. The set T being a left cut, we have $T\cap S_\lambda = T$, so $T \subseteq S_\lambda$ as claimed. □

Let $S \in C^y_k$ with $k > \lambda$. Consider a set $T \in C^y_{<k}$ which contains S, minimal with respect to inclusion. Such a cut T exists as the cut S_λ contains S by Claim 137. Since S is indivisible, there exists an edge e in $\Delta(T)\setminus\Delta(S)$.

Claim 138. *If $S' \in C^y_{\leq k}$ is included in T and $e \notin \Delta(S')$, then $S' = S$.*

Proof. Assume for contradiction that S' is different from S. By Lemma 132, $S\cup S'$ is a left xy-cut, and by Lemma 133, we have $\delta(S\cup S') < k$. Let Y be the component of $G\setminus\overline{(S\cup S')}$ which contains $\overline{T}$. By Lemma 134, the cut $\overline{Y}$ belongs to $C^y_{<k}$. Therefore $\overline{Y} \subseteq T$, and by minimality of T, we have $\overline{Y} = T$. This implies that $e \in \Delta(S\cup S')$, which is a contradiction. □

We turn Claim 138 into an algorithm. For every cut T in $C^y_{<k}$ and every edge $e \in \Delta(T)$, we contract $(G\setminus T)\cup e$ to a single vertex still called y. We call this graph G'. If the xy-edge connectivity of G' is not equal to k, the search stops. Otherwise, we compute the unique indivisible left xy-cut S with border k. This cut S in G is an element of C^y_k. By Claim 138, all the elements of C^y_k can be computed in this way. This algorithm gives the upper bound $|C^y_{\leq k}| \leq |C^y_{<k}|+(k-1)|C^y_{<k}|$, hence $|C^y_{\leq k}| \leq k!$. This concludes the proof of Theorem 136. □

The value $k!$ in Theorem 136 can actually be improved to 4^k [28, 100].

7.4.2 Irrelevant Requests

We denote by C_k, $C_{<k}$ and $C_{\leq k}$ respectively the union over all vertices y of G of C^y_k, $C^y_{<k}$ and $C^y_{\leq k}$ respectively.

A collection of sets is called a Δ-*system* if every two distinct sets have the same intersection. Erdős and Rado [55] proved that there exists a function er such that a collection of $er(k,r)$ sets of size at most k contains a Δ-system consisting of r sets. The bound proved in following result will be immediately improved to a single-exponential bound with Theorem 140, whose proof is conceptually more complicated.

Theorem 139. *Every set* K *of at least* $er(k!, k')$ *vertices contains a subset* K' *of size* k' *such that every left cut* S *with* $\delta(S) \leq k$ *satisfies either* $S \cap K' = \emptyset$ *or* $|K' \setminus S| \leq k$*. In other words, every left cut with border at most* k *isolates either all the elements of* K'*, or at most* k *elements of* K'*. The set* K' *can be computed in FPT time in* k *and* k'*.*

Proof. Let us consider the collection $\mathcal{C}$ of sets $C^y_{\leq k}$, for all $y \in K$. By Theorem 136, the collection $\mathcal{C}$ has size bounded in terms of k and k' and can be computed in FPT time.

The sets $C^y_{\leq k}$ have size at most $k!$ and the set K has size $er(k!, k')$, so there exists a Δ-system of size k', *i.e.* a subset K' of k' vertices of K such that for all $y, y' \in K'$, the set $C^{y'}_{\leq k} \cap C^y_{\leq k}$ is equal to some fixed set C of $C_{\leq k}$. This set K' is computable in FPT time. Every cut S in C satisfies $S \cap K' = \emptyset$, *i.e.* the cuts in C isolate K'. Moreover, if a cut S in $C_{\leq k}$ does not belong to C, then S belongs to at most one $C^y_{\leq k}$, hence S isolates at most one vertex of K'.

We have proved so far that the conclusion of Theorem 139 holds if S is an indivisible left cut with border or size at most k, with the stronger conclusion that S isolates at most one vertex of K' when it does not completely cut K'. To conclude, let us observe that if S is divisible and Z is a component of $G \setminus S$, then by Lemma 134 the cut $\overline{Z}$ belongs to $C_{\leq k}$. Hence either $\overline{Z}$ isolates K', or $\overline{Z}$ isolates at most one vertex of K'. The number of components of $G \setminus S$ is at most k, which concludes the proof of Theorem 139. □

The same result holds with a better (single-exponential) bound, as expressed in the following result. We have kept the simpler Theorem 139 along with its proof for readers more concerned with the simplicity of the argument than by the efficiency of the algorithm.

Theorem 140. *Every set* K *of at least* $\alpha(k, k') = k'^{\binom{k+2}{2}-1}$ *vertices contains a subset* K' *of size* k' *such that every left cut* S *with* $\delta(S) \leq k$ *satisfies either* $S \cap K' = \emptyset$ *or* $|K' \setminus S| \leq k$*. The set* K' *can be computed in FPT (single exponential) time.*

Proof. Observe that the result trivially holds when $k' \leq k$. So we can assume that $k' > k$. We prove the result by induction on k.

For $k = 1$, $\alpha(1, k') = k'^2$. The complements of a left cut with a border of size 1 form a collection of disjoint sets of vertices, which induces a partition of K. There is either a class K' of this partition containing at least $\sqrt{|K|} \geq k'$ elements, or a set K' of size at least $\sqrt{|K|} \geq k'$ whose elements are chosen in different classes. In both cases K' satisfies the induction hypothesis.

Assume now that $k > 1$. We distinguish two cases.

Assume that there exists an indivisible left cut S with $\delta(S) \leq k$ and $|K \setminus S| \geq k'^{\binom{k+1}{2}-1}$. By induction, we extract from $K \setminus S$ a subset K' of size k' such that every left cut S with $\delta(S) \leq k-1$ satisfies either $S \cap K' = \emptyset$ or $|K' \setminus S| \leq k-1$. Consider a left cut S' with $\delta(S') = k$.

If $S' = S$, then S' isolates K' by definition of K', hence we assume that S' is distinct from S. Observe that $K' \setminus S'$ is equal to $K' \setminus (S \cup S')$. As $S \cup S'$ is a left cut with border at most $k-1$, the set $K' \setminus S'$ is either K' or has size at most $k-1$. So the conclusion of Theorem 140 holds.

Conversely, assume that all indivisible left cuts S with $\delta(S) \leq k$ satisfy $|K \setminus S| < k'^{\binom{k+1}{2}-1}$. Consider an auxiliary graph H with vertex set K and where vv' is an edge when there exists an indivisible left cut S with $\delta(S) \leq k$ such that $\{v, v'\} \cap S = \emptyset$. The degree in H of a vertex v is less than $d := k'^{\binom{k+1}{2}-1}.k!$. Indeed, there are at most $k!$ indivisible left xv-cuts with a border of size at most k, and we have assumed that all indivisible left cuts S with $\delta(S) \leq k$ satisfy $|K \setminus S| < k'^{\binom{k+1}{2}-1}$. Note that d is less than $k'^{\binom{k+1}{2}-1}.k'^k$ as $k' \geq k$. There is a stable set K' in H of size at least $\frac{|K|}{d}$, and $\frac{|K|}{d} \geq k'$ since $\binom{k+2}{2} - \binom{k+1}{2} - k = 1$. Every indivisible left cut S with $\delta(S) \leq k$ isolates at most one vertex of K' as K' is stable in H. Thus, every left cut S with $\delta(S) \leq k$ isolates at most k vertices of K'.

The induction hypothesis holds in both cases, which concludes the proof of Theorem 140. □

We will use the bound from Theorem 140 in the following. The next result was essentially our key tool in reducing MULTICUT to graphs of bounded treewidth in Chapter 4, but we give a more straightforward proof here. The proof of Theorem 66 essentially implies that the following result holds with $h(\ell) = \ell^{O(\ell)}$, and can be computed in time $O^*(\ell^{O(\ell)})$.

Theorem 141. *Every set* K *with at least* $h(\ell) := \ell.2^{\ell!} + 1$ *vertices of* G *contains a vertex* y *such that every cut* S *with* $\delta(S) + |S \cap K| \leq \ell$ *is such that* $y \notin S$. *Moreover,* y *can be found in FPT time.*

In other words, the vertex y verifies the following: whenever the deletion of a set of a edges isolates x from all but b elements of K, with $a + b \leq \ell$, then the vertex y is also isolated from x.

Proof. Let G' be the graph obtained from G by adding a new vertex z with neighbourhood K. In G', the set C of indivisible left zx-cuts with border at most ℓ has size at most $\ell!$ by Theorem 136. The size of K is at least $\ell.2^{\ell!} + 1$, so there exists a subset T of K of size at least $\ell+1$ such that for every cut S in C, we have either $T \subseteq S$ or $T \cap S = \emptyset$. For example, originally set $T = K$, and for every cut S in C, do $T := T \cap S$ if $|T \cap S| \geq |T \cap \overline{S}|$, and $T := T \cap \overline{S}$ otherwise.

We pick a vertex y in T. Let us prove that y satisfies the conclusion of Theorem 141.

In the graph G, consider a set A of a edges which isolates x from all the elements of K apart from a subset B of size b with $a + b \leq \ell$. Let F be the set of edges $A \cup \{zb : b \in B\}$ of G'. Note that F is a zx-edge cut. We denote by X the component of x in $G' \setminus F$. Since $V(G') \setminus X$ is an indivisible zx-cut with border at most ℓ, it contains by Corollary 135 an indivisible left zx-cut U with border at most ℓ. In other words, U belongs to C.

Let us first observe that the set T cannot be disjoint from U. Indeed T has size $\ell + 1$ and each vertex of T is adjacent to z, thus the border of U would exceed ℓ. Hence T is included in U, and the set of edges A isolates T from x in G, and in particular separates y from x. This concludes the proof of Theorem 141. □

This connectivity result allows us to bound the *request degree* of a vertex v, *i.e.* the number of requests with v as an endpoint.

Corollary 142. *In a* MULTICUT *instance with deletion allowance* k*, the maximum request degree can be reduced to at most* $h(k+1) = O(2^{k!})$ *in FPT time.*

Proof. Consider a vertex x, and denote by K the set of vertices forming a request with x. Assume that $|K| \geq h(k+1)$. By Theorem 141, there is a vertex y of K such that every subset S containing x and verifying $\delta(S) + |S \cap K| \leq k+1$ is such that $y \notin S$. We simply remove the request xy from the set of requests. Indeed, let F be a multicut of size at most k of this reduced instance. Let S be the component of x in $G \setminus F$. As F is a multicut, no element of $K \setminus y$ belongs to S. Moreover $\delta(S) \leq k$ since at most k edges are deleted. Thus $\delta(S) + |S \cap K|$ is at most $k+1$, which implies that $y \notin S$. In other words, even if we do not require to cut x from y, a multicut of the reduced instance must cut the request xy. Therefore removing the request xy from R is correct. When such a reduction can no longer be performed, each vertex has request degree smaller than $h(k+1)$. □

7.4.3 Cherry Reduction

An x*-cherry*, or simply *cherry* is a connected induced subgraph C of G with a particular vertex x called *attachment vertex* of C such that there is no edge from $C \setminus x$ to $G \setminus C$ and no request has its two terminals in $C \setminus x$. In other words, the requests inside an x-cherry must have x as an endpoint. Note that we can always assume that the restriction of a multicut to an x-cherry C is the border of a left cut of C, where x is the root. If $u \in C \setminus x$, a request $uv \in R$ is *irrelevant* if for every multicut F of at most k edges of $R \setminus uv$ and such that $F \cap C$ is the border of a left cut in C, F actually separates u from v.

Theorem 143. *Let* C *be an* x*-cherry of an instance with deletion allowance* k*. We can find in FPT time a set* $K(C)$ *of at most* $b(k) := h(k+1)\alpha(k, h(2k+1)) = k^{O(k^3)}$ *terminals in* $C \setminus x$*, such that if* F *is a set of at most* k *edges which cuts all requests with one endpoint in* $K(C)$ *and such that* $F \cap C$ *is the border of a left cut, then* F *actually cuts all requests with an endpoint in* $C \setminus x$*.*

Proof. By Corollary 142, we can assume that all terminals have request degree at most $h(k+1)$. Let L be the set of terminals in $C \setminus x$. We assume that $|L| > b(k)$. Our goal is to show that there exists an irrelevant request with one endpoint in L. Let us consider the bipartite request graph B formed by the set of requests with one endpoint in L. The graph B is bipartite since $C \setminus x$ has no internal requests. Recall that if a bipartite graph with vertex bipartition (X, Y) has maximum degree d and minimum degree one, there exists a matching with at least $|X|/d$ edges. Indeed, in this case the edges can be partitioned into d matchings and the graph contains at least $|X|$ edges.

The request graph B thus contains a matching M of size at least $\alpha(k, h(2k+1))$ such that each request in M has one endpoint in L and the other endpoint outside $C \setminus x$. Let $K := V(M) \cap V(C \setminus x)$. We first only consider the cherry C where x is the root. Since the size of K is at least $\alpha(k, h(2k+1))$, the set K contains by Theorem 139 a subset K' of size $h(2k+1)$ such that every left cut S with border at most k verifies $S \cap K' = \emptyset$ or $|K' \setminus S| \leq k$. Let M' be the set of edges of M having an endpoint in K'. We denote by L' the set of vertices $M' \setminus K'$, *i.e.* the endpoints of edges in M' which do not belong to $C \setminus x$. Now let us consider the graph $G' := G \setminus (C \setminus x)$ with root x. The set L' has size at least $h(2k+1)$, thus by Theorem 141 there is a vertex y in L' such that, whenever we delete k edges in G' such that at most k vertices of L' belong to the component of x, then y does not belong to the component of x. The vertex y being an element of L', we consider the request $zy \in M'$, where z belongs to $V(C \setminus x)$.

We claim that the request zy is irrelevant. Indeed, let F be a multicut of $R \setminus zy$ with at most k edges such that $F_C = F \cap C$ is the border of a left cut. Let S be the component of x in $C \setminus F_C$. The set S is a left cut and has a border of size at most k, hence, either S completely isolates x from K' or S isolates at most k vertices of K' from x. If K' is isolated from x, then in particular x is disconnected from y, hence the request zy is cut by F. So we assume that a subset K'' containing all but at most k vertices of K' is included in S. Hence, denoting by L'' the other endpoints of the edges of M' intersecting K'', this means that F must disconnect x from L''. Therefore, the set F of at most k edges disconnects x from at most $k+1$ elements of L' (the k elements of L'' and possibly y), so by definition of y, the set F disconnects x from y. In particular zy is cut by F. Thus the request zy is indeed irrelevant. All the computations so far are FPT.

We repeat this process, removing irrelevant requests until the size of L does not exceed $b(k)$. We then set $K(C) := L$, and the conclusion of Theorem 143 holds. □

Let C be a cherry of a graph G with deletion allowance k. A subset $\mathcal{L}$ of the edges of C is *active* when we can assume that a multicut uses only edges of $\mathcal{L}$ in C, or more formally: if a multicut F of size at most k exists, then there exists a multicut F' of size at most $|F|$ such that $F' \setminus C = F \setminus C$ and $F' \cap C \subseteq \mathcal{L}$. When the set $\mathcal{L}$ is clear from the context, we say by extension that edges of $\mathcal{L}$ themselves are *active*.

Lemma 144. *Let* C *be an* x*-cherry of a graph* G *with deletion allowance* k*, and let* K *be the set of all terminals of* $C \setminus x$*. Let* $\mathcal{L}(C)$ *be the union of all borders of cuts of* $C^y_{\leq k}$*, where* $y \in K$*. Then* $\mathcal{L}(C)$ *is active, and has size at most* $(k+1)!|K|$*.*

Proof. Assume that F is a multicut of size at most k. Let S be the component of x in $C \setminus F$. Let T be a left cut with $T \subseteq S$ and $\delta(T) \leq \delta(S)$. If a component U of $\overline{T}$ does not intersect K, then $F \setminus \Delta(U)$ is still a multicut. Hence, we can assume that all components U of $\overline{T}$ intersect K, in which case $\Delta(U) \in C^y_{\leq k}$ for some y in K, hence $\Delta(T)$ is included in $\mathcal{L}(C)$. The set

$F' = (F \setminus C) \cup \Delta(T)$ is a multicut, and the size bound for $\mathcal{L}(C)$ follows from Theorem 136: $C^{y}_{\leq k}$ contains at most $k!$ cuts of size at most k, and $\mathcal{L}(C)$ is a union of $|K|$ such sets. □

Theorem 145. *Let* $H_1, H_2, \ldots, H_p$ *be* x*-cherries of a graph* G *with deletion allowance* k *such that* $H_1 \setminus x, H_2 \setminus x, \ldots, H_p \setminus x$ *are pairwise disjoint. Assume that for every* i, $U_i := H_1 \cup \cdots \cup H_i$ *is a cherry. Then every set* U_i *has a bounded active set* $\mathcal{L}_i$ *such that* $\mathcal{L}_j \cap U_i \subseteq \mathcal{L}_i$ *whenever* $i \leq j$.

Proof. By Theorem 143, we can reduce the set of terminals in U_1 to a bounded set K_1. The set $\mathcal{L}_1 = \mathcal{L}(U_1)$ is bounded and active by Lemma 144. The requests of $C_1 \setminus K_1$ are irrelevant in U_2 since they are irrelevant in U_1, hence we can assume that Theorem 143 applied to U_2 yields a set of terminals $K_2 \subseteq K_1 \cup C_2$. Let $\mathcal{L}_2$ be the active edges associated to K_2. Note that if an edge $e \in \mathcal{L}_2$ is in U_1, then the edge e must belong to a set $C^{y}_{\leq k}$ for some $y \in K_2 \cap U_1$. Since $K_2 \subseteq K_1 \cup C_2$, we have $y \in K_1$, and so $e \in \mathcal{L}_1$, which is the property we are looking for. We extract K_3 from $K_2 \cup C_3$, and iterate this process to form the sequence $\mathcal{L}_i$. □

7.5 Reducing MULTICUT to COMPONENT MULTICUT

Let $G = (V, E)$ be a connected graph, and R be a set of requests. A *vertex-multicut* Y is a subset of V such that every xy-path of G where $xy \in R$ contains a vertex of Y. Let A be a connected component of $G \setminus Y$. We call Y*-component*, or *component*, the union of A and its set of neighbours in Y. Let C be a Y-component, the vertices of $C \cap Y$ are the *attachment vertices* of C.

7.5.1 Component Multicut

Our first intermediate problem is formally expressed below. Informally, the Y-components have at most two attachment vertices. Each Y-component with two attachment vertices has a distinguished path called *backbone*, and the multicut restricted to a component must consist of exactly one edge of the backbone plus a fixed number of other edges. Finally, the vertex-multicut Y must be split by the solution.

COMPONENT MULTICUT:
Input: A connected graph $G = (V, E)$, a vertex-multicut Y, a set of requests, and q integers integers $f_1, \ldots, f_q$ such that:

1. There are q Y-components $G_1, \ldots, G_q$ with two attachment vertices x_i and y_i. The other Y-components have only one attachment vertex.
2. Every G_i has an $x_i y_i$-path denoted P_i called the *backbone* of G_i, such that the deletion of an edge of P_i decreases the edge connectivity in G_i between x_i and y_i.
3. The integers $f_1, \ldots, f_q$ are such that $f_1 + \cdots + f_q \leq k - q$.

Parameter: k.
Output: TRUE if there exists a multicut F such that:

1. every path P_i contains exactly one edge of F,
2. every component G_i contains exactly $1+f_i$ edges of F,
3. the solution F *splits* Y, *i.e.* each connected component of $G \setminus F$ contains at most one vertex of Y.

Otherwise, the output is FALSE.

The edges of G which do not belong to the backbones are called *free edges*. The backbone P_i, in which only one edge is deleted, is the crucial structure of G_i. Indeed, the whole proof consists of modifying each Y-component G_i step by step to finally completely reduce it to the backbone P_i. Here, f_i is the number of free edges that we can delete in G_i. Observe that $k-q-f_1-\cdots-f_q$ edges can be deleted in Y-components with one attachment vertex. Our first reduction is the following:

Theorem 146. MULTICUT *can be reduced to* COMPONENT MULTICUT *in FPT time.*

The remaining of Section 7.5 is devoted to the proof of Theorem 146. We first construct a vertex-multicut Y through iterative compression. Then, we prove that we can reduce to Y-components with one or two attachment vertices. Finally, we show that we can assume that every component with two attachment vertices has a path in which exactly one edge is chosen in the solution. This is our backbone.

7.5.2 The Vertex-Multicut Y

This subsection is devoted to prove by iterative compression that MULTICUT is equivalent to the following problem, as was first noted in [99]:

RICHER MULTICUT:
Input: A graph G, a set of requests R, an integer k, a vertex multicut Y of size at most $k+1$.
Parameter: k.
Output: TRUE if there is a multicut of size at most k which splits Y, otherwise FALSE.

Lemma 147. MULTICUT *is FPT-equivalent to* RICHER MULTICUT.

Lemma 147 follows from the following two Lemmas:

Lemma 148. MULTICUT *can be solved in time* $O(f(k)n^c)$ *if the* MULTICUT *variant where a vertex multicut of size at most* $k+1$ *is additionally given in the input can be solved in time* $O(f(k)n^{c-1})$.

Proof. By induction on n, we solve MULTICUT in time $f(k)(n-1)^c$ on $G-v$, where $v \in V(G)$. If the output is FALSE, we return FALSE, otherwise the output is a multicut F of size at most k. Let X be a vertex cover of F of size at most k (*i.e.* X contains one endpoint of each edge in F). The set $X \cup \{v\}$ is a vertex-multicut of the original instance, so we solve MULTICUT in time $f(k)n^{c-1} + f(k)(n-1)^c$ which is at most $f(k)n^c$. □

So we can assume that the input of MULTICUT contains a vertex-multicut Y of size at most $k+1$.

Lemma 149. *We can assume that the solution* F *splits* Y.

Proof. To a solution F is associated the partition of $G \setminus F$ into connected components. In particular, this induces a partition of Y. We branch over all possible partitions of Y. In a given branch, we simply contract the elements of Y belonging to a same part of the partition. □

This concludes the proof of Lemma 147.

During the following reduction proof, the size of the set Y will never decrease. Since one needs $k+1$ edges to separate $k+2$ vertices, the size of Y cannot exceed $k+1$, otherwise we return FALSE. Hence the primary invariant is the size of Y, and we immediately conclude if we can increase $|Y|$.

7.5.3 Reducing Attachment Vertices

Our second invariant, which we intend to maximize, is the number of Y-components with at least two attachment vertices. This number cannot exceed k, since a solution must split Y. Our third invariant is the sum of the edge connectivity between all pairs of vertices of Y, which we want to increase. This invariant is bounded by $k\binom{|Y|}{2}$ since the connectivity between two elements of Y is at most k. Note that this third invariant never decreases when we contract vertices.

Lemma 150. *If* C *is a* Y*-component with at least three attachment vertices, we improve the invariant.*

Proof. Let x, y, z be attachment vertices of C. Let λ be the edge-connectivity between x and y in C. Let $P_1, \dots, P_\lambda$ be a set of edge-disjoint xy-paths. A *critical edge* is an edge which belongs to some xy-edge cut of size λ. Note that every critical edge belongs to some path P_i. A *slice* of C is a connected component of C minus the critical edges. Given a vertex v of C, the *slice of* v, denoted by $SL(v)$, is the slice of C containing v. Let $B(z)$ be the *border* of $SL(z)$, *i.e.* the set of vertices of $SL(z)$ which are incident to a critical edge. Note that $B(z)$ intersects every path P_i on at most two vertices, namely the leftmost vertex of P_i belonging to $SL(z)$ and the rightmost vertex of P_i belonging to $SL(z)$. In particular, $B(z)$ has b vertices, where $b \le 2\lambda$.

We branch over $b+1$ choices to decide whether one of the b vertices of $B(z)$ belongs to a component of $G \setminus F$ (where F is the solution) which does not contain a vertex of Y. When this is the case, the vertex is added to Y, which increases the primary invariant. In the last branch, all the vertices of $B(z)$ are connected to a vertex of Y in $G \setminus F$. We branch again over all mappings f from $B(z)$ into Y. In each branch, the vertex $v \in B(z)$ is connected to $f(v) \in Y$ in $G \setminus F$. Hence we can contract every vertex $v \in B(z)$ with the vertex $f(v) \in Y$. This gives a new graph G'. We denote by S' the subgraph $SL(z)$ in G'. Observe that S' is a Y-component of G'.

If x and y belong to S', then the edge connectivity between x and y has increased. Indeed, there is now a path P between x and y inside S', in particular P has no critical edge. Thus the connectivity between x and y has increased, so the invariant has improved. We assume without loss of generality that x does not belong to S'.

If S' contains an element of Y distinct from z, then S' is a Y-component with at least two attachment vertices. Moreover, there exists a path P in $C \setminus S'$ from x to $B(z)$. Hence we have created an extra Y-component with at least two attachment vertices in G', which improves the second invariant.

In the last case, z is the only vertex of Y which belongs to S'. Therefore, $B(z)$ is entirely contracted to z. In particular z is now incident to a critical edge e. So there exists an xy-cut A with $\delta(A) = \lambda$ and $e \in \Delta(A)$. Without loss of generality, we assume that $z \notin A$ (otherwise we consider the yx-cut $\overline{A}$). We denote by B the vertices of $\overline{A}$ with a neighbour in A. In particular, B contains z, has size at most λ, and every xy-path in C contains a vertex of B. Let us denote by L the set $A \cup B$ and by R the set $\overline{A}$. Note that $L \cap R = B$. We now branch to decide in which components of $G \setminus F$ the elements of B are partitioned. If an element of B is not connected to Y in $G \setminus F$, we improve the invariant. If each element of B is contracted to a vertex of Y, both L and R in the contracted graph are Y-components with at least two attachment vertices (respectively $\{x, z\}$ and $\{y, z\}$). We again improve the invariant. □

7.5.4 Backbones

We now assume that every component has at most two attachment vertices. Let $G_1, \ldots, G_q$ be the components of G with two attachment vertices. We denote by λ_i the edge connectivity of G_i between its two attachment vertices x_i and y_i. Recall that the third invariant is the sum of the λ_i for $i = 1, \ldots, q$.

Lemma 151. *We can assume that* x_i *and* y_i *have degree* λ_i *in* G_i.

Proof. Let A be the unique left $x_i y_i$-cut with $\delta(A) = \lambda_i$ in the graph G_i rooted in x_i. Let B be the set of vertices of A with a neighbour in $\overline{A}$. We now branch to decide how the components of $G \setminus F$ partition B. If a vertex of B is not connected to a vertex of Y in $G \setminus F$, we can add it to Y and improve the invariant. If a vertex of B is contracted to a vertex y_i, we increase λ_i. Hence all elements of B are contracted to x_i. Therefore A becomes an x_i-

cherry, hence $A \setminus x_i$ is removed from G_i. The degree of x_i inside G_i is now exactly λ_i. We apply the same argument to reduce the degree of y_i to λ_i. □

We now branch over all partitions of k into $k_0 + k_1 + \cdots + k_q = k$, where k_i is the number of edges of the solution chosen in G_i when $i > 0$, and k_0 is the total number of edges chosen in the y-cherries for $y \in Y$.

Lemma 152. *Every component* G_i *can be deleted or has a backbone.*

Proof. If $k_i \geq 2\lambda_i$, then the whole component G_i can be disconnected from the rest of the graph by removing the edges in G_i incident to x_i and y_i. This reduces the second invariant, the number of components with at least two attachment vertices. So we can assume that $k_i \leq 2\lambda_i - 1$. Let $P_1, P_2, \ldots, P_{\lambda_i}$ be edge-disjoint $x_i y_i$-paths.

Our algorithm now branches $2\lambda_i$ times, where the branches are called B_j and B'_j for $j = 1, \ldots, \lambda_i$. In the branch B_j, we assume that there is only one edge of the solution selected in P_j, and that this edge is critical, *i.e.* belongs to an $x_i y_i$-cut of size λ_i. In the branch B'_j, we assume that all the edges of the solution selected in P_j are not critical. Let us show that every solution F belongs to one of these branches. If F does not belong to any branch B'_j, this means that F uses at least one critical edge in each P_j. But since $k_i \leq 2\lambda_i - 1$, some path P_j only intersects F on one edge, which is therefore critical. Hence F is a solution in the branch B_j. Thus this branching process is valid. In the branch B_j, we contract all non critical edges of P_j, therefore P_j is the backbone we are looking for. In the branch B'_j, we contract all critical edges of P_j, hence the connectivity λ_i increases. We thus improve the invariant. □

This concludes the proof of Theorem 146. To sum up, the invariants in this redution from MULTICUT to COMPONENT MULTICUT are (in decreasing order of importance):

- $|Y|$, to maximize, bounded by $k+1$.
- The number of components with at least two attachment vertices, to maximize, bounded by k.
- The sum of the connectivity of all pairs of vertices of Y, to maximize, bounded by k^3.

The tree which represents the branchings made by the algorithm has depth $O(k^5)$ (the product of the bounds on the invariants). The degree of its nodes is bounded by $O^*(k^{O(k)})$. Indeed, when reducing the components with three or more attachment vertices in Lemma 150, or when ensuring that the degree of attachment vertices in a given component is exactly their connectivity in Lemma 151, we improve the invariant at a cost of $O^*(k^{O(k)})$, and these are the bottlenecks. Finally, the amount of work performed at each node vanishes in front of the number of branches, which gives:

Remark 153. *The reduction from* MULTICUT *to* COMPONENT MULTICUT *is executed in time* $O^*(k^{O(k^6)})$.

Reductions which are only performed once do not impact the overall running time: reducing MULTICUT to RICHER MULTICUT is achieved in time $O^*(k^{O(k)})$ in Lemma 147; also, there are $O(k^2)$ components with at most 2 attachment vertices (all y-cherries are considered as a single component for $y \in Y$), hence branching to decide how many edges of the multicut per component costs $k^{O(k^2)}$. These terms vanish in front of the $O^*(k^{O(k^6)})$ term.

7.6 BACKBONE MULTICUT is FPT

7.6.1 Backbone Multicut

We introduce here the problem BACKBONE MULTICUT, which is a generalisation of COMPONENT MULTICUT. Our goal is to show that BACKBONE MULTICUT is solvable in FPT time, which implies that COMPONENT MULTICUT is FPT, which in turns implies that MULTICUT is FPT thanks to Theorem 146.

BACKBONE MULTICUT, formally defined below, differs from COMPONENT MULTICUT in two ways. BACKBONE MULTICUT contains half-requestd of the type (u, y, v), where $u, v \in V$ and $y \in Y$. Cutting the half-request (u, y, v) means cutting all paths from u to v going through y. Also, an instance of BACKBONE MULTICUT can express simple properties on the edge of a backbone selected in the multicut, which allows us to enrich instances.

BACKBONE MULTICUT:
Input: A connected graph $G = (V, E)$, a set R of half-requests, a set Y of at most $k+1$ vertices, a set B of q variables, a set $\mathcal{C}$ of clauses, and q non negative integers $f_1, \dots, f_q$ such that:

1. G has q Y-components called G_i with two attachment vertices $x_i, y_i \in Y$, where $i = 1, \dots, q$. Moreover, G_i has a backbone Q_i (a prescribed x_iy_i-path) and the x_iy_i-connectivity in G_i is λ_i.
2. The set R contains *half-requests, i.e.* sets of triples (u, y, v), informally meaning that vertex u sends a request to vertex v via y, where $y \in Y$. Also, Y is a u, v-cut for every half-request $(u, y, v) \in R$.
3. The set B contains q integer-valued variables $c_1, \dots, c_q$. Each variable c_i corresponds to the deletion of one edge in the backbone Q_i. Formally, if the edges of Q_i are $e_1, \dots, e_{\ell_i}$, ordered from x_i to y_i, the variable c_i can take all possible values from 1 to ℓ_i, and $c_i = r$ means that we delete the edge e_r in Q_i.
4. The clauses in $\mathcal{C}$ have four possible types: $(c_i \le a \Rightarrow c_j \le b)$, or $(c_i \le a \Rightarrow c_j \ge b)$, or $(c_i \ge a \Rightarrow c_j \ge b)$, or $(c_i \ge a \Rightarrow c_j \le b)$.

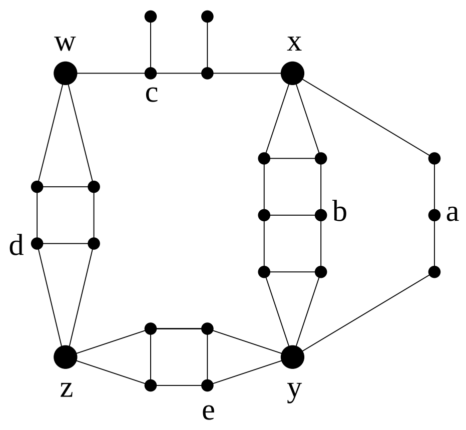


Figure 7.1: Fat vertices w, x, y, z belong to Y. The request ab can be simulated by two half-requests: (a, x, b) and (a, y, b). The request cd can be simulated by the single half-request (c, w, d), as paths between c and d which do not go through w have to go through x and y, and hence will be automatically cut when Y is split. By the same argument, the request ce is irrelevant and can be deleted.

5. The integers $f_1, \dots, f_q$ sum to a value at most k. Each integer f_i corresponds to the number of free edges (*i.e.* edges of G which are not in a backbone) of the solution which are chosen in G_i.

Parameter: k.
Output: TRUE if:

1. There exists an assignment of the variables of B which satisfies $\mathcal{C}$.[1]
2. There exists a subset F of at most k free edges of G, which contains f_i free edges in G_i for $i = 1, \dots, q$.
3. The union F' of the set F together with the backbone edges corresponding to the variables of B splits Y and intersects every half-request of R, *i.e.* for every half-request $(u, y, v) \in R$ every path between u and v containing y intersects F'.

Otherwise, the output is FALSE.

Note that the deletion allowance of BACKBONE MULTICUT is $k+q$. COMPONENT MULTICUT directly translates into BACKBONE MULTICUT with an empty set of clauses, and where each request is simulated by one or two half-requests (see Figure 7.1).

This section is devoted to the proof of the following result, which will conclude the proof of the fixed-parameter tractability of MULTICUT:

1. Formally, if c_i is assigned value r, then variables $c_i \le a$ for $a \ge r$ and variables $c_i \ge a$ for $a \le r$ are true and variables $c_i \ge a$ for $a \ge r+1$ and variables $c_i \le a$ for $a \le r-1$ are false.

Theorem 154. BACKBONE MULTICUT *can be solved in FPT time.*

7.6.2 Invariants

Our primary invariant is the sum of the numbers of free edges f_i for $i = 1, \dots, q$, which starts with value at most k and is non-negative. A branch in which we can decrease this primary invariant will be considered solved. The secondary invariant is the sum of the $\lambda_i - 1$, called the *free connectivity*, which we try to increase. Observe that this invariant is bounded above by k. For the last invariant, recall that the *slice* $SL(v)$ of a vertex v in a component G_i is the connected component containing v of G_i minus the *critical* edges of G_i, *i.e.* edges of λ_i-cuts. Observe that since all edges of a backbone are critical, the slices of distinct vertices in a backbone do not intersect. See Figure 7.2.

The *slice connectivity* of a vertex v in Q_i is the x_iy_i-edge-connectivity of $G_i \setminus SL(v)$ (where $SL(v)$ is considered as a vertex set). We denote it by $sc(v)$. For example, if the set of neighbours of v intersects every x_iy_i-path in $G_i \setminus Q_i$, then we have $sc(v) = 0$. Conversely, if $v \in Q_i$ has only neighbours in Q_i, then $sc(v) = \lambda_i - 1$. The *slice connectivity* sc_i of G_i is the maximum of $sc(v)$, where $v \in Q_i$. The third invariant is the sum sc of the sc_i, for $i = 1, \dots, q$, and we try to minimize this invariant. Observe that sc is always at most k.

Our goal is to show that we can always improve the invariant, or conclude that $\lambda_i = 1$ for all i.

In the following, we consider a component G_i *with* $\lambda_i > 1$*, say* G_1.

To avoid cumbersome indices, we assume that the attachment vertices of G_1 are x and y, and that their edge-connectivity is denoted by λ instead of λ_1. Moreover, we denote by P_1 the backbone of G_1, and we assume that $P_1, P_2, \dots, P_\lambda$ is a set of edge-disjoint xy-paths in G_1. We visualize x to the left and y to the right (see Figure 7.2). Hence when we say that a vertex $u \in P_i$ is to the left of some vertex $v \in P_i$, we mean that u is between x and v on P_i.

7.6.3 Contracting Edges

In our proof, we contract edges of the backbone and also free edges which are not critical. We always preserve the fact that the edges of the backbone are critical.

When contracting an edge of the backbone P_1, we need to modify several parameters. Assume that the edges of P_1 are $e_1, \dots, e_\ell$. The variable c_1 represents the edge of P_1 which belongs to the multicut. Now assume that the edge $e_i = v_iv_{i+1}$ is contracted. All the indices of the edges which are at least $i+1$ are decreased by one. All the constraints associated to the other backbones are not affected by the transformation. However, each time a clause contains a literal $c_1 \geq j$, where $j > i$, this literal must be replaced by $c_1 \geq j-1$. Similarly, each occurrence of $c_1 \leq j'$ for $j' \geq i$ must be replaced by $c_1 \leq j'-1$. If a set of edges is contracted, we perform the contractions one by one.

The collection of paths $P_2, \dots, P_\lambda$ can be affected during our contractions since it can happen that a path P_i with $i \geq 2$ contains both endpoints of a contracted edge uv. In such a

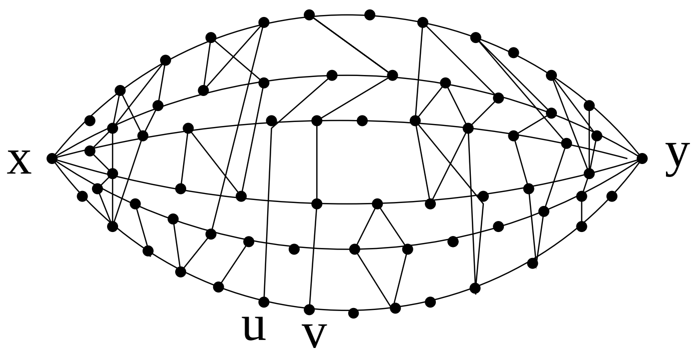


Figure 7.2: A component in a BACKBONE MULTICUT instance with two attachment vertices $x, y \in Y$. The bottom xy-path is the backbone P_1, and the other "horizontal" xy-paths are the prescribed paths $P_2, \dots, P_6$, from bottom to top. Each edge of the backbone does indeed belong to a λ-cut (*i.e.* an xy-cut consisting of six edges). In other words, there exists no path between two distinct vertices of the backbone which consists only of *diagonal edges, i.e.* edges which do not belong to the paths $P_1, \dots, P_6$. Thus the slices of two distinct backbone vertices are disjoint. The tag of vertex u, as defined in Subsection 7.6.4, is $\{1,4,5\}$ and the tag of vertex v is $\{1,3,4,5,6\}$. The slice connectivity of u is $sc(u) = 6-3 = 3$, and $sc(v) = 6-5 = 1$. The slice connectivity of this component is 5, as there exists a vertex of the backbone P_1 with no neighbour outside P_1.

case, we remove from P_i the loop formed by the contraction, *i.e.* the subpath of P_i between u and v. We thus preserve our path collection.

7.6.4 Choosing a Stable Edge

Let v be a vertex of P_1. The *tag* of v is the subset $t(v) := \{i \mid P_i \cap SL(v) \neq \emptyset\}$, *i.e.* the set of indices of the paths intersecting the slice of v. Note that $t(v)$ contains 1. Observe also that the slice connectivity of G_1 is the maximum of $\lambda - |t(v)|$, where v belongs to P_1. See Figure 7.2.

By extension, the *tag* of an edge $v_i v_{i+1}$ of the backbone P_1 is the ordered pair $(t(v_i), t(v_{i+1}))$. When speaking of an XY-*edge*, we implicitly mean that its tag is (X,Y). In particular, the edge of P_1 which is selected in the solution has a given tag. We branch over the possible choices for the tag XY of the deleted edge of P_1. Let us assume that the chosen edge has tag XY.

Lemma 155. *If* $X \neq Y$, *we improve the invariant.*

Proof. Since only one edge is cut in the backbone, we can contract all the edges of P_1 with tags different from XY. Observe that when contracting some UV-edge of P_1, the tag of the resulting vertex contains $U \cup V$ since the slice of the resulting vertex contains the union of

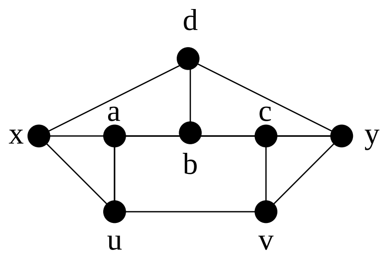


Figure 7.3: This component has two attachment vertices x and y. Its backbone is $P_1 = xuvy$, and we have $P_2 = xabcy$ and $P_3 = xdy$. Both u and v have tag $t(u) = t(v) = \{1,2\}$, since all edges of P_2 are edges of λ-cut. After contracting u and v, edges ab and bc are no longer edges of λ-cut, and the slice of the resulting vertex $u = v$ becomes $\{u = v, a, b, c, d\}$, and its tag becomes $\{1,2,3\}$, which strictly contains $t(u) \cup t(v)$.

both slices (it can actually be larger, see Figure 7.3). After contraction, all the edges of P_1 between two consecutive occurrences of XY-edges are contracted, hence the tag of every vertex of P_1 now contains $X \cup Y$. In particular, the slice connectivity of G_1 decreases while the free connectivity is unchanged. Thus the invariant has improved. □

Therefore we may assume that we choose an XX-edge in the solution. Let us contract all the edges of P_1 which are not XX-edges. By doing so, the tag of every vertex of P_1 contains X. After this contraction, the instance is modified, hence we have to branch again over the choice of the tag of the edge chosen in the solution. Any choice different from XX increases the slice connectivity. Hence we can still assume that the tag of the chosen edge is XX.

The slice connectivity of G_1 is $\lambda - |X|$. An XX-edge uv of the backbone is *unstable* if, when contracting uv, the tag of the vertex $u = v$ increases (*i.e.* strictly contains X). Otherwise uv is *stable*. We branch on the fact that the chosen XX-edge is stable or unstable.

Lemma 156. *If the chosen XX-edge is unstable, we improve the invariant.*

Proof. We enumerate the set of all unstable edges from left to right along P_1, and partition them according to their index into the odd indices and the even indices. We branch according to the index of the chosen unstable edge. Assume for instance that the chosen unstable edge has odd index. We contract all the edges of P_1 except from the odd unstable edges. We claim that the tag of every vertex now strictly contains X. Indeed, all edges of P_1 between two consecutive odd unstable edges are contracted, in particular some even unstable edge. Thus, since this even edge is unstable, the tag now strictly contains X. Hence the slice connectivity decreases. □

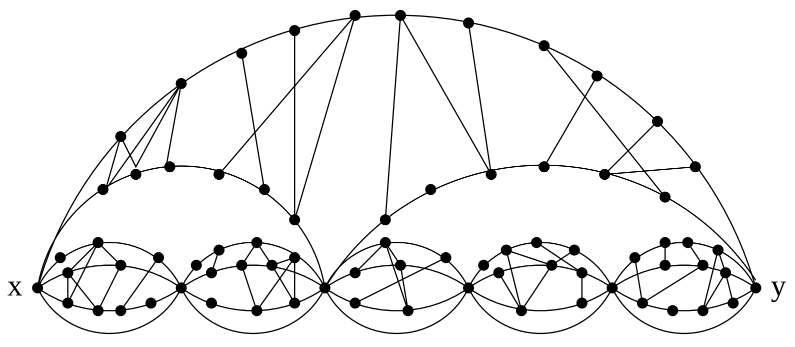


Figure 7.4: All backbone vertices of this component are full, where $X = \{1,2,3,4\}$, with the backbone P_1 at the bottom.

7.6.5 Contracting Slices

In this part, we assume that the chosen edge of P_1 is a stable XX-edge. A vertex v of P_1 is *full* if v belongs to every P_i, where $i \in X$ (see Figure 7.4). Our goal in this subsection is to show that we can reduce to the case where $X = \{1,\dots,\lambda\}$. By the previous section, a branching which increases the tag of the chosen edge would improve the invariant. So we assume that in all our branchings, the chosen edge is still a stable XX-edge.

Lemma 157. *We can assume that all backbone vertices are full.*

Proof. We can first assume that there are at most k vertices with tag X between two full vertices. Indeed, let us enumerate $w_1, w_2, \dots$ the vertices with tag X from left to right along the backbone P_1. A solution F contains at most k free edges and the slices of the vertices of the backbone are disjoint, so at most k slices of vertices w_i contain an edge of F. Hence, if we partition the set of all slices $SL(w_i)$ into $k+1$ classes according to their index i modulo $k+1$, the solution F will not intersect one of these classes. We branch on these $k+1$ choices. Assume for instance that F does not contain an edge in all $SL(w_i)$ where i divides $k+1$. Therefore, we can safely contract each of such slices $SL(w_i)$ onto w_i. This makes w_i a full vertex.

Let us now enumerate the full vertices $z_1, z_2, \dots$ from left to right. Let uv be a stable XX-edge. There exists a full vertex z_i to the left of u (with possibly $z_i = u$) and a full vertex z_{i+1} to the right of v. Since the number of vertices with tag X between z_i and z_{i+1} is at most k, the number of XX-edges between z_i and z_{i+1} is at most $k+1$. The *rank* of uv is the index of uv in the enumeration of the edges between z_i and z_{i+1} from left to right. Every edge of P_1 has some rank between 1 and $k+1$. In particular, we can branch over the rank of the selected stable XX-edge. Assume for instance that the rank of the chosen edge is 1. We then contract all edges which are not stable XX-edges with rank 1. This leaves only full vertices on P_1 since by construction there is a full vertex between two edges of the same rank. □

Note that after performing the reduction of Lemma 157, if v_iv_{i+1} is a stable XX-edge, then for every vertex $w \in P_j$ with $j \in X$ which lies between v_i and v_{i+1} in P_j, every wY-path contains v_i or v_{i+1}. In particular, if $X = \{1,\dots,\lambda\}$ then v_i is an xy-cut-vertex in G_1, and v_{i+1} as well.

Lemma 158. *We can assume that* $X = \{1,\dots,\lambda\}$. *In other words, we can reduce to the case where every vertex of* P_1 *is a cut-vertex of* G_1.

Proof. Assume that X is not equal to $\{1,\dots,\lambda\}$. We show that we can partition the component G_1 into two components G_1^1 and G_1^2. This partition leaves the free-connectivity unchanged, but decreases the slice connectivity. A vertex v_i of the backbone P_1 is *left clean* if the edge $v_{i-1}v_i$ of P_1 is a stable XX-edge, but the edge v_iv_{i+1} of P_1 is not. The vertex v_i is *right clean* if the edge v_iv_{i+1} is a stable XX-edge, but the edge $v_{i-1}v_i$ is not. Finally, v_i is *clean* if both $v_{i-1}v_i$ and v_iv_{i+1} are stable XX-edges. When enumerating all left clean and right clean vertices from left to right, we obtain the sequence of distinct vertices $r_1, l_1, r_2, l_2, \dots, r_p, l_p$ where the r_i are the right clean vertices and the l_i are the left clean vertices. Observe that x and y do not appear in the sequence since their tag is $\{1,\dots,\lambda\}$. Let us consider a pair r_i, l_i. We say that a vertex v of G_1 is *between* r_i and l_i if every path from v to x or y intersects $\{r_i, l_i\}$. Let B_i be the set of vertices which are between r_i and l_i. Let B be the union of B_i for $i = 1,\dots,p$.

Let G_1^1 be a copy of the subgraph induced by B on G_1. Observe that G_1^1 has p connected components, since $l_i \neq r_{i+1}$. We contract in G_1^1 the vertices l_i and r_{i+1}, for all $i = 1,\dots,p-1$, hence making G_1^1 connected. Finally, we contract the vertex r_1 of G_1^1 with x, and we contract the vertex l_p of G_1^1 with y, so that G_1^1 is a Y-component which has x and y as attachment vertices. The backbone P_1^1 of G_1^1 simply consists of the edges of the original backbone.

To construct G_1^2, we remove from G_1 all the vertices of B which are not left clean or right clean vertices. Hence no stable XX-edge is left in G_1^2. We contract backbone edges of G_1^2 as follows. All the backbone vertices between x and r_1 are contracted to a vertex $w_1 := x$, more generally all the vertices between l_i and r_{i+1} are contracted to a new vertex called w_{i+1}, and finally all the vertices between l_p and y are contracted to $w_{p+1} := y$. We add in G_1^2 the path $w_1, w_2, \dots, w_{p+1}$ which is the backbone P_1^2 of G_1^2. We correlate the edges of the backbone of G_1^1 and G_1^2 by adding clauses implying that the chosen edge of P_1^2 is w_iw_{i+1} if and only if the chosen edge of P_1^1 is between r_i and l_i. We finally branch to split the number of free edges f_1 chosen in G_1 into $f_1^1 + f_1^2 = f_1$, the respective free edges deleted in G_1^1 and G_1^2. Let us call G' the graph G in which G_1 is replaced by G_1^1 and G_1^2. Note that the free edges of G_1 are partitioned into the free edges of G_1^1 and of G_1^2. Observe that the free-connectivity of G and G' are equal. However, the slice connectivity has decreased in G', since its value is 0 in G_1^1 and strictly less than $\lambda - |X|$ in G_1^2. Indeed, for $i = 1,\dots,p-1$, either the edge l_il_{i+1} is unstable or the tag of l_i or l_{i+1} strictly contains X. Hence, contracting all

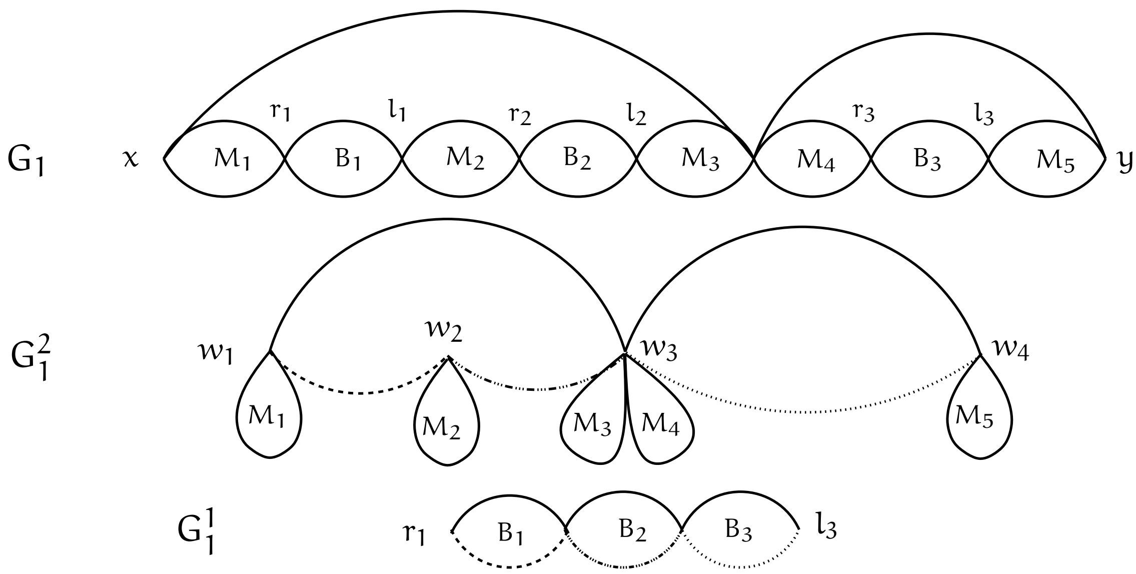


Figure 7.5: At the top, a component G_1 before transformation in the proof of Lemma 158. The new components G_1^1 and G_1^2 are depicted at the bottom and in the center respectively. The backbone edge in G_1^2 between w_i and w_{i+1} (dashed, mixed or dotted) is correlated to the subpath of the backbone contained in B_i in G_1^1 (similarly dashed, mixed or dotted). The vertices w_1 and r_1 are identified with vertex x, and the vertices w_4 and l_3 are identified with y, so that the xy-component G_1 is replaced by two xy-components G_1^1 and G_1^2.[2]

vertices between l_i and r_{i+1} strictly increases the tag of the resulting vertex in G_1^2. So the invariant improves. Figure 7.5 gives an example of this transformation.

Remains to prove that there exists a multicut in G' if and only if there exists a multicut in G which uses a stable XX-edge. This comes from the following observation. Let $e = v_j v_{j+1}$ be a stable XX-edge of P_1 between r_i and l_i. Let G_e be the graph obtained from G by deleting e, contracting x with all the vertices of P_1 to the left of v_j, and contracting y with all the vertices of P_1 to the right of v_{j+1}. Let G'_e be the graph obtained from G' by deleting e in P_1^1, deleting the edge $w_i w_{i+1}$ correlated to e in P_1^2, contracting x with all the vertices of P_1^1 to the left of v_j and all the vertices of P_1^2 to the left of w_i, and contracting y with all the vertices of P_1^1 to the right of v_{j+1} and all the vertices of P_1^2 to the right of w_{i+1}. The key fact is that G_e is equal to G'_e. Hence the multicuts in G and G' selecting the edge e are in one to one correspondence. □

The proof of Lemma 158 produces a new component, hence a new edge to be chosen

2. The names M_i are not used in the proof of Lemma 158, their purpose is just to identify what vertices which do not belong to a set B_i become in G_1^2.

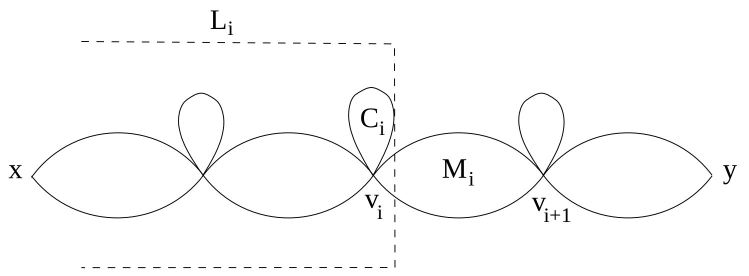


Figure 7.6: The left subgraph L_i of v_i. The set C_i is the v_i-cherry, and M_i is the lemon of the backbone edge v_iv_{i+1}.

in a backbone. This increases the deletion allowance by 1, but the number of free edges has not increased. The slice connectivity decreases, so the the invariant improves.

7.6.6 Reducing the Lemons

Thanks to the results of the previous section, we can assume that each vertex of the backbone P_1 of G_1 intersects all paths P_i for $i = 1, \dots, \lambda$. Let v_iv_{i+1} be an edge of the backbone P_1. The v_i*-cherry* C_i is the set of all vertices u of G_1 such that every uY-path contains v_i.

The *lemon* M_i of v_iv_{i+1} is the set consisting of v_i, v_{i+1} and of all vertices u of G_1 which do not belong to a cherry and such that every ux-path in G_1 contains v_i and every uy-path in G_1 contains v_{i+1}. Observe that when contracting v_iv_{i+1}, the lemon M_i becomes part of the v_i-cherry, where v_i denotes the resulting vertex. We denote by L_i the union of all C_j with $j \le i$ and all M_j with $j < i$. We call L_i the *left subgraph* of v_i. Similarly, the *right subgraph* R_i of v_i is the union of all C_j with $j \ge i$ and all M_j with $j > i$. See Figure 7.6.

If a multicut F selects the edge v_iv_{i+1} in the backbone, then the vertices $x, v_1, \dots, v_i$ all lie in the same connected component of $G \setminus F$. When these vertices $x, v_1, \dots, v_i$ are contracted to x, the set L_i becomes an x-cherry. Half-requests through y with an endpoint in L_i are automatically cut since F splits Y. Consider the terminals T_i of half-requests of L_i which are routed via x. Note that these half-requests are equivalent to usual requests, since L_i is now an x-cherry. By Theorem 143 we can reduce T_i to a bounded set of terminals K_i. This motivates the following key definition.

By Lemma 144, we define $\mathcal{L}_i$ to be a bounded active set of edges in the x-cherry obtained from L_i by contracting vertices $x, v_1, \dots, v_i$. By Theorem 145, we can compute such sets $\mathcal{L}_i$ so that $\mathcal{L}_j \cap L_i \subseteq \mathcal{L}_i$ when $i \le j$.

Let us say that a multicut F selecting v_iv_{i+1} in P_1 is *proper* if $F \cap L_i$ is included in $\mathcal{L}_i$.

Lemma 159. *If there exists a multicut F of size at most k containing the backbone edge v_iv_{i+1}, then there is a proper multicut F' of size at most k containing v_iv_{i+1}.*

Proof. Consider a multicut F containing $v_i v_{i+1}$. As the set $\mathcal{L}_i$ is active in the cherry obtained by contracting the path $x, v_1, \dots, v_i$ in L_i, there exists a multicut F' of size k such that $F' \setminus L_i = F \setminus L_i$ and $F' \cap L_i \subseteq \mathcal{L}_i$. Hence F' is proper and contains $v_i v_{i+1}$. $\square$

We denote by $\mathcal{L}$ the set of all subsets F of size at most k contained in some $\mathcal{L}_i$. We denote by c the maximum size of a set $\mathcal{L}_i$. Note that c is bounded in terms of k.

Given two sets $F_i \subseteq \mathcal{L}_i$ and $F_j \subseteq \mathcal{L}_j$ with $j \geq i$, let us write $F_i \preceq F_j$ when $F_j \cap L_{i+1} \subseteq F_i$. Observe that $\preceq$ is a partial order. A subset $\mathcal{F}$ of $\mathcal{L}$ is *correlated* if:

– elements of $\mathcal{F}$ have the same size, and
– $\mathcal{F}$ is a chain for $\preceq$, *i.e.* for every F_i and F_j in $\mathcal{F}$, with $F_i \subseteq \mathcal{L}_i$, $F_j \subseteq \mathcal{L}_j$ and $j \geq i$, we have $F_j \cap L_{i+1} \subseteq F_i$.

Lemma 160. *There exists a partition* $\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_{k(2c)^k}$ *of* $\mathcal{L}$ *into* $k(2c)^k$ *correlated sets.*

Proof. Let us prove by induction on $\ell = 0, \dots, k$ that there exists no antichain for $\preceq$ in $\mathcal{L}$ consisting of $(2c)^\ell + 1$ sets of size at most l. This clearly holds for $\ell = 0$. Assume that this holds for $\ell - 1$. By contradiction, let $A = \{F_1, F_2, \dots, F_{(2c)^\ell + 1}\}$ be an antichain of sets of size at most ℓ. Let t_i be an integer such that $F_i \subseteq \mathcal{L}_{t_i}$ for $i = 1, \dots, (2c)^\ell + 1$. We assume that the sets F_i are enumerated in such a way that $t_i \leq t_j$ whenever $i \leq j$. The set F_1 is incomparable to all sets F_i with $i > 1$, hence $F_i \cap L_{t_1+1} \not\subseteq F_1$ for all $i > 1$. In particular $F_i \cap L_{t_1+1}$ is nonempty, hence all sets F_i, for $i = 1, \dots, (2c)^\ell + 1$, have an edge in L_{t_1+1}. The sets F_i such that $t_i = t_1$ have an edge in $\mathcal{L}_{t_1}$ by definition. The sets F_i such that $t_i > t_1$ have an edge in $\mathcal{L}_{t_1+1}$ as $\mathcal{L}_{t_i} \cap L_{t_1+1} \subseteq \mathcal{L}_{t_1+1}$, by definition of the sets $\mathcal{L}_i$. Since the size of $\mathcal{L}_{t_1} \cup \mathcal{L}_{t_1+1}$ is at most $2c$, there exists a subset B of A of size at least $(2c)^{\ell-1} + 1$ of sets F_i sharing a same edge $e \in \mathcal{L}_{t_1} \cup \mathcal{L}_{t_1+1}$. The set $\{F \setminus e | F \in B\}$ has size $|B| \geq (2c)^{\ell-1} + 1$ and is an antichain of sets of size at most $\ell - 1$ by definition of $\preceq$. This contradicts the induction hypothesis.

By Dilworth's Theorem, there exists a partition of $\mathcal{L}$ into $(2c)^k$ sets totally ordered by $\preceq$, which can be be refined according to the cardinality to obtain a partition into $k(2c)^k$ correlated sets. Such a partition can be found in FPT time. $\square$

Let us now consider such a partition $\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_{k(2c)^k}$ of $\mathcal{L}$ into correlated sets. Observe that by Lemma 159 we can restrict our search to multicuts of the following type in G_1:

– A backbone edge $v_i v_{i+1}$.
– Other edges in the lemon M_i, which separate v_i from v_{i+1} in M_i.
– Edges in $\mathcal{L}_i$.
– Edges in $\mathcal{R}_i$, which is defined analogously to $\mathcal{L}_i$, with the roles of vertices x and y reversed.

Lemma 161. *We can assume that there are no cherries* C_i. *Moreover, if a multicut of size at most* k *exists, there exists one which contains only edges in one lemon* M_i.

Proof. By Lemma 159, if there exists a multicut F containing the backbone edge $v_t v_{t+1}$, then there exists a proper multicut F' containing $v_t v_{t+1}$. By definition $F' \cap L_t \subseteq \mathcal{L}$.

We branch over the existence of a proper solution F' such that $F' \cap L_t \in \mathcal{F}_j$ for $j = 1, \dots, k(2c)^k$, where t is the integer such that $v_t v_{t+1} \in F'$. Let us assume that we are in the branch where $F' \cap L_t \in \mathcal{F}_j$. A backbone edge $v_i v_{i+1}$ is in the *support* of $\mathcal{F}_j$ if there exists some $F_i \in \mathcal{F}_j$ such that $F_i \subseteq \mathcal{L}_i$. When $v_i v_{i+1}$ is in the support we say that lemon M_i is a *support lemon.* In this case, there actually exists a unique set in $\mathcal{F}_j$, which we denote by F_i, such that $F_i \subseteq \mathcal{L}_i$, as $\mathcal{F}_j$ is totally ordered under $\preceq$. Let ℓ be the number of edges of the sets in $\mathcal{F}_j$.

Claim 162. *For all* $F_a \in \mathcal{F}_j$, *if* M_i *is a support lemon then* $F_a \cap M_i = \emptyset$.

Proof. As $\mathcal{L}$ contains no backbone edge by definition, it is enough to show that u is not disconnected from v_i in $G_1 \setminus F_a$. As M_i is a support lemon, there exists a set $F_i \in \mathcal{F}_j$ such that $F_i \subseteq \mathcal{L}_i$. Consider a set $F_a \in \mathcal{F}_j$ with $F_a \subseteq \mathcal{L}_a$. If $a \le i$, then $F_a \subseteq L_a \subseteq L_i$, so $F_a \cap M_i = \emptyset$. If $a \ge i$, then $F_a \cap L_{i+1} \subseteq F_i \subseteq L_i$ as $\mathcal{F}_j$ is correlated, hence $F_a \cap M_i = \emptyset$ holds as well. This completes the proof of Claim 162. □

Consider a vertex u such that either u belongs to some cherry C_i or u belongs to a lemon M_i which is not a support lemon. An edge $v_a v_{a+1}$ in the support *affects* a half-request (u, x, v) if $a < i$ or if $i \le a$ and the unique set $F_a \in \mathcal{F}_j$ such that $F_a \subseteq \mathcal{L}_a$ separates u from x in G_1. If $v_a v_{a+1}$ does not affect (u, x, v), then neither does $v_b v_{b+1}$ when $b \ge a$. Indeed when $b \ge a$, $F_b \subseteq \mathcal{L}_b$ and $F_b \in \mathcal{F}_j$, we have that $F_b \cap L_a \subseteq F_a$.

Let us now modify the instance. If no edge of the support affects a half request (u, x, v), where either u belongs to some cherry C_i or u belongs to a lemon M_i which is not a support lemon, we remove (u, x, v) from R and add the half-request (x, x, v). Otherwise we let $v_a v_{a+1}$ be the support edge with a maximal which affects (u, x, v). We replace (u, x, v) in R by (v_{a+1}, x, v). We call this process *projecting* the half-request (u, x, v). After projecting all half-requests via x with an endpoint in a cherry or in a lemon M_i which is not a support lemon, we decrease f_i by ℓ and contract every edge of P_1 which is not in the support of $\mathcal{F}_j$. Note that, if $v_i v_{i+1}$ is not in the support, then there remains no half-request via x with an endpoint in M_i.

Assume that F' is a solution of this reduced instance which uses an edge $v_a v_{a+1}$ in the support. Let F_a be the element of $\mathcal{F}_j$ such that $F_a \subseteq \mathcal{L}_a$. Then $F' \cup F_a$ is a solution in the original instance. Indeed the half-requests with an endpoint in the support lemons are cut in $F' \cup F_a$ if and only if they are cut by F', as F_a does not intersect these lemons by Claim 162. Also, the half-requests with an endpoint in the lemons which are not support lemons or with an endpoint in the cherries are cut in the reduced instance if and only if they are cut by F_a in the initial instance by construction.

Conversely, assume that F is a proper solution in the original instance which uses the edge $v_a v_{a+1}$ and such that $F \cap L_a \in \mathcal{F}_j$. In particular, $F_a = F \cap L_a$, so $F \setminus F_a$ is a solution of the reduced instance. Indeed, all half-requests (u, x, v) cut by F_a in the original instance are affected by $v_a v_{a+1}$, hence they have been projected to (v_i, x, v) with $i \geq a+1$, and they are cut by $F \setminus F_a$ in the reduced instance.

The reduction, consisting in projecting all half-requests with an endpoint in a cherry or in a lemon which is not a support lemon, improves the invariant unless $\ell = 0$, *i.e.* unless the proper solution of the original instance with backbone edge $v_i v_{i+1}$ does not use any edge in L_i. In this case, all the requests via x of cherry C_j are projected to v_j, for all j. By the same argument, we can assume that no edge in a proper solution is selected to the right of M_i and that the half-requests via y of C_j are projected to v_j. In this case, there remains no terminal in the cherries, so we simply contract the cherries. We are only left with lemons, and we moreover know that if a solution exists, then there exists a solution which uses only edges in a single lemon. This concludes the proof of Lemma 161. □

Theorem 163. *We can assume that* G_1 *only consists of the backbone* P_1.

Proof. We assume that $\lambda > 1$ and show that we can improve the invariant. Let us consider a backbone edge $v_i v_{i+1}$. We denote by W the multiset of vertices $\{w_2, \dots, w_\lambda\}$ where w_j is the vertex of the slice S_i of v_i in M_i which belongs to the path P_j and has a neighbour in $M_i \setminus S_i$. In other words, w_j is the rightmost vertex of each path P_j in the slice of v_i. These vertices w_j are not necessarily distinct, for instance if v_i has degree λ in M_i, the slice S_i is exactly $\{v_i\}$ hence all w_j are equal to v_i for $j = 2, \dots, \lambda$. We also denote by $Z = \{z_2, \dots, z_\lambda\}$ the multiset of vertices of the slice T_i of v_{i+1} in M_i which belong respectively to the paths $P_2, \dots, P_\lambda$ and have a neighbour in $M_i \setminus T_i$.

A multicut F induces a partition of $W \cup Z$ according to the components of $G \setminus F$. A vertex of $W \cup Z$ has three possible *types*: it can be in the same component as x after the removal of F, in the same component as y, or in another component. Observe that, if two vertices a, b of $W \cup Z$ belong to components distinct from the components of x and y in $G \setminus F$, then F is still a multicut after contracting a and b. Hence F induces a partition of W into three parts, each of which can be contracted, and F remains multicut. We now branch over all partitions of $W \cup Z$ into three parts WZ_x, WZ_y, WZ_u, where WZ_x are vertices which are in the same component as x, WZ_y are vertices which are in the same component as y, and WZ_u are vertices of the same type, possibly disconnected from x and y (but not necessarily so). We branch over all possible partitions of W into WZ_x, WZ_y, WZ_u, and contract in each branch WZ_x to v_i, WZ_y to v_{i+1}, and WZ_u (if not empty) is contracted to a single vertex called u_i. In each branch, these contractions are performed simultaneously in all lemons M_i. We denote by G'_1 the resulting component, by M'_i the contracted lemon M_i, and by S'_i the contracted S_i.

If some vertex of W belongs to WZ_y, or if some vertex of Z belongs to WZ_x, or if WZ_u intersects both W and Z, then the xy edge-connectivity increases in G'_1. Indeed, in all these cases, there exists an xy-path in G'_1 without edges of $\lambda(x,y)$-cut in G_1. This improves the invariant, but we cannot directly conclude since the edges of the backbone may not be critical any longer. Indeed, the connectivity between v_i and v_{i+1} in M'_i could be smaller than the connectivity of another lemon M'_j, in which case the backbone edge v_jv_{j+1} is not critical. To get a correct instance of BACKBONE MULTICUT, we simply branch on the connectivity of the lemon M'_i corresponding to the chosen edge v_iv_{i+1}. In the branch corresponding to connectivity l, we contract the backbone edges v_iv_{i+1} where M'_i has connectivity distinct from l.

Hence we can assume without loss of generality that W is partitioned into WZ_u and WZ_x, and that $Z = WZ_y$. As we contract WZ_y to v_{i+1}, the vertex v_{i+1} has now degree λ in M'_i, and T_i is a v_{i+1}-cherry. Let us assume that $WZ_u \neq \emptyset$. As we have contracted the vertices of W to v_i and u_i, the set S'_i has exactly two vertices with a neighbour in $M'_i \setminus S'_i$, namely v_i and u_i. Note that the degree of v_i in $M'_i \setminus S'_i$ is exactly the number of vertices w_j chosen in WZ_x (with multiplicity, since WZ_x is a multiset). We denote this degree of v_i in $M'_i \setminus S'_i$ by d. Note that d does not depend on i since we have chosen in every M_i the same subset WZ_x inside $\{w_2, \ldots, w_\lambda\}$.

Let λ_S be the v_iu_i edge-connectivity in S'_i. If $\lambda_S > f_1$, then u_i and v_i cannot be separated, so we contract u_i and v_i. We branch in order to assume that λ_S is some fixed value. In the branch corresponding to connectivity λ_S, we contract backbone edges v_iv_{i+1} where S_i has connectivity distinct from λ_S. Let $P'_1, \ldots, P'_{\lambda_S}$ be a collection of edge disjoint paths between u_i and v_i in S'_i. We denote by S' the slice of v_i in S'_i, and once again we consider the rightmost vertices $W' = \{w'_1, \ldots, w'_{\lambda_S}\}$ of S' in the paths P'_j. We branch over all possible partitions of W' into W'_x, W'_y, W'_u. Once again, if W'_y is not empty, we increase the connectivity between x and y. Observe that W'_u can be contracted to WZ_u, hence to u_i. In particular if W'_u is not empty, we increase the connectivity between v_i and u_i in S'_i. We iterate this process in S'_i until either W'_u is empty in which case v_i has degree λ_S in S'_i, or λ_S exceeds f_1 in which case we contract v_i and u_i.

We apply Lemma 161 to G'_1. Therefore, we can assume that no cherries are left and that if a solution exists, one multicut is contained in some M'_i. Two cases can happen:

If $f_1 \geq d+\lambda_S+\lambda-1$, and the edge v_iv_{i+1} is chosen in the backbone, then we can assume that the restriction of the multicut to M'_i simply consists of all the edges incident to v_i and v_{i+1} in M'_i. Indeed v_i is incident to $d+\lambda_S$ free edges, and v_{i+1} is incident to $\lambda-1$ free edges. This is clearly the best solution since it separates all vertices of $M'_i \setminus \{v_i, v_{i+1}\}$ from v_i and v_{i+1}. Therefore, we project every request (u,x,v) where $u \in M'_i$ to (v_{i+1},x,v) and project every request (u,y,v) where $u \in M'_i$ to (v_i,y,v). Finally we reduce f_1 to 0 and we delete all vertices of G'_1 which are not in P_1.

Assume now that $f_1 < d+\lambda_S+\lambda-1$. We branch over $2(\lambda-1)$ choices, where the branches are named B_j and B'_j for all $j = 2, \ldots, \lambda$. In the branch B_j, we assume that only one edge of the solution is selected in P_j, and that this edge is critical. In the branch B'_j, we assume that all the edges of the solution selected in P_j are not critical. In the branch B'_j, we contract non critical edges of P_j and improve the invariant. In the branch B_j, we find a new backbone P_j. In this last case, we delete the edges of P_1 and reduce the number of free edges to $f_1 - 1$. We also translate the clauses in terms of edges of the new backbone P_j.

Indeed the number of edges in the backbone of G_1 has changed. Clauses of the form $c_1 \le i$ become $c_1 \le \epsilon(i)$ where $\epsilon(i)$ denotes the index of the rightmost edge of P_j in the lemon M'_i.

This branching process covers all the cases where $v_i = u_i$ since in this case $f_1 < 2\lambda - 2$ and therefore one path P_j contains only one edge of the multicut. In the case $v_i \neq u_i$, assume that a multicut F is not of a type treated in one of the branches. In other words, F contains at least two edges in each path P_j for $j = 2, \dots, \lambda$, and at least one of them is critical. Then F contains two edges in each of the d paths P_j not containing u_i since F does not respect the branches B_j for $j = 2, \dots, \lambda$. Also, F contains one edge outside S'_i in each path P_j containing u_i since edges in S'_i are not critical and F is not treated in the branches B'_j. Thus F contains at least $2d + (\lambda - d - 1)$ free edges outside S'_i. Hence less than λ_S edges of F lie in S'_i, thus v_i and u_i belong to the same component in $G - F$. This case is covered in another branch in which v_i and u_i are contracted. Hence this branching process is exhaustive, and this completes the proof of Theorem 163. □

7.6.7 Reducing to 2-SAT

We are left with instances in which the Y-components with two attachment vertices only consist of a backbone. We now reduce the last components.

Lemma 164. *We can assume that there is no component with one attachment vertex.*

Proof. Let $Y = \{y_1, \dots, y_p\}$ and let k be the number of free edges in the multicut. A vertex $y_i \in Y$ is *safe* if there is no request between two components attached only to y_i. If y_i is not safe then there is a request (u, y_i, v), with u and y in components attached to y_i, hence y_i must be either disconnected from u or disconnected from v by the solution. We explore one branch where u is added to Y, and one branch where v is added to Y. This creates a component with two attachment vertices. This component has a backbone, and the number of free edges decreases.

Hence, we can assume that all the vertices of Y are safe. The y_i-cherry is the union of all the components attached to y_i. We branch over all possible integer partitions of k into a sum $k_1 + k_2 + \cdots + k_p = k$. In each branch, we require that k_i edges are deleted in the y_i-cherry for $i = 1, \dots, p$. By Lemma 144, the y_i-cherry has a bounded active set $\mathcal{L}_i$, hence in the y_i-cherry we can consider only a bounded number of cuts of size k_i: all subsets of $\mathcal{L}_i$ of size k_i. We branch over these different choices. In a given branch, we delete a particular set of edges F_i in the y_i-cherry. Thus, we delete the vertices of the y_i-cherry isolated from y_i by F_i, and contract the other vertices of the y_i-cherry to y_i. Finally, no Y-cherry remains. □

Theorem 165. *Multicut is FPT.*

Proof. By Lemma 164, to prove that BACKBONE MULTICUT is FPT, we only have to deal with an input graph G which is a subdivision of a graph with at most k edges, and where a multicut must consist of exactly one edge in each subdivided path. Let us consider a half-request (v_i, x, v'_j). Assume without loss of generality that $v_i \in G_1$, $v'_j \in G_2$, and x belongs to G_1 and G_2 (if x does not belong to G_1 or G_2, then splitting Y automatically results in cutting the half-request (v_i, x, v'_j)). For simplicity, we assume that the edges of both P_1 and P_2 are enumerated in increasing order from x. We add to $\mathcal{C}$ the clauses $x_1 \geq i \Rightarrow x_2 \leq j-1$ and $x_2 \geq j \Rightarrow x_1 \leq i-1$. We transform all the half requests in this way. We are only left with a set of clauses which we have to satisfy.

We finally add all the relations $x_i \geq a \Rightarrow x_i \geq a-1$ and $x_i \leq a \Rightarrow x_i \leq a+1$ and $x_i \geq a \Rightarrow \neg(x_i \leq a-1)$ and $x_i \leq a \Rightarrow \neg(x_i \geq a+1)$ to ensure the coherence of a satisfying assignment. We now have a 2-SAT instance which is equivalent to the original multicut instance. As 2-SAT is solvable in polynomial time, this shows that BACKBONE MULTICUT is FPT. Hence the simpler COMPONENT MULTICUT problem is FPT. Together with Theorem 146 which reduces MULTICUT to COMPONENT MULTICUT, this concludes the proof of Theorem 165. □

To sum up, the invariants in the proof that BACKBONE MULTICUT is FPT are (in decreasing order of importance):

- The total number of free edges in the multicut, to minimize, bounded by k.
- The sum of the free connectivity in each component, to maximize, bounded by k.
- The sum of the slice connectivity in each component, to minimize, bounded by k. [3]

The algorithm starts by branching over the tag XY of the backbone edge chosen in the multicut in a given component. When $X \neq Y$, the invariant improves by Lemma 155. The dominant complexity term comes from the $O(2^k)$ branches where $X = Y$. Tags may change, and another such branching is done, and again the dominant term comes from the 2^{2k} cases where $X = Y$. If the chosen edge is unstable, we improve the invariant with a factor two branching in Lemma 156. If the edge is stable, we branch over k choices to decide which part of a partition modulo $k+1$ does not intersect the solution F in Lemma157, and branch again over $k+1$ choices for the rank of the backbone edge chosen in the solution. This yields $O(k^2 2^{2k})$ cases where all vertices are full. If a backbone vertex is not a cut-vertex, we increase the invariant by Lemma 158. Otherwise, we apply Theorem 163, which branches over $3^{O(k)}$ cases, in which either the component consists only in its backbone, or the invariant has improved.

When the whole process described in the previous paragraph has been performed over all components with two attachment vertices, yielding $O^*(k^{O(k)})$ branches, we apply Lemma 164. If a vertex in Y is not safe, the invariant improves. When all vertices are safe, the tree which represents the branchings made by the algorithm thus far (where child nodes are instances with a better invariant) has depth at most k^3 and the degree of its nodes

3. By "bounded" we mean bounded above, all the invariants described in this chapter being trivially non-negative. In both cases, maximize or minimize, the upper bound corresponds to the maximum number of times an invariant can be improved.

is bounded by $O^*(k^{O(k)})$. Hence the total number of leaves in the branching process thus far is $O^*(k^{O(k^4)})$.

Lemma 164 proceeds by branching over $O(k^k)$ cases to fix the number of edges chosen by the solution in each component, and then applies Lemma 144, branching exhaustively over all subsets with the adequate number of active edges in each component. This gives a branching factor of $O((|K|(k+1)!)^k)$, where K is the set of terminals in a given cherry, which is bounded by $O(k^{O(k^3)})$ by Theorem 143. The total number of branches obtained thus far is $O(k^{O(k^4)})$. Finally, Theorem 165 directly translates to a 2-SAT instance. Thus:

Remark 166. *The FPT algorithm for* BACKBONE MULTICUT *runs in time* $O^*(k^{O(k^4)})$.

7.7 Hints for Vertex-Multicut

This section contains a sketch of a translation of our proof for edge-multicut in terms of vertex-multicut. The proof has the same outline, and we just explain how the notions introduced for edge-multicut can be transferred to the vertex-multicut setting. What follows is more intended as a hint rather than a complete proof of the fact that the following version of Multicut is FPT:

VERTEX MULTICUT:
Input: A graph G, a set of requests R, a subset of vertices S, an integer k.
Parameter: k.
Output: TRUE if there is a vertex-multicut of size at most k which does not intersect S, otherwise FALSE.

In the standard version of VERTEX MULTICUT, the set S is empty. We use this slightly more general version for technical reasons. Let us now explain how we can translate the results of the previous sections for VERTEX MULTICUT.

For Section 7.4, the results are based on the submodularity of edge cuts. The vertex cuts being also submodular, we can transfer the results for vertices. Here an indivisible xy-cut is a set of vertices K which deletion separates x from y and such that no strict subset of K separates x from y. For the reduction from VERTEX MULTICUT to COMPONENT MULTICUT, the proof is essentially the same. One particularity of VERTEX MULTICUT is the following. When we contract vertices, we have to add the resulting vertex to S, the set of non-deletable vertices. Let Y be the vertex-multicut of size $k+1$ given by iterative compression. We can branch to decide which vertices of Y belong to the solution and then branch over the possible contractions of the set Y. Hence we can assume that $Y \subseteq S$. We have to replace arguments of the type "we add a vertex to Y" by "we branch to know if the vertex is added to Y or if it belongs to the solution". The connectivity between x and y is the maximum number of paths between x and y whose intersections with the set of deletable vertices are pairwise disjoint. The connectivity can be computed by flows with weight 1 for deletable

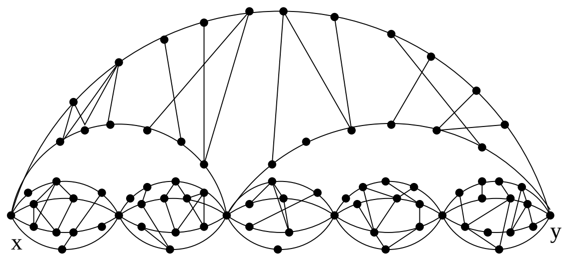


Figure 7.7: The transformation into lemons for VERTEX MULTICUT. The backbone is the path at the bottom. All even vertices of the backbone are full, with $X = \{1,2,3,4\}$, and all odd vertices belong to S.

vertices and ∞ for non-deletable vertices. A vertex of λ-cut is a deletable vertex which deletion decreases the connectivity. In the vertex-multicut context, a *backbone* is a path in which only one vertex will be selected in the multicut and where every odd vertex belongs to the set S. Additionally, all the deletable vertices of the backbone must be vertices of λ-cut.

To prove the existence of a backbone, we have to generalise Lemma 151. The border of the slice of x_i has size at most k but the number of vertices which touch this border can be arbitrarily large. We can branch to know if a vertex is deleted in the slice. If this is not the case then the slice can be contracted to x_i, hence x_i has only λ neighbours. Otherwise, we can branch to know if each vertex in the border is in the component of y_i or a new component. In each of these case the invariant improves. Hence, the only relevant case is when all the vertices of the border of x_i will lie in the same connected component as x_i. In this case, we can contract x_i with the border of its slice, which yields a cherry in which we have to delete vertices. By Lemma 144, we can bound the number of possible cuts. We can branch these cuts and decrease the deletion allowance.

Let us now turn to the BACKBONE MULTICUT problem for vertex-multicuts. A key definition of Section 7.6 is the notion of full vertex. We have to modify this notion, as contracted vertices are not deletable. Hence all the vertices of the backbone cannot be full as for edge-multicuts. Instead, we transform the instance to make all non-deletable vertices of the backbone full (see Figure 7.7). The slice $S(v)$ of a non-deletable vertex v is the connected component of v in G minus the vertices of λ-cut. We define the tag as for edge-multicut. A vertex v of the backbone is X-*stable* if v is deletable, and the tag of each of its two neighbours in the backbone is X, and the tag after the contraction of v with its two neighbours is still X. As for edge-multicuts, we can assume that we delete an X-stable vertex in the backbone. We can similarly define classes for Lemma 157, and remark that one class does not intersect the solution. All the vertices of each slice in this class can be contracted. This ensures that all the vertices which are non-deletable are full. We can write as in Lemma 158 that a non-deletable vertex is *left* (resp. *right*) *clean* if the vertex to its left (resp. right) is X-stable and then we can assume that $X = \{1, ..., \lambda\}$ as for edge-multicut.

In the reduction of the lemons for Vertex-Multicut, we cannot contract x with the border of its slice since it does not ensure that the degree of x is λ. Hence we have to contract x with vertices of its slice which touch the vertices of the border. The set of such vertices can be restricted to a bounded size with Lemma 144. Hence the same inductive method used for edge-multicut also holds.

7.8 Conclusion

7.8.1 A Single-Exponential Algorithm

Our proof was originally designed only to prove that MULTICUT is FPT, with no particular focus on algorithmic efficiency. For the sake of completeness, we briefly analysed the complexity after each of the two major parts of the proof. MULTICUT is reduced to COMPONENT MULTICUT in time $O^*(k^{O(k^6)})$ by Remark 153, and BACKBONE MULTICUT is solved in time $O^*(k^{O(k^4)})$ by Remark 166. Hence the overall running time of the algorithm is $O^*(k^{O(k^6)})$. This algorithm and its complexity analyis are definitely not fine tuned, and the running time could probably be vastly improved with slight changes to the proof and to the analysis.

7.8.2 Comparison with Marx and Razgon's Proof in [100]

Marx and Razgon independently proved that MULTICUT is FPT in [100]. Both proofs start with the Iterative Compression technique originally used in [99]. Reducing Y-components (our vertex-multicut Y is denoted by W in [100]) with three or more attachment vertices as we do in Lemma 150 essentially corresponds to Lemma 5.3 in [100]. The basic connectivity tools, important cuts in [100] and left cuts here are identical. At this point, the two proofs drastically diverge. While we concentrate on linearly structuring the (restriction inside a component of the) multicuts, Marx and Razgon focus on *non-isolating solutions, i.e.* solutions where no vertex is disconnected from Y. They exhibit a transformation from a positive instance to an instance which admits a non-isolating solution with a large enough probability ($2^{-O(k^3)}$). This probabilistic transformation can be derandomized into a single-exponential algorithm running in time $O^*(2^{O(k^3)})$ (Lemma 4.1). Finally, with an instance admitting a non-isolating solution, they reduce to Almost 2-SAT, which was proved FPT in [109].

Roughly speaking, the two proofs are about as much technically intricate. On the plus side, our proof is self-contained, while [100] uses Almost 2-SAT, which Fixed-Parameter Tractability had remained an important open question until recently. Also, not going

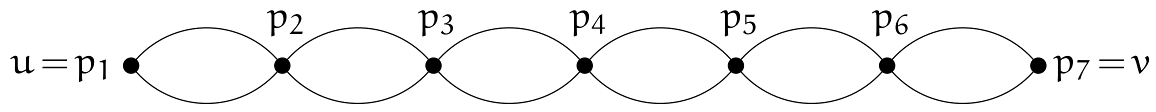


Figure 7.8: A double path on seven vertices.

through randomization and through Almost 2-SAT allows us to (arguably) get more insight on the structure of the problem. On the minus side, Marx and Razgon's algorithm is more efficient and "cleaner". Also, their proof works directly with vertex-multicuts, while our proof has been originally written in terms of edge-multicut. Finding a shorter proof retaining the best characteristics of each would be very interesting.

7.8.3 Other Leads

The approach developed in this chapter was not our original idea of how an FPT algorithm for MULTICUT should be designed. We made our first significant progress with a structural approach based on treewidth, pathwidth and minors. Chapter 4 uses the extra structure given by a large grid minor to reduce MULTICUT to bounded treewidth, and we hoped to proceed with the structure given by arbitrarily large pathwidth to reduce to graphs of bounded pathwidth, and hopefully deal with bounded pathwidth instances in a similar fashion. We could not execute this strategy, but this is one interesting direction for attempts towards a different proof / a better proof / a simpler algorithm / a faster algorithm.

In our solution, we do not use the fact that the treewidth can be assumed bounded, but this is a fact which can be used if helpful thanks to Chapter 4.

Considering finer concepts than the notion of request can also be envisioned. In the simpler case of MULTICUT IN TREES, a request is simply a path. In general graphs, a request can be seen as the set of paths between its endpoints. In our proof, we simulate requests by half-requests, partitioning the set of paths naturally associated to a request according to an intermediate point. This could be done thanks to the vertex-multicut Y. But we originally wanted to go much further, and consider the more general problem of cutting a prescribed set of paths, not necessarily all paths between given pairs of vertices. The obvious problem is that a request can be realized by exponentially many paths (exponentially many in n), but we loosely conjectured that this difficulty can be avoided as follows:

Question 167. *Given a graph G on n vertices, an integer k and two vertices u, v of G, does there always exist a set S of at most $f(k) * \mathrm{poly}(n)$ paths between u and v such that removing at most k edges of G to disconnect all paths in S must actually disconnect u and v? In other words, can an FPT number of paths simulate a request with respect to k-multicuts?*

Consider for starters the multigraph G consisting of a path on n vertices with endpoints $u = p_1$ and $v = p_n$, where each edge has been duplicated (into a 2-cycle), as in Figure 7.8.

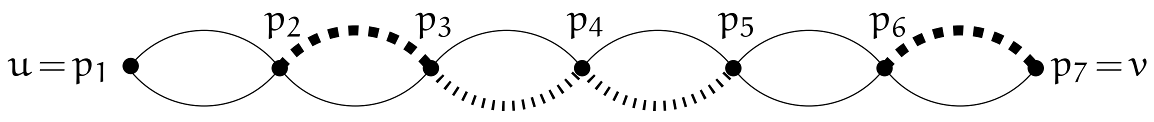


Figure 7.9: Assuming $k=4$, one constraint would be generated by $T=\{2,3,4,6\}$ partitioned into $\overline{T}=\{2,6\}$ (corresponding to the dashed edges) in and $\underline{T}=\{3,4\}$ (corresponding to the dotted edges). A solution must contain a uv-path avoiding all dotted edges and dashed edges.

We are looking for a small set of uv-paths, such that every hitting set of these paths is a uv-edge-cut.

Given a (simple) uv-path P and an integer $i \in \{1,\dots,n-1\}$, let us say that P *takes* i if P contains the top edge in *position* i, *i.e.* between p_i and p_{i+1}. We can reformulate the constraint on the solution set S as follows: for every set T of k positions and for every bipartition of T into $\overline{T}$ and $\underline{T}$, there must exist a path in S which takes all positions in $\overline{T}$ and takes no position in $\underline{T}$. Indeed, if this is not the case, then the bottom edges for positions in $\overline{T}$ and the top edges for positions in $\underline{T}$ form a set of k edges which hits S but does not cut u from v. See Figure 7.9 for an example.

Hence every bipartitioned set of k positions gives a constraint. There are $\binom{n}{k}2^k$ such constraints and a given simple uv-path satisfies (in the above sense) $\binom{n}{k}$ such constraints. Hence a random simple path satisfies a fraction $\frac{1}{2^k}$ of the constraints. In particular, there exists one path P which satisfies at least a fraction $\frac{1}{2^k}$ of the constraints. We pick the path P in our solution S and repeat. This process (not taking into account the computability of P) finds a solution of size at most $\log_{2^k}(\binom{n}{k}2^k)$ paths, which is actually even better than needed, with a logarithmic dependence on n rather than a polynomial dependence.

Question 168. *If Question 167 has a positive answer, can such an FPT set of paths emulating the request uv be computed in FPT time?*

In the simple example worked out above, the randomized process should easily be derandomizable into an FPT algorithm.

If Question 167 and Question 168 turn out to have positive answers, then MULTICUT can be reduced to the following problem:

PATH HITTING:
Input: A graph G, a set R of paths in G, an integer k.
Parameter: k.
Output: TRUE if there is a set at most k edges of G which hits R, otherwise FALSE.

It is not clear a priori whether this problem should be easier or harder than MULTICUT. Directly emulating a MULTICUT instance with PATH HITTING would require an exponential number of paths, and conversely the structure of the objects to be hit can be more complicated in PATH HITTING than in MULTICUT.

Question 169. *Is* PATH HITTING *FPT?*

We did not pursue this insight further when the ideas exposed in this chapter proved to be fruitful, but Questions 167, 168 and 169 remain very interesting nonetheless, on their own right as well as with respect to MULTICUT.

CHAPTER

8 Parameterized Algorithms for Finding Trees with Many Leaves

Sections 8.1 and 8.2 are devoted to a parameterized algorithm for MAXIMUM LEAF OUT-BRANCHING running in time $O^*(3.72^k)$. Section 8.3 applies this algorithm to derive an exponential exact algorithm for MAXIMUM LEAF OUT-BRANCHING, and was actually suggested by Serge Gaspers. These sections are based on [39], a joint work with Gregory Gutin, Eun-Jung Kim and Anders Yeo. In Section 8.4, we briefly discuss the randomized monomial technique of Koutis and Williams and its (non)-applicability to MAXLEAF, and we use this technique to provide a fast randomized algorithm for finding trees with many internal vertices.

Throughout this chapter, we denote the set of arcs of a digraph D by $A(D)$. Given a vertex x which belongs to a subgraph H of a digraph D, $N^+_H(x)$ and $N^-_H(x)$ denote the sets of out-neighbours and in-neighbours of x in H, respectively. Also, let $A^+_H(x) = \{xy : y \in N^+_H(x)\}$, $d^+_H(x) = |N^+_H(x)|$, and $d^-_H(x) = |N^-_H(x)|$. When $H = D$ we omit the subscripts in the notation above.

Let D be a digraph, let T be an out-tree and consider a set of vertices $L \subseteq V(D)$. A (T,L)-*out-tree* of D is an out-tree T' of D such that:

1. $A(T) \subseteq A(T')$.
2. Vertices of L are leaves in T'.
3. T and T' have the same root.

A (T,L)-*out-branching* is a (T,L)-out-tree which is spanning. Let $\ell_{\max}(D,T,L)$ be the maximum number of leaves over all (T,L)-out-branchings of D. We set this number to 0 if there is no (T,L)-out-branching. For an out-tree T in a digraph D, Leaf(T) denotes the set of leaves in T and Int$(T) = V(T) - $Leaf$(T)$, the set of *internal vertices* of T. Given a vertex x in an out-tree T let T_x denote the subtree of T rooted at x.

Throughout this chapter we use a triple (D,T,L) to denote a given digraph D, an out-tree T of D and a set of vertices $L \subseteq V(D) - \mathrm{Int}(T)$. We denote by $\hat{D}(T,L)$ the subgraph of D obtained from D by:

– deleting all out-arcs of vertices in L.

– deleting all arcs not in $A(T)$ which go into a vertex in $V(T)$.

Informally, $\hat{D}(T,L)$ corresponds to the available arcs when we want to extend a partial solution T into an out-branching, having decided that vertices in L must be leaves. This will be formally stated in Lemma 170. Note that by definition, vertices in L are either leaves of T or vertices not in T. When T and L are clear from the context we will omit them and denote $\hat{D}(T,L)$ by $\hat{D}$.

8.1 Another $4^k n^{O(1)}$ Time Algorithm

The algorithm of this section is similar to the branching algorithm in [88] briefly described on pages 117-118, with the following differences. We decide at an earlier stage which of the current leaves of T cannot be leaves in a final (T,L)-out-branching and turn them directly into internal vertices thanks to Lemma 171. This is Step 2 in Algorithm $\mathcal{A}(D,T,L)$. This decision works as a preprocessing of the given instance and gives us a better chance to come up with a (T,L)-out-tree with at least k leaves more quickly. A more important reason for this step is the fact that our algorithm is easier than the main algorithm in [88] to transform into a faster algorithm.

In the following, r denotes the root of the desired out-branching.

Lemma 170. *Given a triple* (D,T,L)*, we have* $\ell_{\max}(D,T,L) = \ell_{\max}(\hat{D},T,L)$.

Proof. If there is no (T,L)-out-branching in D, the subgraph $\hat{D}$ does not have a (T,L)-out-branching either and the equality holds trivially. Hence suppose that T^* is a (T,L)-out-branching in D with $\ell_{\max}(D,T,L)$ leaves. Clearly, $\ell_{\max}(D,T,L) \geq \ell_{\max}(\hat{D},T,L)$. Since the vertices of L are leaves in T^*, all arcs out of vertices in L do not appear in T^*, *i.e.* $A(T^*) \subseteq A(D) \setminus \{A^+_D(x) | x \in L\}$. Moreover, $A(T) \subseteq A(T^*)$ and thus all arcs not in $A(T)$ which go into a vertex in $V(T)$ do not appear in T^*. Indeed, otherwise we would have a vertex in $V(T)$ with more than one in-arc in T^*, or r would have an in-arc. Hence $A(T^*) \subseteq A(\hat{D})$ and the above equality holds. □

Lemma 171. *Given a triple* (D,T,L)*, the following equality holds for each leaf* x *of* T.

$$\ell_{\max}(D,T,L) = \max\{\ell_{\max}(D,T,L \cup \{x\}), \ell_{\max}(D,T \cup A^+_{\hat{D}}(x), L)\}$$

Proof. If $\ell_{\max}(D,T,L) = 0$ then the equality trivially holds, so we assume that $\ell_{\max}(D,T,L) \geq 1$. Since a $(T,L \cup \{x\})$-out-branching or $(T \cup A^+_{\hat{D}}(x), L)$-out-branching is also a (T,L)-out-branching, the inequality $\geq$ holds. To prove the opposite direction, assume T' is an optimal (T,L)-out-branching. If x is a leaf in T', then T' is a $(T,L \cup \{x\})$-out-branching and $\ell_{\max}(D,T,L) \leq \ell_{\max}(D,T,L \cup \{x\})$.

Suppose x is not a leaf in T'. Denote by T'' the digraph obtained from T' by deleting all arcs entering $N^+_{\hat{D}}(x)$ in T', and adding the arcs $A^+_{\hat{D}}(x)$. Note that $d^-_{T''}(y) = 1$ for each vertex $y \neq r$ in T'', and $A(T'') \subseteq A(\hat{D})$. In order to show that T'' is an out-branching it suffices to see that there is no cycle in T'' containing x. If there is a cycle C containing x in T'' and $xy \in A(C)$, then $C - \{xy\}$ forms a directed (y,x)-path in $\hat{D}$. As $x \in V(T)$ and $y \not\in V(T)$, this contradicts the fact that there is no path from $V(D) - V(T)$ to $V(T)$ in $\hat{D}$. Hence, T'' is an out-branching.

As no vertex in L has an out-arc in $\hat{D}$ we have that $L \subseteq \text{Leaf}(T'')$. Furthermore, $A(T) \subseteq A(T'')$ as $A(T) \subseteq A(T')$ and all arcs we deleted from $A(T')$ go to a vertex not in $V(T)$. Therefore T'' is a (T,L)-out-branching with as many leaves as T'. This proves that $\ell_{\max}(D,T,L) \leq \ell_{\max}(D,T \cup A^+_{D'}(x),L)$. □

Given a triple (D,T,L) and a vertex $x \in \text{Leaf}(T) - L$, define $T^{\text{root}}_{D,L}(x)$ as follows:

(1) Let $T^{\text{root}}_{D,L}(x) := \emptyset$ and $x' := x$.
(2) While $d^+_{\hat{D}}(x') = 1$, add $A^+_{\hat{D}}(x') = \{x'y\}$ to $T^{\text{root}}_{D,L}(x)$ and let $x' := y$.
(3) Add $A^+_{\hat{D}}(x')$ to $T^{\text{root}}_{D,L}(x)$.

The idea behind this definition is the following: during the algorithm, we will decide that a given leaf x of the partial out-tree T built thus far is not a leaf of the out-branching we are looking for. Then adding the out-arcs of x to T is correct. To make sure that the number of leaves of T has increased even when $d^+_{V-V(T)}(x) = 1$, we add $T^{\text{root}}(x)$ to T instead of just adding the single out-arc of x, as described in the following.

Lemma 172. *Consider a triple* (D,T,L) *and a leaf* $x \in$ *Leaf*$(T) - L$. *If* $\ell_{\max}(D,T,L \cup \{x\}) \geq 1$ *then the following holds:*

(i) *If* $|$*Leaf*$(T^{\text{root}}_{D,L}(x))| \geq 2$ *then* $\ell_{\max}(D,T,L) = \max\{\ell_{\max}(D,T,L \cup \{x\}), \ell_{\max}(D,T \cup T^{\text{root}}_{D,L}(x),L)$.

(ii) *If* $|$*Leaf*$(T^{\text{root}}_{D,L}(x))| = 1$ *then* $\ell_{\max}(D,T,L) = \ell_{\max}(D,T,L \cup \{x\})$.

Proof. Let us start with (ii). Note that if $|\text{Leaf}(T^{\text{root}}_{D,L}(x))| = 1$ then an optimal (T,L)-out-branching T' must verify $|\text{Leaf}(T'_x)| = 1$. Assume that T' is an optimal (T,L)-out-branching and that $|\text{Leaf}(T'_x)| = 1$. Let us show that $\ell_{\max}(D,T,L \cup \{x\}) = |\text{Leaf}(T')| = \ell_{\max}(D,T,L)$. If x is a leaf of T' then this is clearly the case, so assume that x is not a leaf of T'. Let y be the unique out-neighbour of x in T'. As $\ell_{\max}(D,T,L \cup \{x\}) \geq 1$, there exists a path $P = p_0(= r)p_1p_2 \ldots p_r(= y)$ from the root r of T to y in $\hat{D}(T,L \cup \{x\})$. Assume that q is chosen such that $p_q \not\in T'_x$ and $\{p_{q+1}, p_{q+2}, \ldots, p_r\} \subseteq V(T'_x)$. In the digraph $D^* = D[V(T'_x) \cup \{p_q\} - \{x\}]$, every vertex is reachable from p_q in D^*. Therefore there exists an out-branching in D^*, say T^*, with p_q as the root. Let T'' be the out-branching obtained from T' by deleting all arcs in T'_x and adding all arcs in T^*. We have that $|\text{Leaf}(T'')| \geq |\text{Leaf}(T')|$ as vertices in $\text{Leaf}(T^*) \cup \{x\}$ are leaves in T'' and vertices in $\text{Leaf}(T'_x) \cup \{p_q\}$ are the only leaves in T' which may not be leaves in T'', and $|\text{Leaf}(T'_x) \cup \{p_q\}| = 2$. Therefore, $\ell_{\max}(D,T,L \cup \{x\}) \geq |\text{Leaf}(T')| =$

$\ell_{\max}(D,T,L)$. We always have $\ell_{\max}(D,T,L) \geq \ell_{\max}(D,T,L\cup\{x\})$, which completes the proof of (ii).

Let us turn to the proof of (i), where $|\text{Leaf}(T_{D,L}^{\text{root}}(x))| \geq 2$. Let Q denote the set of leaves of $T_{D,L}^{\text{root}}(x)$ and let $R = V(T_{D,L}^{\text{root}}(x)) - Q$. Note that by the construction of $T_{D,L}^{\text{root}}(x)$ the vertices of R can be ordered $(x =)r_1, r_2, \dots, r_i$ such that $r_1r_2\dots, r_i$ is a path in $T_{D,L}^{\text{root}}(x)$. Once again, let T' be an optimal (T,L)-out-branching. If a vertex r_j with $1 \leq j \leq i$ is a leaf of T' then $|\text{Leaf}(T'_x)| = 1$ and the above gives us $\ell_{\max}(D,T,L\cup\{x\}) = \ell_{\max}(D,T,L)$. This proves part (i) in this case, as we always have $\ell_{\max}(D,T,L) \geq \ell_{\max}(D,T\cup T_{D,L}^{\text{root}}(x),L)$. Therefore, let us assume that no vertex in $\{r_1, r_2, \dots, r_i\}$ is a leaf of T' and that all arcs $(x =)r_1r_2, r_2r_3, \dots, r_{i-1}r_i$ belong to T'. By Lemma 171 we may furthermore assume that T' contains all the arcs from r_i to vertices in Q. Thus, $T_{D,L}^{\text{root}}(x)$ is a subtree of T' and $\ell_{\max}(D,T,L) = \ell_{\max}(D,T\cup T_{D,L}^{\text{root}}(x),L)$. This completes the proof of part (i). □

The following is an $O(4^k n^{O(1)})$ algorithm. Its complexity can be obtained similarly to [88], and was briefly sketched on pages 117-118. We restrict ourselves only to proving its correctness.

```
For every vertex $x \in V(D)$, do $\mathcal{A}(D,\{x\},\emptyset)$.
If one of the returns of $\mathcal{A}(D,\{x\},\emptyset)$ is "TRUE" then output "TRUE".
Otherwise, output "FALSE".


$\mathcal{A}(D,T,L)$:
(1) If $\ell_{\max}(D,T,L) = 0$, return "FALSE".
(2) While there is a vertex $x \in \text{Leaf}(T) - L$ such that $\ell_{\max}(D,T,L\cup\{x\}) = 0$, add
      the arcs $A^+_{\hat{D}}(x)$ to $T$.
(3) If $|L| \geq k$, return "TRUE".
      If the number of leaves in $T$ is at least $k$, return "TRUE".
      If all leaves in $T$ belong to $L$, return "FALSE".
(4) Choose a vertex $x \in \text{Leaf}(T) - L$.
      $B_1 := \mathcal{A}(D,T,L\cup\{x\})$ and $B_2 :=$"FALSE".
      If $|\text{Leaf}(T_{D,L}^{\text{root}}(x))| \geq 2$ then let $B_2 := \mathcal{A}(D,T\cup T_{D,L}^{\text{root}}(x),L)$.
      Return "TRUE" if either $B_1$ or $B_2$ is "TRUE". Otherwise return "FALSE".
```

Remark 173. *While the first line in Step 3 is unnecessary, we keep it since it is needed in the next algorithm where* $L \subseteq$ *Leaf*(T) *is not necessarily true, see (4.2) in the next algorithm* $\mathcal{B}(D,T,L)$, *where* $p_0 \notin V(T)$.

Theorem 174. *Algorithm* $\mathcal{A}(D,T,L)$ *works correctly. In other words,* D *has a* (T,L)*-out-branching with at least* k *leaves if and only if Algorithm* $\mathcal{A}(D,T,L)$ *returns "TRUE".*

Proof. Let us first show that in a call to $\mathcal{A}(D,T,L)$ the argument (D,T,L) is *proper, i.e.* T is an out-tree of D and $L \cap \text{Int}(T) = \emptyset$. Obviously the initial argument $(D,\{x\},\emptyset)$ is proper. Assuming that (D,T,L) is a proper argument, then clearly $(D,T,L\cup\{x\})$ is proper, so let us consider $(D,T\cup T^{\text{root}}_{D,L}(x),L)$. By the definition of $T^{\text{root}}_{D,L}(x)$, $T\cup T^{\text{root}}_{D,L}(x)$ is an out-tree in D. Since we consider the digraph $\hat{D}$ at each step in the definition of $T^{\text{root}}_{D,L}(x)$, no vertex in L is an internal vertex of $T\cup T^{\text{root}}_{D,L}(x)$. Hence $(D,T\cup T^{\text{root}}_{D,L}(x),L)$ is a proper argument.

Consider the search tree ST that we obtain by running the algorithm $\mathcal{A}(D,T,L)$. First consider the case when ST consists of a single node. If $\mathcal{A}(D,T,L)$ returns "NO" in Step 1, then clearly we do not have a (T,L)-out-branching. Step 2 is valid by Lemma 171, *i.e.* it does not change the return of $\mathcal{A}(D,T,L)$. Let us now consider Step 3. As $\ell_{\max}(D,T,L)\geq 1$ after Step 1, and by Lemma 171 the value of $\ell_{\max}(D,T,L)$ does not change with Step 2, we note that $\ell_{\max}(D,T,L)\geq 1$ before we perform Step 3. Therefore there exists a (T,L)-out-branching in D. If $|L|\geq k$ or $|\text{Leaf}(T)|\geq k$ then, by Observation 42, a (T,L)-out-branching in D has at least k leaves and the algorithm returns "TRUE". If $\text{Leaf}(T)\subseteq L$ then the only (T,L)-out-branching in D is T itself and as $|\text{Leaf}(T)|<k$ the algorithm correctly returns "FALSE". Thus, the Algorithm $\mathcal{A}(D,T,L)$ is correct when the search tree ST is just a node.

Assume now that ST has at least two nodes and the theorem holds for all successors of the root R of ST. As R makes further recursive calls, we have $\ell_{\max}(D,T,L)\geq 1$ and there exists a vertex $x\in \text{Leaf}(T)-L$. If there is a (T,L)-out-branching with at least k leaves, then by Lemma 172 there is a $(T,L\cup\{x\})$-out-branching with at least k leaves or $(T\cup T^{\text{root}}_{D,L}(x),L)$-out-branching with at least k leaves. By induction hypothesis, one of B_1 or B_2 is "TRUE" and thus $\mathcal{A}(D,T,L)$ correctly returns "TRUE". Else if $\ell_{\max}(D,T,L)<k$, then again by Lemma 172 and induction hypothesis both B_1 and B_2 are "FALSE". Therefore the theorem holds for the root R of ST, which completes the proof. □

8.2 Faster Algorithm

We make the algorithm from the previous section faster by adding an extra vertex to the set L in certain circumstances. Recall that Step 2 in the above algorithm $\mathcal{A}(D,T,L)$ and in our new algorithm $\mathcal{B}(D,T,L)$ is new compared to the algorithm in [88]. We will also allow L to contain vertices which are not leaves of the current out-tree T. The improved algorithm is now described.

For every vertex $x\in V(D)$, do $\mathcal{B}(D,\{x\},\emptyset)$.
If one of the returns of $\mathcal{B}(D,\{x\},\emptyset)$ is "TRUE" then output "TRUE".
Otherwise, output "FALSE".

$\mathcal{B}(D,T,L)$:

(1) If $\ell_{\max}(D,T,L)=0$, return "FALSE".

(2) While there is a vertex $x \in \text{Leaf}(T) - L$ such that $\ell_{\max}(D, T, L \cup \{x\}) = 0$, add the arcs $A^+_{\hat{D}}(x)$ to T.

(3) If $|L| \geq k$, return "TRUE".
If the number of leaves in T is at least k, return "TRUE".
If all leaves in T belong to L, return "FALSE".

(4) Choose a vertex $x \in \text{Leaf}(T) - L$, colour x red and let $H_x := \hat{D}$.

(4.1) Let z be the nearest ancestor of x in T coloured red, if it exists.

(4.2) Let $L' := L \cup \{x\}$.
If z exists and T_z has exactly two leaves x and x' and $x' \in L$ then:
Let $P = p_0 p_1 \dots p_r$ be a path in $H_z - A^+_{\hat{D}}(z)$ such that $V(P) - V(T_z) = \{p_0\}$ and $p_r \in N^+_{\hat{D}}(z)$, and let $L' := L \cup \{p_0, x\}$.

(4.3) $B_1 := \mathcal{B}(D, T, L')$ and $B_2 :=$"FALSE".

(4.4) If $|\text{Leaf}(T^{\text{root}}_{D,L}(x))| \geq 2$ then let $B_2 := \mathcal{B}(D, T \cup T^{\text{root}}_{D,L}(x), L)$.

(4.5) Return "TRUE" if either B_1 or B_2 is "TRUE". Otherwise return "FALSE".

The existence of P in Step (4.2) follows from the fact that z was coloured red, hence adding z to L would not have destroyed all out-branchings. Note that p_0 does not necessarily belong to T.

For the sake of simplifying the proof of Theorem 176 below we furthermore assume that the above algorithm picks the vertex x in Step 4 in a depth-first manner. That is, the vertex x is chosen to be the last vertex added to T such that $x \in \text{Leaf}(T) - L$.

Theorem 175. *Algorithm $\mathcal{B}(D, T, L)$ works correctly. In other words, D has a (T, L)-out-branching with at least k leaves if and only if Algorithm $\mathcal{B}(D, T, L)$ returns "TRUE".*

Proof. The only difference between $\mathcal{B}(D, T, L)$ and $\mathcal{A}(D, T, L)$ is that in Step (4.2) we may add an extra vertex p_0 to L which was not done in $\mathcal{A}(D, T, L)$. We will now prove that this addition does not change the correctness of the algorithm.

Assume that there is an optimal (T, L)-out-branching T' with $x \in \text{Leaf}(T')$ but $p_0 \notin \text{Leaf}(T')$. Let us show that an optimal solution is found in the branch of the search tree where we put z into L. This will complete the proof as if an optimal (T, L)-out-branching T' does not contain x as a leaf, then by Lemma 172 it is found in $\mathcal{B}(D, T \cup T^{\text{root}}_{D,L}(x), L)$, and if it includes both x and p_0 as leaves then it is found in $B(D, T, L')$ in Step (4.3).

Note that $T_z = T'_z$ as T_z had exactly two leaves x and x' and $x' \in L$ and we have just assumed that x is a leaf of T'. Let $D^* = D[V(T'_z) \cup \{p_0\} - \{z\}]$ and consider the following two cases.

Assume first that p_0 can reach all vertices of D^* in D^*. Let T^* be an out-branching in D^* with p_0 as the root. Let T'' be the out-branching obtained from T' by deleting all arcs in T'_z and adding all arcs in T^*. Note that $|\text{Leaf}(T'')| \geq |\text{Leaf}(T')|$ as $\text{Leaf}(T^*) \cup \{z\}$ are leaves

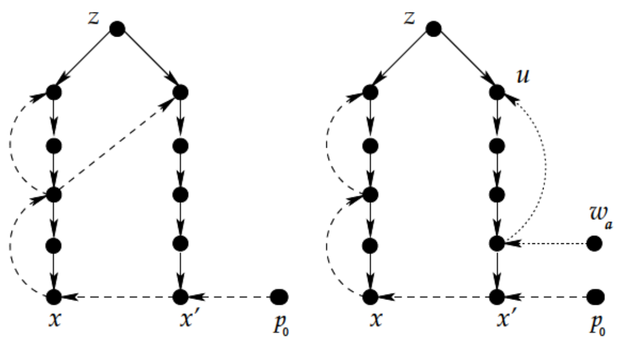


Figure 8.1: Real lines represents T'_z arcs; dashed lines represent the reachability of p_0; dotted lines represent the reachability of w_a.

in T'' and Leaf(T'_z) are the only two leaves in T' which may not be leaves in T''. Therefore an optimal solution is found when we add z to L.

Assume now that p_0 cannot reach all vertices of D^* in D^*. There exists a vertex $u \in N^+_T(z)$ which cannot be reached by p_0 in D^*. All such unreachable vertices lie on the same branch of T_z (the branch not containing p_r). Let $W = w_0w_1w_2\dots w_lu$ be a path from the root of T to u, which does not use any arcs out of z. Such a path exists as z was coloured red in Step (4.1), so adding z to L at this stage would not destroy all out-branchings. Let a be such that $w_a \not\in T'_z$ and $\{w_{a+1}, w_{a+2}, \dots, w_l, u\} \subseteq V(T'_z)$ (see Figure 8.1).

Consider the digraph $D'' = D[V(T'_z) \cup \{p_0, w_a\} - \{z\}]$. Every vertex in D'' can be reached by either p_0 or w_a in D''. Therefore, there exists two vertex disjoint out-trees T_{p_0} and T_{w_a} rooted at p_0 and w_a, respectively, such that $V(T_{p_0}) \cup V(T_{w_a}) = V(D'')$. Furthermore, since p_0 cannot reach u in D^*, both T_{p_0} and T_{w_a} must contain at least two vertices. Let T''' be the out-branching obtained from T' by deleting all arcs in T'_z and adding all arcs in T_{p_0} and in T_{w_a}. We have that $|\text{Leaf}(T''')| \geq |\text{Leaf}(T')|$ as vertices in $\text{Leaf}(T_{p_0}) \cup \text{Leaf}(T_{w_a}) \cup \{z\}$ are leaves in T''' and vertices x, x' and w_a are the only three vertices which may be leaves in T' but not in T'''. Once again, an optimal solution is found when we add z to L. □

Theorem 176. *Algorithm* $\mathcal{B}(D,T,L)$ *runs in time* $O(3.72^k n^{O(1)})$.

Proof. For an out-tree Q, let $\ell(Q) = |Leaf(Q)|$. Recall that we have assumed that $\mathcal{B}(D,T,L)$ picks the vertex x in Step 4 in a depth-first manner.

Consider the search tree ST obtained by running the algorithm $\mathcal{B}(D,\{x\},\emptyset)$. The root of ST is the triple $(D,\{x\},\emptyset)$. The children of this root is $(D,\{x\},L')$ when we make a recursive call in Step (4.3) and $(D, T^{\text{root}}_{D,L}(x), \emptyset)$ if we make a recursive call in Step (4.4). The children of these nodes are again triples corresponding to the recursive calls.

Let $g(T,L)$ be the number of leaves in a subtree R of ST with triple (D,T,L). Clearly, $g(T,L) = 1$ when (D,T,L) is a leaf of ST. For a non-trivial subtree R of ST, we will prove by induction that $g(T,L) \le c\alpha^{k-\ell(T)}\beta^{k-|L|}$, where $\alpha = 1.96$, $\beta = 1.896$ and $c \ge \alpha^2\beta^2$. Assume that this holds for all smaller non-trivial subtrees. Note that the value of c is chosen in such a way that in the inequalities in the rest of the proof, we have upper bounds for $g(T^*,L^*)$ being at least 1 when (D,T^*,L^*) is a leaf of ST.

Recall that $x \in \mathrm{Leaf}(T) - L$ was picked in Step (4). Now consider the following possibilities.

Assume that $|L'| = |L|+2$. If a call is made to $\mathcal{B}(D, T \cup T_{D,L}^{\mathrm{root}}(x), L)$ in (4.4) then the number of leaves of T increases by at least one, so the number of leaves of R is at most the following:

$$\begin{aligned} g(T,L') + g(T \cup T_{D,L}^{\mathrm{root}}(x), L) &\le c\alpha^{k-\ell(T)}\beta^{k-|L|-2} + c\alpha^{k-\ell(T)-1}\beta^{k-|L|} \\ &= c\alpha^{k-\ell(T)}\beta^{k-|L|}(\tfrac{1}{\beta^2} + \tfrac{1}{\alpha}) \\ &\le c\alpha^{k-\ell(T)}\beta^{k-|L|}. \end{aligned}$$

So we may assume that $|L'| = |L|+1$ in (4.3). Assume now that $|\mathrm{Leaf}(T_{D,L}^{\mathrm{root}}(x))| \ne 2$ in (4.4). In this case, either no recursive call is made in (4.4) or we increase the number of leaves in T by at least two. Therefore, the number of leaves of R is at most:

$$\begin{aligned} c\alpha^{k-\ell(T)}\beta^{k-|L|-1} + c\alpha^{k-\ell(T)-2}\beta^{k-|L|} &= c\alpha^{k-\ell(T)}\beta^{k-|L|}(\tfrac{1}{\beta} + \tfrac{1}{\alpha^2}) \\ &\le c\alpha^{k-\ell(T)}\beta^{k-|L|}. \end{aligned}$$

So we may assume that $|L'| = |L|+1$ in (4.3) and $|\mathrm{Leaf}(T_{D,L}^{\mathrm{root}}(x))| = 2$ in (4.4). Let $T' = T \cup T_{D,L}^{\mathrm{root}}(x)$ and consider the recursive call to $\mathcal{B}(D,T',L)$. If we increase the number of leaves in T' in Step (2) of this recursive call, then the number of leaves of the subtree of ST rooted at (D,T',L) is at most:

$$c\alpha^{k-\ell(T')-1}\beta^{k-|L|-1} + c\alpha^{k-\ell(T')-2}\beta^{k-|L|} = c\alpha^{k-\ell(T')}\beta^{k-|L|}(\tfrac{1}{\alpha\beta} + \tfrac{1}{\alpha^2}).$$

Therefore, as $\ell(T') = \ell(T)+1$, the number of leaves in R is at most:

$$\begin{aligned} g(T,L') + g(T',L) &\le c\alpha^{k-\ell(T)}\beta^{k-|L|-1} + c\alpha^{k-\ell(T)-1}\beta^{k-|L|}(\tfrac{1}{\alpha\beta} + \tfrac{1}{\alpha^2}) \\ &= c\alpha^{k-\ell(T)}\beta^{k-|L|}(\tfrac{1}{\beta} + \tfrac{1}{\alpha^2\beta} + \tfrac{1}{\alpha^3}) \\ &\le c\alpha^{k-\ell(T)}\beta^{k-|L|}. \end{aligned}$$

So we may assume that we do not increase the number of leaves in Step (2) when we consider (D,T',L). Let y and y' denote the two leaves of T'_x (after possibly adding some arcs in Step (2)). Consider the recursive call to $\mathcal{B}(D,T',L\cup\{y\})$. If we increase the number of leaves of T' in Step (2) in this call, then the number of leaves in R is at most

$$\begin{aligned} g(T,L\cup\{x\}) \quad + \quad & g(T',L\cup\{y\}) \ + \ g(T' \cup (T')_{D,L}^{\mathrm{root}}(y), L) \\ &\le c\alpha^{k-\ell(T)}\beta^{k-|L|}(\tfrac{1}{\beta} + (\tfrac{1}{\alpha^2\beta^2} + \tfrac{1}{\alpha^3\beta}) + \tfrac{1}{\alpha^2}) \\ &\le c\alpha^{k-\ell(T)}\beta^{k-|L|}. \end{aligned}$$

So we may assume that we do not increase the number of leaves in Step (2) when we consider $(D,T',L\cup\{y\})$. However, in this case $|L'| = |L|+2$ in this recursive call. Indeed, when we consider y' the conditions of (4.2) are satisfied as, in particular, T_x has exactly two leaves. So, in this last case, the number of leaves in R is at most

$$\begin{aligned} g(T, L\cup\{x\}) &+ g(T', L\cup\{y\}) + g(T'\cup(T')^{root}_{D,L}(y), L) \\ &\leq c\alpha^{k-\ell(T)}\beta^{k-|L|}(\tfrac{1}{\beta}+(\tfrac{1}{\alpha\beta^3}+\tfrac{1}{\alpha^2\beta})+\tfrac{1}{\alpha^2}) \\ &\leq c\alpha^{k-\ell(T)}\beta^{k-|L|}. \end{aligned}$$

We increase either $|L|$ or $\ell(T)$ whenever we consider a child in the search tree and no non-leaf in ST has $|L| \geq k$ or $\ell(T) \geq k$. Therefore, the number of nodes in ST is at most $O(k\alpha^k\beta^k) = O(3.72^k)$. The amount of work we do in each recursive call is polynomial, which concludes the proof of Theorem 176. □

It would be interesting to see whether DIRECTED k-LEAF admits an algorithm of significantly smaller running time, say $O(3^k n^{O(1)})$.

8.3 Exponential Algorithm for DIRECTED MAXIMUM LEAF

The unparameterized DIRECTED MAXIMUM LEAF problem can be solved in time $O^*(2^n)$ by an exhaustive search using Observation 47. The same complexity would be reached by using a $O^*(4^k)$ parameterized algorithm to check the existence of an out-branching with at least k leaves for $k = 1, \ldots, \frac{n}{2}$, and using the brute-force algorithm for $k = \frac{n}{2}, \ldots, n$. Our improvement over the $O^*(4^k)$ algorithm breaks the bottleneck in the first case (the values of k close to $\frac{n}{2}$) and allows to shift the threshold for the algorithm swap away from $\frac{n}{2}$, which gets rid of the bottleneck in the second case as well.

More formally, let $a = 0.526$. The value an is our new threshold for the algorithm swap. We can solve DIRECTED MAXIMUM LEAF for a digraph D on n vertices with the following algorithm ADML:

Stage 1. Set $k := \lceil an \rceil$. For each $x \in V(D)$, apply $\mathcal{B}(D, \{x\}, \emptyset)$ to decide whether D contains an out-branching with at least k leaves. If D contains such an out-branching, go to Stage 2. Otherwise, using binary search and $\mathcal{B}(D, \{x\}, \emptyset)$, return the maximum integer ℓ for which D contains an out-branching with ℓ leaves.

Stage 2. Set $\ell := \lceil an \rceil$. For $k = \ell+1, \ell+2, \ldots, n$, using Observation 47, decide whether there exists a vertex set S of D of cardinality k such that $\hat{D}(\emptyset, S)$ has an out-branching, and when the answer is "FALSE", return $k-1$.

The correctness of Algorithm ADML is obvious. Let us now evaluate its time complexity. Let $r = \lceil an \rceil$. Since $3.72^a < 1.996$, Stage 1 takes time at most $3.72^r n^{O(1)} = O(1.996^n)$. Since $\frac{1}{a^a(1-a)^{1-a}} < 1.9973$, Stage 2 takes time at most

$$\binom{n}{r} \cdot n^{O(1)} = \left(\frac{1}{a^a(1-a)^{1-a}}\right)^n n^{O(1)} = O(1.9973^n).$$

Thus, we obtain the following:

Theorem 177. *There is an algorithm to solve* Directed Maximum Leaf *in time* $O(1.9973^n)$.

The same also holds for the undirected MaxLeaf problem, which has been improved by Raible and Fernau. In [108] they obtain a parameterized algorithm for MaxLeaf running in time $O^*(3.4581^k)$, which gives an exponential algorithm running in time $O^*(1.8961^n)$ for finding the spanning trees with the maximum number of leaves in undirected graphs.

8.4 A Randomized Algorithm for Finding Trees with many Internal Nodes

Koutis and Williams [89] have developed a randomized technique to find a multilinear monomial of degree k in a polynomial presented as a circuit, in randomized time $O^*(2^k)$. Several problems can be expressed in this paradigm, such as finding a path with k vertices, or a tree on k vertices, as a subgraph. More formally:

Lemma 178 (Lemma 1 of [89])**.** *Let* $P(X,z)$ *be a polynomial represented by a commutative arithmetic circuit C. The existence of a term of the form* $z^tQ(X)$ *in* $P(X,z)$, *where* $Q(X)$ *is a multilinear monomial of degree at most* k, *can be decided in time* $O^*(2^kt^2|C|)$ *and space* $O(t|C|)$.

Originally, [89] claimed that this statement also directly applies to the Maximum Leaf Spanning Tree problem, but the construction is flawed, and does not appear to be easily patchable.

This technique fits well with problems where we search for a structure of bounded size. In MaxLeaf, we have to find a structure with k leaves but an arbitrary number of internal vertices, which we could not express in this framework. In the rest of this section, we describe how to apply this randomized monomial method to the dual MinLeaf problem (the construction will also work for the directed version).

Minimum Leaf Spanning Tree:

Input: A graph G, an integer k.

Parameter: k.

Output: TRUE if there is a spanning tree of G with at least k internal nodes, otherwise FALSE.

Minimum Leaf Out-Branching and Rooted Minimum Leaf Out-Branching are defined similarly for directed graphs and rooted directed graphs respectively. As in the case of MaxLeaf, the dual parameterization by $n-k$ gives a $W[1]$-hard problem. Indeed, the case $n-k=2$ is the NP-hard problem Hamiltonian Path. In other words, finding a spanning tree with at most k leaves is not tractable when parameterized by k.

To explicitely express MINIMUM LEAF SPANNING TREE in terms of the search for a bounded size subgraph, let us reformulate as follows:

Lemma 179. *A graph G contains a spanning tree with at least k internal nodes if and only if G contains a tree with exactly k internal vertices and at most k leaves.*

Lemma 179 also holds for digraphs and rooted digraphs. Hence we can restrict ourselves to searching for a structure of bounded size (at most $2k$ in all), and this makes the problem tractable with the randomized monomial method. Note that, contrary to the maximum leaf problems, a vertex can never become unreachable due to a choice in the construction of a partial solution. When deciding that a vertex will be a leaf in the final solution for MAXIMUM LEAF SPANNING TREE, one might make another vertex unreachable from the root, but deciding that a vertex will be internal in minimum leaf problems does not rule out any edge or arc.

The following result has been obtained with EunJung Kim:

Theorem 180. *There exists a randomized $O^*(4^k)$ algorithm to decide* MINIMUM LEAF SPANNING TREE, MINIMUM LEAF OUT-BRANCHING *and* ROOTED MINIMUM LEAF OUT-BRANCHING.

The best non-randomized algorithms known for MINIMUM LEAF SPANNING TREE and MINIMUM LEAF OUT-BRANCHING run in time $O^*(8^k)$ [63] and $O^*(16^{k+o(k)})$ [65] respectively. [63] actually gives a linear $3k$ kernel for MINIMUM LEAF SPANNING TREE. The rest of this section is devoted to the proof of Theorem 180.

We define a u-rooted tree to be a tree containing u where u is counted as an internal vertex even if u a leaf.

Given a graph $G = (V, E)$, and an integer $k \geq 2$ we define a polynomial $P(G, k)$:

$$P(G,k) = \sum_{t=k+2}^{2k} \sum_{(u,v)\in E} \sum_{t'=2}^{t-2} \sum_{k'=1}^{k-1} x_u x_v C_{k',t',u} C_{k-k',t-t',v}$$

We want $C_{k,t,u}$ to contain the monomial $x_1 \dots x_t$ if and only if $u, x_1, \dots, x_t$ is a u-rooted tree with k internal nodes. Informally, the above definition of $P(G)$ enforces that in the original calls to $C_{k',t',u}$ and $C_{k-k',t-t',v}$, u and v will indeed be internal nodes in the global tree, hence it is safe to consider u-rooted and v-rooted trees. Indeed, u (resp. v) must have a neighbour besides v (resp. u) as $t' \geq 2$ and $t \geq 2$ in the above formula.

We define recursively the polynomials $C_{k,t,u}$:

$$C_{k,t,u} = \sum_{(u,v)\in E} x_v C_{k,t-1,u} + \sum_{(u,v)\in E} \sum_{t'=2}^{t-1} \sum_{k'=1}^{k-1} x_v C_{k',t',v} C_{k-k',t-t',u}$$

$C_{1,1,u} = 1$
$C_{k,t,u} = 0$ when $k \geq t$ and $k \geq 2$

Lemma 181. $C_{k,t,u}$ *contains only terms of degree* $t-1$ *when* $C_{k,t,u} \neq 0$.

Proof. We prove this by induction. The hypothesis clearly holds for the base cases, and also holds in the induction step as well: the term $x_v C_{k,t-1,u}$ contains only terms of degree $(t-1)-1+1 = t-1$ and in the term $x_v C_{k',t',v} C_{k-k',t-t',u}$, the degree is $(t'-1)+(t-t'-1)+1 = t-1$. □

Lemma 182. $C_{k,t+1,u}$ *contains the monomial* $x_1 \dots x_t$ *if and only if* $u, x_1, \dots, x_t$ *is a* u*-rooted tree with* k *internal nodes.*

Proof. We prove this by induction on $C_{k,t+1,u}$. The hypothesis holds for the base cases.

Assume that $C_{k,t,u}$ contains the monomial $x_1 \dots x_t$. This monomial either comes from a term $x_v C_{k,t-1,u}$ or from a term $x_v C_{k',t',v} C_{k-k',t-t',u}$ with $v \in N(u)$, say $v = x_1$. In the former case, $C_{k,t-1,u}$ contains the monomial $x_2 \dots x_t$, so by induction $u, x_2, \dots, x_t$ is a u-rooted tree with k internal nodes, hence $u, x_1, \dots, x_t$ is a u-rooted tree with k internal nodes since $v = x_1$ is adjacent to u. In the latter case, by Lemma 181, we can assume w.l.o.g that $C_{k',t',v}$ contains the monomial $x_2 \dots x_{t'}$ and that $C_{k-k',t-t',u}$ contains the monomial $x_{t'+1} \dots x_t$. By induction $v, x_2, \dots, x_{t'}$ is a v-rooted tree T_v with k' internal nodes and $u, x_{t'+1}, \dots, x_t$ is a u-rooted tree T_u with $k-k'$ internal nodes. T_u and T_v are disjoint as $x_1 \dots x_t$ is a monomial. Also, $T_u \cup T_v$ is a tree as $(u, v) \in E$. Vertex v is internal in $T_u \cup T_v$ as v has at least two neighbours: u and another neighbour in T_v. Indeed $t' \geq 2$, hence T_v is not reduced to its root v. Thus, $T_u \cup T_v$ is a u-rooted tree with $k' + (k-k') = k$ internal nodes.

Conversely, assume that $u, x_1, \dots, x_t$ is a u-rooted tree T with k internal nodes. Let v be a neighbour of u in T. If v is a leaf, then $T - v$ is a u-rooted tree on $t-1$ vertices and k internal vertices. Hence, by induction $C_{k,t-1,u}$ contains a monomial corresponding to $T - v$, so the term $\sum_{(u,v)\in E} x_v C_{k,t-1,u}$ contains the monomial $x_1 \dots x_t$. If v is internal in T, consider the subtree T_v of $T - u$ rooted at v and the subtree T_u of $T - v$ rooted at u. We have $T_u \cup T_v = T$. Denote by I the set of internal nodes of T. By induction, $C_{k',t',v}$ contains a monomial corresponding to the vertices of $T_v - v$, and $C_{k-k',t-t',u}$ contains a monomial corresponding to the vertices of $T_u - u$. Hence $x_v C_{k',t',v} C_{k-k',t-t',u}$ contains a monomial corresponding to the vertices of $T - u$. □

Lemma 183. *When* $k \geq 2$, $P(G,k)$ *contains a monomial of degree* $t \geq 1$ *if and only if* G *contains a tree with* t *vertices and* k *internal nodes.*

Proof. Note that by Lemma 181, $P(G,k)$ contains only terms of degree $2 + (t'-1) + (t-t'-1) = t$.

If G contains a tree T with t vertices and k internal nodes, let u and v be adjacent internal nodes of T. Denote by I the set of internal vertices of T, and by T_u (resp. T_v) the subtree of T containing u (resp. v) in $T-v$ (resp. $T-u$). Let $k' = |I \cap T_v|$ and $t' = |V(T_v)|$. The polynomial $P(G,k)$ contains the term $x_u x_v C_{k',t',v} C_{k-k',t-t',u}$. By Lemma 182 and as T_u and T_v are disjoint, the term $C_{k',t',v} C_{k-k',t-t',u}$ contains a monomial, which is the product of the variables x_w for $w \in V(T) - \{u,v\}$. Hence $P(G,k)$ contains a monomial of degree t, namely $\prod_{w \in V(T)} x_w$.

Conversely, if $P(G,k)$ contains a monomial M of degree t, let (u,v) be the corresponding edge in the $P(G,k)$ construction. Let M_u and M_v be the monomials associated to $C_{k',t',v}$ and $C_{k-k',t-t',u}$. With a slight abuse of language we identify a monomial M with the set of vertices associated to the variables appearing in M. We have that $M = M_u \cup M_v \cup \{u,v\}$. By Lemma 182, let T_u be a tree associated to $M_u \cup u$ and let T_v be the tree associated to $M_v \cup v$. The trees T_u and T_v are disjoint, and $T_u \cup T_v$ is a tree in which both u and v are internal since $t \geq 2$ and $t' \geq 2$. Hence T is a tree on t vertices with $k' + (k - k') = k$ internal nodes. □

By Lemma 183 we only have to test whether $P(G)$ contains a monomial of degree t with $k+2 \leq t \leq 2k$ to decide if G contains a tree with at least k internal nodes. This can be done in randomized time $O^*(4^k)$ by Lemma 178.

The same construction works for directed graphs. It is even somewhat simpler, since we do not need to be careful with the internality of the root. The polynomial $P(D,k)$ for a digraph D becomes:

$$P(D,k) = \sum_{t=k+2}^{2k} \sum_{u \in V(D)} x_u C_{k,t,u}$$

The other definitions do not change; $(u,v) \in E$ becomes $(u,v) \in A$. The same construction also holds for rooted directed graphs: $P(D,k) = \sum_{t=k+2}^{2k} x_u C_{k,t,u}$ where u is the root of D.

Concluding Remarks

Results

Throughout this thesis, we have stated the following results in the area of parameterized complexity:

- MULTICUT parameterized by the solution size k is FPT. The running time is single exponential in k.
- MAXIMUM LEAF OUT-BRANCHING has an $O^*(3.72^k)$ algorithm.
- MAXIMUM LEAF OUT-BRANCHING has an $O(1.9973^n)$ algorithm, combining the parameterized algorithm with brute force search.
- ROOTED MAXIMUM LEAF OUT-BRANCHING has an edge-quadratic kernel.
- ROOTED MAXIMUM LEAF OUT-BRANCHING has a constant factor approximation algorithm.
- MULTICUT IN TREES has a poynomial $O(k^6)$ kernel.

Open Problems

The following questions are left open in this thesis and seem worth investigating:

- To obtain a polynomial kernel for MULTICUT IN TREES, can the Wingspan rule be replaced by a simpler and more natural reduction rule?
- Is it possible to significantly improve the kernel for MULTICUT IN TREES, to a cubic kernel for instance?
- The proof that Vertex-Multicut is FPT mentioned in Section 7.7 is very sketchy. Is there a simple proof of the fact that the Fixed-Parameter Tractability of Edge-Multicut implies that of Vertex-Multicut?

- Is it always possible to emulate a request with a bounded (in k) number of paths, as formulated in Question 167?
- Is there a linear kernel for ROOTED MAXIMUM LEAF OUT-BRANCHING? For a positive answer, focusing on 2-cycles seems logical. For a negative answer, the lower bound technique developped in [45] leading to results such as Theorem 5 seems like a good starting point.
- Is it possible to significantly improve over the factor 92 in a constant approximation algorithm for MAXIMUM LEAF OUT-BRANCHING, say to a single-digit factor?
- Is it possible to significantly improve over the $O^*(3.72^k)$ algorithm for MAXIMUM LEAF OUT-BRANCHING, to an $O^*(3^k)$ algorithm for example?
- Can a more efficient exponential algorithm for MAXIMUM LEAF OUT-BRANCHING be designed without the help of a parameterized algorithm? Alternatively, can the parameterized / brute force paradigm be improved with a parameterized / cleverer algorithm approach?
- Can the randomized monomial detection technique be used for finding trees with many leaves?

Bibliography

[1] F. Abu-Khzam. Kernelization algorithms for d-hitting set problems. In *Algorithms and Data Structures*, volume 4619 of *Lecture Notes in Computer Science*, pages 434–445. 2007. Cited page 25.

[2] N. Alon. Transversal numbers of uniform hypergraphs. *Graphs and Combinatorics*, 6:1–4, 1990. Cited page 29.

[3] N. Alon, F. Fomin, G. Gutin, M. Krivelevich, and S. Saurabh. Parameterized algorithms for directed maximum leaf problems. In *Proc. ICALP 2007, LNCS 4596*, pages 352–362, 2007. Cited page 30.

[4] N. Alon, F. Fomin, G. Gutin, M. Krivelevich, and S. Saurabh. Spanning directed trees with many leaves. *SIAM Journal on Discrete Mathematics*, 23(1):466–476, 2009. Cited page 30.

[5] N. Alon, R. Yuster, and U. Zwick. Color-coding: a new method for finding simple paths, cycles and other small subgraphs within large graphs. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, STOC '94, pages 326–335, 1994. Cited page 19.

[6] A. Asratian, T. Denley, and R. Häggkvist. *Bipartite Graphs and Their Applications*. Cambridge, 1998. Cited page 59.

[7] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer, 2nd edition, 2008. Cited page 59.

[8] J. Barrionuevo, A. Calvo, G. Durán, and F. Protti. New advances about a conjecture on Helly circle graphs. *Electronic Notes in Discrete Mathematics*, 18:31–36, 2004. Latin-American Conference on Combinatorics, Graphs and Applications. Cited page 110.

[9] H. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996. Cited pages 21 and 64.

[10] H. Bodlaender. Treewidth: Structure and algorithms. In *Structural Information and Communication Complexity*, volume 4474 of *Lecture Notes in Computer Science*, pages 11–25. Springer Berlin / Heidelberg, 2007. Cited page 63.

[11] H. Bodlaender, L. Cai, J. Chen, M. Fellows, J.A. Telle, and D. Marx. Open problems in parameterized and exact computation. In *Proceedings of the 2nd International Workshop on Parameterized and Exact Computation*, IWPEC 2006, 2006. Cited pages 6, 28, and 31.

[12] H. Bodlaender, R. Downey, M. Fellows, and D. Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009. Cited pages 23, 24, and 30.

[13] H. Bodlaender, M. Fellows, P. Heggernes, F. Mancini, C. Papadopoulos, and F. Rosamond. Clustering with partial information. In *Proceedings of the 33rd international symposium on Mathematical Foundations of Computer Science*, MFCS ’08, pages 144–155, 2008. Cited page 27.

[14] H. Bodlaender, F. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. Thilikos. (Meta) kernelization. In *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS ’09, pages 629–638, 2009. Cited pages 24 and 25.

[15] H. Bodlaender, B. Jansen, and S. Kratsch. Cross-composition: A new technique for kernelization lower bounds. In *STACS*, pages 165–176, 2011. Cited page 24.

[16] H. Bodlaender, S. Thomassé, and A. Yeo. Kernel bounds for disjoint cycles and disjoint paths. In *ESA’09*, pages 635–646, 2009. Cited page 24.

[17] P. Bonsma, T. Brueggemann, and G. Woeginger. A faster FPT algorithm for finding spanning trees with many leaves. In *Mathematical Foundations of Computer Science 2003*, volume 2747 of *Lecture Notes in Computer Science*, pages 259–268. 2003. Cited page 29.

[18] P. Bonsma and F. Dorn. An FPT algorithm for directed spanning k-leaf. *Manuscript*. Cited page 30.

[19] P. Bonsmal and F. Zickfeld. A 3/2-approximation algorithm for finding spanning trees with many leaves in cubic graphs. In *Graph-Theoretic Concepts in Computer Science*, pages 66–77. Springer-Verlag, 2008. Cited page 29.

[20] A. Bouchet. Reducing prime graphs and recognizing circle graphs. *Combinatorica*, 7:243–254, 1987. Cited page 115.

[21] A. Bouchet. Circle graph obstructions. *Journal of Combinatorial Theory, Series B*, 60:107–144, 1994. Cited page 109.

[22] N. Bousquet, J. Daligault, S. Thomassé, and A. Yeo. A polynomial kernel for multicut in trees. In *STACS*, pages 183–194, 2009. Cited pages 28 and 31.

[23] L. Brunetta, M. Conforti, and M. Fischetti. A polyhedral approach to an integer multicommodity flow problem. *Discrete Applied Mathematics*, 101(1-3):13 – 36, 2000. Cited page 26.

[24] J. Buss and J. Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, 22:560–572, 1993. Cited page 24.

[25] S. Chawla, R. Krauthgamer, R. Kumar, Y. Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Computational Complexity*, 15(2):94–114, 2006. Cited page 26.

[26] J. Chen, Y. Kanj, and G. Xia. Improved parameterized upper bounds for vertex cover. In *Mathematical Foundations of Computer Science 2006*, volume 4162 of *Lecture Notes in Computer Science*, pages 238–249. 2006. Cited pages 19 and 25.

[27] J. Chen and Y. Liu. On the parameterized max-leaf problems: digraphs and undirected graphs. Technical report, Department of Computer Science, Texas A&M University, 2008. Cited page 30.

[28] J. Chen, Y. Liu, and S. Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009. Cited pages 26 and 126.

[29] J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 177–186, 2008. Cited pages 21 and 124.

[30] Y. Chen., J. Flum, and M. Müller. Lower bounds for kernelizations and other preprocessing procedures. In *Mathematical Theory and Computational Practice*, volume 5635 of *Lecture Notes in Computer Science*, pages 118–128. 2009. Cited pages 21 and 23.

[31] J. Cheriyan and J. Reif. Directed s-t numberings, rubber bands, and testing digraph k-vertex connectivity. *Combinatorica*, 14(4):435–451, 1994. Cited page 48.

[32] M.-C. Costa, L. Létocart, and F. Roupin. Minimal multicut and maximal integer multiflow: A survey. *European Journal of Operational Research*, 162(1):55–69, 2005. Cited page 26.

[33] B. Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990. Cited pages 20 and 64.

[34] B. Courcelle. Circle graphs and monadic second-order logic. *Journal of Applied Logic*, 6(3):416–442, 2008. Cited page 109.

[35] B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46:218–270, 1993. Cited page 95.

[36] W. Cunningham. Decomposition of directed graphs. *SIAM Journal on Algebraic and Discrete Methods*, 3(2):214–228, 1982. Cited page 115.

[37] J. Daligault, N. Bousquet, and S. Thomassé. Multicut is FPT. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 459–468, 2011. Cited page 117.

[38] J. Daligault, D. Gonçalves, and M. Rao. Diamond-free circle graphs are Helly circle. *Discrete Mathematics*, 310(4):845–849, 2010. Cited page 67.

[39] J. Daligault, G. Gutin, E.J. Kim, and A. Yeo. FPT algorithms and kernels for the directed k-leaf problem. *Journal of Computer and System Sciences*, 76(2):144–152, 2010. Cited pages 32, 117, and 157.

[40] J. Daligault, C. Paul, A. Perez, and S. Thomassé. Reducing multicut to bounded treewidth. *http://www.lirmm.fr/~daligault/MulticutTreewidthReduction.pdf*, 2010. Cited page 117.

[41] J. Daligault, M. Rao, and S. Thomassé. Well-quasi-order of relabel functions. *Order*, 27:301–315, 2010. Cited page 67.

[42] J. Daligault and S. Thomassé. On finding directed trees with many leaves. In *Parameterized and Exact Computation*, volume 5917 of *Lecture Notes in Computer Science*, pages 86–97. 2009. Cited page 32.

[43] P. Damaschke. Induced subgraphs and well-quasi-ordering. *Journal of Graph Theory*, 14:427–435, 1990. Cited page 95.

[44] H. de Fraysseix. A characterization of circle graphs. *European Journal of Combinatorics*, 5(3):223–238, 1984. Cited page 109.

[45] H. Dell and D. van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 251–260, 2010. Cited pages 8, 24, and 172.

[46] E. Demaine, M. Hajiaghayi, and D. Marx. Open problems – parameterized complexity and approximation algorithms. Number 09511 in Dagstuhl Seminar Proceedings, 2010. Cited pages 26 and 90.

[47] E. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974. Cited page 29.

[48] G. Ding, T. Johnson, and P. Seymour. Spanning trees with many leaves. *Journal of Graph Theory*, 37(4):189–197, 2001. Cited page 29.

[49] R. Downey and M. Fellows. *Parameterized Complexity*. Springer, 1999. Cited pages 19 and 25.

[50] M. Drescher and A. Vetta. An approximation algorithm for the maximum leaf spanning arborescence problem. *ACM Transactions on Algorithms*, 6:46:1–46:18, 2010. Cited pages 7, 30, and 57.

[51] G. Duràn. *On Intersection Graphs of Arcs and Chords in a Circle*. PhD thesis, Universidad de Buenos Aires, Argentina, 2000. Cited pages 12, 67, and 110.

[52] G. Duràn, A. Gravano, M. Groshaus, F. Protti, and J. Szwarcfiter. On a conjecture concerning helly circle graphs. *Pesquisa Operacional*, 23:221 – 229, 2003. Cited page 110.

[53] M. Tedder D. Corneil E. Gioan, C. Paul. Practical split decomposition via graph-labelled trees. *Manuscript*, 2010. Cited page 115.

[54] M. Tedder D. Corneil E. Gioan, C. Paul. Quasi linear-time circle graph recognition. *Manuscript*, 2010. Cited pages 12, 110, and 116.

[55] P. Erdős and R. Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 35:85–90, 1960. Cited page 126.

[56] V. Estivill-Castro, M. Fellows, M. Langston, and F. Rosamond. Fixed-parameter tractability is polynomial-time extremal structure theory I: The case of max leaf. In *Proc. of ACiD 2005*. Cited page 29.

[57] V. Estivill-Castro, M. Fellows, M. Langston, and F. Rosamond. FPT is P-time extremal structure I. In *Algorithms and Complexity in Durham 2005, Proceedings of the first ACiD Workshop, volume 4 of Texts in Algorithmics*, pages 1–41. King's College Publications, 2005. Cited page 29.

[58] U. Feige, M. Hajiaghayi, and J. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38:629–657, 2008. Cited page 64.

[59] M. Fellows, J. Guo, H. Moser, and R. Niedermeier. A generalization of Nemhauser and Trotter's local optimization theorem. In *26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009)*, volume 3, pages 409–420, 2009. Cited page 24.

[60] M. Fellows, C. McCartin, F. Rosamond, and U. Stege. Coordinatized kernels and catalytic reductions: An improved FPT algorithm for max leaf spanning tree and other problems. In *FSTTCS*, pages 240–251, 2000. Cited page 29.

[61] H. Fernau, F. Fomin, D. Lokshtanov, D. Raible, S. Saurabh, and Y. Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. In *STACS*, pages 421–432, 2009. Cited pages 7, 24, and 30.

[62] J. Flum and M. Grohe. *Parameterized complexity theory*. Springer-Verlag, 2006. Cited pages 19 and 25.

[63] F. Fomin, S. Gaspers, S. Saurabh, and S. Thomassé. A linear vertex kernel for maximum internal spanning tree. In *Algorithms and Computation*, volume 5878 of *Lecture Notes in Computer Science*, pages 275–282. 2009. Cited page 167.

[64] F. Fomin, F. Grandoni, and D. Kratsch. Solving connected dominating set faster than 2^n. *Algorithmica*, 52(2):153–166, 2008. Cited page 29.

[65] F. Fomin, D. Lokshtanov, F. Grandoni, and S. Saurabh. Sharp separation and applications to exact and parameterized algorithms. In *LATIN*, pages 72–83, 2010. Cited page 167.

[66] F. Fomin, D. Lokshtanov, S. Saurabh, and D. Thilikos. Bidimensionality and kernels. In *SODA*, pages 503–510, 2010. Cited page 25.

[67] L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 133–142, 2008. Cited page 23.

[68] R. Fraïssé. *Theory of Relations*, volume 118 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1986. Cited page 94.

[69] C. Gabor, K. Supowit, and W.-L. Hsu. Recognizing circle graphs in polynomial time. *Journal of ACM*, 36:435–473, 1989. Cited page 115.

[70] G. Galbiati, F. Maffioli, and A. Morzenti. A short note on the approximability of the maximum leaves spanning tree problem. *Information Processing Letters*, 52(1):45–49, 1994. Cited page 29.

[71] N. Garg, V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. In *STOC*, pages 698–707, 1993. Cited page 26.

[72] N. Garg, V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997. Cited pages 26 and 36.

[73] J. Geelen and S. Oum. Circle graph obstructions under pivoting. *Journal of Graph Theory*, 61:1–11, 2009. Cited page 109.

[74] G. Gottlob and S. Tien Lee. A logical approach to multicut problems. *Information Processing Letters*, 103(4):136–141, 2007. Cited page 26.

[75] J. Griggs and M. Wu. Spanning trees in graphs of minimum degree 4 or 5. *Discrete Mathematics*, 104(2):167–183, 1992. Cited page 29.

[76] M. Grohe, K. Kawarabayashi, D. Marx, and P. Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 479–488. ACM, 2011. Cited page 21.

[77] S. Guillemot. FPT algorithms for path-transversals and cycle-transversals problems in graphs. In *IWPEC*, pages 129–140, 2008. Cited page 26.

[78] J. Guo. *Algorithm design techniques for parameterized graph modification problems.* PhD thesis, 2006. Cited pages 28 and 33.

[79] J. Guo, F. Hüffner, E. Kenar, R. Niedermeier, and J. Uhlmann. Complexity and exact algorithms for multicut. In *SOFSEM*, pages 303–312, 2006. Cited page 26.

[80] J. Guo and R. Niedermeier. Fixed-parameter tractability and data reduction for multicut in trees. *Networks*, 46(3):124–135, 2005. Cited pages 6, 26, 28, 31, 33, and 69.

[81] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38:31–45, 2007. Cited page 23.

[82] F. Gurski and E. Wanke. Minimizing NLC-width is NP-complete. In *Graph-Theoretic Concepts in Computer Science*, volume 3787 of *Lecture Notes in Computer Science*, pages 69–80. 2005. Cited page 95.

[83] G. Higman. Ordering by Divisibility in Abstract Algebras. *Proceedings of the London Mathematical Society*, s3-2(1):326–336, 1952. Cited pages 93, 98, and 99.

[84] O. Johansson. NLC2-decomposition in polynomial time. In *Graph-Theoretic Concepts in Computer Science*, volume 1665 of *Lecture Notes in Computer Science*, pages 110–121. 1999. Cited page 95.

[85] M. Kaminski, V. Lozin, and M. Milanic. Recent developments on graphs of bounded clique-width. *Discrete Applied Mathematics*, 157(12):2747–2761, 2009. Second Workshop on Graph Classes, Optimization, and Width Parameters. Cited page 107.

[86] P. Klein, A. Agrawal, R. Ravi, and S. Rao. Approximation through multicommodity flow. *Foundations of Computer Science*, 2:726–737, 1990. Cited page 27.

[87] D. Kleitman and D. West. Spanning trees with many leaves. *SIAM Journal on Discrete Mathematics*, 4:99–106, 1991. Cited page 29.

[88] J. Kneis, A. Langer, and P. Rossmanith. A new algorithm for finding trees with many leaves. In *Algorithms and Computation*, volume 5369 of *Lecture Notes in Computer Science*, pages 270–281. 2008. Cited pages 15, 30, 117, 158, 160, and 161.

[89] I. Koutis and R. Williams. Limits and applications of group algebras for parameterized problems. In *Automata, Languages and Programming*, volume 5555 of *Lecture Notes in Computer Science*, pages 653–664. 2009. Cited pages 15, 20, and 166.

[90] I. Kríẑ. Well-quasiordering finite trees with gap-condition. Proof of Harvey Friedman's conjecture. *Annals of Mathematics*, 130:215–226, 1989. Cited page 102.

[91] I. Kríẑ and J. Sgall. Well quasi ordering depends on the labels. *Acta Scientiarum Mathematicarum*, 55:59–65, 1991. Cited page 94.

[92] J. Kruskal. Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95:210–225, 1960. Cited pages 93, 95, and 102.

[93] J. Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *Journal of Combinatorial Theory, Series A*, 13(3):297 – 305, 1972. Cited page 93.

[94] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *Theory of Graphs: International Symposium: Rome*, pages 215–232, 1966. Cited pages 47 and 48.

[95] N. Linial, L. Lovasz, and A. Wigderson. Rubber bands, convex embeddings and graph connectivity. *Combinatorica*, 8:91–102, 1988. Cited page 48.

[96] D. Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006. Cited pages 26 and 124.

[97] D. Marx. Chordal deletion is fixed-parameter tractable. *Algorithmica*, 57(4):747–768, 2010. Cited page 25.

[98] D. Marx, B. O'Sullivan, and I. Razgon. Treewidth reduction for constrained separation and bipartization problems. In *STACS*, pages 561–572, 2010. Cited pages 26 and 72.

[99] D. Marx and I. Razgon. Constant ratio fixed-parameter approximation of the edge multicut problem. In *ESA*, volume 5757 of *Lecture Notes in Computer Science*, pages 647–658. Springer, 2009. Cited pages 13, 26, 132, and 153.

[100] D. Marx and I. Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 469–478. ACM, 2011. Cited pages iii, 21, 117, 119, 124, 126, and 153.

[101] K. Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10:96–115, 1927. Cited page 76.

[102] H. Nagamochi and T. Ibaraki. *Algorithmic Aspects of Graph Connectivity*. Cambridge University Press, 1st edition, 2008. Cited page 124.

[103] W. Naji. Reconnaissance des graphes de cordes. *Discrete Mathematics*, 54(3):329 – 337, 1985. Cited page 115.

[104] C. Nash-Williams. On well-quasi-ordering infinite trees. *Mathematical Proceedings of the Cambridge Philosophical Society*, 61(03):697–720, 1965. Cited pages 100 and 107.

[105] R. Niedermeier. *Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications)*. Oxford University Press, 2006. Cited page 25.

[106] S. Oum. Excluding a bipartite circle graph from line graphs. *Journal of Graph Theory*, 60:183–203, 2009. Cited page 109.

[107] M. Pouzet. Un bel ordre d'abritement et ses rapports avec les bornes d'une multirelation. *Comptes Rendus de l'Académie des Sciences, Paris, Séries A-B*, 274:1677–1680, 1972. Cited pages 11, 94, and 106.

[108] D. Raible and H. Fernau. An amortized search tree analysis for k-leaf spanning tree. In *SOFSEM*, pages 672–684, 2010. Cited pages 15 and 166.

[109] I. Razgon and B. O'Sullivan. Almost 2-SAT is fixed-parameter tractable. *Journal of Computer and System Sciences*, 75:435–450, 2009. Cited pages 21 and 153.

[110] B. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004. Cited page 21.

[111] N. Robertson and P. Seymour. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, 1986. Cited page 65.

[112] N. Robertson and P. Seymour. Graph minors. IV. Tree-width and well-quasi-ordering. *Journal of Combinatorial Theory, Series B*, 48:227–254, 1990. Cited page 65.

[113] N. Robertson and P. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995. Cited page 82.

[114] N. Robertson and P. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92:325–357, 2004. Cited page 93.

[115] N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 62(2):323–348, 1994. Cited pages 65 and 81.

[116] P. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, 1993. Cited pages 63 and 64.

[117] D. Shmoys. Cut problems and their application to divide-and-conquer. In *Approximation algorithms for NP-hard problems*. PWS, 1996. Cited page 27.

[118] R. Solis-Olba. 2-approximation for finding trees with many leaves. In *Proc. of ESA 1998*, pages 441–452. Cited page 29.

[119] J. Spinrad. Recognition of circle graphs. *Journal of Algorithms*, 16:264–282, 1994. Cited pages 12, 110, and 115.

[120] J. Storer. Constructing full spanning trees for cubic graphs. *Information Processing Letters*, 13:8–11, 1981. Cited page 29.

[121] S. Thomassé. On better-quasi-ordering countable series-parallel orders. *Transactions of the American Mathematical Society*, 352:2491–2505, 2000. Cited pages 95 and 107.

[122] S. Thomassé. A quadratic kernel for feedback vertex set. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 115–119, 2009. Cited page 24.

[123] E. Wanke. k-NLC graphs and polynomial algorithms. *Discrete Applied Mathematics*, 54:251–266, 1994. Cited page 94.

[124] J. Wu and H. Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *DIALM '99: Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 7–14, 1999. Cited page 29.

Abstract

This thesis tackles NP-hard problems with combinatorial techniques, focusing on the framework of Fixed-Parameter Tractability. The main problems considered here are MULTICUT and MAXIMUM LEAF OUT-BRANCHING. MULTICUT is a natural generalisation of the cut problem, and consists in simultaneously separating prescribed pairs of vertices by removing as few edges as possible in a graph. MAXIMUM LEAF OUT-BRANCHING consists in finding a spanning directed tree with as many leaves as possible in a directed graph. The main results of this thesis are the following. We show that MULTICUT is FPT when parameterized by the solution size, *i.e.* deciding the existence of a multicut of size k in a graph with n vertices can be done in time $f(k) * \mathrm{poly}(n)$. We show that MULTICUT IN TREES admits a polynomial kernel, *i.e.* can be reduced to instances of size polynomial in k. We give an $O^*(3.72^k)$ algorithm for MAXIMUM LEAF OUT-BRANCHING and the first non-trivial (better than 2^n) exact algorithm. We also provide a quadratic kernel and a constant factor approximation algorithm. These algorithmic results are based on combinatorial results and structural properties, involving tree decompositions, minors, reduction rules and $s-t$ numberings, among others. We present results obtained with combinatorial techniques outside the scope of parameterized complexity: a characterization of Helly circle graphs as the diamond-free circle graphs, and a partial characterisation of 2-well-quasi-ordered classes of graphs.

Keywords: *Parameterized Complexity, FPT, Parameterized Algorithms, Exponential Algorithms, Approximation Algorithms, Kernels, Multicut, Trees with Many Leaves, Helly Circle Graphs, Well-Quasi-Order*

Résumé

Dans cette thèse, nous abordons des problèmes NP-difficiles à l'aide de techniques combinatoires, en se focalisant sur le domaine de la complexité paramétrée. Les principaux problèmes que nous considérons sont les problèmes de Multicoupe et d'Arbre Orienté Couvrant avec Beaucoup de Feuilles. La Multicoupe est une généralisation naturelle du très classique problème de coupe, et consiste à séparer un ensemble donné de paires de sommets en supprimant le moins d'arêtes possible dans un graphe. Le problème d'Arbre Orienté Couvrant avec Beaucoup de Feuilles consiste à trouver un arbre couvrant avec le plus de feuilles possible dans un graphe dirigé. Les résultats principaux de cette thèse sont les suivants. Nous montrons que le problème de Multicoupe paramétré par la taille de la solution est FPT (soluble à paramètre fixé), c'est-à-dire que l'existence d'une multicoupe de taille k dans un graphe à n sommets peut être décidée en temps $f(k) * \mathrm{poly}(n)$. Nous montrons que Multicoupe dans les arbres admet un noyau polynomial, c'est-à-dire est réductible aux instances de taille polynomiale en k. Nous donnons un algorithme en temps $O^*(3.72^k)$ pour le problème d'Arbre Orienté Couvrant avec Beaucoup de Feuilles et le premier algorithme exponentiel exact non trivial (c'est-à-dire meilleur que 2^n). Nous fournissons aussi un noyau quadratique et une approximation à facteur constant. Ces résultats algorithmiques sont basés sur des résultats combinatoires et des propriétés structurelles qui concernent, entre autres, les décompositions arborescentes, les mineurs, des règles de réduction et les $s - t$ numberings. Nous présentons des résultats combinatoires hors du domaine de la complexité paramétrée: une caractérisation des graphes de cercle Helly comme les graphes de cercle sans diamant induit, et une caractérisation partielle des classes de graphes 2-bel-ordonnées.

Mots clefs : *Complexité Paramétrée, FPT, Algorithmes Paramétrés, Algorithmes Exponentiels, Algorithmes d'Approximation, Noyaux, Multicoupe, Arbres avec beaucoup de Feuilles, Graphes de Cercle Helly, Bel-Ordre*

LIRMM — 161, rue Ada — 34095 Montpellier cedex 5 — France