Matrix decompositions and algorithmic applications to (hyper)graphs

Benjamin Bergougnoux

THÈSE

Présentée par

Benjamin BERGOUGNOUX

Pour obtenir le diplôme de

Docteur d'université

Spécialité : Informatique

Matrix Decompositions and Algorithmic Applications to (Hyper)Graphs

---

Soutenue publiquement le 13 février 2019 devant le jury composé de :

Cristina BAZGAN	Rapporteure
Nicolas TROTIGNON	Rapporteur
Mourad BAÏOU	Président du jury
Christophe PAUL	Examinateur
Mamadou Moustapha KANTÉ	Directeur de thèse

École Doctorale Sciences Pour l'Ingénieur
Université Clermont Auvergne

Abstract

In the last decades, considerable efforts have been spent to characterize what makes NP-hard problems tractable. A successful approach in this line of research is the theory of *parameterized complexity* introduced by Downey and Fellows in the nineties. In this framework, the complexity of a problem is not measured only in terms of the input size, but also in terms of a *parameter* on the input. One of the most well-studied parameters is *tree-width*, a graph parameter which measures how close a graph is to the topological structure of a tree. It appears that tree-width has numerous structural properties and algorithmic applications.

However, only sparse graph classes can have bounded tree-width. But, many NP-hard problems are tractable on dense graph classes. Most of the time, this tractability can be explained by the ability of these graphs to be recursively decomposable along vertex bipartitions (A, B) where the adjacency between A and B is simple to describe. A lot of graph parameters – called *width measures* – have been defined to characterize this ability, the most remarkable ones are certainly clique-width, rank-width, and mim-width. In this thesis, we study the algorithmic properties of these width measures.

We provide a framework that generalizes and simplifies the tools developed for tree-width and for problems with a constraint of acyclicity or connectivity such as Connected Vertex Cover, Connected Dominating Set, Feedback Vertex Set, etc. For all these problems, we obtain $2^{O(k)} \cdot n^{O(1)}$, $2^{O(k \log(k))} \cdot n^{O(1)}$, $2^{O(k^2)} \cdot n^{O(1)}$ and $n^{O(k)}$ time algorithms parameterized respectively by clique-width, $\mathbb{Q}$-rank-width, rank-width and mim-width. We also prove that there exists an algorithm solving Hamiltonian Cycle in time $n^{O(k)}$, when a clique-width decomposition of width k is given. Finally, we prove that we can count in polynomial time the minimal transversals of β-acyclic hypergraphs and the minimal dominating sets of strongly chordal graphs. All these results offer promising perspectives towards a generalization of width measures and their algorithmic applications.

Resumé

Durant ces dernières décennies, d'importants efforts et beaucoup de café ont été dépensés en vue de caractériser les instances faciles des problèmes NP-difficiles. Dans ce domaine de recherche, une approche s'avère être redoutablement efficace : la théorie de la *complexité paramétrée* introduite par Downey et Fellows dans les années 90. Dans cette théorie, la complexité d'un problème n'est plus mesurée uniquement en fonction de la taille de l'instance, mais aussi en fonction d'un *paramètre*. Dans cette boite à outils, la *largeur arborescente* est sans nul doute un des paramètres de graphe les plus étudiés. Ce paramètre mesure à quel point un graphe est proche de la structure topologique d'un arbre. La largeur arborescente a de nombreuses propriétés algorithmiques et structurelles.

Néanmoins, malgré l'immense intérêt suscité par la largeur arborescente, seules les classes de graphes peu denses peuvent avoir une largeur arborescente bornée. Mais, de nombreux problèmes NP-difficiles s'avèrent faciles dans des classes de graphes denses. La plupart du temps, cela peut s'expliquer par l'aptitude de ces graphes à se décomposer récursivement en bipartitions de sommets (A, B) où le voisinage entre A et B possède une structure simple. De nombreux paramètres – appelés largeurs – ont été introduits pour caractériser cette aptitude, les plus remarquables sont certainement la *largeur de clique*, la *largeur de rang*, la *largeur booléenne* et la *largeur de couplage induit*. Dans cette thèse, nous étudions les propriétés algorithmiques de ces largeurs.

Nous proposons une méthode qui généralise et simplifie les outils développés pour la largeur arborescente et les problèmes admettant une contrainte d'acyclicité ou de connexité tel que

Couverture Connexe, Dominant Connexe, Coupe Cycle, etc. Pour tous ces problèmes, nous obtenons des algorithmes s'exécutant en temps $2^{O(k)} \cdot n^{O(1)}$, $2^{O(k \log(k))} \cdot n^{O(1)}$, $2^{O(k^2)} \cdot n^{O(1)}$ et $n^{O(k)}$ avec k étant, respectivement, la largeur de clique, la largeur de $\mathbb{Q}$-rang, la larguer de rang et la largueur de couplage induit. On prouve aussi qu'il existe un algorithme pour Cycle Hamiltonien s'exécutant en temps $n^{O(k)}$ quand une décomposition de largeur de clique k est donné en entrée. Finalement, nous prouvons qu'on peut compter en temps polynomial le nombre de transversaux minimaux d'hypergraphes β-acyclique ainsi que le nombre de dominants minimaux de graphes fortement triangulés. Tous ces résultats offrent des pistes prometteuses en vue d'une généralisation des largeurs et de leurs applications algorithmiques.

Remerciements

Une thèse n'est pas une aventure solitaire, c'est une réussite collective et humaine qui repose sur une multitude de personnes du maraîcher qui m'a vendu ses légumes au barman qui m'a servi ses bières. Tout cela repose sur ce qu'il y a de meilleur dans nos sociétés : la solidarité. Cette thèse n'aurait jamais vu le jour sans sécurité sociale, sans bourses d'études, sans aides sociales, sans écoles et universités publiques, etc. Ainsi mes premiers remerciements vont à ces femmes et ces hommes qui ont combattu et combattent, parfois au prix de leurs vies, pour la démocratie, la République, les droits humains et l'intérêt général.

Mes deuxièmes remerciements sont naturellement destinés à mon directeur de thèse Mamadou Moustapha Kanté. Merci pour toute cette énergie, cette sagesse (e.g., *la Planète des singes*) et cet ensemble indénombrable de conseils que tu as partagés avec moi.

J'aimerais remercier Cristina Bazgan et Nicolas Trotignon pour avoir accepté de rapporter ma thèse. Merci également à Mourad Baïou et Christophe Paul qui m'ont fait l'honneur d'être membre de mon jury de soutenance.

Merci à Clermont-Ferrand (une ville géniale, je conseille) et plus spécifiquement au LIMOS de m'avoir accueilli. Je remercie tout spécialement Béa et sa légendaire efficacité d'avoir été si patiente avec moi malgré un nombre non-négligeable de bourdes :P.

Durant ces années de thèse, j'ai eu la chance de travailler avec plein d'êtres humains très sympathiques. Je pense notamment à deux de mes co-auteurs : à Thomas Bellitto, pour cette escapade totalement inattendue en transitions interdites et à Florent Capelli, pour ses précieux et nombreux conseils de toutes sortes.

I would like to say "감사합니다" to O-joung Kwon, I will never forget this unexpected afternoon where we found the main idea of our "beautiful" [60] algorithm for Hamiltonian Cycle!

I would like to thank Binh-Minh Bui-Xuan, Jan Arne Telle and Martin Vatshelle for the creation of the d-neighbor equivalence relation. This beautiful concept is at the core of the result I am the most proud of. Now, I cannot design an algorithm without trying to use the d-neighbor equivalence relation.

Psychologiquement, la recherche n'est certainement pas un long fleuve tranquille en temps normal et c'est encore pire avec ces satanées deadlines et le syndrome de l'imposteur. Contre ces maux, rien de mieux que le rire, le jeu, le sport et tous ces bons moments passés en si bonne compagnie. J'aimerais remercier ici tous ces amis qui ont contribué, parfois sans s'en douter, à cette thèse.

Commençons par le commencement et donc par remercier tous les anciens colocataires de mon auberge espagnole. Merci à Amélie *lalala*-lemande pour sa bonne humeur contagieuse et son caractère inoubliable. Thank you Alex, I hope that one day, we will watch England win something! Merci à mes culs terreux : Camille avec sa Toul et ses larges épaules et François, fils de charron et dompteur de 407 break ! Je remercie Hanne et Shannen pour leur belgitude et les soirées inoubliables que j'ai passées en leur compagnie. Thank you Laura for all these pub crawls

we did! Un merci à Pierre pour tes Picons, tes Santiags et ces fameuses virées nocturnes... Et pour finir, merci à toi Zurab, de ton salami à ta façon de « faire la vaisselle », tu nous auras bien fait rigoler quand même !

J'aimerais remercier les thésards du LIMOS pour cette excellente ambiance de « travail » et pour m'avoir retrouvé dans ce champ ! Merci à Arnaud et à son légendaire sang froid dans la défaite, aux Benjamins, pour cette fameuse saint Patrick et leurs contre-pépouze-abricot-pastèque, à Giacomo avec qui j'ai été vice-champion olympique (d'Europe) de kayak, à Docteur Perret du Cray et Mister Riton pour son déhanché de folie lors de nos nombreuses danses, à Matthieu (l'être de lumière) mon premier fournisseur d'occasions de trinquer, et enfin à Rafael (et sa fameuse crampe) pour m'avoir amené à ce « bal populaire ». À toutes ces parties endiablées de *Hanabi* et cie[1] et aux pauses travail qui les entrecoupaient !

Merci à mes braves et spirituels[2] co-bureaux, Alexis, David et Kévin pour toutes ces discussions passionnantes et passionnées sur tout et (surtout) n'importe quoi:

$$0, 1, 2, 3, \ldots, 9, 01, 02, 03, \ldots, 09, 10, 11, \ldots, 99, 001, 002, \ldots$$

Un petit merci à Isabelle et Renaud du Beerland (le meilleur bar du monde) pour m'avoir fait découvrir tant de bonnes bières artisanales.

Il me faut remercier aussi ceux qui ont le plus contribué à ma construction intellectuelle. Merci à tous ces profs géniaux de la primaire au Master qui ont su me donner le goût des études, l'amour des mathématiques et ma passion pour l'algorithmique. Cela inclut de nombreux enseignants en informatique de l'université de Montpellier.

Et bien sûr, merci avant tout à mes parents – sans qui je ne serai pas né – pour m'avoir donné autant d'amour et de liberté. Je vous dois la curiosité et la persévérance nécessaire à une telle aventure. Merci pour ces innombrables efforts et sacrifices que vous avez faits pour moi. Je suis fier de vous.

Je remercie également ma sœur Salomé pour tous ces précieux moments qu'on a partagés et ces fous rires qui font un bien fou.

Pour terminer, j'aimerais remercier Eva de m'avoir supporté (dans les deux sens du terme) jusqu'aux bouts[3]. Merci d'avoir changé ma vie et de continuer à partager la tienne avec moi.

[1] Mais, au fait, il sert à quoi le 10 finalement ?

[2] Pas dans le sens mormons du terme.

[3] Notons l'utilisation du pluriel me permettant de rajouter gratuitement un « x » à cette thèse comme celui qu'on prononce fièrement dans « Bergougnoux ».

Contents

Introduction en français

Depuis la construction des premiers ordinateurs dans les années 40, le besoin d'algorithmes efficaces pour résoudre une multitudes de problèmes n'a jamais cessé de croître. Cela motiva les chercheurs à étudier la *complexité* des problèmes, i.e., les ressources, tel que le temps et la mémoire, qu'un ordinateur doit utiliser pour résoudre un problème. Les chercheurs constatèrent assez rapidement que certains problèmes semblaient plus difficiles à résoudre que d'autres. Par exemple, calculer le plus grand diviseur commun de deux entiers peut être fait assez rapidement tandis que trouver tous les diviseurs d'un entier semble demander beaucoup plus de temps et de mémoire. Malgré des décennies de recherche intensive, il existe de nombreux problèmes intéressants pour lesquels personne n'a pu trouver d'algorithmes rapides les résolvant ni prouver qu'ils n'en admettaient pas. Néanmoins, il est possible de caractériser la difficulté des problèmes en comparant leurs complexités relatives. Par exemple, étant donné les diviseurs de deux entiers a et b, il est facile de calculer le plus grand diviseur commun de a et de b. Ainsi, trouver les diviseurs d'un entier est au moins aussi difficile que calculer le plus grand diviseur commun de deux entiers. Cette approche a permis de classifier les problèmes en fonction de leurs complexités relatives.

La classification la plus célèbre est sans aucun doute celle émergeant de la théorie de la NP-complétude introduite indépendamment par Cook [25] et Levin [104] au début des années 70. Intuitivement, un problème est dans la classe NP si il peut être formulé comme une question dont la réponse est soit « oui » soit « non » et dont les instances où la réponse est « oui » peuvent facilement être vérifiée. Le Traveling Salesman Problem est un exemple typique de problème difficile dans NP. Étant donné un entier L, une liste de villes et les distances entre chaque paire de villes, ce problème demande si il existe une route visitant chaque ville et revenant à son point de départ de longueur inférieure à L. Malgré son apparente simplicité, personne jusqu'à présent n'a pu trouver d'algorithme rapide pour résoudre ce problème. Il se trouve que ce problème a une propriété étonnante : il est NP-*difficile* [97], c'est à dire qu'il est au moins aussi difficile que chacun des problèmes dans NP. En d'autres mots, si le Traveling Salesman Problem admet un algorithme rapide alors tous les problèmes dans NP admettent un algorithme rapide. Des milliers de problèmes intéressants se sont révélés être des problèmes NP-difficiles [68, 97] et aucun d'entre eux ne semblent admettre d'algorithme rapide. Ceci motiva les chercheurs à conjecturer que $\mathsf{P} \neq \mathsf{NP}$ où P est la classe des problèmes admettant un algorithme dont le temps d'exécution est polynomial en la taille de l'instance. Cette conjecture est considérée par de nombreux chercheurs comme la plus importante conjecture mathématique, malgré des décennies de recherche intensive elle est reste encore ouverte [63].

À première vue, $\mathsf{P} \neq \mathsf{NP}$ ressemble à un mur insurmontable et l'étude des problèmes NP-difficiles semble être une impasse. Mais les problèmes NP-difficiles sont connus pour avoir d'innombrables applications dans la vie réelle, on ne peut tout simplement pas adopter la politique de l'autruche à leurs égards. Par exemple, le Traveling Salesman Problem a de nombreuses applications en logistique et en planification mais aussi en génétique au niveau du séquençage ADN [107].

En dépit de leurs difficultés théoriques, en pratique, des softwares rapides tel les solveurs SAT ont été développés pour résoudre des problèmes NP-difficiles. Bien sur, ça ne prouve pas que P = NP car ces derniers n'arrivent pas à résoudre rapidement toutes les instances d'un problème NP-difficile. Les bonnes performances des ces softwares peuvent s'expliquer en partie par le fait que les instances rencontrées en pratique ne sont pas arbitraires. Ces instances cachent de nombreuses structures qui expliquent pourquoi elles sont faciles à résoudre [52, 53]. Par exemple, en pratique, une instance du Traveling Salesman Problem peut consister en un ensemble de villes existantes reliées par un réseau routier. Pour résoudre le problème sur un telle instance, un algorithme peut utiliser le fait qu'un réseau routier, étant construit par des homo sapiens, est structuré et optimisé pour le transport. Durant ces dernières décennies, d'importants efforts et beaucoup de café ont été dépensés en vue de caractériser ces structures cachées et de les exploiter pour obtenir des algorithmes rapides.

Dans ce domaine de recherche, une approche s'avère être redoutablement rapide : la théorie de la *complexité paramétrée* introduite par Downey et Fellows [42] dans les années 90. Dans cette théorie, la complexité d'un problème n'est plus mesurée uniquement en fonction de la taille de l'instance, mais aussi en fonction d'un *paramètre*. Cette nouvelle dimension permet d'obtenir une classification des problèmes NP-difficiles beaucoup plus fine qu'en théorie de la NP-complétude. Durant ces 30 années d'existence, la complexité paramétrée est devenue un domaine important de l'informatique théorique, elle est au cœur de plus d'un millier de publications, en particulier, trois livres récents [10, 37, 43] lui sont consacrés. Formellement, un *problème paramétré* est un problème dont chaque instance est associée avec une valeur numérique : le *paramètre*. Un problème donné peut être paramétré par une multitude de paramètres. Le choix du paramètre a un immense impact sur la complexité du problème. Définissons et commentons rapidement les concepts les plus importants de la complexité paramétrée.

- Un problème paramétré est dit FPT s'il admet un algorithme FPT : un algorithme dont le temps d'exécution est $f(k) \cdot n^{O(1)}$ avec k le paramètre, n la taille de l'instance et f une fonction. Un des objectifs principaux en complexité paramétrée est de créer des algorithmes FPT où la fonction f et l'exposant de n sont les plus petits possibles. En plus de leurs indéniable intérêt théorique, les algorithmes FPT sont également intéressants en pratique [24, 50, 53, 102, 103].

- Un problème paramétré est dit XP s'il admet un algorithme XP : un algorithme dont le temps d'exécution est $f(k) \cdot n^{g(k)}$ avec k le paramètre, n la taille de l'instance et avec f et g deux fonctions. En d'autres mots, un problème paramétré est XP si on peut le résoudre en temps polynomial sur les instances où le paramètre est considéré comme une constante. Contrairement aux algorithmes FPT, l'intérêt des algorithmes XP est avant tout théorique car ils sont significativement moins performants que les algorithmes FPT [52].

  Tandis que tous les problèmes FPT sont par définition XP, certains problèmes XP ne semblent pas admettre d'algorithmes FPT. Downey et Fellows [41] ont défini une classe de problèmes dans XP appelée W[1]. Il est conjecturé que FPT $\neq$ W[1], i.e., les problèmes dans W[1] ne sont pas FPT. Cette conjecture est pratiquement aussi raisonnable que P $\neq$ NP; dans les faits, les deux conjectures se ressemblent beaucoup [52]. Similairement aux réductions polynomiales utilisées pour prouver la NP-difficulté d'un problème, il existe une notion de réduction entre problèmes paramétrés pour prouver qu'un problème est W[1]-difficile (l'analogue de NP-difficile) et ainsi qu'il n'admet vraisemblablement pas d'algorithme FPT.

En plus de ces concepts, la complexité paramétrée offre de nombreuses techniques pour créer des algorithmes rapides [37]. Elle fournit aussi des outils pour obtenir des bornes inférieures

conditionnelles sur la complexité paramétré d'un problème tel que la W[1]-difficulté [37, 43]. Cette thèse se consacre à la création d'algorithmes FPT et XP rapides. Une partie des algorithmes que nous allons présenter sont asymptotiquement optimaux sous une hypothèse de complexité appelée *Exponential Time Hypothesis* (abrégée en ETH). Cette hypothèse de complexité a été introduite en 2001 par Impagliazzo et Paturi [82]. Informellement, ETH suppose qu'on ne peut pas résoudre le problème 3-SAT en temps $2^{o(n)}$ avec n le nombre de variables. Grâce à ETH, on peut prouver des bornes inférieures sur la complexité classique de problèmes NP-difficiles mais aussi sur leurs complexités paramétrées.

La grande majorité des résultats obtenus concernant la complexité paramétrée sont de près ou de loin rattachés aux *graphes*. Un graphe est une structure de donnée qui modélise une relation binaire entre des entités. Formellement, un graphe est une paire ordonnée $G = (V(G), E(G))$ avec $V(G)$ un ensemble d'éléments appelés *sommets* et $E(G)$ un ensemble de paires non-ordonnées de sommets appelées *arêtes*. Malgré leurs extrême simplicité, les graphes permettent de modéliser un nombre incalculable de concepts. On les utilise dans de nombreux domaines, en informatique mais aussi en biologie, en chimie, en physique, en économie, en électronique, en industrie, etc. Par exemple, les graphes excellent dans la modélisation de réseaux tel que les réseaux sociaux et les réseaux routiers en identifiant les sommets avec les carrefours et les arrêtes avec les routes qui les séparent.

Les graphes offrent un riche panel de paramètres tel que le degré maximum (nombre maximum de voisins d'un sommet), le diamètre (distance maximum entre deux sommets), le genre, la dégénérescence, etc. C'est donc sans surprise qu'ils s'avèrent être un terreaux fertiles pour de nombreux résultats en complexité paramétrée, même pour des problèmes qui ne sont pas directement liés aux graphes tel que des problèmes en intelligence artificielle [20, 79, 113].

La *largeur arborescente* est sans aucun doute un des paramètres de graphe les plus étudiés. Ce paramètre a été redécouvert plusieurs fois dans différents contextes et notamment par Robertson et Seymour [126] dans leurs monumentale projet sur les mineurs de graphes. Intuitivement, la largeur arborescente mesure à quel point un graphe est proche de la structure topologique d'un arbre. Plus précisément, un graphe a une largeur arborescente au plus k si il peut être représenté par une décomposition récursive (appelée *décomposition arborescente*) constituée d'ensembles de sommets (appelés *sacs*) de tailles au plus k connectés entre eux à la manière d'un arbre. La largeur arborescente a de nombreuses propriétés algorithmiques et structurelles, voir [8] pour une vue d'ensemble. Une de ces propriétés algorithmiques les importantes se retrouve dans le théorème de Courcelle [29] : un des plus célèbre méta-théorèmes en informatique. Il énonce que tout problème exprimable en logique monadique du second ordre (appelée MSO_2) peut être résolu en temps $f(k) \cdot n$ sur un graphe de n sommets et avec une décomposition arborescente de largeur au plus k. Bodlaender [7] prouva un résultat tout aussi fameux qui énonce qu'on peut calculer en temps $f(k) \cdot n$ une décomposition de largeur arborescente optimale d'un graphe de largeur arborescente au plus k. Ensemble, ces deux résultats prouvent qu'un nombre gargantuesque de problèmes sont FPT paramétrés par la largeur arborescente. Parmi ces problèmes, on retrouve des problèmes NP-difficiles célèbres et intensivement étudiés tel que Dominating Set, Hamiltonian Cycle et 3-Coloring. Cependant, les algorithmes qu'on obtient à travers le théorème de Courcelle sont trop souvent inutilisables en pratique car la fonction f dans leurs temps d'exécution est énorme. Beaucoup d'effort ont été porté pour obtenir de meilleurs algorithmes. De nos jours, pour de nombreux problèmes NP-difficiles, nous avons un algorithme paramétré par la largeur arborescente qui est optimal sous ETH.

Néanmoins, malgré l'immense intérêt suscité par la largeur arborescente, seul les classes de graphes peu denses peuvent avoir une largeur arborescente bornée. Mais, de nombreux problèmes NP-difficiles s'avèrent faciles dans des classes de graphes denses. La plupart du temps, cela

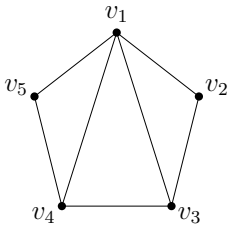
peut s'expliquer par l'aptitude de ces graphes à se décomposer récursivement en bipartitions de sommets (A, B) où le voisinage entre A et B possède une structure simple. De nombreux paramètres – appelés largeurs – ont été introduits pour caractériser cette aptitude, les plus remarquables sont certainement la *largeur de clique* [35], la *largeur de rang* [116], la *largeur booléenne* [17] et la *largeur de couplage induit* [132]. Dans cette thèse, nous étudions les propriétés algorithmiques de ces largeurs.

Expliquons comment ces paramètres sont définis. La largeur de clique est défini à partir des opérations suivantes dans les graphes labellisés : (1) la création d'un sommet avec un label $i \in \mathbb{N}$, (2) la relabellisation de tous les sommets labellisés i avec le label j, (3) l'union disjointe de deux graphes labellisés, (4) l'ajout de toutes les arêtes entre les sommets labellisés i et ceux labellisés j. La largeur de clique d'un graphe est le plus petit nombre de labels nécessaires pour le construire à partir de ces opérations. Les expressions construisant un graphe avec au plus k labels sont appelées k-expressions.

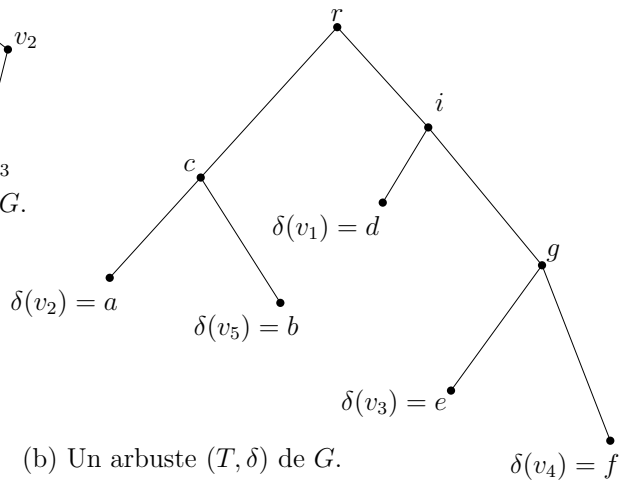
De leurs coté, la largeur de rang, la largeur de couplage induit et de nombreuses autres largeurs sont définis à partir de la notion d'*arbuste*. Un arbuste d'un graphe G est une paire (T, δ) où T est un arbre enraciné et δ est une bijection entre les sommets de G et les feuilles de T. Chaque nœud x de T est associé à un ensemble de sommets V_x dans G correspondant aux sommets qui sont en bijection avec les descendants de x. Étant donné une fonction $\mathsf{f} : 2^{V(G)} \to \mathbb{N}$, nous pouvons associer à chaque arbuste (T, δ) de G, une mesure – appelée f-largeur – correspondant au maximum $\mathsf{f}(V_x)$ parmi tous les nœuds x de T. La f-largeur de G est alors la f-largeur minimum parmi tous les arbustes de G.

Par exemple, la largeur de rang est définie à partir de la fonction $\mathsf{f}(A)$ qui correspond au rang dans $GF(2)$ de la matrice d'adjacence entre A et $V(G) \setminus A$; si on prend le rang dans $\mathbb{Q}$, on obtient une variante de la largeur de rang introduite dans [120] appelée *largeur de $\mathbb{Q}$-rang*. Pour la largeur de couplage induit, $\mathsf{f}(A)$ correspond à la taille maximum d'un couplage induit dans le graphe biparti associé à A et $V(G) \setminus A$. La Figure 1 présente un exemple d'arbuste (T, δ) d'un graphe G. Remarquez que la largeur de ($\mathbb{Q}$-)rang de (T, δ) égale 2 et ceci est optimal, i.e., la largeur de ($\mathbb{Q}$-)rang de G est 2. Cependant, la largeur de couplage induit de (T, δ) est 2 mais cela n'est pas optimal car on peut trouver un arbuste de largeur de couplage induit 1.

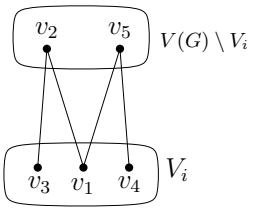
Les notions d'arbuste, de décomposition arborescente et de k-expression sont très efficaces pour résoudre rapidement les problèmes NP-difficiles avec des algorithmes de programmation dynamique. Expliquons les grandes idées utilisées pour créer de tels algorithmes avec les arbustes (les mêmes idées marchent sur les décompositions arborescente et les k-expressions). Supposons qu'on veut trouver un ensemble de sommet satisfaisant une propriété $\mathcal{P}$ de taille maximum dans un graphe G. Pour ce faire, on dispose d'un arbuste (T, δ) de G avec une petite f-largeur pour une certaine fonction f. Afin de résoudre ce problème, nous allons parcourir les nœuds de T de bas en haut et à chaque nœud x de T, nous allons calculer un ensemble de *solutions partielles*, une solution partielle étant une partie de $G[V_x]$ pouvant potentiellement s'étendre en une solution. Puisqu'on recherche un ensemble de sommets, les solutions partielles associées à un nœud x seront des sous-ensembles de V_x. L'idée principale est de calculer, pour chaque nœud x de T, un ensemble de solutions partielles $\mathcal{S}_x$ qui *représente* 2^{V_x}, i.e., pour chaque ensemble $X \subseteq V_x$ et chaque $Y \subseteq V(G) \setminus V_x$, si $X \cup Y$ vérifie $\mathcal{P}$, alors il existe un ensemble $X' \in \mathcal{S}_x$ tel que $X' \cup Y$ vérifie $\mathcal{P}$ et avec $|X'| \geq |X|$. Avec cette notion de représentativité, on est assuré de trouver une solution à notre problème (si il en existe une) dans l'ensemble de solutions partielles calculé pour la racine de T. La plupart des algorithmes de programmation dynamique sont basés sur ce concept de représentant, et proposer un algorithme rapide se réduit souvent à trouver une manière de calculer des petits représentants. Généralement, ceci est fait en définissant une relation d'équivalence $\sim$ sur 2^{V_x} avec « peu » de classes d'équivalence et tel que $X \sim W$ si pour tout $Y \subseteq V(G) \setminus V_x$, l'ensemble $X \cup Y$ vérifie $\mathcal{P}$ si et seulement si $W \cup Y$ vérifie aussi $\mathcal{P}$. Pour



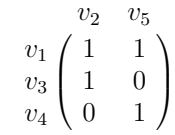
(a) Un graphe G.



(b) Un arbuste (T, δ) de G.



(c) Graphe bipartite associé avec $(V_i, V(G) \setminus V_i)$.



(d) Matrice d'adjacence entre V_i et $V(G) \setminus V_i$.

Figure 1 – La Sous-figure (a) présente un graphe G. La Sous-figure (b) montre un arbuste (T, δ) de G avec r comme racine de T. Observez que l'ensemble de sommets associé au nœud i est $V_i := \{v_1, v_3, v_4\}$. Les Sous-figures (c) et (d) représentent, respectivement, le graphe biparti et la matrice d'adjacence associés avec les ensembles de sommets V_i et $V(G) \setminus V_i$.

créer un algorithme FPT (respectivement XP), le nombre N de classes d'équivalence de $\sim$ ainsi que le temps T pour décider si $X \sim W$ doivent être bornés par $f(k) \cdot n^{O(1)}$ (resp. $f(k) \cdot n^{g(k)}$) avec k la f-largeur de (T, δ) et f, g des fonctions. Muni d'une telle relation d'équivalence, nous pouvons, étant donné un représentant $\mathcal{S}_x^\star$ de 2^{V_x}, calculer en temps $|\mathcal{S}_x^\star| \cdot N \cdot T$ un ensemble $\mathcal{S}_x$ de taille N qui représente 2^{V_x}. Effectivement, il suffit pour cela de garder un ensemble de taille maximum pour chaque classe d'équivalence de $\sim$ dans $\mathcal{S}_x^\star$.

Pour calculer les ensembles $\mathcal{S}_x$, pour tous les nœuds x de T, nous commençons par les calculer pour les feuilles de T. Cette étape est triviale car $|V_x| = 1$. Pour chaque nœud interne x de T avec deux enfants a et b, nous calculons $\mathcal{S}_x$ à partir de $\mathcal{S}_a$ et de $\mathcal{S}_b$. Généralement, on calcule un ensemble $\mathcal{S}_x^\star$ de taille au plus N^2 en prenant l'ensemble des solutions partielles obtenues par l'union d'une solution partielle de $\mathcal{S}_a$ et d'une solution partielle de $\mathcal{S}_b$. Normalement, il est facile de prouver que $\mathcal{S}_x^\star$ représente 2^{V_x} à partir du fait que $\mathcal{S}_a$ et $\mathcal{S}_b$ représentent, respectivement, 2^{V_a} et 2^{V_b}. Maintenant, on calcule $\mathcal{S}_x$ en temps $N^3 \cdot T$ à partir de $\mathcal{S}_x^\star$ comme expliqué précédemment. Vu qu'un arbuste possèdent $2n - 1$ nœuds, en appliquant ces idées, on obtient un algorithme s'exécutant en temps $O(N^3 \cdot T \cdot n)$.

Il y a trois aspects importants à considérer quand on étudie et compare des largeurs.

(a) La complexité paramétrée du calcul d'une décomposition de largeur optimale ou approchée.

(b) L'ensemble des graphes où cette largeur est petite.

(c) La complexité paramétrée des problèmes NP-difficiles paramétrés par cette largeur en supposant qu'une décomposition soit donnée en entrée.

L'Aspect (a) doit être considéré comme une étape de pré-calcul car une fois calculée la décomposition peut servir à résoudre rapidement de nombreux problèmes. Malheureusement, pour toutes les largeurs abordées dans cette thèse, calculer une décomposition de largeur optimale s'avère être NP-difficile. Cependant, pour la largeur arborescente et la largeur de ($\mathbb{Q}$-)rang, il existe des algorithmes FPT rapides pour calculer une décomposition qui approxime à une

constante près la largeur du graphe en entrée. Concernant la largeur de clique et de couplage induit, les algorithmes utilisant ces largeurs requièrent qu'une k-expression ou qu'un arbuste soit donné en entrée. En effet, on ne sait pas si on peut approximer à un facteur constant près la largeur de clique (ou de couplage induit) d'un graphe avec un algorithme XP. Nous donnons une vue d'ensemble concernant cet aspect en Section 2.5.

L'Aspect (b) est capital. En effet, même si une largeur est facile à calculer (ou à approximer), cette dernière peut être énorme, i.e., proche du nombre de sommets. Il est important de comparer les largeurs sur cet aspect car deux largeurs peuvent résoudre un problème avec le même temps d'exécution mais il peut y avoir une différence immense entre les valeur de ces deux largeurs sur un graphe.

Par exemple, la largeur de clique est plus générale que la largeur arborescente. En d'autres mots, si une classe de graphes a une largeur arborescente bornée alors sa largeur de clique est également bornée [35], mais le contraire est faux : la largeur arborescente d'une clique de taille n est $n-1$ tandis que sa largeur de clique est au plus de 2. La largeur de rang et de $\mathbb{Q}$-rang sont aussi générales que la largeur de clique, i.e., ces trois largeurs sont bornées dans les mêmes classes de graphes. Cependant, la largeur de clique d'un graphe peut être exponentiellement plus grande que sa largeur de ($\mathbb{Q}$-)rang. De son coté, la largeur de couplage induit est plus générale que les quatre autres largeurs, contrairement à ces dernières, elle est même bornée dans les graphes d'intervalles et de permutation où les autres largeurs ne sont pas bornées [5, 76].

Intuitivement, plus une largeur est générale plus il va être dur de résoudre des problèmes NP-difficiles avec celle-ci. Comparer la généralité des largeurs est utile car si une largeur f est plus générale qu'une largeur g alors :

- si un problème P paramétré par f est FPT, alors P est FPT paramétré par g et
- si un problème P' est W[1]-difficile paramétré par g, alors P' est aussi W[1]-difficile paramétré par f.

Par conséquent, un problème est FPT paramétré par la largeur de clique si et seulement si il est FPT paramétré par la largeur de ($\mathbb{Q}$-)rang. Cependant, même si deux largeurs sont équivalentes en terme de généralité, ça ne veut pas dire qu'elles ont les même propriétés ni que les techniques algorithmiques employées pour une vont s'avérer efficaces pour l'autre. Dans la Section 2.3, nous donnons un état de l'art sur les relations qu'entretiennent les différentes largeurs au niveau de l'Aspect (b).

De ce qu'on a dit, la largeur de rang semble surpasser la largeur de clique à la fois sur l'Aspect (a) et sur l'Aspect (b). Cependant, cette supériorité se paie sur l'Aspect (c) : la largeur de rang est plus difficile à manipuler que la largeur de clique pour résoudre des problèmes.

L'Aspect (c) est le thème central de cette thèse. Cet aspect a été intensivement étudié durant ces dernières décennies. Grâce au théorème de Courcelle [29] et à ses variantes [3, 13] (présentés dans la Sous-section 2.6.1), on connaît un nombre incommensurable de problèmes NP-difficiles qui sont FPT paramétrés par la largeur arborescente. Courcelle, Makowsky et Rotics prouvèrent une variante du théorème de Courcelle [28] pour la largeur de clique et de ($\mathbb{Q}$-)rang. Plus précisément, ils démontrèrent que tout problème exprimable en logique monadique du premier ordre (appelée MSO_1) – une restriction de MSO_2 – est FPT paramétré par la largeur de clique et de ($\mathbb{Q}$-)rang. De nombreux problèmes classiques sont exprimables en MSO_1 comme Dominating Set et Feedback Vertex Set. Cependant, les temps d'exécution des algorithmes qu'on obtient à partir de ces méta-théorèmes sont épouvantables. Pour obtenir des algorithmes efficaces, il est nécessaire de mettre les mains dans le cambouis.

Pour la largeur arborescente et la largeur de clique, nous avons une idée précise de la complexité de nombreux problèmes classiques tel que Vertex Cover, Hamiltonian Cycle et

Graph Coloring. C'est à dire que pour ces problèmes, nous connaissons des algorithmes paramétrés par la largeur arborescente du graphe en entrée (ou la largeur de clique d'une k-expression donnée en entrée) dont la dépendance sur le paramètre est asymptotiquement optimale sous ETH. En revanche, pour les autres largeurs, notre connaissance est assez limitée. Pour de nombreux problèmes NP-difficiles, il a une grosse différence entre le temps d'exécution des meilleurs algorithmes et les meilleures bornes inférieures. Dans la Section 2.6, nous donnons une vue d'ensemble des meilleurs algorithmes connus en incluant les résultats de cette thèse et les meilleures bornes inférieures connues pour plusieurs problèmes NP-difficiles classiques.

Dans le reste de cette introduction, nous allons présenter en détails les questions traitées dans cette thèse ainsi que ses contributions.

Tandis que les largeurs définies à partir du rang de matrices binaires[4] dans différents corps ont été intensivement étudiées [92, 95, 115, 120], on ne connaît rien des largeurs qu'on pourrait obtenir à partir de décompositions de matrices dérivées d'autres structures algébriques. Dans le Chapitre 3, nous introduisons deux nouvelles largeurs, qui à notre connaissance n'ont jamais été étudiées auparavant. On les a appelées respectivement *largeur de* $\mathbb{N}$*-rang* et *largeur de* $\mathcal{B}$*-rang*. Ces deux largeurs sont définies à partir de notions de décomposition de matrices dans des semi-anneaux. Un semi-anneaux est une structure algébrique ressemblant aux anneaux mais sans la contrainte que chaque élément doit avoir un inverse pour l'addition. Pour définir ces deux largeurs, nous utilisons les semi-anneaux suivants :

- le semi-anneau de Boole $\mathcal{B} = (\{0,1\}, \vee, \wedge)$ où $\vee$ et $\wedge$ sont, respectivement, le **ou** et le **et** logique,
- le semi-anneau $(\mathbb{N}, +, \cdot)$ où $+$ et $\cdot$ correspondent à l'addition et la multiplication qu'on apprend à l'école primaire.

Malgré le fait que ces semi-anneaux n'ont pas toutes les propriétés d'un corps, Froidure [64] a prouvé qu'on pouvait définir à partir d'eux une notion de rang dans les matrices binaires. Étant donné une matrice binaire M, on définit ainsi $\mathsf{rw}_{\mathcal{B}}(M)$ (resp. $\mathsf{rw}_{\mathbb{N}}(M)$) comme étant le nombre minimum de vecteurs nécessaires pour générer les lignes de M dans $\mathcal{B}$ (resp. $(\mathbb{N}, +, \cdot)$). Par exemple, pour la matrice M dont les lignes sont $(1,1,0), (0,1,1)$ et $(1,1,1)$. La somme des vecteurs $(1,1,0)$ et $(0,1,1)$ dans $\mathcal{B}$, $(\mathbb{N}, +, \cdot)$ et $GF(2)$ sont égales, respectivement, à $(1,1,1)$, $(1,2,1)$ et $(1,0,1)$. Par conséquent, nous avons $\mathsf{rw}_{\mathcal{B}}(M) = 2$, $\mathsf{rw}_{\mathbb{N}}(M) = 3$ et le rang de M dans $GF(2)$ égale 3.

La largeur de $\mathcal{B}$-rang (respectivement $\mathbb{N}$-rang) d'un graphe G correspond à la $\mathsf{rw}_{\mathcal{B}}$-largeur (resp. $\mathsf{rw}_{\mathbb{N}}$-largeur) de G où $\mathsf{rw}_{\mathcal{B}}(A) := \mathsf{rw}_{\mathcal{B}}(M)$ (resp. $\mathsf{rw}_{\mathbb{N}}(A) := \mathsf{rw}_{\mathbb{N}}(M)$) avec M la matrice d'adjacence entre A et $V(G) \setminus A$.

Dans cette thèse, nous prouvons que ces deux paramètres ont la même généralité que la largeur de clique et de rang. Pour cela, nous démontrons que $\mathsf{rw}_{\mathcal{B}}(A)$ (resp. $\mathsf{rw}_{\mathbb{N}}(A)$) correspond au nombre minimum de bicliques nécessaires pour couvrir (resp. partitionner) les arêtes du graphe biparti associé à A et $V(G) \setminus A$. À travers ces équivalences et grâce à [22], nous déduisons que les calculs de $\mathsf{rw}_{\mathcal{B}}(A)$ et de $\mathsf{rw}_{\mathbb{N}}(A)$ sont NP-difficiles et qu'on ne peut calculer $\mathsf{rw}_{\mathcal{B}}(A)$ en temps $2^{2^{o(\mathsf{rw}_{\mathcal{B}}(A))}}$ en supposant ETH. De plus, grâce à un méta-théorème de Sæther et Vatshelle [128], nous pouvons étendre ces résultats de difficulté aux calculs des deux largeurs dans les graphes.

Au vu de ces résultats, ces deux largeurs se révèlent décevantes : elles ont l'air très difficiles à calculer et elles ne se démarquent pas assez de la largeur de clique. Pour toutes ces raisons, nous n'avons pas continué l'étude de la largeur de $\mathcal{B}$-rang et de la largeur de $\mathbb{N}$-rang.

[4] Matrices dont les entrées sont 0 ou 1.

À la place, nous nous sommes concentrés sur les applications algorithmiques de la largeur de clique, la largeur de rang et la largeur de couplage induit. Apprivoiser un problème NP-difficile – trouver un bon algorithme – avec ces largeurs peut être aussi bien une simple formalité qu'un calvaire. Les problèmes NP-difficiles les plus gentils sont certainement ceux dont on peut vérifier une solution localement : en regardant juste le voisinage de chaque sommets séparément. C'est le cas des problèmes Independent Set, Dominating Set, et Maximum Induced Matching. Ces trois problèmes sont des problèmes de (σ, ρ)-*domination*. Étant donné une paire (σ, ρ) de sous-ensembles finis ou co-finis de $\mathbb{N}$ et un graphe G, un (σ, ρ)-dominant de G est un sous-ensemble D de $V(G)$ tel que, pour tout sommet $v \in V(G)$, le nombre de voisins de v dans D appartient à σ si $v \in D$ ou à ρ sinon. Un problème est un problème de (σ, ρ)-*domination* si il consiste à trouver un (σ, ρ)-dominant de poids maximum ou minimum. Par exemple, le problème Dominating Set consiste à trouver un $(\mathbb{N}, \mathbb{N} \setminus \{0\})$-dominant de poids minimum. Énormément de problème NP-difficiles appartiennent à cette famille de problèmes voir [18, Table 1].

Cette famille de problème a été introduite et apprivoisée avec la largeur arborescente par Telle et Proskurowski [129]. Elle a été aussi étudiée avec les largeurs de clique, de ($\mathbb{Q}$-)rang et de couplage induit dans [18, 73, 120]. Grâce à [18, 129], nous avons une idée précise de la complexité des problèmes de (σ, ρ)-domination paramétrés par la largeur arborescente ou la largeur de clique. C'est à dire, qu'on sait résoudre ces problèmes en temps $2^{O(k)} \cdot n$ et on connaît de nombreux problèmes de (σ, ρ)-domination qui ne peuvent pas être résolu en temps $2^{o(k)} \cdot n^{O(1)}$ en supposant ETH. Pour les autres paramètres, on sait grâce à [18, 120] qu'on peut résoudre chaque problème de (σ, ρ)-domination en temps $2^{O(k \log(k))} \cdot n^{O(1)}$, $2^{O(k^2)} \cdot n^{O(1)}$ et $n^{O(k)}$ quand k est, respectivement, la largeur de $\mathbb{Q}$-rang, la largeur de rang et la largeur de couplage induit. Cependant, nous n'avons pas sous ETH de bornes inférieures serrées pour ces largeurs.

A contrario, les problèmes incorporant une contrainte globale – e.g., connexité ou acyclicité – sont beaucoup plus durs à apprivoiser avec des largeurs. Parmi ces problèmes, on retrouve des classiques comme Hamiltonian Cycle, Connected Vertex Cover et Feedback Vertex Set. Notre connaissance sur la complexité de ces problèmes paramétrés par les largeurs était assez limitée même pour la largeur arborescente. Pendant un moment, la plupart des informaticiens pensaient que pour nombre de ces problèmes, les algorithmes naïfs en temps $k^{O(k)} \cdot n$, avec k la largeur arborescente du graphe, ne pouvait être améliorés. En effet, il semblait nécessaire de connaître les composantes connexes des solutions partielles dans l'optique de les mettre à jour et d'assurer la connexité de la solution calculée. Algorithmiquement, à chaque sac de la décomposition arborée, les composantes connexes des solutions partielles sont représentées par des partitions sur les k sommets du sac. Conserver toutes les partitions de k éléments possibles conduit inévitablement à un temps d'exécution de $k^{O(k)} \cdot n$.

Mais étonnamment, en 2011, Cygan et al. ont démontré [38] qu'il existait des algorithmes Monte Carlo s'exécutant en temps $2^{O(k)} \cdot n^{O(1)}$ pour une grande variété de problèmes avec une contrainte globale tel que Connected Dominating Set, Hamiltonian Cycle et Feedback Vertex Set. Pour obtenir ces algorithmes, Cygan et al. introduisent une technique appelée *Cut & Count*. On explique les grandes idées de cette technique dans la Section 4.3. Les principaux inconvénients des algorithmes obtenus avec cette approche sont les suivants : ils ne sont pas déterministes, leurs dépendances dans le nombre de sommets du graphe n'est pas linéaire et leurs dépendances par rapport aux poids des sommets ou des arrêtes est pseudo-polynomiale.

En 2013, Bodlaender et al. ont proposé dans [9] une approche générale appelée *rank-based approach* pour créer des algorithmes déterministes en temps $2^{O(k)} \cdot n$ pour une plus grande variété de problèmes. Les algorithmes obtenus avec la *rank-based approach* ne souffrent pas des inconvénients de ceux obtenus avec l'approche *Cut & Count*. La principale contribution de [9] a été de prouver qu'à partir d'un ensemble de solutions partielles $\mathcal{A}$, on pouvait calculer, en temps $|\mathcal{A}| \cdot 2^{O(k)}$, un sous-ensemble $\mathcal{B}$ de $\mathcal{A}$ de taille 2^{k-1} et qui représente $\mathcal{A}$. Ainsi, pour résoudre un

problème de connectivité, on peut se contenter de manipuler des ensembles de solutions partielles de taille $2^{O(k)}$. Pour prouver cela, étant donné un ensemble de solutions partielles $\mathcal{A}$ associé à un sac B, les auteurs définissent une matrice binaire $\mathcal{M}$ tel que :

- les lignes de $\mathcal{M}$ correspondent aux partitions de $\{1, \ldots, k\}$ représentant les composantes connexes des solutions partielles dans $\mathcal{A}$,
- les colonnes de $\mathcal{M}$ correspondent à toutes les partitions de $\{1, \ldots, k\}$ et représentent toutes les manières de connecter les sommets dans le sac B,
- $\mathcal{M}[p, q] = 1$ si et seulement si la « fusion » de p et de q égale $\{\{1, \ldots, k\}\}$, i.e., cette fusion représente une solution connectée.

Bodlaender et al. ont démontré que le rang de $\mathcal{M}$ est borné par 2^{k-1}; de plus, une base de poids maximum (ou minimum si on cherche à minimiser) générant les lignes de $\mathcal{M}$ peut être calculée en temps $|\mathcal{A}| \cdot 2^{O(k)}$ et correspond à un représentant de $\mathcal{A}$. Nous donnons plus d'explications sur cette approche dans les Sections 4.1 et 4.2. En plus de son indéniable intérêt théorique, la *rank-based approach* est intéressante en pratique [50].

Il est assez naturel de se demander si on peut adapter ces approches aux autres largeurs. Instinctivement, on a commencé à étudier cette question avec la largeur de clique car cette dernière est plus facile à manipuler que les autres. À ce moment là, notre connaissance sur la complexité des problèmes avec une contrainte globale paramétrés par la largeur de clique était encore plus incomplète que pour la largeur arborescente avant l'introduction du *Cut & Count*. Quelques problèmes comme Connected Dominating Set et Connected Vertex Cover étaient connus pour être FPT paramétrés par la largeur de clique car ils sont exprimables en MSO_1. Cependant, nous n'avions pas d'idée précise sur leurs complexités paramétrées en dehors de certains cas spéciaux comme Feedback Vertex Set qu'on peut résoudre en temps $k^{O(k)} \cdot n^{O(1)}$ en supposant qu'une k-expression est donnée [16]. Similairement à la largeur arborescente, les meilleures bornes inférieures pour ces problèmes stipulent qu'on ne peut les résoudre en temps $2^{o(k)} \cdot n^{O(1)}$ en supposant ETH.

Dans la Section 4.1, nous démontrons qu'on peut adapter la *rank-based approach* à la largeur de clique. Nous nous concentrons sur les variantes connexes des problèmes de (σ, ρ)-domination (on demande à la solution ou à son complément d'induire un graphe connexe), e.g., Connected Dominating Set et Connected Vertex Cover. Il n'est pas difficile de modifier l'algorithme de [18] pour les problème de (σ, ρ)-domination afin de résoudre leurs variantes connexes en temps $k^{O(k)} \cdot n^{O(1)}$. En effet, il suffit de garder une solution partielle pour chaque partition des labels possible correspond à la façon dont le graphe induit par cette solution partielle connecte ces classes de labels. On peut modifier légèrement cet algorithme naïf et prouver qu'on peut calculer à chaque étape, un représentant de taille $2^{O(k)}$ pour obtenir un algorithme s'exécutant en temps $2^{O(k)} \cdot n$.

Nous considérons aussi le problème Feedback Vertex Set, qui demande le calcul d'un ensemble de sommet de poids minimum à enlever pour obtenir un graphe acyclique. Nous démontrons qu'on peut résoudre ce problème en temps $2^{O(k)} \cdot n$. Même si notre algorithme est dans le même esprit que celui pour les variantes connexes des problèmes de (σ, ρ)-domination, il est beaucoup moins trivial car on doit gérer l'acyclicité. Ceci est dû au fait qu'avec les opérations de la largeur de clique, une multitude d'arêtes peuvent être ajoutées à la fois. Ainsi, contrairement à la largeur arborescente, à chaque étape de calcul, le nombre de sommets ayant un voisinage dans le reste du graphe n'est pas borné.

Dans nos deux algorithmes, nous utilisons la même approche que [9], mais pour Feedback Vertex Set, nous devons adapter celle-ci pour gérer l'acyclicité. Contrairement à [9], nous

ne pouvons pas garantir l'acyclicité en comptant le nombre d'arêtes induites par les solutions partielles. Dans notre cas, utiliser cette astuce, nous conduirait à un temps d'exécution de $n^{O(k)}$ (on explique pourquoi dans la Section 4.1).

Dans la Section 4.2, nous généralisons et étendons les résultats obtenus pour la largeur de clique aux largeurs de ($\mathbb{Q}$-)rang et de couplage induit. Nous étendons aussi la classe de problèmes étudiés en y incluant les variantes acycliques (ou acycliques et connexes) des problèmes de (σ, ρ)-domination, où on demande aux solutions d'induire un graphe acyclique (ou un arbre). Parmi ces problèmes, on retrouve des classiques comme Maximum Induced Tree et Longest Induced Path. Nous obtenons des algorithmes rapides pour résoudre toutes ces variantes de problèmes de (σ, ρ)-domination, avec les largeurs de clique, de ($\mathbb{Q}$-)rang et de couplage induit. Les temps d'exécution de ces algorithmes sont présentés dans la Table 1. À une constante prés dans les exposants, ces temps d'exécutions sont équivalents aux temps d'exécutions des meilleurs algorithmes connus pour les problèmes de (σ, ρ)-domination dans [18, 120], même pour des problèmes très basiques comme Independent Set et Dominating Set. Pour la largeur de clique, cela prouve que rajouter une contrainte de connexité ou d'acyclicité à ces problèmes n'ajoute pas de grande difficulté. Pour les autres largeurs, cela semble être également le cas mais nous ne pouvons confirmer cette intuition sans de meilleures bornes inférieures.

Table 1 – Temps d'exécution de nos algorithmes pour les différentes largeurs, ici n est le nombre de sommets du graphe en entrée et k la largeur de la décomposition en entrée.

Clique-width	Rank-width	$\mathbb{Q}$-rank-width	Mim-width
$2^{O(k)} \cdot n^{O(1)}$	$2^{O(k^2)} \cdot n^{O(1)}$	$2^{O(k \log(k))} \cdot n^{O(1)}$	$n^{O(k)}$

Pour obtenir ces algorithmes, nous généralisons et simplifions la *rank-based approach* de [9]. Pour y arriver, nous utilisons la notion de *d-voisins équivalence*. Ce concept a été introduit dans [18] et utilisé dans [18, 120] pour obtenir les meilleurs algorithmes connus pour les problèmes de (σ, ρ)-domination paramétrés par les largeurs de clique, de ($\mathbb{Q}$-)rang et de couplage induit. Formellement, étant donné $A \subseteq V(G)$ et $d \in \mathbb{N} \setminus \{0\}$, deux ensembles $X, Y \subseteq A$ sont d-voisins équivalents envers A si, pour chaque sommet v de $V(G) \setminus A$, nous avons $\mathsf{min}(d, |N(v) \cap X|) = \mathsf{min}(d, |N(v) \cap Y|)$ où $N(v)$ est le voisinage de v dans G. Il est assez simple de vérifier que c'est une relation d'équivalence, et si $d = 1$, alors celle relation mesure le nombre de voisinages différents dans $V(G) \setminus A$ générés par les sous-ensembles de A.

Dans la Section 4.2, nous utilisons une largeur définie à partir de la relation de d-voisinage appelée la *largeur de d-voisins*. Cette dernière correspond à la $\mathsf{s\text{-}nec}_d$-largeur où $\mathsf{s\text{-}nec}_d(A)$ est le maximum entre $\mathsf{nec}_d(A)$ et $\mathsf{nec}_d(A \setminus V(G))$ avec $\mathsf{nec}_d(B)$, le nombre de classes d'équivalence de la d-voisins équivalence sur B. Notons que la largeur booléenne introduite dans [17] correspond au logarithme binaire de la largeur de 1-voisins.

Similairement à [18] pour les problèmes de (σ, ρ)-domination, nous prouvons qu'on peut résoudre leurs variantes connexes en temps polynomial en n et en la largeur de d-voisins d'un arbuste donné en entrée, avec d une constante ne dépendant que de σ et de ρ. Les temps d'exécutions obtenus dans la Table 1 proviennent du fait que la largeur de d-voisins d'un arbuste $\mathcal{L}$ est bornée par $(d+1)^{\mathsf{mw}}$, $2^{\mathsf{rw}_{\mathbb{Q}} \log(d \cdot \mathsf{rw}_{\mathbb{Q}}+1)}$, $2^{d \cdot \mathsf{rw}^2}$, et $n^{d \cdot \mathsf{mim}}$ où $\mathsf{mw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{rw}$ et mim sont, respectivement, les largeurs de module (une largeur équivalent à la largeur de clique), de $\mathbb{Q}$-rang, de rang et de couplage induit de $\mathcal{L}$. Jusqu'à maintenant la d-voisins équivalence n'avait été utilisée que pour des problèmes avec des contraintes locales [18, 73, 120]. Étonnamment, nous prouvons qu'elle peut être utilisée pour les variantes connexes des ces problèmes. Cela démontre l'importance et la transversalité de la largeur de d-voisins concernant les largeurs de clique, de

($\mathbb{Q}$-)rang et de couplage induit.

Malheureusement, pour les variantes acycliques (ou acyclique et connexe) nous n'avons pas réussi à obtenir un algorithme en temps polynomial en n et en la largeur de d-voisins pour une certaine constante d. Malgré le fait, que les algorithmes que nous obtenons pour ces problèmes s'appuient énormément la largeur de d-voisins, à un moment, nous avons dû utiliser les propriétés des autres largeurs pour obtenir les temps d'exécution présentés dans la Table 1.

Les algorithmes que nous présentons dans la Section 4.2 généralisent et unifient de nombreux résultats dont notamment la *rank-based approach* de [9]. En effet, notre approche peut être utilisée pour résoudre en temps $2^{O(k)} \cdot n$ tous les problèmes que nous considérons, avec k la largeur arborescente du graphe en entrée. Ceci est dû au fait que si un ensemble de sommets S est un séparateur de taille k, alors $\mathsf{s\text{-}nec}_d(S) \leq (d+1)^k$.

De plus, les algorithmes présentés dans la Section 4.2 sont plus simples que ceux présentés dans [9] et dans la Section 4.1. Cela est dû principalement au fait que nous évitons d'utiliser des partitions pondérées pour représenter nos solutions partielles. Ceci simplifie grandement les opérations utilisées par les algorithmes, les étapes de calculs et les preuves. En particulier, l'utilisation de partitions pondérées pour représenter les solutions partielles dans la Section 4.1 implique de prendre soin de nombreux détails techniques concernant l'acyclicité. Notre approche simplifie l'algorithme pour Feedback Vertex Set paramétré par la largeur de rang de Ganian et Hliněný [65] et les algorithmes paramétrés par la largeur de couplage induit de Jaffke, Kwon et Telle pour Longest Induced Path [85] et Feedback Vertex Set [86].

Dans la Section 4.3, nous démontrons que la portée de nos idées n'est pas limitée à la *rank-based approach*. Nous prouvons qu'on peut utiliser les mêmes idées à l'approche *Cut & Count* pour obtenir un algorithme Monte Carlo pour les variantes connexes de problèmes de (σ, ρ)-domination. Ceci montre la généralité de nos idées et permet d'envisager de les appliquer à d'autres types de problèmes et d'autres approches.

Vous avez peut-être remarqué que nos généralisations de la *rank-based approach* et du *Cut & Count* ne s'appliquent pas à tous les problèmes considérés dans [9, 38]. C'est le cas pour les problèmes Steiner Tree et Hamiltonian Cycle. En fait, le problème Steiner Tree est NP-difficile dans les cliques et donc dans les graphes de largeurs de clique 2. Cependant, sa variante où les poids sont sur les sommets et non sur les arêtes correspond à la variante connexe d'un problème de (σ, ρ)-domination et ainsi cette variante fait parti des problèmes considérées dans les Sections 4.1 et 4.2.

De son coté, le problème Hamiltonian Cycle[5] n'est pas la variante d'un problème de (σ, ρ)-domination. En fait, ce problème est connu pour être exprimable en MSO_2 mais pas en MSO_1. C'est aussi le cas pour d'autres problèmes tout aussi classiques tels que Edge Dominating Set, Graph Coloring et Max Cut. Ces quatre problèmes sont connus pour être FPT paramétrés par la largeur arborescente grâce au théorème de Courcelle [29] et à ses variantes [3, 13]. Paramétrés par la largeur de clique, on sait qu'ils sont tous les quatre XP [49, 100]. Dans de nombreux papiers [71, 99, 100, 106], les auteurs se demandèrent si il existait pour ces problèmes des d'algorithmes FPT paramétrés par la largeur de clique. Fomin, Golovach, Lokshtanov et Saurabh [58] prouvèrent que les problèmes Edge Dominating Set, Graph Coloring et Hamiltonian Cycle sont W[1]-difficiles paramétrés par la largeur de clique. En 2014, les même auteurs [59] prouvèrent que Edge Dominating Set et Max Cut pouvaient être résolus en temps $n^{O(k)}$, à partir d'une k-expression, mais qu'on ne pouvait pas les résoudre en temps $n^{o(k)}$ en supposant ETH. Dans la conclusion de [59], les auteurs affirment qu'en supposant ETH, on ne peut pas résoudre Hamiltonian Cycle en temps $n^{o(k)}$ (ce résultat est prouvé dans [60]).

[5] Voir la Section 1.4 pour la définition du problème Hamiltonian Cycle.

Cependant, les auteurs ne purent pas prouver qu'on pouvait résoudre Hamiltonian Cycle en temps $n^{O(k)}$. À l'époque, les meilleurs algorithmes pour Hamiltonian Cycle et Graph Coloring s'exécutaient, respectivement, en temps $n^{O(k^2)}$ [49] et $n^{2^{O(k)}}$ [100]. Récemment, Golovach et al. [74] ont prouvé qu'étonnamment Graph Coloring ne pouvait pas être résolu en temps $n^{2^{o(k)}}$ en supposant ETH.

Dans la Section 4.4, nous prouvons qu'on peut résoudre Hamiltonian Cycle en temps $n^{O(k)}$ à partir d'une k-expression. Notre algorithme utilise une équivalence entre l'existence d'un cycle Hamiltonien dans un graphe et l'existence d'une marche Eulérienne qui utilise des arêtes de couleurs différentes alternativement dans un multigraphe dont les arêtes sont colorées avec deux couleurs. L'idée essentielle est que dans un tel multigraphe, l'existence d'une marche Eulérienne « alternante » peut être déterminée par les informations suivantes : le nombre d'arêtes colorée incidentes à chaque sommets et la connexité du multigraphe. Avec ces idées, nous évitons l'obstacle majeur de l'algorithme naïf dans [49] qui garde tous les multigraphes possibles sur k sommets et avec au plus n arêtes.

Grâce à notre résultat, nous avons maintenant une idée précise de la complexité des problèmes classiques qui sont W[1]-difficiles paramétrés par la largeur de clique. Même si notre approche semble assez ad hoc à première vue, elle pourrait nous aider à étendre le champ d'application de l'approche que nous avons créée pour les variantes de problèmes de (σ, ρ)-domination.

Jusqu'à maintenant, nous n'avons parlé que de problèmes d'optimisation ou de décision. La grande majorité des problèmes étudiés à travers la complexité paramétrée concerne ces deux types de problèmes. Laissés derrière pendant un moment, les problèmes de comptage reçoivent de plus en plus d'attention de la part de la communauté [9, 36, 57, 73]. Tandis qu'un problème de décision demande si il existe une solution, un problème de comptage demande le nombre de solutions. Ainsi, la variante de comptage d'un problème de décision NP-difficile est au moins aussi difficile que ce dernier. A contrario, si un problème de décision est dans P, sa variante de comptage n'admet pas nécessairement d'algorithme en temps polynomial [130].

À l'instar des problèmes de décision, il existe pour les problèmes de comptage deux classes de complexité jouant le même rôle que P et NP. L'ensemble des problèmes de comptage qu'on peut résoudre en temps polynomial est dénoté par FP (pour *Functional* P). Valiant [130] introduisit la classe de complexité #P l'analogue de NP pour les problèmes de comptage. Intuitivement, #P désigne l'ensemble des fonctions qui correspondent aux nombre de chemins acceptants d'une machine de Turing non-déterministe. En supposant #P $\neq$ FP, on peut s'intéresser à classifier les problèmes de comptage entre ceux qui sont faciles à résoudre, i.e., qui appartiennent à FP, et ceux qui sont difficiles, c'est à dire #P-difficiles [130] ou même difficiles à approximer [72].

La variante de comptage d'un problème de décision peut être beaucoup plus difficile que ce dernier. Par exemple, décider si un graphe admet un couplage parfait est décidable en temps polynomial tandis que compter le nombre de couplages parfaits d'un graphe est #P-difficile même dans les graphes bipartis [130]. Dans le même esprit, décider si une formule 2-SAT admet un modèle est un problème dans P alors que sa variante de comptage est #P-difficile [131].

Il est assez naturel d'étudier les problèmes de comptage avec la boite à outils fournie par la complexité paramétrée. Dans la Section 5.1, nous donnons une vue d'ensemble des résultats obtenus sur le comptage avec les largeurs.

Cependant, il reste énormément de zones d'ombre, les largeurs qu'on connaît ne parviennent pas à expliquer tous les résultats d'appartenance dans P et FP quand on restreint les instances à certaines classes de graphes. Ceci est particulièrement vrai pour les problèmes de comptage dont la variante de décision est facile. Par exemple, compter le nombre de *vertex covers* est #P-difficile en général mais restreint aux graphes cordaux, ce problème appartient à FP [112]. Or aucune largeur qu'on connaît et qui est bornée dans les graphes cordaux n'arrive à expliquer

ce résultat. Par conséquent, il faut découvrir de nouvelles largeurs et pour ce faire, nous devons améliorer notre compréhension de ces classes de graphes et des problèmes qu'on peut résoudre en temps polynomial sur ces classes.

Dans cette optique, nous démontrons, dans le Chapitre 5, plusieurs résultats sur la complexité classique de problèmes de comptage. Notre principal sujet d'étude est le comptage des *transversaux minimaux* dans les *hypergraphes*. Un hypergraphe est une collection de sous-ensembles – appelés hyper-arêtes – d'un ensemble fini d'éléments appelés sommets. Les hypergraphes généralisent la notion de graphes car un graphe est un hypergraphe dont les hyper-arêtes sont de taille 2. Un transversal est un ensemble de sommets qui intersecte toutes les hyper-arêtes de l'hypergraphe. Le comptage des transversaux minimaux est intimement lié au problème d'énumération des transversaux minimaux. Ce problème d'énumération a énormément d'applications dans de nombreux domaines [47]. Malgré des décennies de recherche intensives, on ne sait toujours pas si on peut le résoudre en temps polynomial en la taille de l'instance et du nombre de solutions. Compter les transversaux minimaux a également de nombreuses applications dans des domaines variés comme dans le *model checking* [44]. Ce problème a également un intérêt en théorie des graphes car il est intimement proche du comptage des dominants minimaux dans les graphes. Ces deux problèmes de comptage se révèlent être #P-difficiles en général. Cependant, la littérature regorge de résultats [20, 73, 94] démontrant qu'on peut résoudre ces problèmes en se restreignant à certaines classes de graphes ou d'hypergraphes.

Les hypergraphes sont connus pour être plus compliqués à manipuler que les graphes, même des notions basiques comme l'acyclicité ne sont pas triviale à définir dans les hypergraphes. En fait, plusieurs notions d'acyclicité coexistent [20]. Similairement, il existe différentes notions de largeurs pour les hypergraphes qui ont été introduites pour généraliser la largeur arborescente [77, 78]. À notre connaissance, personne n'a pu utiliser ces largeurs d'hypergraphes pour créer un algorithme FPT ou XP pour résoudre un problème NP-difficile ou #P-difficile. En fait, les largeurs d'hypergraphes qu'on connaît sont soit trop dures à manipuler, soit elles sont trop générales, i.e., bornées dans des instances difficiles.

Dans la Section 5.2, nous prouvons qu'on peut compter en temps polynomial les transversaux minimaux des hypergraphes β-acycliques. La β-acyclicité est une des notions généralisant l'acyclicité des graphes aux hypergraphes. En corollaire, nous déduisons qu'on peut compter en temps polynomial le nombre de dominant minimaux dans les graphes fortement cordaux, une classe de graphes où même la largeur de couplage induit n'est pas bornée. Le comptage des dominants minimaux dans les graphes cordaux est connu pour être #P-difficile [96]. Ces deux résultats offrent de prometteuses pistes pouvant mener à la découverte de nouvelles largeurs de graphes et d'hypergraphes intéressantes.

Introduction

The needs of efficient algorithms to solve real-world problems never ceased to increase since the construction of the first computers in the forties. This leads researchers to study the *complexity* of problems, i.e., the minimal amounts of resources – such as time and memory – a computer needs to solve a problem. It quickly became clear that all problems are not equally hard to solve. For example, computing the greatest common divisor of two integers can be done in a few steps of computation (Euclid's algorithm), while finding the divisors of an integer needs, apparently, much more time and memory to be computed. Despite decades of research, no one was able to find efficient algorithms for many interesting problems or to rule out the existence of efficient algorithms for these problems. Nevertheless, one can characterize the hardness of problems by comparing their relative complexity. For example, given the divisors of two integers a and b, we can easily find the greatest common divisor of a and b, hence finding the divisors of an integer is at least as hard as finding the greatest common divisor of two integers. This approach turned out to be successful for classifying the problems with respect to their relative complexity.

The most famous classification was due to the theory of NP-completeness introduced independently by Cook [25] and Levin [104] in the early seventies. Informally, a problem is in the class NP if it can be formulated as a yes-no question of the input value and the instances where the answer is "yes" have efficiently verifiable proofs. The Traveling Salesman Problem is a typical example of hard problem in NP. This problem asks, given an integer L, a list of cities and the distances between each pair of cities, whether there exists a route of length at most L that visits each city and returns to the origin city. This problem is in NP since it is quite easy given a route to check whether its length is smaller than L. Despite its apparent simplicity, no one was able to find an efficient algorithm for this problem. In fact, the Traveling Salesman Problem has a surprising property: it is NP-*hard* [97], meaning that it is at least as hard as any problem in NP. That is, the existence of an efficient algorithm for the Traveling Salesman Problem implies that every problem in NP admits an efficient algorithm. Thousands of interesting problems turned out to be NP-hard [68, 97] and none of them seems to admit an efficient algorithm. This leads researchers to conjecture that $\mathsf{P} \neq \mathsf{NP}$ where P is the class of problems in NP that admit an algorithm whose running time is polynomial in the input size. Proving or refuting this conjecture is one of the most important open problems in computer science [63].

At first glance, $\mathsf{P} \neq \mathsf{NP}$ may seem to be an insurmountable wall and the study of NP-hard problems may look like a dead-end. On the other hand, NP-hard problems have tons of applications in practice, we cannot just ignore them and pretend they do not exist. For instance, the Traveling Salesman Problem has several applications in transportation but also in planning, logistics, manufacture of microchips, DNA sequencing, etc. [107].

In spite of their theoretical hardness, in practice, efficient softwares such as SAT-solvers [87] have been developed for handling some NP-hard problems. Of course, this does not prove that $\mathsf{P} = \mathsf{NP}$ as these tools do not solve quickly all instances of an NP-hard problem. The good performances of these software in practice can be partially explained by the fact that real inputs are not arbitrary. That is, these real-world instances have a lots of hidden structures which

explain why they are easy to solve [52, 53]. For example, a real instance of the Traveling Salesman Problem may consist in a set of existing cities joined by a road network. An algorithm for this problem could use the fact that human networks are very structured and constructed to optimize transportation. In the last decades, considerable efforts have been spent to characterize these hidden structures and to use them in order to design efficient algorithms.

A successful approach in this line of research is the theory of *parameterized complexity* introduced by Downey and Fellows [42] in the nineties. In this framework, the complexity of a problem is not measured only in terms of the input size, but also in terms of a *parameter* on the input. This additional dimension allows the classification of NP-hard problems on a finer scale than in the classical complexity theory. During the 30 years of its existence, the area has transformed into a mainstream topic of theoretical computer science with thousands of papers and three recent books [10, 37, 43]. Formally, a *parameterized problem* is a problem whose instances are associated with a numerical value called *parameter*. A given problem can be parameterized by a multitude of different parameters. The choice of a parameter has a great impact on the complexity of the problem. Let us define and comment, briefly, some of the most important concepts in parameterized complexity.

- A parameterized problem is FPT if it admits an algorithm – called FPT algorithm – running in time $f(k) \cdot n^{O(1)}$ with n the input size, k the parameter, and f a function. Typically, one goal in parameterized complexity is to design FPT algorithms, trying to make both $f(k)$ and the exponent of n as small as possible. If a parameterized problem is FPT, then the problem is easy to solve for all instances where the parameter value is small. Additionally to their theoretical interests, FPT algorithms have practical applications [24, 50, 53, 102, 103].

- A parameterized problem is XP if it admits an algorithm – called XP algorithm – running in time $f(k) \cdot n^{g(k)}$ with f and g two functions, n the input size, and k the parameter. Observe that a problem is XP if it can be solved in polynomial time on every instance where the parameter value is fixed. Unlike FPT, the interest on XP is mostly theoretical since an XP algorithm is significantly less efficient than an FPT algorithm [52]. While every FPT parameterized problem is XP by definition, it is conjectured that some XP problems are not FPT. To support this conjecture, Downey and Fellows [41] define a class of XP problems called W[1]. It is conjectured that FPT $\neq$ W[1], i.e., the problems in W[1] are not FPT. Assuming FPT $\neq$ W[1] is almost as reasonable as assuming P $\neq$ NP; in fact, the two conjectures have a lot in common [52]. Similarly to polynomial reductions, there exists a notion of reduction between parameterized problems to prove that a parameterized problem is W[1]-hard (analog of NP-hard) and thus unlikely to be FPT.

Additionally to these concepts, parameterized complexity theory provides plenty of techniques to design efficient algorithms. It also provides tools to establish conditional lower bounds on the complexity of problems such as W[1]-hardness [37]. An important part of this thesis consists in designing efficient FPT and XP algorithms. Some of our algorithms are asymptotically optimal under a reasonable complexity assumption called *Exponential Time Hypothesis* (ETH for short). This complexity assumption was introduced in 2001 by Impagliazzo and Paturi in [82]. Roughly speaking, ETH states that the 3-SAT problem cannot be solved in time $2^{o(n)}$ with n the number of variables. Under ETH, one can prove lower bounds on the classical complexities of NP-hard problems but also on their parameterized complexities.

The vast majority of results obtained through the parameterized complexity toolbox are closely related to *graphs*. A graph is a data structure that models a binary relation between entities. Formally, a graph is an ordered pair $G = (V(G), E(G))$ comprising a set V of elements, called *vertices*, together with a set E of unordered pairs of vertices called *edges*. Despite their

simplicity, graphs can model a huge amount of concepts. We use graphs in many areas, in computer science but also in biology, chemistry, physic, economics, electrical engineering, industry, etc. For instance, graphs excel at modeling any kind of network such as friendship relations between people or a road network by identifying crossroads as vertices and roads as edges.

Graphs offer a wide spectrum of interesting parameters such as the maximum degree (the largest number of neighbors of a vertex), the diameter (the largest distance between two vertices), the degeneracy, the genus, etc. It comes with no surprise that graphs make a fertile ground for many results in parameterized complexity theory, even for studying problems which are not directly related to graphs such as AI problems [20, 79, 113].

One of the most well-studied graph parameters is *tree-width*. The concept of tree-width was rediscovered several times in different settings, in particular, by Robertson and Seymour [126] in their monumental Graph Minors project. Intuitively, tree-width measures how close a graph is to the topological structure of a tree. More precisely, a graph has tree-width at most k if it can be represented by a structural decomposition (called *tree decomposition*) into vertex sets of size k (called *bags*) that are connected in a tree-like fashion. It appears that tree-width has numerous structural properties and algorithmic applications, see [8] for a survey. A celebrated algorithmic meta-theorem by Courcelle [29] states that every graph problem expressible in monadic second-order logic (MSO_2) can be decided in time $f(k) \cdot n$ on an n-vertex graph given with a tree-decomposition of tree-width k. Another celebrated result proved by Bodlaender [7] states that an optimal tree-decomposition of a graph can be computed in time $f(k) \cdot n$. Consequently, Courcelle's Theorem and Bodlaender's algorithm prove that a huge number of problems are FPT parameterized by tree-width and that they are solvable in linear time on graphs of bounded tree-width. Among these problems, there are some well-studied and well-known NP-hard problems such as Dominating Set, Hamiltonian Cycle, and 3-Coloring. However, the FPT algorithms we obtain through Courcelle's meta-theorem have a huge dependency on tree-width. Many efforts have been spent to design better FPT algorithms. These efforts lead to the creation of fruitful techniques to obtain efficient algorithms. Nowadays, for many NP-hard problems, we have an FPT algorithm parameterized by tree-width whose dependency on tree-width is optimal under ETH.

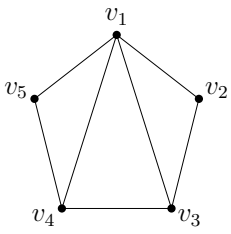
Nevertheless, despite the broad interest on tree-width, only sparse graph classes can have bounded tree-width. But many NP-hard problems are tractable on dense graph classes. Most of the time, this tractability can be explained by the ability of these graphs to be recursively decomposable along vertex bipartitions (A, B) where the adjacency between A and B has a simple structure. A lot of graph parameters – called *width measures* – have been defined to characterize this ability, the most remarkable ones are certainly clique-width [35], rank-width [116], boolean-width [17], and maximum induced matching width (called mim-width) [132]. In this thesis, we study the algorithmic properties of these width measures.

Let us explain how these width measures are defined. Clique-width is defined in terms of the following graph operations: (1) addition of a single vertex labeled $i \in \mathbb{N}$, (2) renaming label i into j, (3) addition of edges between vertices labeled i and those labeled j, (4) disjoint union. The *clique-width* of a graph is the minimum number of labels needed to construct it, and the expressions constructing a graph with at most k labels are called *k-expressions*.

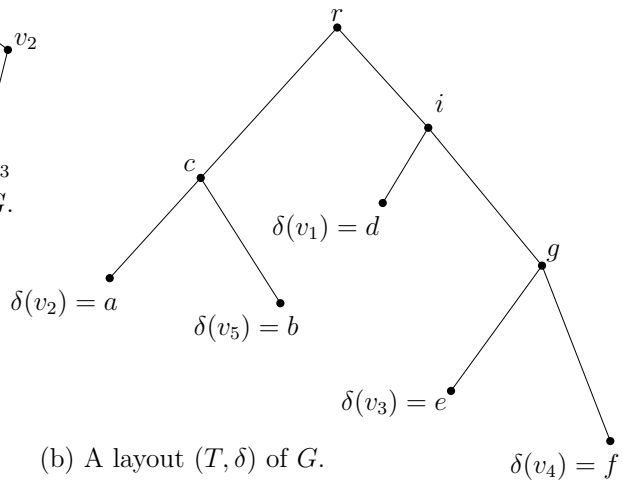
Rank-width, mim-width, and many other width measures are obtained through the notion of *rooted layout*. A rooted layout of a graph G is a pair (T, δ) where T is a rooted tree and δ is a bijection between the leaves of T and the vertices of G. Every node x of T is associated with a vertex set V_x of G which are the vertices of G in bijection with the leaves of T that are the descendants of x. Given a set function $\mathsf{f} : 2^{V(G)} \to \mathbb{N}$, one can associate with each rooted layout (T, δ) a measure, called usually f-width, defined as the maximum $\mathsf{f}(V_x)$ over all the nodes x of

T. The f-width of G is the minimum f-width over all rooted layouts of G.

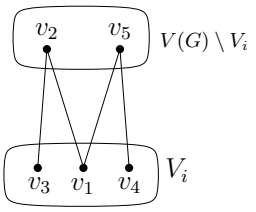
For instance, rank-width is defined from the function $\mathsf{f}(A)$ which corresponds to the rank over $GF(2)$ of the adjacency matrix between the vertex sets A and $V(G) \setminus A$; if we take the rank over $\mathbb{Q}$, we obtain a useful variant of rank-width introduced in [120], called $\mathbb{Q}$-rank-with. For mim-width, $\mathsf{f}(A)$ is the maximum size of an induced matching in the bipartite graph associated with $(A, V(G) \setminus A)$. See Figure 2 for an example of a rooted layout (T, δ) of a graph G. Notice that the rank-width and the $\mathbb{Q}$-rank-width of (T, δ) equal 2 and this is optimal, i.e., the rank-width and $\mathbb{Q}$-rank-width of G is 2. The mim-width of (T, δ) is 2 because $\{v_2v_3, v_4v_5\}$ is an induced matching of the bipartite graph associated with V_i and $V(G) \setminus V_i$. However, this is not optimal as one can find a rooted layout of G of mim-width 1.



(a) A graph G.



(b) A layout (T, δ) of G.



(c) Bipartite graph associated with $(V_i, V(G) \setminus V_i)$.

	v_2	v_5
v_1	1	1
v_3	1	0
v_4	0	1

(d) Adjacency matrix between V_i and $V(G) \setminus V_i$.

Figure 2 – Subfigure (a) shows a graph G. Subfigure (b) shows a layout (T, δ) of G with r as root of T. Observe that the vertex set associated with the node i is $V_i := \{v_1, v_3, v_4\}$. Subfigures (c) and (d) show, respectively, the bipartite graph and the adjacency matrix associated with the vertex sets V_i and $V(G) \setminus V_i$.

The notions of tree-decomposition, k-expression, and rooted layout are particularly suited to design fast dynamic programming algorithms for NP-hard problems. Let us explain the general ideas used to design such algorithms on rooted layouts (the same ideas hold for tree-decompositions and k-expressions). Suppose that we want to find a vertex subset of maximum size that satisfies some property $\mathcal{P}$ in a graph G. For doing so, we dispose of a rooted layout (T, δ) of G of small f-width for some set function f. We will solve this problem by doing a bottom-up traversal of T and at each node x of T, we will compute a set of *partial solutions*. As we seek a vertex subset, the *partial solutions* associated with a node x of T are subsets of V_x. The main idea is to compute, for each node x of T, a small set of partial solutions $\mathcal{S}_x$ that *represents* 2^{V_x}, i.e., for every $X \subseteq V_x$ and every $Y \subseteq V(G) \setminus V_x$, if $X \cup Y$ satisfies $\mathcal{P}$, then there exists $X' \in \mathcal{S}_x$ such that $X' \cup Y$ satisfies $\mathcal{P}$ and $|X'| \geq |X|$. With this notion of representativity, the set computed for the root of T must contain a set of maximum size that satisfies $\mathcal{P}$ in G, assuming that such a set exists. Most of the dynamic programming algorithms are based on this concept of representativity, and proposing a fast algorithm is usually reduced to finding a way of computing small representative sets. Typically, this is done by defining an equivalence relation $\sim$ on 2^{V_x} with "few" equivalence classes and such that $X \sim W$ if, for all $Y \subseteq V(G) \setminus V_x$, we have $X \cup Y$ satisfies $\mathcal{P}$ if and only if $W \cup Y$ satisfies $\mathcal{P}$. To design an FPT (resp. XP) algorithm, the number N of equivalence classes of $\sim$ and the time T to decide whether $X \sim W$ must be

bounded by $f(k) \cdot n^{O(1)}$ (resp. $f(k) \cdot n^{g(k)}$) with k the f-width of (T, δ) and f, g some functions. This way, given any representative set $\mathcal{S}_x^\star$ of 2^{V_x}, we can compute in time $|\mathcal{S}_x^\star| \cdot N \cdot T$ a set $\mathcal{S}_x$ of size N that represents 2^{V_x} by taking a set of maximum size in each equivalence class of $\sim$ over $\mathcal{S}_x^\star$.

In order to compute the sets $\mathcal{S}_x$, for each node x of T, we start by computing them for each leaf x of T, this is quite easy since $|V_x| = 1$. For every internal node x of T with two children a and b, we compute $\mathcal{S}_x$ from $\mathcal{S}_a$ and $\mathcal{S}_b$. Typically, we compute a set of partial solutions $\mathcal{S}_x^\star$ of size at most N^2 and in time $N^2 \cdot n^{O(1)}$, by taking the partial solutions of x obtained by the union of a set in $\mathcal{S}_a$ and one in $\mathcal{S}_b$. Normally, it is easy to prove that $\mathcal{S}_x^\star$ represents 2^{V_x} from the fact that $\mathcal{S}_a$ and $\mathcal{S}_b$ represent, respectively, 2^{V_a} and 2^{V_b}. Observe that computing $\mathcal{S}_x$ from $\mathcal{S}_x^\star$ can be done in time $N^2 \cdot T$ from the above explanations. As a layout has $2n - 1$ nodes, these ideas lead to an $O(N^2 \cdot T \cdot n)$ time algorithm.

There are three important aspects to consider when studying and comparing width measures.

(a) The parameterized complexity of computing a decomposition and the width of this decomposition with respect to the width of the input graph.

(b) The graph classes where this width is bounded.

(c) How fast we can solve NP-hard problems with a given decomposition.

Aspect (a) should be considered as a precomputation step since once a rooted layout is computed, we can use it to solve quickly many problems. Unfortunately, for all the width measures we deal with in this thesis and tree-width, computing an optimal decomposition is an NP-hard problem. However, for tree-width, rank-width and $\mathbb{Q}$-rank-width, there exist "efficient" FPT algorithms that, given a graph G, compute a corresponding decomposition of width that is within a constant factor from the width of G. On the other hand, all the algorithms parameterized by clique-width (or mim-width) require that a k-expression (resp. a rooted layout) is given as input. Indeed, it is not known whether the clique-width (respectively mim-width) of a graph can be approximated within a constant factor by an FPT or XP algorithm. We give an overview of this aspect in Section 2.5.

Aspect (b) is crucial. Indeed, even if the width of the computed decomposition is within a constant factor from the width of a graph, this latter may be huge, i.e., close to the number of vertices. It is primordial to compare width measures on this aspect because two width measures could have algorithms with the same running time but the values of these two width measures on the input graph could differ greatly.

For instance, the modeling power of clique-width is strictly stronger than the modeling power of tree-width. In other words, if a graph class has bounded tree-width, then it has bounded clique-width [35], but the converse is false as cliques have clique-width at most 2 and unbounded tree-width. Rank-width and $\mathbb{Q}$-rank-width have the same modeling power as clique-width, i.e., a graph has bounded ($\mathbb{Q}$-)rank-width if and only if it has bounded clique-width, but the clique-width of a graph can be exponentially bigger than its ($\mathbb{Q}$-)rank-width. In contrast, mim-width has the strongest modeling power among all these width measures and is even bounded on interval graphs and permutation graphs [5, 76] where the other complexity measures are unbounded.

Intuitively, the stronger the modeling power of a parameter is, the harder it will be to design efficient algorithms for this parameter. Comparing the modeling power of these width measures is quite useful since if a width measure p has a stronger modeling power than a width measure $\mathsf{p}^\star$, then:

- if a problem P parameterized by p is FPT, then P is FPT parameterized by $\mathsf{p}^\star$ and

- if a problem P' is W[1]-hard parameterized by $\mathsf{p}^\star$, then P' is W[1]-hard parameterized by p.

For instance, a problem is FPT parameterized by clique-width if and only if it is FPT parameterized by ($\mathbb{Q}$-)rank-width. However, even if two parameters have the same modeling power, it does not mean that they have the same properties or that the best algorithm for one will also give the best algorithm for the other. From what we said, rank-width seems to overcome clique-width both on Aspects (a) and (b) but this superiority has a price on Aspect (c): rank-width is harder to manipulate than clique-width when it comes to solve problems. In Sections 2.3, we give an overview of the known relations between the values of these parameters on graphs. Section 2.4 gives an overview of the well-known graph classes where the value of one of these parameters is bounded.

Aspect (c) is the main focus of this thesis. This aspect has been intensively studied over the last decades. Thanks to Courcelle's theorem [29] and its variants [3, 13], we know a huge number of NP-hard problems which are FPT parameterized by tree-width. Courcelle, Makowsky, and Rotics provided a variant of Courcelle's theorem [28] for clique-width and ($\mathbb{Q}$-)rank-width. More precisely, they showed that every problem expressible in monadic first order logic (called MSO_1) – a restriction of MSO_2 – is FPT parameterized by clique-width, rank-width, or $\mathbb{Q}$-rank-width (see Subsection 2.6.1 for an overview of these meta-theorems). Many famous problems are expressible in MSO_1 such as Feedback Vertex Set and Dominating Set. But, the running time of the algorithms we obtained from these meta-theorems are awful. If we want efficient algorithms, we need to get our hands dirty.

For tree-width and clique-width, we have a precise idea on the parameterized complexity of many classical NP-hard problems such as Vertex Cover, Hamiltonian Cycle, and Graph Coloring. That is, for these problems, we know algorithms parameterized by the tree-width of the input graph or the clique-width of a given k-expression whose dependency on the parameter is asymptotically optimal under ETH. In contrast, for the other parameters, there is a huge gap between the best lower bounds and the running times of the best-known algorithms. Section 2.6 gives an overview of the best (up to a constant in the exponent) algorithms (including our results) and the best lower bounds for several classical NP-hard problems.

In the rest of this introduction, we explain in detail the questions that are addressed in this thesis and we review our contributions.

While width measures defined through the rank of binary matrices[6] over some field have been intensively studied [92, 95, 115, 120], nothing is known on the width measures we could obtain via the matrix decompositions over other algebraic structures. In Chapter 3, we introduce two new width measures which, to the best of our knowledge, has never been studied. We call these parameters $\mathbb{N}$-rank-width and $\mathcal{B}$-rank-width. Both parameters are defined through the notion of rooted layout as the f-width of some function f defined via matrix decompositions over a *semiring*. A semiring is an algebraic structure similar to a ring, but without the requirement that each element has an additive inverse. To define these parameters, we use the following semirings:

- the Boole semiring $\mathcal{B} := (\{0,1\}, \vee, \wedge)$ where $\vee$ and $\wedge$ correspond, respectively, to the logical disjunction and the logical conjunction,
- the semiring $(\mathbb{N}, +, \cdot)$ where $+$ and $\cdot$ are the addition and multiplication we learn in elementary school.

[6] Matrix whose entries are either 0 or 1.

Despite the fact that these semirings do not have all the properties of a field, Froidure [64] has showed that we can define for them a rank-like notion on binary matrices. Given a binary matrix M, we define $\mathsf{rw}_{\mathcal{B}}(M)$ (resp. $\mathsf{rw}_{\mathbb{N}}(M)$) as the minimum number of vectors we need to generate the rows of M over $\mathcal{B}$ (resp. $(\mathbb{N}, +, \cdot)$).

For example, let M be the matrix with rows $(1,1,0), (0,1,1)$ and $(1,1,1)$. The sum of the vectors $(1,1,0)$ and $(0,1,1)$ over $\mathcal{B}, \mathbb{N}$ and $GF(2)$ equals, respectively, $(1,1,1)$, $(1,2,1)$, and $(1,0,1)$. Consequently, we have $\mathsf{rw}_{\mathcal{B}}(M) = 2$, $\mathsf{rw}_{\mathbb{N}}(M) = 3$, and the rank of M over $GF(2)$ equals 3.

The $\mathcal{B}$-rank-width (resp. $\mathbb{N}$-rank-width) of a graph G is the $\mathsf{rw}_{\mathcal{B}}$-width (resp. $\mathsf{rw}_{\mathbb{N}}$-width) of G where $\mathsf{rw}_{\mathcal{B}}(A) := \mathsf{rw}_{\mathcal{B}}(M)$ (resp. $\mathsf{rw}_{\mathbb{N}}(A) := \mathsf{rw}_{\mathbb{N}}(M)$) with M the adjacency matrix between A and $V(G) \setminus A$.

In this thesis, we prove that these two parameters are equivalent in terms of modeling power to clique-width and rank-width. For doing so, we prove that $\mathsf{rw}_{\mathcal{B}}(A)$ (resp. $\mathsf{rw}_{\mathbb{N}}(A)$) equals the minimum number of bicliques that cover (resp. partition) the edges of the bipartite graph associated with A and $V(G) \setminus A$. Moreover, from these equivalences, we deduce from [90, 114, 22] that computing $\mathsf{rw}_{\mathcal{B}}(A)$ and $\mathsf{rw}_{\mathbb{N}}(A)$ is NP-hard and that we cannot compute $\mathsf{rw}_{\mathcal{B}}(A)$ in time $2^{2^{\mathsf{rw}_{\mathcal{B}}(A)}}$ unless ETH fails. We can extend these hardness results to the computation of the $\mathcal{B}$-rank-width and the $\mathbb{N}$-rank-width of a graph thanks to a meta-theorem of Sæther and Vatshelle [128] on the computation of width measures.

These two new parameters turn out to be rather disappointing: they are too hard to compute and they do not seem to differ by much from rank-width and clique-width. For all these reasons, we did not continue to study $\mathcal{B}$-rank-width and $\mathbb{N}$-rank-width.

Instead, we focus on the algorithmic applications of clique-width, rank-width and mim-width. Taming an NP-hard problem with a width measure – finding an efficient algorithm parameterized by this width measure – could be a formality as much as a calvary. The kindest NP-hard problems are those for which the property of the object to be found can be verified by looking separately at the neighborhood of each vertex. This is the case for Independent Set, Dominating Set, and Maximum Induced Matching. These three problems belong to the family of problems called (σ, ρ)-Dominating Set problems. Given a pair (σ, ρ) of finite or co-finite subsets of $\mathbb{N}$ and a graph G, a (σ, ρ)-*dominating set* of G is a subset D of $V(G)$ such that, for each vertex $x \in V(G)$, the number of neighbors of x in X is in σ if $x \in X$ and in ρ otherwise. A problem is a (σ, ρ)-Dominating Set problem if it consists in finding a minimum (or maximum) (σ, ρ)-*dominating set*. For instance, the Dominating Set problem asks for the computation of a minimum $(\mathbb{N}, \mathbb{N}\setminus\{0\})$-dominating set. Many NP-hard problems belong to this family of problems, see [18, Table 1].

This family was introduced and tamed with tree-width in [129]. It was also studied with clique-width, ($\mathbb{Q}$)-rank-width, and mim-width in [18, 73, 120]. Thanks to [18, 129], we have a precise idea on the parameterized complexity of (σ, ρ)-Dominating Set problems with tree-width and clique-width. That is, we can solve them in time $2^{O(k)} \cdot n$ and, unless ETH fails, there is no $2^{o(k)} \cdot n^{O(1)}$ time algorithm for several (σ, ρ)-Dominating Set problems, e.g., Independent Set and Dominating Set. For the other parameters, we know from [18, 120] that any (σ, ρ)-Dominating Set problem can be solved in time $2^{O(k \log(k))} \cdot n^{O(1)}$, $2^{O(k^2)} \cdot n^{O(1)}$ and $n^{O(k)}$ when parameterized respectively by $\mathbb{Q}$-rank-width, rank-width and mim-width. However, we do not have tight lower bounds under ETH for these parameters.

In contrast, taming problems involving a global constraint – e.g., connectivity or acyclicity – require to get our hands dirty. Among these problems, we have some well-known and well-studied problems such as Hamiltonian Cycle, Steiner Tree, Connected Vertex Cover, and Feedback Vertex Set. For a long time, our knowledge on the parameterized complexity of

this kind of problems with width measures was quite limited even for tree-width. For a while, people used to think that for many of these problems the naive $k^{O(k)} \cdot n^{O(1)}$ time algorithms, k the tree-width of the input graph, could not be improved. Indeed, it seemed necessary to know the connected components of the partial solutions in order to be able to extend them and also certify that the computed solution is really connected. Algorithmically, at each bag of a tree-decomposition, the connected components of a partial solution can be represented by a partition on the k vertices of the bag and storing all possible partitions leads to a $k^{O(k)} \cdot n^{O(1)}$ time algorithm.

But, quite surprisingly, in 2011, Cygan et al. showed in [38] that we can design Monte Carlo $2^{O(k)} \cdot n^{O(1)}$ time algorithms for a wide range of problems with a global constraint, including Hamiltonian Cycle, Feedback Vertex Set, and Connected Dominating Set. For doing so, they introduced a technique that they called *Cut & Count*. We explain the main ideas of this approach in Section 4.3. The main drawbacks of the algorithms obtained from the Cut & Count approach are (1) they are not deterministic, (2) the dependence on the number of vertices of the input graph is not linear, and (3) the dependence on the inputs weights is pseudo-polynomial.

In 2013, Bodlaender et al. proposed in [9] a general toolkit called *rank-based approach* to design deterministic $2^{O(k)} \cdot n$ time algorithms, where k is the tree-width of the input graph, to solve a wider range of connectivity problems. The algorithms obtained from the rank-based approach do not suffer from the drawbacks of those obtained from the Cut & Count approach; on the other hand, they have a slightly worse dependence on the tree-width. The main contribution of [9] was to prove that, for some connectivity constraints problems, and for each bag B of a tree-decomposition, we can compute in time $2^{O(k)} \cdot n$ a set of partial solutions of size 2^{k-1} that represents the set of all partial solutions. For doing so, given a set of partial solutions $\mathcal{A}$ associated with a bag B, the authors define a binary matrix $\mathcal{M}$ such that:

- the rows of $\mathcal{M}$ correspond to the partitions of $\{1, \ldots, k\}$ that represent the connected components of the partial solutions in $\mathcal{A}$,
- the columns of $\mathcal{M}$ correspond to all the partitions of $\{1, \ldots, k\}$ and they represent all the potential ways of connecting the vertices of B,
- $\mathcal{M}[p, q] = 1$ if and only if the "join" of p and q equals $\{\{1, \ldots, k\}\}$, i.e., represents a connected solution.

Bodlaender et al. showed that the rank of $\mathcal{M}$ is at most 2^{k-1}; moreover, a maximum (or minimum if it is a minimization problem) weighted basis corresponds to a representative set of $\mathcal{A}$ and such a representative can be computed in time $|\mathcal{A}| \cdot 2^{O(k)} \cdot n^{O(1)}$. More explanations on the rank-based approach are given in Sections 4.1 and 4.2. Besides its theoretical interest, the rank-based approach is also a viable approach in practice [50].

It is quite natural to ask whether we can adapt these approaches to other width measures. Instinctively, we began to look at these questions with clique-width because this latter is easier to handle than the other width measures. At the time, our knowledge on the complexity of problems with a global constraint parameterized by clique-width was even more incomplete than for tree-width before the introduction of the Cut & Count approach. Some problems Connected Vertex Cover and Connected Dominating Set were known to be FPT parameterized by clique-width because they are expressible in MSO_1. However, we do not have a clear picture of their parameterized complexities, except for some special cases such as Feedback Vertex Set which was proved to admit a $k^{O(k)} \cdot n^{O(1)}$ time algorithm in [16], provided the graph is given with a k-expression. Similarly to tree-width, it is known that these three problems do not admit $2^{o(k)} \cdot n^{O(1)}$ time algorithms unless ETH fails.

In Section 4.1, we prove that we can adapt the rank based approach to clique-width. We investigate the connected variants of (σ, ρ)-Dominating Set problems (we ask the solution or the complement of the solution to induce a connected graph) e.g., Connected Dominating Set and Connected Vertex Cover. It is not hard to modify the dynamic programming algorithm from [18] for (σ, ρ)-Dominating Set problems in order to solve their connected variants in time $k^{O(k)} \cdot n^{O(1)}$, since it suffices to keep track, for each family of partial solutions, of the possible partitions of the label classes induced by them. We modify this naive algorithm slightly and prove that one can compute, at each step of computation, a representative set of size $2^{O(k)}$, yielding $2^{O(k)} \cdot n$ time algorithms resolving this family of problems.

We also consider the Feedback Vertex Set problem, which asks to compute a minimum set of vertices to delete so that the resulting graph is acyclic, and propose similarly a $2^{O(k)} \cdot n$ time algorithm. But, the algorithm, even in the same spirit as the one for connected (σ, ρ)-dominating set, is less trivial since one has to manage also the acyclicity. A task that is not trivial when dealing with clique-width operations as a bunch of edges can be added at the same time. Indeed, at each step of the dynamic programming algorithm, when dealing with tree-width the number of vertices that have a neighbor in the rest of the graph is at most the tree-width of the given decomposition, but for clique-width this number can belong to $O(n)$.

In both cases, we use the same rank-based approach as in [9], but we need to adapt it in the Feedback Vertex Set's case to manage the acyclicity. Contrary to [9], we cannot guarantee the acyclicity of the final solutions by just counting the number of vertices and edges. In our case, counting the number of edges would yield an $n^{O(k)}$ time algorithm (we explain why in Section 4.1).

In Section 4.2, we generalize and extend the results we obtained for clique-width to rank-width, $\mathbb{Q}$-rank-width, and mim-width. We also extend the class of problems considered to include the acyclic (or acyclic and connected) variants of (σ, ρ)-Dominating Set problems, where we ask the solution to induce an acyclic graph (or a tree). Among these variants of (σ, ρ)-Dominating Set problems we have some famous problems such as Maximum Induced Tree and Longest Induced Path. Consequently, we obtain efficient algorithms to solve all these variants of (σ, ρ)-Dominating Set problems, with parameters clique-width, ($\mathbb{Q}$)-rank-width, and mim-width. The running times of these algorithms are given in Table 2. Up to a constant in the exponent, these running times match those obtained for (σ, ρ)-Dominating Set problems in [18, 120] even for "basic" problems such as Independent Set and Dominating Set. For clique-width, it proves that adding connectivity or acyclicity constraints on a problem based on locally checkable constraints does not make these problems significantly harder. For the other width measures, it seems that this is also the case but we cannot confirm this intuition without tight lower bounds.

Table 2 – Running times of our algorithms for the different parameters, where n is the number of vertices of the given graph and k the width of the input decomposition.

Clique-width	Rank-width	$\mathbb{Q}$-rank-width	Mim-width
$2^{O(k)} \cdot n^{O(1)}$	$2^{O(k^2)} \cdot n^{O(1)}$	$2^{O(k \log(k))} \cdot n^{O(1)}$	$n^{O(k)}$

To obtain these algorithms, we generalize and simplify the rank-based approach from [9]. For doing so, we use the notion of *d-neighbor equivalence*. This concept was introduced in [18] and it was used in [18, 120] to obtain the best algorithms for (σ, ρ)-Dominating Set parameterized by clique-width, rank-width, $\mathbb{Q}$-rank-width or mim-width. Formally, given $A \subseteq V(G)$ and $d \in \mathbb{N} \setminus \{0\}$, two sets $X, Y \subseteq A$ are d-neighbor equivalent w.r.t. A if for all $v \in V(G) \setminus A$, we have

$\mathsf{min}(d, |N(v) \cap X|) = \mathsf{min}(d, |N(v) \cap Y|)$, where $N(v)$ is the set of neighbors of v in G. One easily checks that it is an equivalence relation, and if $d = 1$, then it measures the number of subsets of A with different neighborhoods in $V(G) \setminus A$.

In Section 4.2, we use a parameter related to the d-neighbor equivalence relation that we call *d-neighbor-width*. The d-neighbor-width of a graph G is defined from the function $\mathsf{s\text{-}nec}_d(A)$ where $\mathsf{s\text{-}nec}_d(A)$ is the maximum number of equivalence classes of the d-neighbor equivalence over A and $V(G) \setminus A$. It is worth noticing that the boolean-width introduced in [17] corresponds to the binary logarithm of the 1-neighbor-width.

Analogously to [18] for (σ, ρ)-Dominating Set problems, we prove that any connected variant of a (σ, ρ)-Dominating Set problem can be solved in time polynomial in n and the d-neighbor-width of the given layout, with d a constant that depends only on σ and ρ. The running times obtained in Table 2 follow from the fact that the d-neighbor-width of a rooted layout $\mathcal{L}$ is upper bounded by $(d+1)^{\mathsf{mw}}$, $2^{\mathsf{rw}_{\mathbb{Q}} \log(d \cdot \mathsf{rw}_{\mathbb{Q}}+1)}$, $2^{d \cdot \mathsf{rw}^2}$, and $n^{d \cdot \mathsf{mim}}$ with $\mathsf{mw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{rw}$ and mim being, respectively, the module-width (a width equivalent to clique-width), the $\mathbb{Q}$-rank-width, the rank-width, and the mim-width of $\mathcal{L}$. Until now, the d-neighbor equivalence relation was only used for problems with a locally checkable constraint [18, 73, 120]. We showed that, surprisingly, we can use it also for the connected variant of these problems. This highlights the importance and the transversality of the d-neighbor-width for clique-width, ($\mathbb{Q}$-)rank-width and mim-width.

Unfortunately, for the acyclic (or acyclic and connected) variants of (σ, ρ)-Dominating Set problems, we were not able to obtain algorithms whose running time is polynomial in n and the d-neighbor-width of the given layout (for some constant d). Despite the fact that we heavily rely on the d-neighbor width, we had to use the properties of the different width measures in order to obtain the running times presented in Table 2.

The framework presented in Section 4.2 can be used on tree-decomposition to obtain $2^{O(k)} \cdot n^{O(1)}$ time algorithms parameterized by tree-width for the variants of (σ, ρ)-Dominating Set problems. Indeed, given a vertex separator S of size k, the number of d-neighbor equivalence classes over S (resp. $V(G) \setminus S$) is upper bounded by 2^k (resp. $(d+1)^k$). For this reason, we can consider our framework as a generalization of the rank-based approach of [9]. Our framework generalizes also the clique-width adaptation of the rank-based approach used in Section 4.1 to obtain $2^{O(k)} \cdot n$ time algorithms, k being the clique-width of a given decomposition, for Connected (σ, ρ)-Dominating Set problem and Feedback Vertex Set. However, the constant in the running time of the algorithms in Section 4.1 and [9] are better than those of our algorithms. For instance, in Section 4.1, we obtain a $15^k \cdot 2^{(\omega+1) \cdot k} \cdot k^{O(1)} \cdot n$ time algorithm for Feedback Vertex Set, while in Section 4.2, we design a $54^k \cdot 2^{2(\omega+1) \cdot k} \cdot n^4$ time algorithm for this latter problem. Indeed, the approach in 4.2 is based on a more general parameter and is not optimized neither for tree-width nor clique-width.

Anyway, this approach simplifies the algorithms in Section 4.1 and [9] because we do not use weighted partitions to encode the partial solutions. Consequently, the definitions of the dynamic programming tables and the computational steps of our algorithms are simpler than those in Section 4.1 and [9]. This is particularly true for Feedback Vertex Set where the use of weighted partitions to encode the partial solutions in Section 4.1 implies to take care of many technical details concerning the acyclicity.

Moreover, The results we obtain simplify the $2^{O(k^2)} \cdot n^{O(1)}$ time algorithm parameterized by rank-width for Feedback Vertex Set from [65], and the $n^{O(k)}$ time algorithms parameterized by mim-width for Feedback Vertex Set and Longest Induced Path from [85, 86].

In Section 4.3, we show that the ambit of our ideas are not limited to the rank-based approach. We prove that the same ideas can be used to generalize the Cut & Count approach in order

to obtain an efficient Monte-Carlo algorithm for any connected variant of a (σ, ρ)-Dominating Set problem. This offers promising prospects to extend our ideas to other kinds of problems and other approaches.

One might notice that we were not able to apply our generalizations of the Cut & Count and rank-based approaches to all the problems considered in the original frameworks. This is the case of the problems Steiner Tree and Hamiltonian Cycle. In fact, the Steiner Tree problem is NP-hard on graphs of clique-width 2, i.e., cliques. However, its node-weighted variant corresponds to a connected variant of a (σ, ρ)-Dominating Set problem and thus fits into our framework.

On the other hand, we can prove that the Hamiltonian Cycle[7] problem is not a variant of a (σ, ρ)-Dominating Set problem. In fact, Hamiltonian Cycle is known to be expressible in MSO_2 but not in MSO_1. This is also the case for other well-known and well-studied problems such as Edge Dominating Set, Graph Coloring, and Max Cut. These four problems were known to be FPT parameterized by tree-width thanks to (variants of [3, 13]) Courcelle's theorem [29] and to be XP parameterized by clique-width [49, 100]. The existence of FPT algorithms parameterized by clique-width for these problems was asked in several papers [71, 99, 100, 106]. Fomin, Golovach, Lokshtanov, and Saurabh [58] proved the W[1]-hardness of Edge Dominating Set, Graph Coloring, and Hamiltonian Cycle parameterized by clique-width. In 2014, the same authors [59] proved that, given a k-expression, Max-Cut and Edge Dominating Set admit $n^{O(k)}$-time algorithms, and they do not admit $f(k) \cdot n^{o(k)}$-time algorithms unless ETH fails. In the conclusion of [59], the authors state that, unless ETH fails, there is no $f(k) \cdot n^{o(k)}$ time algorithm for Hamiltonian Cycle (this lower-bound is proved in [60]) and they left open the question of finding an algorithm with running time $f(k) \cdot n^{O(k)}$. At that time, the best known running time parameterized by clique-width for Hamiltonian Cycle and Graph Coloring were respectively $n^{O(k^2)}$ [49] and $n^{2^{O(k)}}$ [100]. Recently, Golovach et al. [74] proved that, surprisingly, Graph Coloring cannot be solved in time $f(k) \cdot n^{2^{o(k)}}$ unless ETH fails.

In Section 4.4, we prove that there exists an algorithm solving Hamiltonian Cycle in time $n^{O(k)}$, when a clique-width k-expression is given. We present a technique of representative sets using two-edge colored multigraphs on k vertices. The essential idea is that, for a two-edge colored multigraph, the existence of an Eulerian trail that uses edges with different colors alternately can be determined by two information: the number of colored edges incident with each vertex, and the connectedness of the multigraph. With this idea, we avoid the bottleneck of the naive algorithm, which stores all the possible multigraphs on k vertices with at most n edges.

Thanks to this result, we now have a precise idea on the complexity of the most classical problems that are W[1]-hard parameterized by clique-width. At first glance, the approach we used to design our algorithm seems quite ad hoc, but it could help to extend the application scope of the framework we designed for the connected and acyclic variants of (σ, ρ)-Dominating Set problems.

Until now, we have only talked about decision and optimization problems. The vast majority of problems studied through the parameterized complexity theory concerns these two kinds of problems. Left behind for a while, counting problems are receiving increasing attention from the parameterized complexity community [9, 36, 57, 73]. While decision problems ask whether there exists a solution, counting problems ask for the number of solutions. Hence, the counting variant of an NP-hard problem is unlikely to be solvable in polynomial time. On the other hand, if a decision problem is in P, its counting variant is not necessarily solvable in polynomial time [130].

[7] See Subsection 1.4 for the definitions of Hamiltonian Cycle.

As for decision problems, there exist two analog complexity classes to P and NP. The set of counting problems solvable in polynomial time is denoted by FP (for Functional P). Valiant [130] introduced a class of problems call #P which are an analog of NP for counting problems. Informally, #P denotes the set of functions corresponding to the number of accepting paths of a non-deterministic Turing machine. Under the assumption $\mathsf{FP} \neq \#\mathsf{P}$, one may be interested in classifying counting problems between those that are easy to compute, i.e., belong to FP, and those that are hard, i.e., are #P-hard [130], or even hard to approximate [72].

The counting variant of a decision problem could be much more tough than this later. For example, deciding whether a graph admits a perfect matching can be done in polynomial time while counting the number of perfect matchings is #P-hard even on bipartite graphs [131]. In the same spirit, deciding whether a 2-SAT formula admits a satisfying assignment is decidable in polynomial time but counting the number of satisfying assignments of a 2-SAT formula is #P-hard [131].

It is quite natural to study counting problems through the toolbox provided by the parameterized complexity theory. In Section 5.1, we give an overview on some results obtained on counting problems with width measures.

Nevertheless, the known width measures are not able to explain every tractability result on graph classes. This is especially true for counting problems whose decision variants are trivial. For instance, counting the number of vertex covers is #P-hard in general but restricted to chordal graphs, the problem belongs to FP [112]. To the best of our knowledge, we are not able to explain this tractability result in terms of width measures, i.e., they are not implied by a known XP or FPT algorithm. Consequently, we have to discover new width measures and for doing so, we need a better understanding of these graph classes and of the problems that are tractable on them.

In Chapter 5, we provide some results on the classical complexity of counting problems which could lead to the creation of interesting width measures. The main problem we study is the counting of the minimal *transversals* of a *hypergraph*. A hypergraph is a collection of subsets - called *hyperedges* - of a finite ground set, and a transversal is a subset of the ground set that intersects every hyperedge. Hypergraphs generalize the notion of graphs, i.e., a graph is a hypergraph whose hyperedges have size 2. The problem of computing the number of minimal transversals of a hypergraph – denoted by #Minimal Transversal – is closely related to the Transversal Enumeration problem which asks for the enumeration of all (inclusion-wise) minimal transversals of a given hypergraph. This enumeration problem has a lots of applications [47]. Despite the fact that it has been extensively studied over the last decades, it is not yet known whether this problem can be solved by an algorithm that runs in time polynomial in the size of the output. The #Minimal Transversal problem has also many applications in several areas, see for instance the description given in [44] in the case of model checking. This problem also has applications in graph theory as it is closely related to the problem of counting the *minimal dominating sets* of a graph, we denote this latter problem by #Minimal Dominating Set. It turns out that these counting problems are both #P-hard in general. However, there is a rich literature on solving these problems in polynomial time by restricting the input (hyper)graph to a specific class, see for example [20, 73, 94].

Hypergraphs are known to be more complex to manipulate than graphs, even basic notions on graphs such as acyclicity are not trivial to extend to hypergraphs. In fact, several concepts have been introduced to characterize the acyclicity of hypergraphs [20]. Similarly, several width measures on hypergraphs have been introduced to generalize tree-width [77, 78]. To the best of our knowledge, no one was able to use these width measures to design an FPT or an XP algorithm for an NP-hard (or #P-hard) problem. In fact, the width measures we know are either

too general, i.e., they are bounded on hard instances, or they are too hard to manipulate.

In Section 5.2, we prove that we can solve #Minimal Transversal in polynomial time on β-acyclic hypergraphs. The β-acyclicity is one generalization of graph acyclicity on hypergraphs. As a corollary, we deduce that we can solve #Minimal Dominating Set in polynomial time in strongly chordal graphs, a class of graphs of unbounded mim-width. This latter problem was known to be #P-hard on chordal graphs [96]. These two results offer promising prospects for the design of new width measures in graphs and hypergraphs.

Organization of this thesis

In Chapter 1, we present the definitions of the concepts we manipulate through this thesis. In particular, we give the definitions of d-neighbor equivalence and the definitions of the graph problems we deal with. We also present the concepts related to parameterized complexity and the *Exponential Time Hypothesis*.

In Chapter 2, we give the definitions of the width measures we deal with in this thesis and we give an overview of their properties. In particular, we compare their values on graphs and we discuss about their computations. We also present the asymptotically best algorithms for several classical problems parameterized by these width measures.

In Chapter 3, we define and study the $\mathcal{B}$-rank-width and $\mathbb{N}$-rank-width.

In Chapter 4, we present our results on connectivity problems: the acyclic and connected variants of (σ, ρ)-Dominating Set problems and the Hamiltonian Cycle problem.

In Chapter 5, we summarize some parameterized complexity results on counting problems especially those concerning width measures and we present our results on #Minimal Transversal and #Minimal Dominating Set.

In Chapter 6, we conclude by some perspectives and open questions concerning the study of *width measures*.

Publications

The results presented in this thesis have led to following publications:

- **More applications of the d-neighbor equivalence: acyclicity and connectivity constraints**,
  with Mamadou Moustapha Kanté,
  in Proceedings of the European Symposium on Algorithms (ESA 2019),
  Also on Arxiv: CoRR, abs/1805.11275, 2018.
  These results are presented in Section 4.2.

- **Fast exact algorithms for some connectivity problems parametrized by clique-width**,
  with Mamadou Moustapha Kanté,
  Theoretical Computer Science 2019,
  Also on Arxiv: CoRR, abs/1707.03584, 2017.
  These results are presented in Section 4.1.

- **Counting Minimal Transversals of β-Acyclic Hypergraphs**,
  with Florent Capelli and Mamadou Moustapha Kanté,
  Journal of Computer and System Sciences 2019,

Also on Arxiv: CoRR, abs/1808.05017, 2018.
These results are presented in Section 5.2.

- **An optimal XP algorithm for Hamiltonian cycle on graphs of bounded clique-width**,
with O-joung Kwon and Mamadou Moustapha Kanté,
submitted to *Algorithmica* in 2018, short version in Proceedings of the 15th International Symposium Algorithms and Data Structures (WADS 2017), volume 10389 of LNCS, pages 121–132, 2017,
Also on Arxiv: CoRR, abs/1702.06095, 2017.
These results are presented in Section 4.4.

During this Ph.D., I also work on projects that are not directly related to width measures. For the sake of coherence, we did not present these results in this thesis. These results lead to the following two publications:

- **Towards a Polynomial Kernel for Directed Feedback Vertex Set**,
with Eiben Eduard, Ganian Robert, Ordyniak Sebastian, and Ramanujan M. S.,
in Proceedings of the 42nd International Symposium on Mathematical Foundations of Computer Science,(MFCS 2017), volume 83 of LIPIcs, pages 36:1–36:15, 2017.
Kernelization is one of the most interesting facets of the parameterized complexity theory. The Kernelization theory provides many tools and fruitful techniques for a theoretical study of preprocessing. Here the goal is to obtain a *polynomial kernel* which is essentially a polynomial-time preprocessing algorithm that transforms the given instance of the problem into an equivalent one whose size is bounded polynomially in the parameter. In this appendix, we study Directed Feedback Vertex Set parameterized by the feedback vertex set number of the underlying *undirected graph*. We provide two main contributions: a polynomial kernel for this problem on general instances, and a linear kernel for the case where the input digraph is embeddable on a surface of bounded genus.

- **On Minimum Connecting Transition Sets in Graphs**,
with Thomas Bellitto,
in Proceedings of the 44th International Workshop on Graph-Theoretic Concepts in Computer Science, (WG 2018), volume 11159 of LNCS, pages 40–51, 2018.
Also on Arxiv: CoRR, abs/1808.05017, 2018.
These results concern *forbidden transition graphs*. A forbidden transition graph is a graph defined together with a set of permitted *transitions*, i.e., unordered pairs of adjacent edges that one may use consecutively in a walk in the graph. In this appendix, we look for the smallest set of transitions needed to be able to go from any vertex of the given graph to any other. We prove that this problem is NP-hard and study approximation algorithms. We develop theoretical tools that help to study this problem.

Chapter 1

Preliminaries

In this chapter, we give a formal definition of all the recurrent concepts used in this thesis. In Sections 1.1 and 1.2, we consider definitions from, respectively, set theory and graph theory. Section 1.3 is dedicated to the notion of d-neighbor equivalence which is used in our algorithms (presented in Section 4.2) and also to define two width measures in Chapter 2. We give the definition of this equivalence relation and some results. In Section 1.4, we define the graph problems we deal with in this thesis. In Section 1.5, we review notions of parameterized complexity. We conclude with a section presenting the Exponential Time Hypothesis.

1.1 Set Theory

The size of a set V is denoted by $|V|$ or $\#V$, and 2^V denotes the power-set of V. When the context is clear, we often write x to denote the singleton $\{x\}$. For two sets A and B, we denote by $A \setminus B$ the set $\{a \in a \,:\, a \notin B\}$, and we write $A \uplus B$ for the disjoint union of A and B. Moreover, we denote by $A \triangle B$ the symmetric difference of A and B, i.e. $(A \setminus B) \cup (B \setminus A)$.

Special Sets. We denote by $\mathbb{N}$ the set of non-negative integers and $\mathbb{N}^+$ the set $\mathbb{N} \setminus \{0\}$. For $k \in \mathbb{N}^+$, we denote by $[k]$ the set $\{1, \ldots, k\}$. The binary field, the rational field and the real field are denoted, respectively, by $GF(2)$, $\mathbb{Q}$, and $\mathbb{R}$.

Mapping. For a mapping $f : A \to B$ and $U \subseteq B$, we let $f^{-1}(U) := \{a \in A \,:\, f(a) \in U\}$. When $U = \{b\}$, we write $f^{-1}(b)$ instead of $f^{-1}(\{b\})$. We let $\mathsf{min}(\varnothing) := +\infty$ and $\mathsf{max}(\varnothing) := -\infty$.

Set function. Let V be a finite set and $\mathsf{f} : 2^V \to \mathbb{R}$ be a set function. If f satisfies $\mathsf{f}(A) = \mathsf{f}(V \setminus A)$ for all $A \subseteq V$, then f is said to be *symmetric*. If $\mathsf{f}(X) + \mathsf{f}(Y) \geq \mathsf{f}(X \cap Y) + \mathsf{f}(X \cup Y)$, then f is said to be *submodular*.

Merging Operation. For a ground set V and subsets $\mathcal{A}_1, \ldots, \mathcal{A}_k$ of 2^V, with $k \geq 2$, we let

$$\bigotimes_{1 \leq i \leq k} \mathcal{A}_i := \begin{cases} \varnothing & \text{if } \mathcal{A}_i = \varnothing \text{ for some } 1 \leq i \leq k, \\ \{T_1 \cup \cdots \cup T_k \,:\, \forall i \leq k,\ T_i \in \mathcal{A}_i\} & \text{otherwise.} \end{cases}$$

If $k = 2$, then we write $\mathcal{A}_1 \bigotimes \mathcal{A}_2$. We use this operator in almost all of our dynamic programming algorithms.

1.2 Graph Theory

Our graph terminology is standard, and we refer to R. Diestel's book [40].

Graph. We use the term graph to denote loopless simple undirected graphs. A graph is an ordered pair $G = (V, E)$ where V is a set whose elements are called *vertices* and E is a set of *edges*, which are unordered pairs of vertices. For a graph G, we denote its vertex set by $V(G)$, and its edge set by $E(G)$. We write xy or yx to denote an edge $\{x, y\}$. For every vertex set $X \subseteq V(G)$, when the underlying graph is clear from context, we denote by $\overline{X}$, the set $V(G) \setminus X$. Given a weight function $\mathsf{w} : V(G) \to \mathbb{Q}$ on the vertices of G and a set $X \subseteq V(G)$, we write $\mathsf{w}(X) := \sum_{x \in X} \mathsf{w}(x)$.

Neighborhood/degree. For a graph G and a vertex $v \in V(G)$, the neighborhood of v, denoted by $N_G(v)$, is the set of all vertices adjacent to v in G. The closed neighborhood of v is denoted by $N_G[v] := N_G(v) \cup \{v\}$. We extend these notions to sets of vertices, for $X \subseteq V(G)$, we define $N_G(X) := \bigcup_{v \in X} N_G(v)$ and $N_G[X] := N_G(X) \cup X$. We denote by $\mathsf{deg}_G(v)$ the number of vertices adjacent to v in G. If the underlying graph is clear, then we may remove G from the subscripts.

Subgraphs. Let G be a graph. A subgraph of G is a graph H such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For $X \subseteq V(G)$, we denote by $G[X]$ the subgraph of G induced by X, i.e., $(X, \{uv \in E(G) : u, v \in X\})$. For $F \subseteq E(G)$, we write $G - F$ for the subgraph $(V(G), E(G) \setminus F)$, and $G_{|F}$ for the subgraph $(V(G), F)$. For an edge e of G, we simply write $G - e$ for $G - \{e\}$.

For $X, Y \subseteq V(G)$, we denote by $G[X, Y]$ the bipartite graph with vertex set $X \cup Y$ and edge set $\{xy \in E(G) : x \in X \wedge y \in Y\}$. Moreover, we denote by $M_{X,Y}$ the (X, Y)-matrix such that $M_{X,Y}[x, y] = 1$ if $y \in N(x)$, 0 otherwise.

Walk/Trail/Path/Cycle. A *walk* of a graph G is a sequence $(v_1, e_1, v_2, e_2, \ldots, v_{t-1}, e_{t-1}, v_t)$ of vertices $v_1, \ldots, v_t$ and edges $e_1, \ldots, e_{t-1}$ such that, for every $i \in [t-1]$, the endpoints of e_i are v_i and v_{i+1}. The vertices v_1 and v_t are called *end-vertices* and the vertices $v_2, \ldots, v_{t-1}$ are called *internal* vertices. Notice that, for every $v \in V(G)$, we consider the sequence (v) as a walk. A walk is *closed* if its first and last vertices are the same. For convenience sometimes, we use the shortcut $(e_1, e_2, \ldots, e_{t-1})$ for a walk $(v_1, e_1, \ldots, v_{t-1}, e_{t-1}, v_t)$. This is possible because an edge always determines the two vertices with which it is incident. A *trail* of a graph is a walk where each edge is used at most once. A *path* of a graph is a trail in which all vertices (except possibly the first and the last) are pairwise distinct. A *cycle* of a graph is a path whose end vertices are the same.

Connected Component. A graph is *connected* if there exists a path between every pair of vertices. A *connected component* of a graph G is a maximal connected subgraph of G. For a graph G, we denote by $\mathsf{cc}(G)$ the following set

$$\mathsf{cc}(G) := \{V(C) : C \text{ is a connected component of } G\}.$$

Tree/Forest. A *forest* is an acyclic graph, i.e., a graph without cycles. A *tree* is a connected forest. Since we manipulate at the same time graphs and trees representing them, the vertices of trees will be called *nodes*. A node of degree at most one is called a *leaf* and a node of degree at least two is called an *internal* node. A *rooted tree* is a tree with a distinguished vertex called the *root*. For a rooted tree T and two adjacent nodes u and v of $V(T)$, we say that v is a child

of u if u lies in the path between v and the root of T. A *rooted binary tree* is a rooted tree where every internal node has exactly two children. A rooted caterpillar is a rooted binary tree where every internal node has at least a child that is a leaf.

1.3 d-neighbor equivalence

The following definition is from [18].

Definition 1.1 (d-neighbor equivalence [16])**.** *Let G be a graph, $A \subseteq V(G)$, and $d \in \mathbb{N}^+$. Two subsets X and Y of A are* d-neighbor equivalent w.r.t. A*, denoted by $X \equiv_A^d Y$, if $\mathsf{min}(d, |X \cap N(u)|) = \mathsf{min}(d, |Y \cap N(u)|)$ for all $u \in \overline{A}$.*

It is not hard to check that $\equiv_A^d$ is an equivalence relation. See Figure 1.1 for an example of 2-neighbor equivalent sets.

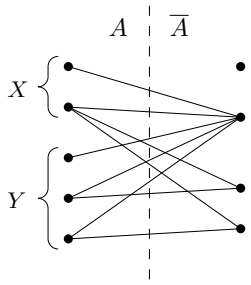


Figure 1.1 – We have $X \equiv_A^2 Y$, but it is not the case that $X \equiv_A^3 Y$.

We use the following definition to define two width measures in Chapter 2.

Definition 1.2 (nec_d)**.** *For a graph G and $d \in \mathbb{N}^+$, we let $\mathsf{nec}_d : 2^{V(G)} \to \mathbb{N}$ where for all $A \subseteq V(G)$, $\mathsf{nec}_d(A)$ is the number of equivalence classes of $\equiv_A^d$.*

Observation 1.3. *Notice that while nec_1 is a symmetric function [98, Theorem 1.2.3], nec_d is not necessarily symmetric for $d \geq 2$. For example, if a vertex x of G has c neighbors, then for every $d \in \mathbb{N}^+$, we have $\mathsf{nec}_d(\{x\}) = 2$ and $\mathsf{nec}_d(\overline{\{x\}}) = 1 + \mathsf{min}(d, c)$.*

The following lemma from [18] upper bounds $\mathsf{nec}_d(A)$ in terms of $\mathsf{nec}_1(A)$ for all $A \subseteq V(G)$.

Lemma 1.4 ([18])**.** *For every graph G and $d \in \mathbb{N}^+$, $\mathsf{nec}_d(A)$ and $\mathsf{nec}_d(\overline{A})$ are both upper bounded by $\mathsf{nec}_1(A)^{d \cdot \log_2(\mathsf{nec}_1(A))}$, for each $A \subseteq V(G)$.*

1.4 Some graph properties and problems

This section is dedicated to the definition of the graph problems we deal with in this thesis. We begin by the definition of some classical graph problems. We also give the definition of a family of problems called (σ, ρ)-Dominating Set problems and we introduce the connected and acyclic variants of this family of problems.

For a graph G and $X \subseteq V(G)$, we define the following graph properties.

Dominating set. A set $D \subseteq V(G)$ *dominates* a set $U \subseteq V(G)$ if every vertex in U is either in D or is adjacent to a vertex in D. A *dominating set* of G is a set that dominates $V(G)$.

Clique. A set $K \subseteq V(G)$ is a clique if the vertices in K are pairwise adjacent.

Feedback vertex set. A set $X \subseteq V(G)$ is a *feedback vertex set* if $G[V(G) \setminus X]$ is a forest.

Hamiltonian Cycle. A *Hamiltonian cycle* is a cycle that contains every vertex of G.

Independent Set. A set $I \subseteq V(G)$ is an *independent set* of G if the graph $G[I]$ has no edges.

Matching. A *matching* is a set of edges $M \subseteq E(G)$ such that every vertex in G is incident to at most one edge in M.

Induced matching. A set $M \subseteq E(G)$ is an *induced matching* if M is a matching of G and every edge in $E(G)$ intersects at most one edge in M.

q-coloring. A *q-coloring* of G is a partition of the vertices into at most q independent sets.

Vertex cover. A set $X \subseteq V(G)$ is a *vertex cover* of G if every edge in $E(G)$ has at least one endpoint in X.

We also define the following problems:

Dominating Set
Input: A graph G and a weight function $\mathsf{w} : V(G) \to \mathbb{N}$. **Output:** A dominating set D of G such that $\mathsf{w}(D)$ is minimum.

Clique
Input: A graph G and a weight function $\mathsf{w} : V(G) \to \mathbb{N}$. **Output:** A clique K of G such that $\mathsf{w}(K)$ is minimum.

Edge Dominating Set
Input: A graph G. **Output:** A set $D \subseteq E(G)$ of minimum size such that every edge in $E(G) \setminus D$ is incident with at least one edge in D.

Feedback Vertex Set
Input: A graph G and a weight function $\mathsf{w} : V(G) \to \mathbb{N}$. **Output:** A feedback vertex set X of G such that $\mathsf{w}(X)$ is minimum.

Graph Coloring
Input: A graph G. **Output:** The minimum $q \in \mathbb{N}$ such that G admits a q-coloring.

Hamiltonian Cycle
Input: A graph G. **Output:** Does G admit a Hamiltonian cycle?

Independent Set
Input: A graph G and a weight function $\mathsf{w} : V(G) \to \mathbb{N}$. **Output:** An independent set I of G such that $\mathsf{w}(I)$ is maximum.

Max-Cut
Input: A graph G. **Output:** A set $A \subseteq V(G)$ such that $\lvert E(G[A, \overline{A}])\rvert$ is maximum.

Node-Weighted Steiner Tree
Input: A graph G, a weight function $\mathsf{w} : V(G) \to \mathbb{N}$, and a set $K \subseteq V(G)$. **Output:** A subset $X \subseteq V(G)$ such that $K \subseteq V(G)$, $G[X]$ is connected, and $\mathsf{w}(X)$ is minimum.

Vertex Cover
Input: A graph G and a weight function $\mathsf{w} : V(G) \to \mathbb{N}$. **Output:** A vertex cover X of G such that $\mathsf{w}(X)$ is minimum.

Longest Induced Path
Input: A graph G. **Output:** An induced path of G of maximum length.

In general, let $\mathcal{P}$ be a vertex subset property. The maximization (resp. minimization) problem associated with $\mathcal{P}$ asks, given a graph G and a weight function $\mathsf{w} : V(G) \to \mathbb{N}$, for the computation of a set $X \subseteq V(G)$ that satisfies $\mathcal{P}$ and such that $\mathsf{w}(X)$ is maximum (resp. minimum).

Given a graph G, we say that a set X satisfies the *acyclic*, *connected* and *AC* variant of $\mathcal{P}$, if X satisfies $\mathcal{P}$ and $G[X]$ is, respectively, acyclic, connected, and a tree. The acyclic, the connected, and the AC variants of a maximization (resp. minimization) problem Π associated with $\mathcal{P}$ are the maximization (resp. minimization) problems associated with, respectively, the acyclic, the connected, and the AC variant of $\mathcal{P}$. The acyclic, the connected, and the AC variants of Π are denoted, respectively, by Acyclic Π, Connected Π, and AC-Π. For example, the Connected Dominating Set problem is the connected variant of the Dominating Set problem.

(σ, ρ)-Dominating Set problems. Now, we give the definition of the family of problems called (σ, ρ)-Dominating Set problems. This family of problems was introduced in [129] as a generalization of Dominating Set. Many NP-hard problems based on a locally checkable constraint belong to the family of (σ, ρ)-Dominating Set problems. Bui-Xuan, Telle and Vatshelle [18] designed a beautiful algorithm that solves any (σ, ρ)-Dominating Set problem. This algorithm is based on the d-neighbor equivalence where d depends on the considered problem. See Subsection 4.1.3 for some explanation of the value of d. We will show in Section 4.2 that similar algorithms exist for the acyclic and the connected variants of the (σ, ρ)-Dominating

Set problems. We review the algorithmic results concerning the (σ,ρ)-Dominating Set problems and their variants in Subsection 2.6.2. To the best of our knowledge, these variants were never studied until now.

Let σ and ρ be two (non-empty) finite or co-finite subsets of $\mathbb{N}$. We say that a subset D of $V(G)$ (σ,ρ)-*dominates* a subset $U \subseteq V(G)$ if

- for every vertex $u \in U \cap D$, we have $|N(u) \cap D| \in \sigma$ and
- for every vertex $u \in U \setminus D$, we have $|N(u) \cap D| \in \rho$.

A subset D of $V(G)$ is a (σ,ρ)-*dominating set* if D (σ,ρ)-*dominates* $V(G)$.

A problem Π is a (σ,ρ)-Dominating Set problem if Π is the maximization or minimization problem associated with a (σ,ρ)-dominating set property.

Notice that vertex cover is not a property expressible as a (σ,ρ)-dominating set, but the complement of a vertex cover, i.e., an independent set, is a $(\{0\},\mathbb{N})$-dominating set. In order to catch problems like Connected Vertex Cover into the framework designed for (σ,ρ)-dominating sets, we introduce the following notion of *co-(σ,ρ)-dominating set*. A subset X of $V(G)$ is a *co-(σ,ρ)-dominating set* if $V(G) \setminus X$ (σ,ρ)-*dominates* $V(G)$. We denote by Co-(σ,ρ)-Dominating Set problems the family of problems that are the maximization or minimization problems associated with the co-(σ,ρ)-dominating set properties.

Examples of some vertex subset properties expressible as (σ,ρ)-dominating sets or as a variant of a (σ,ρ)-dominating set are shown in Table 1.1. Many (σ,ρ)-Dominating Set problems and their connected, acyclic and AC variants associated with these vertex subset properties are NP-hard. These NP-hardness results are proved in [19, 68, 135]. It is worth mentioning that the connected variants of the minimization problems associated with the properties vertex cover, dominating set, and total dominating set are NP-hard [68]. Moreover, the connected variant of the maximization problem associated with the induced q-regular subgraph property is also NP-hard even when $q = 2$ [68], in this case, the problem asks for the computation of a longest induced cycle.

1.5 Parameterized Complexity

In this section, we give a formal definition of some classes of problems such as FPT, XP, and W[1]. We begin by recalling the definition of the big O notation that we use to describe the running times of our algorithm.

Definition 1.5. *Let $f, g : \mathbb{R} \to \mathbb{R}$ be two functions. We say that $g(n) \in O(f(n))$ if there exists $c \in \mathbb{N}$ and $N \in \mathbb{R}$ such that for all $n > N$, we have $g(n) \leq c \cdot f(n)$. Moreover, we say that $f(n) \in \Omega(g(n))$ if $g(n) \in O(f(n))$. Finally, we say that $f(n) \in \Theta(g(n))$ if $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$.*

The following is a formal definition of what is a parameterized decision problem.

Definition 1.6. *A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{R}$, where Σ is a fixed, finite alphabet. For an instance $(x,k) \in \Sigma^* \times \mathbb{R}$, k is called the parameter.*

As explained in the introduction, one way to describe the easy instances of an NP-hard problem is to find parameterization of this problem for which the problem is FPT.

Definition 1.7 (FPT)**.** *A parameterized problem $L \subseteq \Sigma^* \times \mathbb{R}$ is called* fixed-parameter tractable *(FPT) if there exists an algorithm $\mathcal{A}$ (called FPT algorithm), a computable function $f : \mathbb{R} \to \mathbb{R}$, and a constant c such that, given $(x,k) \in \Sigma^* \times \mathbb{R}$, the algorithm $\mathcal{A}$ correctly decides whether $(x,k) \in L$ in time bounded by $f(k) \cdot |(x,k)|^c$. We denote by FPT the complexity class containing all fixed-parameter tractable problems.*

Table 1.1 – Examples of (co)-(σ, ρ)-dominating sets and some connected, acyclic and AC variants, with $\{\geq d\} := \mathbb{N} \setminus \{0, \ldots, d-1\}$. Columns MAX and MIN show the complexity of the maximization and minimization problems associated with these subset properties, with P and NP-h denoting, respectively, Polytime and NP-hard.

σ	ρ	Variant	Standard name	MAX	MIN
$\{0\}$	$\mathbb{N}$		Independent set	NP-h	P
$\{0\}$	$\mathbb{N}$	*Co*	Vertex cover	P	NP-h
$\mathbb{N}$	$\mathbb{N}^+$		Dominating set	P	NP-h
$\mathbb{N}$	$\{1\}$		Perfect dominating set	P	NP-h
$\mathbb{N}^+$	$\mathbb{N}^+$		Total dominating set	P	NP-h
$\mathbb{N}$	$\{\geq d\}$		d-dominating set	P	NP-h
$\{q\}$	$\mathbb{N}$		Induced q-regular subgraph	NP-h	P
$\{1\}$	$\mathbb{N}$		Induced matching	NP-h	P
$\mathbb{N}$	$\mathbb{N}$	*Acyclic*	Induced forest	NP-h	P
$\mathbb{N}$	$\mathbb{N}$	*AC*	Induced tree	NP-h	P
$\{1,2\}$	$\mathbb{N}$	*Acyclic*	Induced linear forest	NP-h	P
$\{1,2\}$	$\mathbb{N}$	*AC*	Induced path	NP-h	P

Example 1.8. *The most famous example of an* FPT *problem is certainly the* k-Vertex Cover *problem parameterized by the size of the solution, whose definition is the following.*

k-Vertex Cover

Input: *A graph* G *and* $k \in \mathbb{N}$.
Parameter: k.
Output: *Does* G *admit a vertex cover of size* k*?*

Algorithm 1 presents the pseudo-code of the famous $O(2^k \cdot n)$ *time algorithm for* k-Vertex Cover.

Algorithm 1: $\mathcal{A}(G, k)$

```
Data: A graph G and k ∈ ℕ.
Result: Answer whether G admits a vertex cover of size at most k.
if E(G) = ∅ then
    return yes;
else if k = 0 then
    return no;
else
    take an edge uv ∈ E(G);
    return A(G − u, k − 1) or A(G − v, k − 1);
```

Definition 1.9 (XP)**.** *A parameterized problem* $L \subseteq \Sigma^* \times \mathbb{R}$ *is called* slice-wise polynomial *(XP) if there exists an algorithm* $\mathcal{A}$ *(called XP algorithm), two computable functions* $f : \mathbb{R} \to \mathbb{R}$ *and* $g : \mathbb{R} \to \mathbb{R}$ *such that, given* $(x, k) \in \Sigma^* \times \mathbb{R}$*, the algorithm* $\mathcal{A}$ *correctly decides whether*

$(x,k) \in L$ *in time bounded by* $f(k) \cdot |(x,k)|^{g(k)}$. *We denote by XP the complexity class containing all slice-wise polynomial problems.*

Example 1.10. *By definition, any parameterized problem which is* FPT *is also* XP. *The converse is not true, the following two problems are in* XP *but they are unlikely to be in* FPT.

k-Clique
Input: *A graph* G *and* $k \in \mathbb{N}^+$.
Parameter: k.
Output: *Does* G *admit a clique of size* k*?*

k-Dominating Set
Input: *A graph* G *and* $k \in \mathbb{N}$.
Parameter: k.
Output: *Does* G *admit a dominating-set of size* k*?*

Given an n*-vertex graph, we can solve both problems in time* $O(n^{k+1})$ *with a naive algorithm that checks for each subset* X *of vertices of size* k*, if* X *is a solution. As we will see in the following,* k-Clique *and* k-Dominating Set *are two typical* XP *problems that are unlikely to be* FPT *under a reasonable complexity assumption.*

The analog of polynomial reduction in parameterized complexity is called parameterized reduction whose definition is the following.

Definition 1.11 (Parameterized reduction)**.** *Let two parameterized problems* $A, B \subseteq \Sigma^* \times \mathbb{R}$. *A parameterized reduction from* A *to* B *is an algorithm* $\mathcal{A}$ *that, given an instance* (x,k) *of* A*, outputs an instance* (x',k') *of* B *such that*

- $(x,k) \in A$ *if and only if* $(x',k') \in B$,
- $k' \leq g(k)$ *for some computable function* g, *and*
- *the running time is* $f(k) \cdot |x|^{O(1)}$ *for some computable function* f.

As intended, if there exists a parameterized reduction from a problem A to a problem B, and B is FPT, then A is FPT.

The W-hierarchy is a set of complexity classes introduced by Downey and Fellows [43] in order to capture the hardness of problems which are unlikely to be FPT. In this thesis, we just need to know from this hierarchy the class of problems called W[1]-hard. We will not give a formal definition of this class based on Boolean circuits, instead, we give the following equivalent definition.

Definition 1.12. *A parameterized problem* A *is* W[1]*-hard if there exists a parameterized reduction from* k-Clique *to* A.

Analogously to $\mathsf{P} \neq \mathsf{NP}$, it is conjectured that $\mathsf{FPT} \neq \mathsf{W}[1]$. These two conjectures are very similar (both are closely related to the Halting Problem [52]) and $\mathsf{FPT} \neq \mathsf{W}[1]$ is nearly as reasonable as $\mathsf{P} \neq \mathsf{NP}$. More arguments in favor of the conjecture $\mathsf{FPT} \neq \mathsf{W}[1]$ can be found in [52].

It is worth noticing that $\mathsf{FPT} \neq \mathsf{W}[1]$ implies $\mathsf{P} \neq \mathsf{NP}$, because every polynomial time algorithm is, by definition, an FPT algorithm.

As suggested by the definition, k-Clique is a W[1]-hard problem. This problem and some variants such as k-Multicolored Clique are the starting point of many W[1]-hardness proof.

The complexity classes we have introduced so far form the following chain of inclusion:

$$\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{XP}.$$

The following class is analog to NP-hard.

Definition 1.13. *A parameterized problem is para* NP*-hard if it is* NP*-hard for some fixed constant value of its parameter.*

It is worth mentioning that, unless $\mathsf{P} = \mathsf{NP}$, any parameterized problem that is para NP-hard cannot be in XP and consequently it cannot be in FPT either.

Example 1.14. *A famous example of a para* NP*-hard problem is* q-Coloring *parameterized by the number of colors whose definition is the following.*

q-Coloring
Input: *A graph G and $q \in \mathbb{N}$.* ***Parameter:*** *q.* ***Output:*** *Does G admit a q-coloring?*

It is well-known that this problem is NP*-complete even when $q = 3$.*

1.6 Exponential Time Hypothesis

As explained in the previous section, the conjecture $\mathsf{FPT} \neq \mathsf{W}[1]$ can be used to prove that a parameterized problem is unlikely to admit an FPT algorithm. Similarly, the conjecture $\mathsf{P} \neq \mathsf{NP}$ is used to prove that a problem is unlikely to admit a polynomial time algorithm. However, these two conjectures do not give any hint about how fast we can solve a problem. To get more precise lower bounds, we need a stronger assumption. The Exponential Time Hypothesis (ETH) is a conjecture stating that, roughly speaking, 3-SAT has no algorithm subexponential in the number of variables. Introduced in 2001 by Impagliazzo and Paturi [82], ETH has been used to prove many lower-bounds for classical problems, but also FPT and XP problems. Before giving its formal definition, we recall the definition of the small o notation.

Definition 1.15. *Let $f, g : \mathbb{R} \to \mathbb{R}$ be two functions. We say that $f(n) \in o(g(n))$ if for all $k > 0$, there exists $N \in \mathbb{R}$ such that $k \cdot f(n) \leq g(n)$ for all $n \geq N$.*

Example 1.16. *For example, we have* $\log(n) \in o(\sqrt{n})$, $\sqrt{n} \in o(n)$, *and* $2^{\sqrt{n}} \in 2^{o(n)}$.

We recall the definition of the q-SAT problem.

q-SAT
Input: A CNF formula φ with n variables and m clauses where every clause contains at most q literals. **Output:** Does there exist a satisfying assignment for φ?

For $q \geq 3$, let δ_q be the infimum of the set of constants c for which there exists a deterministic algorithm solving q-SAT in time $O(2^{cn} \cdot (n+m)^{O(1)})$. The *Exponential-Time Hypothesis* is then defined as follows.

Conjecture 1.17 (Exponential Time Hypothesis, ETH)**.**

$$\delta_3 > 0.$$

Example 1.18. *The Exponential Time Hypothesis implies the following.*

- k-Clique *cannot be solved in time* $f(k) \cdot n^{o(k)}$ *with* k *the size of the solution. Consequently,* ETH *implies* FPT $\neq$ W[1].
- Vertex Cover *cannot be solved in time* $2^{o(n)} \cdot n^{O(1)}$*. This implies in particular that* k-Vertex Cover *cannot be solved in time* $2^{o(k)} \cdot n^{O(1)}$ *with* k *the size of the solution because* $k \leq n$*.*

It is worth mentioning a stronger variant of ETH called Strong Exponential Time Hypothesis (SETH), whose definition is the following.

Conjecture 1.19 (Strong Exponential Time Hypothesis, SETH)**.**

$$\lim_{q \to +\infty} \delta_q = 1.$$

Intuitively, SETH states that, for any $\epsilon > 0$, there are no $(2-\epsilon)^n \cdot (n+m)^{O(1)}$ time algorithms solving CNF-SAT. The Strong Exponential Time Hypothesis implies ETH and gives even more refined lower bounds. For example, unless SETH fails, there do not exist $\epsilon > 0$ and an algorithm solving k-Dominating Set in time $O(n^{k-\epsilon})$ with n the number of vertices of the input graph [37, Theorem 14.42].

While ETH is generally considered a plausible complexity assumption, it is not the case for SETH which is regarded by many as a quite doubtful working hypothesis that can be refuted any time [37]. It is worth noticing that, assuming SETH, some lower bounds have been proved for problems in P [1, 4, 23, 134].

Chapter 2

Overview of width measures

This chapter gives the definitions and the characteristics of the width measures we studied in this thesis. In Section 2.1, we define the notion of rooted layout and the parameters obtained from this notion. In Section 2.2, we define clique-width and we give some of its properties. We give an overview of the knowledge and the open problems concerning:

- the relations between the value of these width measures in Section 2.3,
- the relations between these width measures and some well-known graph classes in Section 2.4,
- the computations of these width measures in Section 2.5,
- how these width measures tame NP-hard problems in Section 2.6.

2.1 Rooted layout of a graph

In [127], Robertson and Seymour define the concept of *branch-decomposition* and *branch-width* to define a variant of tree-width. These concepts allow to define a width measure from a symmetric set function f called the *branch-width of* f. By now, these two notions are standard and they were used to define many width measures on graphs and matroids, see [70, 115].

In this thesis, we use the rooted variant of branch-decomposition called *rooted layout* of a graph (or *rooted decomposition tree*) whose definition follows. Contrary to branch-decomposition, rooted layout allows to define width measure from a set function that is not symmetric. Rooted layout is also more practical than branch-decomposition when designing algorithms.

Definition 2.1 (Rooted layout of a graph)**.** *Let G be a graph. A* rooted layout *of G is a pair $\mathcal{L} = (T, \delta)$ of a rooted binary tree T and a bijective function δ between $V(G)$ and the leaves of T. If T is a rooted caterpillar, then we said that (T, δ) is a* linear layout *of G.*

Each node u of a rooted layout (T, δ) of a graph G is associated with a set of vertices V_u. The following definition explains how we obtain this vertex set.

Definition 2.2. *For each node x of T, let L_x be the set of all the leaves l of T such that the path from the root of T to l contains x. We denote by $V_x^{\mathcal{L}}$ the set of vertices that are in bijection with L_x, i.e., $V_x^{\mathcal{L}} := \delta^{-1}(L_x)$. When $\mathcal{L}$ is clear from the context, we may remove $\mathcal{L}$ from the superscript.*

See Figure 2.1 for an example of a layout of a graph and a vertex set associated with a node of this layout. An example of a linear layout is presented in Figure 2.2.

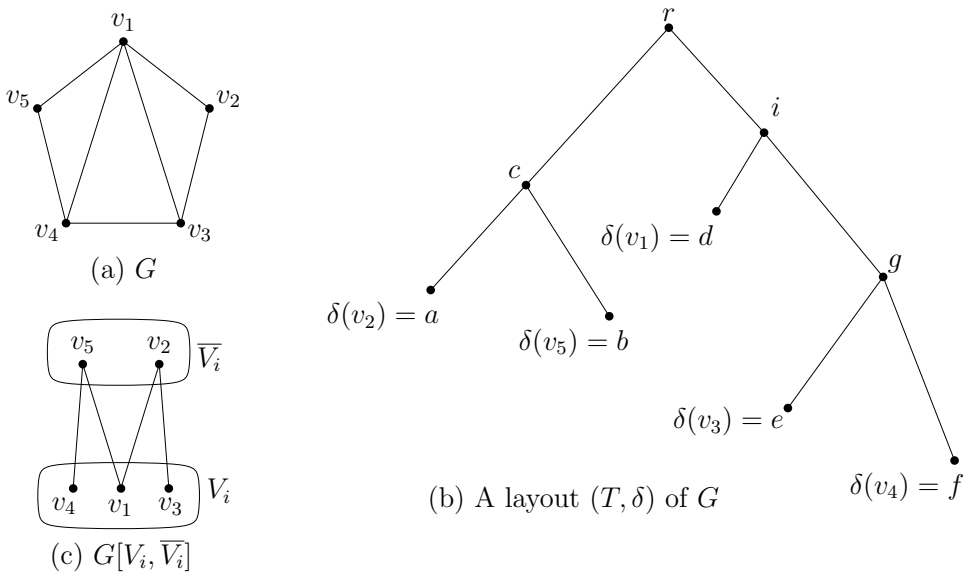


Figure 2.1 – Subfigure (a) shows a graph G. Subfigure (b) shows a layout (T, δ) of G with r as root of T. Observe that the vertex set associated with the node i is $V_i := \{v_1, v_3, v_4\}$. Subfigure (c) shows the bipartite graph $G[V_i, \overline{V_i}]$.

In order to define a width measure from the notion of rooted layout, all we need is a set function $\mathsf{f} : 2^{V(G)} \to \mathbb{R}$. The function f is intended to measure the simplicity of the graph $G[A, \overline{A}]$ through the value of $\mathsf{f}(A)$. The following definition shows how we obtain a width measure from a set function f, we call the resulting parameter f-*width*. All the width measures dealt with in this thesis are the f-width of some set function f.

Definition 2.3 (f-width)**.** *Let G be a graph. Given a set function $\mathsf{f} : 2^{V(G)} \to \mathbb{R}$ and a rooted layout (T, δ), the f-width of (T, δ), denoted by $\mathsf{f}(T, \delta)$, is $\max\{\mathsf{f}(V_x) : x \in V(T)\}$. Finally, the f-width of G, denoted by $\mathsf{f}(G)$, is the minimum f-width over all rooted layouts of G.*

The linear version of a width measure is obtained from the notion of linear layout.

Definition 2.4 (Linear f-width)**.** *Let G be a graph. Given a set function $\mathsf{f} : 2^{V(G)} \to \mathbb{R}$, the linear f-width of G is the minimum f-width over all linear layouts (T, δ) of G.*

The rest of this section is dedicated to the definition of all the parameters we deal with in this thesis that are the f-width of some set function.

Module-width

Introduced in [125], *module-width* is a width measure closely related to clique-width (see Theorem 2.21).

Definition 2.5 (Module-width)**.** *The module-width of a graph G is the mw-width of G where $\mathsf{mw}(A)$ is the cardinal of $\{N(v) \cap \overline{A} : v \in A\}$ for all $A \subseteq V(G)$.*

One also observes that $\mathsf{mw}(A)$ is the number of different rows in $M_{A,\overline{A}}$. Moreover, mw is not a symmetric function, i.e., $\mathsf{mw}(A)$ is not necessarily equal to $\mathsf{mw}(\overline{A})$ (see the following example).

Example 2.6. *Notice that for the rooted layout (T, δ) of the graph G presented in Figure 2.1, we have $\mathsf{mw}(\overline{V_i}) = 2 \neq \mathsf{mw}(V_i) = \mathsf{mw}(T, \delta) = 3$. Moreover, it is easy to check that $\mathsf{mw}(G) = 3$.*

Rank-width and $\mathbb{Q}$-rank-width

Rank-width and $\mathbb{Q}$-rank-width were introduced, respectively, in [115, 116] and [120].

Definition 2.7 (Rank-width)**.** *The rank-width of a graph G corresponds to the* rw*-width of G where $\mathsf{rw}(A)$ is the rank over $GF(2)$ of the matrix $M_{A,\overline{A}}$ for all $A \subseteq V(G)$.*

Definition 2.8 ($\mathbb{Q}$-rank-width)**.** *The $\mathbb{Q}$-rank-width of a graph G corresponds to the $\mathsf{rw}_{\mathbb{Q}}$-width of G where $\mathsf{rw}_{\mathbb{Q}}(A)$ is the rank over $\mathbb{Q}$ of the matrix $M_{A,\overline{A}}$ for all $A \subseteq V(G)$.*

Example 2.9. *Observe that for the rooted layout (T, δ) of the graph G presented in Figure 2.1, we have $\mathsf{rw}(T, \delta) = \mathsf{rw}_{\mathbb{Q}}(T, \delta) = 2$. This is optimal because G is not a distance hereditary graph and Oum [115] proved that a graph has rank-width 1 if and only if it is a distance hereditary graph. The same statement holds if we consider $\mathbb{Q}$-rank-width instead of rank-width because the proof does not depend on the $GF(2)$ field.*

Boolean-width and d-neighbor-width

Boolean-width and d-neighbor-width are both defined from the notion of d-neighbor equivalence introduced in [18] (see Definition 1.1) Boolean-width was introduced in [17]. The d-neighbor-width was first used in [5, 18] as a with measure on layouts called *number of d-neighborhood*. Golovach et al. [73] used the term d-neighbor-width to describe a parameter on two-colored graphs that is closely related to the one we define in this section (both notions collapse when every vertex is two-colored).

Definition 2.10 (Boolean-width)**.** *For every graph G, the* boolean-width *corresponds to the* boolw*-width of G where $\mathsf{boolw} : 2^{V(G)} \to \mathbb{R}$ is the symmetric function such that $\mathsf{boolw}(A) = \log_2(\mathsf{nec}_1(A))$.*

Observe that boolw is a symmetric function because nec_1 is also a symmetric function (see Observation 1.3).

Definition 2.11 (d-neighbor-width)**.** *For every graph G and $d \in \mathbb{N}^+$, the* d-neighbor-width *of G corresponds to the $\mathsf{s\text{-}nec}_d$-width of G where $\mathsf{s\text{-}nec}_d : 2^{V(G)} \to \mathbb{N}$ is the symmetric function such that $\mathsf{s\text{-}nec}_d(A) = \mathsf{max}(\mathsf{nec}_d(A), \mathsf{nec}_d(\overline{A}))$.*

Example 2.12. *Notice that for the rooted layout (T, δ) of the graph G presented in Figure 2.1, we have $\mathsf{s\text{-}nec}_1(T, \delta) = \mathsf{nec}_1(V_i) = 4$. since $\{N(X) \cap \overline{V_i} : X \subseteq V_i\} = \{\varnothing, \{v_2\}, \{v_5\}, \{v_2, v_5\}\}$. Consequently, we have $\mathsf{boolw}(T, \delta) = 2$. However, this is not optimal because the linear layout (T', δ') presented in Figure 2.2 has boolean-width $\log_2(3)$.*

Damiand et al. [39] have proved that a graph has boolean-width 1 if and only if it is a distance hereditary graph. As G is not a distance hereditary graph, we conclude that the (linear) boolean-width of G is $\log_2(3)$.

Maximum Induced Matching-Width

The *maximum induced matching-width* (*mim-width* for short) was introduced in [132].

Definition 2.13. *The mim-width of a graph G is the* mim*-width of G where $\mathsf{mim}(A)$ is the size of a maximum induced matching of the graph $G[A, \overline{A}]$ for all $A \subseteq V(G)$.*

It is worth noticing that if $\mathsf{mim}(A) = k$, then, for every $X \subseteq A$, there exists $W \subseteq X$ such that $|W| \leq k$ and $N(X) \cap \overline{A} = N(W) \cap \overline{A}$. This property is very useful to design algorithms parameterized by mim-width.

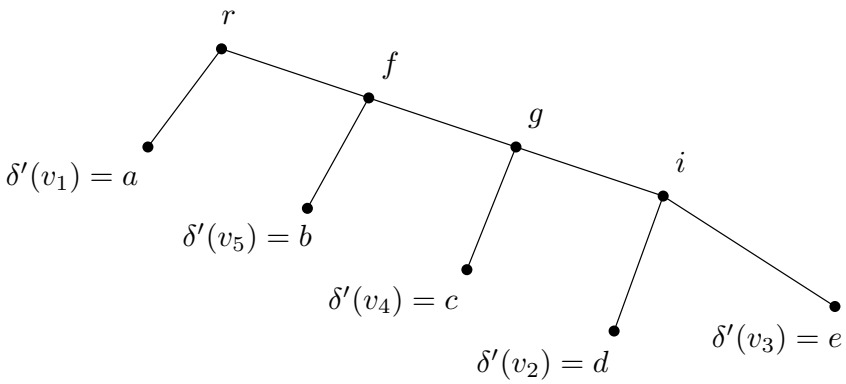


Figure 2.2 – Linear layout (T', δ') of the graph G presented in Figure 2.1 with $\mathsf{boolw}(T', \delta') = \mathsf{boolw}(V_g) = \log_2(3)$ and $\mathsf{mim}(T', \delta') = 1$.

Example 2.14. *Observe that for the rooted layout (T, δ) presented in Figure 2.1, we have $\mathsf{mim}(T, \delta) = \mathsf{mim}(V_i) = 2$ because the edges v_4v_5 and v_3v_2 form an induced matching in the graph $G[V_i, \overline{V_i}]$. But this is not a rooted layout of optimal mim-width as the linear layout presented in Figure 2.2 has mim-width 1.*

2.2 Clique-witdh

In this section, we define *clique-width* and some related concepts. Clique-width is a graph parameter that originally emerges from the theory of graph grammars [32] and the terminology was first introduced by Courcelle and Olariu [35] (see also the book [31]). The definition of clique-width is algorithmic and uses the following definition of labeled graphs.

Definition 2.15 (k-labeled graph)**.** *A* k-labeled graph *is a pair (G, lab_G) with G a graph and lab_G a function from $V(G)$ to $[k]$, called the* labeling function*; each set $lab_G^{-1}(i)$ is called a* label class*. The vertices in $lab_G^{-1}(i)$ are called* i-vertices *of G or just i-vertices if the underlying graph is clear from the context.*

The graph decomposition associated with clique-width is called a k-expression.

Definition 2.16 (k-expression)**.** *Let $k \in \mathbb{N}^+$. We define the following four operations on k-labeled graphs:*

$i(x)$: *for $i \in [k]$, create a graph, with a single vertex x labeled i,*

$\rho_{i \to j}(G)$: *for a labeled graph G and distinct labels $i, j \in [k]$, relabel the vertices of G with label i into j,*

$\eta_{i,j}(G)$: *for a labeled graph G and distinct labels $i, j \in [k]$, add all the non-existing edges between vertices with label i and vertices with label j,*

$G \oplus H$: *take the disjoint union of two labeled graphs G and H, with*

$$\mathsf{lab}_{G \oplus H}(v) := \begin{cases} \mathsf{lab}_G(v) & \text{if } v \in V(G), \\ \mathsf{lab}_H(v) & \text{otherwise.} \end{cases}$$

A term ϕ built with the four operations defined above is well-formed *if either*

- $\phi = i(x)$ *for some* $i \in [k]$,
- $\phi = f(\phi')$ *with* $f \in \{\eta_{i,j}, \rho_{i\to j} : i,j \in [k] \wedge i \neq j\}$ *and* ϕ' *is a well-formed term, or*
- $\phi = \oplus(\phi', \phi^\star)$ *with* ϕ' *and* $\phi^\star$ *two well-formed terms.*

A clique-width k-expression*, or shortly a* k-expression*, is a finite well-formed term built with the four operations defined above. Each* k*-expression* ϕ *evaluates into a* k*-labeled graph* $(\mathsf{val}(\phi), \mathsf{lab}_{\mathsf{val}(\phi)})$.

Definition 2.17 (Clique-width)**.** *The* clique-width *of a graph* G*, denoted by* $\mathsf{cw}(G)$*, is the minimum* k *such that* G *is isomorphic to* $\mathsf{val}(\phi)$ *for some* k*-expression* ϕ*.*

For example, the cycle *abcdea* of length 5 can be constructed using the 3-expression represented as a tree-structure in Figure 2.3. It is not difficult to check that the clique-width of the cycle *abcdea* is 3.

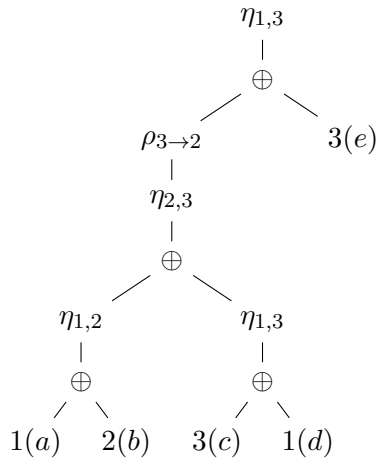


Figure 2.3 – An irredundant 3-expression of C_5.

We can assume without loss of generality that any k-expression defining a graph G uses $O(n)$ disjoint union operations and $O(nk^2)$ unary operations [35].

It is worth noticing, from the recursive definition of k-expressions, that one can compute in time linear in $|\phi|$ the labeling function $\mathsf{lab}_{\mathsf{val}(\phi)}$ of $\mathsf{val}(\phi)$, and hence we will always assume that it is given.

In our algorithm, for a k-expression ϕ, we use the notions of *subexpressions* of ϕ and the concept of k-labeled graphs arising in ϕ whose definitions are the following.

Definition 2.18 (Subexpressions and k-labeled graphs arising in a k-expression)**.** *The set of subexpressions of a* k*-expression* ϕ*, denoted by* $\mathsf{Sub}(\phi)$*, is defined by the following induction:*

$$\mathsf{Sub}(\phi) := \begin{cases} \{\phi\} \text{ if } \phi := i(x) \text{ with } i \in [k], \\ \{\phi\} \cup \mathsf{Sub}(\phi') \cup \mathsf{Sub}(\phi^\star) \text{ if } \phi = \phi' \oplus \ \phi^\star, \\ \{\phi\} \cup \mathsf{Sub}(\phi') \text{ if } \phi = f(\phi') \text{ with } f \in \{\rho_{i\to j}, \eta_{i,j} : i,j \in [k]\}. \end{cases}$$

We say that a k*-labeled graph* (H, lab_H) *arises in a* k*-expression* ϕ *if* $H = \mathsf{val}(\phi')$ *and* $\mathsf{lab}_H = \mathsf{lab}_{\mathsf{val}(\phi')}$ *for some* $\phi' \in \mathsf{Sub}(\phi)$*.*

Let G be a graph, ϕ be a k-expression of G and $\eta_{i,j}(\phi')$ be a subexpression of ϕ. Observe that any subexpression $\eta_{a,b}(\phi^\star)$ of ϕ' that adds edges between some vertices labeled i and j in $\mathsf{val}(\phi')$ are useless and can be "removed" from ϕ without changing $\mathsf{val}(\phi)$. When designing a dynamic programming algorithm on a k-expression, it may be desirable to remove these useless operations that can generate some technical issues as we would like to treat all the i-vertices of $\mathsf{val}(\phi')$ in the same way. But, if some are adjacent to some j-vertices in $\mathsf{val}(\phi')$, then when we compute the partial solutions associated with $\eta_{i,j}(\phi')$, we may need to identify the two cases, which can prevent the algorithm from being FPT as we do not have a control on these edges. In order to avoid these difficulties, our algorithms use the following definition of *irredundant k-expressions*.

Definition 2.19 (Irredundant k-expressions [35])**.** *A k-expression ϕ is called irredundant if, for every subexpression $\eta_{i,j}(\phi')$ of ϕ, there are no edges in $\mathsf{val}(\phi')$ between the vertices labeled i and the vertices labeled j.*

Courcelle and Olariu [35] proved that given a clique-width k-expression, it can be transformed into an irredundant k-expression in linear time.

The following lemma presents some useful properties of irredundant k-expressions.

Lemma 2.20. *Let H be a k-labeled graph arising in an irredundant k-expression ϕ of a graph G. For all $u, v \in V(H)$ with $\mathsf{lab}_H(u) = i$ and $\mathsf{lab}_H(v) = j$, we have the following.*

1. *If $i = j$, then $N_G(u) \setminus V(H) = N_G(v) \setminus V(H)$.*
2. *If $uv \in E(G) \setminus E(H)$, then $i \neq j$ and, for all $x, y \in V(H)$ with $\mathsf{lab}_H(x) = i$ and $\mathsf{lab}_H(y) = j$, we have $xy \in E(G) \setminus E(H)$.*

Proof. (1) Assume that $i = j$. Let ϕ' be the subexpression of ϕ such that $H = \mathsf{val}(\phi')$. As u and v have the same label in H, in every subexpression of ϕ having ϕ' as a subexpression, u and v have the same label. Since edges are added only through the operation $\eta_{a,b}$, we conclude that $N_G(u) \cap (V(G) \setminus V(H)) = N_G(v) \cap (V(G) \setminus V(H))$.

(2) Assume that $uv \in E(G) \setminus E(H)$. Then, we have $i \neq j$ because the operation $\eta_{a,b}$ add edges only between vertices with distinct labels. Let ϕ' be the minimal (size wise) subexpression of ϕ such that $uv \in E(\mathsf{val}(\phi'))$. It follows that $\phi' = \eta_{a,b}(\phi^\star)$, with $\phi^\star \in \mathsf{Sub}(\phi)$, $\mathsf{lab}_{\mathsf{val}(\phi')}(u) = a$ and $\mathsf{lab}_{\mathsf{val}(\phi')}(v) = b$. Let $D := \mathsf{val}(\phi^\star)$. Observe that we have $V(H) \subseteq V(D)$ and $E(H) \subseteq E(D)$. Moreover, all vertices labeled i in H are labeled a in D and those labeled j in H are labeled b in D. Since ϕ is irredundant, there are no edges in D between a vertex labeled a and one labeled b. Thus, for all vertices $x \in \mathsf{lab}_H^{-1}(i)$ and $y \in \mathsf{lab}_H^{-1}(j)$, we have $xy \notin E(H)$ and $xy \in E(G)$. □

The following theorem shows that clique-width and module-width are linearly equivalent, i.e., for every graph G, we have $\mathsf{cw}(G) \in \Theta(\mathsf{mw}(G))$.

Theorem 2.21 ([125, Theorem 6.6])**.** *For every n-vertex graph G, $\mathsf{mw}(G) \leq \mathsf{cw}(G) \leq 2\mathsf{mw}(G)$. One can moreover translate, in time at most $O(n^2)$, a given decomposition into the other one with width at most the given bounds.*

2.3 Hierarchy of width measures

In this section, we compare the properties of the following seven width measures: tree-width, clique-width, rank-width, $\mathbb{Q}$-rank-width, Boolean-width, d-neighbor-width, and mim-width. For a graph G, we denote by $\mathsf{tw}(G)$ the tree-width of G, see [127] for a definition. In the following,

for each parameter f, we give some upper bounds on the value of f over graphs or over vertex sets in terms of the other parameters. We conclude by representing the hierarchy formed by these seven width measures with two diagrams.

When we compare the f-width and g-width of two set functions $\mathsf{f}, \mathsf{g} : 2^{V(G)} \to \mathbb{R}$, it is more convenient and easier to compare the values of these functions over vertex sets. Notice that an upper bound over vertex sets implies, by Definition 2.3 of f-width, the same upper bound over rooted layouts and over graphs. The following lemma formalizes this statement.

Lemma 2.22. *Let G be a graph and $\mathsf{f}, \mathsf{g} : 2^{V(G)} \to \mathbb{R}$ two set functions. If, for all $A \subseteq V(G)$, we have $\mathsf{f}(A) \leq h(\mathsf{g}(A))$ for some function h, then we have*

- *$\mathsf{f}(T, \delta) \leq h(\mathsf{g}(T, \delta))$ for all rooted layouts (T, δ) of G, and*
- *$\mathsf{f}(G) \leq h(\mathsf{g}(G))$.*

Before giving the known upper bounds between the different width measures, let us explain the absence of upper bounds between some width measures. As discussed in the introduction, the tree-width of a graph class can be arbitrarily bigger than the f-width of this graph, for all $\mathsf{f} \in \{\mathsf{cw}, \mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{boolw}, \mathsf{mim}, \mathsf{nec}_d : d \in \mathbb{N}^+\}$. For instance, the tree-width of a complete graph G of size n is $n-1$ while we have $\mathsf{cw}(G) \leq 2$, $\mathsf{f}(G) = 1$, for all $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{boolw}, \mathsf{mim}\}$, and $\mathsf{nec}_d(G) = \mathsf{min}(d, n)$ for every $d \in \mathbb{N}^+$. Moreover, the f-width of a graph class, for $\mathsf{f} \in \{\mathsf{cw}, \mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{boolw}, \mathsf{nec}_d : d \in \mathbb{N}^+\}$, can be arbitrarily larger than the mim-width of this graph class. For example, the (linear) mim-width of any interval graph is 1 [5] while there exist interval graphs G on n vertices where $\mathsf{cw}(G), \mathsf{rw}(G) \in \Theta(\sqrt{n})$, $\mathsf{tw}(G) = n-1$, and $\mathsf{boolw}(G) \leq \log_2(n)$ [132, Theorem 4.3.3].

2.3.1 Upper bounds on clique-width

The following theorem shows how clique-width or module-width are upper bounded by the other width measures.

Theorem 2.23 ([27, 132])**.** *For any graph G and every $A \subseteq V(G)$, we have:*

(a) $\mathsf{cw}(G) \leq \frac{3}{2} \cdot 2^{\mathsf{tw}(G)}$,

(b) $\mathsf{mw}(A) \leq 2^{\mathsf{rw}(A)} \leq 2^{\mathsf{rw}_{\mathbb{Q}}(A)}$,

(c) $\mathsf{mw}(A) \leq 2^{\mathsf{boolw}(A)-1}$,

(d) $\mathsf{mw}(A) \leq \mathsf{nec}_1(A)$.

Proof. Upper bound (a) was proved in [27]. Upper bound (b) is due to the fact that any binary matrix M of rank k (over $GF(2)$) has at most 2^k different rows and because $\mathsf{rw}(A) \leq \mathsf{rw}_{\mathbb{Q}}(A)$ for all $A \subseteq V(G)$. Upper bound (c) was proved in the proof of [132, Theorem 4.2.9]. The last upper bound follows from the definition of mw and nec_1 and the fact that, for every $A \subseteq V(G)$, we have $\{N(x) \cap \overline{A} : x \in A\} \subseteq \{N(X) \cap \overline{A} : X \subseteq A\}$.

□

Upper bounds (a)-(c) of Theorem 2.23 is almost tight due to the following theorem from [27] and the fact that $\mathsf{rw}(G) \leq \mathsf{rw}_{\mathbb{Q}}(G) \leq \mathsf{tw}(G) + 1$ and $\mathsf{boolw}(G) \leq \mathsf{tw}(G) + 1$ (see the following upper bounds).

Theorem 2.24 ([27])**.** *For any $k \in \mathbb{N}$, there is a graph G such that $\mathsf{tw}(G) = k$ and $2^{\lfloor k/2 \rfloor - 1} \leq \mathsf{cw}(G)$.*

Upper bound (d) of Theorem 2.23 is almost tight. Indeed, for any $k \in \mathbb{N}$, let $G[A, B]$ be the bipartite graph with $A = \{a_1, \ldots, a_k\}$, $B = \{b_1, \ldots, b_k\}$ and $E(G[A, B]) = \{a_i b_j : i, j \in [k] \wedge i \neq j\}$. We have $\mathsf{mw}(A) = k$ and $\mathsf{nec}_1(A) = |\{N(X) \cap B : X \subseteq A\}| = |\{\varnothing, N(a_1), \ldots, N(a_k), B\}| = k + 2$.

2.3.2 Upper bounds on rank-width and $\mathbb{Q}$-rank-width

Theorem 2.25 ([17, 117, 120]). *For any graph G and for every $A \subseteq V(G)$, we have:*

(a) $\mathsf{rw}(G) \leq \mathsf{tw}(G) + 1$,

(b) $\mathsf{rw}(A) \leq \mathsf{rw}_{\mathbb{Q}}(A) \leq \mathsf{mw}(A)$,

(c) $\mathsf{rw}(A) \leq 2^{\mathsf{boolw}(A)}$,

(d) $\mathsf{rw}(A) \leq \mathsf{nec}_1(A)$.

Proof. Upper bound (a) was proved in [117]. The first inequality of Upper bound (b) is implicitly proved in [120, Lemma 3.2] and is due to the fact that a set of 0-1 vectors linearly dependent over $\mathbb{Q}$ must also be linearly dependent over $GF(2)$. The second inequality of Upper bound (b) follows from the fact that the rank of a matrix over any field is at most its number of different rows. Upper bound (c) was proved in [17, Lemma 1]. The last upper bound is an implication of Upper bound (b) and Theorem 2.23(d). □

Theorem 2.26 ([120]). *For any graph G and for every $A \subseteq V(G)$, we have:*

(a) $\mathsf{rw}_{\mathbb{Q}}(G) \leq \mathsf{tw}(G) + 1$,

(b) $\mathsf{rw}_{\mathbb{Q}}(A) \leq \mathsf{mw}(A)$,

(c) $\mathsf{rw}_{\mathbb{Q}}(A) \leq 2^{\mathsf{rw}(A)}$,

(d) $\mathsf{rw}_{\mathbb{Q}}(A) \leq \mathsf{nec}_1(A) = 2^{\mathsf{boolw}(A)}$.

Proof. Upper bound (a) was proved in [120, Theorem 3.6]. Upper bound (b) was proved in [120, Lemma 3.2]. Upper bound (c) was proved in [120, Lemma 3.2] and follows from the fact that $\mathsf{rw}_{\mathbb{Q}}(A) \leq \mathsf{mw}(A)$. Upper bound (d) is an implication of Upper bound (b) and Theorem 2.25(c). □

Observe that some of the inequalities of Theorems 2.25 and 2.26 are almost tight owing to the following theorems and the fact that $\mathsf{rw}(G) \leq \mathsf{rw}_{\mathbb{Q}}(G) \leq \mathsf{cw}(G)$.

Theorem 2.27 ([76, 88]). *An $n \times n$ grid has tree-width n, rank-width $n - 1$ and clique-width $n + 1$.*

Theorem 2.28 ([17]). *For large enough integer k, there exists a graph G with $\mathsf{rw}(G) \geq k$ and $\mathsf{boolw}(G) \leq 2\log_2(\mathsf{rw}(G)) + 4$.*

2.3.3 Upper bounds on boolean-width

Theorem 2.29 ([5, 17, 120, 132]). *For any n-vertex graph G and for every $A \subseteq V(G)$, we have*

(a) $\mathsf{boolw}(G) \leq \mathsf{tw}(G) + 1$,

(b) $\mathsf{boolw}(A) \leq \mathsf{mw}(A)$,

(c) $\mathsf{boolw}(A) \leq \frac{1}{4}\mathsf{rw}(A)^2 + O(\mathsf{rw}(A))$,

(d) $\mathsf{boolw}(A) \leq \mathsf{rw}_{\mathbb{Q}}(A) \cdot \log_2(\mathsf{rw}_{\mathbb{Q}}(A) + 1)$,

(e) $\mathsf{boolw}(A) \leq \mathsf{mim}(A) \cdot \log_2(n)$,

(f) $\mathsf{boolw}(A) \leq \log_2(\mathsf{nec}_d(A))$ *for all* $d \in \mathbb{N}^+$.

Proof. Upper bound (a) was proved in [132, Theorem 4.2.8]. Upper bound (b) was proved in [132, Proof of Theorem 4.2.9]. Upper bound (c) was proved in [17, Lemma 1]. Upper bound (d) follows from a result of [120, Theorem 4.2] stating that $\mathsf{nec}_d(A) \leq (d \cdot \mathsf{rw}_{\mathbb{Q}}(A) + 1)^{\mathsf{rw}_{\mathbb{Q}}(A)}$ for all $d \in \mathbb{N}^+$ and $A \subseteq V(G)$. In particular, this result implies $\mathsf{nec}_1(A) \leq (\mathsf{rw}_{\mathbb{Q}}(A) + 1)^{\mathsf{rw}_{\mathbb{Q}}(A)}$. As $\mathsf{boolw}(A) = \log_2(\mathsf{nec}_1(A))$, we deduce that $\mathsf{boolw}(A) \leq \mathsf{rw}_{\mathbb{Q}}(A) \cdot \log_2(\mathsf{rw}_{\mathbb{Q}}(A) + 1)$. Upper bound (e) was proved in [5, Lemma 2]. The last upper bound follows from Definition 2.10 of boolean-width ($\mathsf{boolw}(A) = \log_2(\mathsf{nec}_1(A))$) and the fact that $\mathsf{nec}_1(A) \leq \mathsf{nec}_d(A)$ for all graphs G and every $A \subseteq V(G)$. □

Upper bounds (a) and (d) presented in Theorem 2.29 are tight due to Theorem 2.27.

The following theorem shows that Upper bound (c) of Theorem 2.29 is tight on vertex sets.

Theorem 2.30 ([17])**.** *For any* $k \in \mathbb{N}^+$*, there exists a graph* G *and* $A \subseteq V(G)$ *such that* $\mathsf{rw}(A) = k$ *and* $\mathsf{boolw}(A) \in \Theta(k^2)$.

However, we do not know if, for any $k \in \mathbb{N}^+$, there exists a graph G such that $\mathsf{rw}(G) = k$ and $\mathsf{boolw}(G) \in \Theta(k^2)$. This leads to the following open question.

Open Question 2.31. *Is the boolean-width of every graph subquadratic in its rank-width?*

2.3.4 Upper bounds on mim-width

The following lemma provides some upper bounds between mim-width and the other parameters. All of these upper bounds were proved in [132] but we give a proof for completeness.

Theorem 2.32 ([132])**.** *Let* G *be a graph. For every* $A \subseteq V(G)$, $\mathsf{mim}(A)$ *is upper bounded by* $\mathsf{mw}(A)$, $\mathsf{rw}(A), \mathsf{rw}_{\mathbb{Q}}(A)$ *and* $\mathsf{boolw}(A) = \log_2(\mathsf{nec}_1(A))$.

Proof. Let S be the vertex set of a maximum induced matching of the graph $G[A, \overline{A}]$. Observe that the restriction of the matrix $M_{A,\overline{A}}$ to rows and columns in S is the identity matrix. Hence, $\mathsf{mim}(A)$ is upper bounded both by $\mathsf{rw}(A)$ and $\mathsf{rw}_{\mathbb{Q}}(A)$. It is clear that every pair of subsets of $S \cap A$ have a different neighborhood in $\overline{A}$. Thus, we have $2^{\mathsf{mim}(A)} \leq \mathsf{nec}_1(A)$. We deduce that $\mathsf{mim}(A) \leq \mathsf{boolw}(A) = \log_2(\mathsf{nec}_1(A))$. □

2.3.5 Upper bounds on d-neighbor-width

Theorem 2.33 ([5, 120])**.** *Let* $d \in \mathbb{N}^+$*. For all graphs* G *and for every* $A \subseteq V(G)$*, we have*

(a) $\mathsf{nec}_d(A) \leq (d+1)^{\mathsf{mw}(A)}$,

(b) $\mathsf{nec}_d(A) \leq 2^{d \cdot \mathsf{rw}(A)^2}$,

(c) $\mathsf{nec}_d(A) \leq 2^{\mathsf{rw}_{\mathbb{Q}}(A) \cdot \log_2(d \cdot \mathsf{rw}_{\mathbb{Q}}(A)+1)}$,

(d) $\mathsf{nec}_d(A) \leq \mathsf{mw}(A)^{\mathsf{mim}(A)} \leq n^{d \cdot \mathsf{mim}(A)}$,

(e) $\mathsf{nec}_d(A) \leq 2^{d \cdot \mathsf{boolw}(A)^2} = \mathsf{nec}_1(A)^{d \cdot \log_2(\mathsf{nec}_1(A))}$.

Proof. Upper bound (b) follows directly from $\mathsf{nec}_d(A) \leq \mathsf{mw}(A)^{\mathsf{mim}(A)}$ [132, Lemma 5.2.3]. As $\mathsf{mim}(A) \leq \mathsf{rw}(A)$ by Theorem 2.32 and $\mathsf{mw}(A) \leq 2^{\mathsf{rw}(A)}$ [121], we deduce that $\mathsf{nec}_d(A) \leq 2^{d \cdot \mathsf{rw}(A)^2}$. Upper bound (c) was proved in [120, Theorem 4.2]. Upper bound (d) was proved in [5, Lemma 2]. Upper bound (e) follows from Lemma 1.4.

Upper bound (a) was proved in [132, Lemma 5.2.2] but we give an alternative proof. Let $d \in \mathbb{N}^+$. We want to prove that $\mathsf{nec}_d(A)$ and $\mathsf{nec}_d(\overline{A})$ are at most $(d+1)^{\mathsf{mw}(A)}$. First, notice that, for all vertices x and y in A, we have the following equivalences:

$$\{x\} \equiv_A^d \{y\} \iff \{x\} \equiv_A^1 \{y\} \iff x \text{ and } y \text{ have the same row in } M_{A,\overline{A}}.$$

We deduce that there are $\mathsf{mw}(A)$ equivalence classes $A_1, \ldots, A_{\mathsf{mw}(A)}$ of $\equiv_A^1$ over A. Let $X, Y \subseteq A$. By definition of $\equiv_A^d$ and the construction of $A_1, \ldots, A_{\mathsf{mw}(A)}$, if we have $\mathsf{min}(d, |X \cap A_i|) = \mathsf{min}(d, |Y \cap A_i|)$ for all $i \in \{1, \ldots, \mathsf{mw}(A)\}$, then $X \equiv_A^d Y$. We deduce that $\mathsf{nec}_d(A)$ is at most the cardinal of

$$\{(\mathsf{min}(d, |X \cap A_1|), \ldots, \mathsf{min}(d, |X \cap A_{\mathsf{mw}(A)}|)) : X \subseteq A)\}.$$

Hence, we deduce that $\mathsf{nec}_d(A) \leq (d+1)^{\mathsf{mw}(A)}$ because, for every $X \subseteq A$ and $i \in \{1, \ldots, \mathsf{mw}(A)\}$, we have $\mathsf{min}(d, |X \cap A_i|) \in \{0, \ldots, d\}$.

Let $Y, W \subseteq \overline{A}$. For every $i \in \{1, \ldots, \mathsf{mw}(A)\}$, we let $a_i \in A_i$. Observe that, for every $X \subseteq \overline{A}$, every $i \in \{1, \ldots, \mathsf{mw}(A)\}$, and all $v \in A_i$, we have $|N(a_i) \cap X| = |N(v) \cap X|$. Hence, from the definition of $\equiv_{\overline{A}}^d$, we deduce that if we have $\mathsf{max}(d, |N(a_i) \cap Y|) = \mathsf{max}(d, |N(a_i) \cap W|)$ for all $i \in \{1, \ldots, \mathsf{mw}(A)\}$, then $Y \equiv_{\overline{A}}^d W$. Consequently, we deduce that $\mathsf{nec}_d(\overline{A}) \leq (d+1)^{\mathsf{mw}(A)}$. □

2.3.6 Overview of the Hierarchy

Figure 2.4 represents the following relations $\leq_f$ and $\leq_{poly}$, where

- $\mathsf{a} \leq_f \mathsf{b}$ if there exists a function f such that, for all graphs G, we have $\mathsf{a}(G) \leq f(\mathsf{b}(G))$,
- $\mathsf{a} \leq_{poly} \mathsf{b}$ if there exists a polynomial P such that, for all graphs G, we have $\mathsf{a}(G) \leq P(\mathsf{b}(G))$.

Observe that these two relations are reflexive and transitive. The parameters which are equivalent for a relation are drawn inside the same node. For each pair (a, b) of parameters, there exists a directed path from a to b in the diagram associated with $\leq_f$ (resp. $\leq_{poly}$) if and only if $\mathsf{b} \leq_f \mathsf{a}$ (resp. $\mathsf{b} \leq_{poly} \mathsf{a}$).

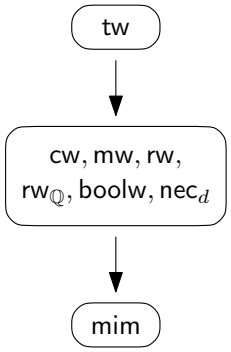
The correctness of the diagrams presented in Figure 2.4 follows from Theorems 2.23-2.33.

Observe that we can use the following lemma and the hierarchy of parameters formed by $\leq_f$ to extend some results concerning one parameter to the others.

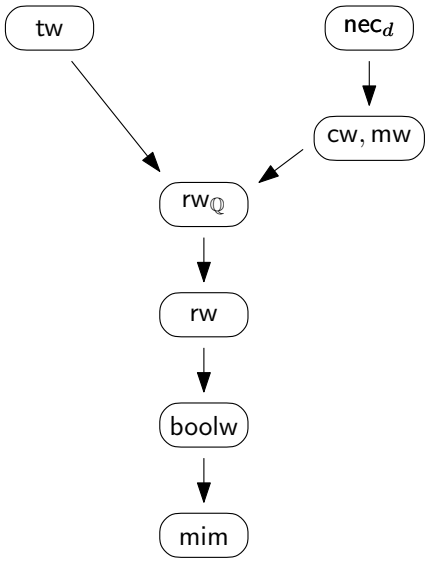
Lemma 2.34. *For any graph parameters* a, b*, if* $\mathsf{a} \leq_f \mathsf{b}$*, then, for any graph problem* Π*, we have the following:*

- *if* Π *is* FPT *parameterized by* a*, then* Π *is* FPT *parameterized by* b*,*
- *if* Π *is* XP *parameterized by* a*, then* Π *is* XP *parameterized by* b*,*
- *if* Π *is para* NP*-hard parameterized by* b*, then* Π *is also para* NP*-hard parameterized by* a*.*

Proof. Let a and b be two parameters such that $\mathsf{a} \leq_f \mathsf{b}$. Then, there exists a function f such that $\mathsf{a}(G) \leq f(\mathsf{b}(G))$ for all graphs G. Let Π be a graph problem. Suppose that Π is FPT when parameterized by a. Thus, there exists an FPT algorithm $\mathcal{A}$ that, given an n-vertex graph G,



(a) Diagram representing the relation $\leq_f$.



(b) Diagram representing the relation $\leq_{poly}$.

Figure 2.4 – Diagrams representing the relations $\leq_f$ and $\leq_{poly}$, with d a constant in $\mathbb{N}^+$.

solves Π in time $g(\mathsf{a}(G)) \cdot n^{O(1)}$. As $\mathsf{a}(G) \leq f(\mathsf{b}(G))$ for all graphs G, the running time of $\mathcal{A}$ is also $O(g(f(\mathsf{b}(G))) \cdot n^{O(1)})$. Hence, Π is also FPT when parameterized by b. Similarly, we can prove that if Π is XP parameterized by a, then Π is XP parameterized by b.

Now suppose that Π is a para NP-hard graph problem parameterized by b. Then, there exists $c \in \mathbb{R}$ such that Π is NP-hard on graphs with $\mathsf{b}(G) \leq c$. Therefore, Π is NP-hard on graphs with $\mathsf{a}(G) \leq f(c)$. Hence, Π is para NP-hard parameterized by a. □

2.4 Relation between width measures and graph classes

As explained in the introduction, many polynomial time algorithms for NP-hard problems on graph classes can be explained through the width measures we have defined in this chapter. In this section, we present some lower bounds on the value of these parameters on some well-known graph classes. For the definition of the graph classes introduced in this section we refer to www.graphclasses.org/.

It is easy to see that graphs of clique-width 1 are graphs with no edges. The graphs of clique-width at most 2 correspond to the cographs [35]. The following theorem from [26] shows that we can recognize the graphs of clique-width at most 3 in polynomial time.

Theorem 2.35 ([26])**.** *There exists an algorithm that, given a graph G with n vertices and m edges, decides whether* $\mathsf{cw}(G) \leq 3$.

The following theorem characterizes the graphs of f-width at most 1 for $\mathsf{f} \in \{\mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{boolw}\}$.

Theorem 2.36 ([115, 39])**.** *For every* $\mathsf{f} \in \{\mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{boolw}\}$ *and every graph G, we have* $\mathsf{f}(G) \leq 1$ *if and only if G is a distance hereditary graph.*

Proof. Oum [115] proved that a graph has rank-width at most 1 if and only if it is a distance hereditary graph. The same statement holds if we consider $\mathbb{Q}$-rank-width instead of rank-width because the proof does not depend on the $GF(2)$ field. Damiand et al. [39] proved that a graph has boolean-width at most 1 if and only if it is a distance hereditary graph. □

Belmonte and Vatshelle [5] proved that there are many well-known classes of graphs where mim-width is bounded by a constant and boolean-width by $O(\log(n))$. They also proved that we can compute in polynomial time a rooted layout of bounded mim-width for any graph that belongs to these classes of graphs. The following theorem lists these graph classes.

Theorem 2.37 ([5])**.** *The following graph classes have bounded mim-width: interval graphs, circular arc graphs, circular permutation and permutation graphs, convex graphs, circular k-Trapezoid graphs, k-polygon graphs, Dilworth-k graphs, and co-k-degenerate graphs for fixed k.*

Moreover, for any n-vertex graph G that belongs to these classes, we have $\mathsf{boolw}(G) \in O(log(n))$ *and we can compute in polynomial time a rooted layout of constant mim-width and of boolean-width* $O(\log(n))$ *(for circular k-Trapezoid graphs, an intersection model is needed in the input).*

In [5], the authors asked whether Theorem 2.37 can be extended to the graph classes called *strongly chordal* graphs and *tolerance* graphs. Later, Mengel [108] proved that mim-width is unbounded in several graph classes including strongly chordal graphs. However, the following question remains open.

Open Question 2.38. *Is there a constant* $c \in \mathbb{N}$ *such that, for every tolerance graph G, we have* $\mathsf{mim}(G) \leq c$*?*

Recently, Fomin, Golovach, and Raymond [61] proved the following theorem that extends Theorem 2.37 to the class of *H-graphs* for every fixed graph H. For a graph H, a graph G is an H-graph if it is an intersection graph of connected subgraphs of some subdivision of H. It is worth mentioning that H-graphs generalize interval graphs (with H a complete graph with two vertices) and circular arc graphs (with H a cycle of length 3).

Theorem 2.39 ([61])**.** *Let H be a graph. For every H-graph G whose intersection model is given, we can compute in polynomial time a linear layout* (T, δ) *with* $\mathsf{mim}(T, \delta) \leq 2|E(H)|$ *and* $\mathsf{boolw}(T, \delta) \leq 2|E(H)| \cdot \log_2(n)$*.*

We conclude this section by the following open question.

Open Question 2.40. *Can we characterize and recognize in polynomial time the following graph classes?*

- *The graphs of clique-width at most 4.*
- *The graphs of (linear) boolean-width at most* $\log_2(3)$*.*
- *The graphs of (linear) mim-width 1.*

Vatshelle [132] proved that any graph of mim-width 1 is a perfect graph. Recently, Jaffke et al. [86] proved that Hamiltonian Cycle is NP-hard on graphs of linear mim-width 1 even when a rooted layout is given. Until now, on all graph classes of linear mim-width 1 (e.g. interval graphs, permutation graphs, convex graphs), Hamiltonian Cycle was known to be polynomial time solvable. This result reveals our lack of knowledge concerning the nature of the graphs of linear mim-width 1.

2.5 Computation

In this section, we will discuss the complexity of computing the parameters we have defined in this chapter. For a parameter $\mathsf{f} \in \{\mathsf{cw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}, \mathsf{boolw}, \mathsf{nec}_1\}$ and a polynomial P, let us define the following parameterized problems.

f-Width
Input: A graph G and an integer $k \in \mathbb{N}$. **Parameter:** k. **Output:** Is the f-width of G at most k?

P-Approx f-Width
Input: A graph G and an integer $k \in \mathbb{N}$. **Parameter:** k. **Output:** Is the f-width of G at most $P(k)$?

Observe that if P is a linear polynomial (of maximum degree 1) then the problem P-Approx f-Width consists in approximating the f-width of graphs within a constant factor. It is worth noticing that, for every f, if f-Width is FPT, then the problem of computing the f-width of a graph G is FPT parameterized by $\mathsf{f}(G)$. Moreover, if, for some polynomial P, the problem P-Approx f-Width is FPT, then the problem of computing, given a graph G, an integer $k \in \mathbb{N}$ such that $\mathsf{f}(G) \leq k \leq P(\mathsf{f}(G))$ is FPT parameterized by $\mathsf{f}(G)$.

The first subsection presents known complexity lower bounds such as NP-hardness concerning these two problems. The last subsection summarizes the known algorithms for f-Width and P-Approx f-Width with $\mathsf{f} \in \{\mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}\}$ and discusses the implications of these algorithms for the other parameters via the different upper bounds presented in the above section.

2.5.1 Complexity lower bounds

The NP-hardness of computing the clique-width of graphs was a problem open since the introduction of clique-width in the early 1990s. It was solved by Fellows et al. with the following theorem.

Theorem 2.41 ([54]). *If* $\mathsf{P} \neq \mathsf{NP}$*, then, for any* $\epsilon \in \mathbb{R}$ *with* $0 < \epsilon < 1$*, there are no polynomial time algorithms that, given an* n*-vertex graph* G*, output an integer* k *such that* $k - \mathsf{cw}(G) \leq n^{\epsilon}$*. In particular, the problem* cw-Width *is* NP*-hard.*

Theorem 2.42 ([118]). *The problem* rw-Width *is* NP*-hard.*

The two following theorems from [128] gives some complexity lower bounds on the computation of mim-width and boolean-width. In order to obtain these results, Sæther and Vatshelle prove that computing $\mathsf{boolw}(G)$ and $\mathsf{mim}(G)$ for a graph G is at least as hard as computing, respectively, $\mathsf{boolw}(A)$ and $\mathsf{mim}(A)$ for a vertex set A of a graph (see Theorem 3.6).

They deduce the two following theorems from the facts that computing $\mathsf{boolw}(A)$ is NP-hard [124] and that computing $\mathsf{mim}(A)$ is W[1]-hard [110] and not approximable within a constant factor unless NP = ZPP [48] where ZPP corresponds to the class of problems that admit a probabilistic Turing machine that always returns the correct answer and whose expected running time is polynomial.

Theorem 2.43 ([128]). *The problem* boolw-Width *is* NP*-hard.*

By Definition 2.10 of boolean-width, we deduce that Theorem 2.43 implies that the problem nec_1-Width is NP-hard.

Theorem 2.44 ([128]). *The problem* mim-Width *is* W[1]*-hard. Moreover, for any constant* $c \in \mathbb{R}$*, there is no polynomial time algorithm that solves the problem* c-Approx mim-Width *unless* NP = *ZPP.*

We conclude this subsection with some open questions concerning the problems f-WIDTH. The following open question summarizes the long standing open questions surrounding the computation of clique-width and boolean-width (or equivalently 1-neighbor-width).

Open Question 2.45. *For each* $\mathsf{f} \in \{\mathsf{cw}, \mathsf{boolw}, \mathsf{nec}_1\}$*, is the problem* f-WIDTH FPT*,* XP*,* W[1]*-hard or para* NP*-hard? Does there exist a polynomial* P *such that* P*-*APPROX f-WIDTH *is* FPT *or* XP*?*

The following open question concerns the computation of mim-width.

Open Question 2.46. *Is the problem* mim-WIDTH XP *or para* NP*-hard? Does there exist a polynomial* P *such that* P*-*APPROX mim-WIDTH *is* FPT *or* XP*?*

2.5.2 Computation of Rank-width and $\mathbb{Q}$-rank-width

Among the parameters defined in this chapter, rank-width and $\mathbb{Q}$-rank-width are the only ones known to be efficiently computable thanks to the following theorem.

Theorem 2.47 (Oum and Seymour [121])**.** *For each* $\mathsf{f} \in \{\mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}\}$*, there is a* $8^k \cdot n^9 \cdot \log(n)$ *time algorithm that, given a graph* G *as input and* $k \in \mathbb{N}$*, either outputs a rooted layout for* G *of* f*-width at most* $3k+1$ *or confirms that the* f*-width of* G *is more than* k*.*

This result relies on the fact that the set functions rw and $\mathsf{rw}_{\mathbb{Q}}$ are both symmetric, submodular and computable in polynomial time. Properties that none of the other set functions defined in this chapter fulfill.

For rank-width, this result can be improved [118] by using its properties (in particular, the fact that $GF(2)$ is a finite field). In fact, it is known that rw-WIDTH is FPT [81, 89]. Moreover, for any $k \in \mathbb{N}$ and for any n-vertex graph G, we can compute a rooted layout (T, δ) of rank-width k or confirm that $\mathsf{rw}(G) > k$ in time $O(h(k) \cdot n^{O(1)})$ for some function h.

We summarize the current knowledge concerning the computation of rank-width with FPT algorithms in Table 2.1.

The following theorem shows the implications of the algorithms presented in Table 2.1 for the other width measures which are equivalent to rank-width for the relation $\leq_f$.

Theorem 2.48 ([17])**.** *Let* G *be a graph and* $h : \mathbb{R} \to \mathbb{R}$ *be a function. For every rooted layout* (T, δ) *of* G *such that* $\mathsf{rw}(T, \delta) = h(\mathsf{rw}(G))$*, we have:*

(a) $\mathsf{f}(T, \mathcal{L}) \leq 2^{h(\mathsf{f}(G))}$ *for each* $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}_{\mathbb{Q}}\}$ *and*

(b) $\mathsf{boolw}(T, \mathcal{L}) \leq 2^{2h(\mathsf{boolw}(G))}$*.*

Proof. Inequality (a) follows from the upper bounds of Theorems 2.23-2.26 and Lemma 2.22. Indeed, by Lemma 2.22, for each $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}_{\mathbb{Q}}\}$, we have $\mathsf{rw}(G) \leq \mathsf{f}(G)$ and, for every rooted layout (T, δ) of G, we have $\mathsf{f}(T, \delta) \leq 2^{\mathsf{rw}(T,\delta)}$. Thus, for every rooted layout (T, δ) of G with $\mathsf{rw}(T, \delta) = h(\mathsf{rw}(G))$, we can conclude that $\mathsf{f}(T, \delta) \leq 2^{h(\mathsf{f}(G))}$. The proof for Inequality (b) is similar and was proved in [17]. □

We conclude this section by two open questions concerning the computation of rank-width raised in [119]. The first one is quite natural in view of Table 2.1.

Open Question 2.49. *Does there exist an algorithm with some function* f *that, given a graph* G *and* $k \in \mathbb{N}$*, finds a rooted layout of rank-width at most* $f(k)$ *or confirms that* $\mathsf{rw}(G) > k$ *in time* $O(2^{O(k)} \cdot n^3)$*?*

Table 2.1 – Values of $g(k)$, running time and paper of some FPT-algorithms that, given an n-vertex graph G and $k \in \mathbb{N}$, compute a rooted layout of rank-width at most $g(k)$ or confirm that $\mathsf{rw}(G) > k$.

Paper	$g(k)$	Running time	Observations
[121]	$3k+1$	$O(8^k \cdot n^9 \cdot \log(n))$	Works also for any symmetric submodular function under some conditions, in particular, it works for $\mathsf{rw}_{\mathbb{Q}}$.
[118]	$3k+1$	$O(8^k \cdot n^4)$	Algorithm from [121] with a faster subroutine designed for rank-width that uses *vertex-minor* to confirm $\mathsf{rw}(G) > k$.
[118]	$24k$	$O(h(k) \cdot n^3)$	Based on a result from [80] and the equivalence between rank-width of bipartite graphs and branch-width of binary matroids. The function $h(k)$ is not given explicitly in [80, 118]
[118]	$3k-1$	$O(f(k) \cdot n^3)$	The algorithm uses monadic second order logic and Courcelle's theorem to speed up a subroutine of [121]. The function $f(k)$ is a huge function.
[81]	k	$O(g(k) \cdot n^3)$	Uses a huge list of forbidden minors in matroids, $g(k)$ is a huge function.
[89]	k	$O(r(k) \cdot n^3)$	Solve a more general problem. Self contained algorithm based on a compact representation of rooted layouts, the function $r(k)$ is not given explicitly.

The best algorithms we know for approximating the rank-width of a graph have a cubic dependence in the number of vertices. Can we avoid this? It is worth noticing that for tree-width, Bodlaender et al. presented an algorithm that, given a graph G and an integer k, computes a tree-decomposition of tree-width at most $5k + 4$ or confirms that $\mathsf{tw}(G) > k$ in time $2^{O(k)} \cdot n$ [11].

Open Question 2.50. *Does there exist an algorithm with some functions f,g and a constant $c < 3$ that, given an n-vertex graph G, finds a rooted layout of rank-width at most $f(k)$ or confirms that $\mathsf{rw}(G) > k$ in time $O(g(\mathsf{rw}(G)) \cdot n^c)$?*

Ideally, the function $f(k)$ in Questions 2.49 and 2.50 should be a linear polynomial.

2.6 Width measures versus NP-hard problems

In this section, we discuss about how the seven parameters defined in this chapter catch the hardness of NP-hard problems. We begin by presenting Courcelle's meta-theorem and its variant for clique-width. Finally, we will discuss the parameterized complexity of some classical problems such as Minimum Vertex Cover, Minimum Dominating Set, Hamiltonian Cycle, Graph Coloring, Max Cut, etc.

2.6.1 Monadic second order logic

Many graph problems are expressible in monadic second order logic (MSO_2 for short). Informally, an MSO_2 formula is a well-formed term built with:

- the quantifiers $\forall$ and $\exists$,
- variables for vertices, edges, sets of vertices and sets of edges,
- the logical connectives $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$,
- the binary relations $=, \in, \mathsf{inc}, \mathsf{adj}$ where inc and adj correspond, respectively, to the predicates "this edge is incident to this vertex" and "these two vertices are adjacent".

See [31] for a more formal definition. Many well-known and well-studied NP-hard problems are expressible in MSO_2 such as Hamiltonian Cycle, 3-Colorability, etc.

The following theorem is certainly the most famous theorem in parameterized complexity.

Theorem 2.51 (Courcelle's theorem [29])**.** *For every problem Π expressible in MSO_2 by a formula ϕ, there exists a function f such that Π is solvable in time $f(|\phi|, k) \cdot n$ on graphs with n vertices and with a tree decomposition of tree-width k.*

In fact, the requirement that G is provided with a tree decomposition is not necessary since Bodlaender [7] proved that, for any graph, we can compute an optimal tree decomposition in time $k^{O(k^3)} \cdot n$. Consequently, the problems expressible in MSO_2 are FPT parameterized by tree-width and the length of the formula. Observe that there exist some variants [3, 13, 34] of Theorem 2.51 which are useful for optimization and counting problems.

Courcelle, Makowsky, and Rotics [28] extended Theorem 2.51 to graphs of bounded clique-width at a cost of a smaller set of problems. These problems are those expressible by an MSO_1 formula, i.e., an MSO_2 formula that does not use edge set quantifications.

Theorem 2.52 ([28])**.** *For every problem expressible in MSO_1 by a formula ϕ, there exists an algorithm that, given a graph G and a k-expression of G, solves this problem in time $f(|\phi|, k) \cdot n$.*

Observe that we can compute a $2^{\mathsf{cw}(G)+1}$-expression for any graph G in time $f(\mathsf{cw}(G)) \cdot n^3$ thanks to the algorithm from [118] and Theorems 2.21 and 2.48. Consequently, for any problem Π expressible in MSO_1 by a formula ϕ, there exists an algorithm, that given a graph G, solves Π in time $g(|\phi|, \mathsf{cw}(G)) \cdot n^3$. That is, every problem expressible in MSO_1 is FPT parameterized by clique-width and the length of the formula.

Notice that Theorem 2.52 holds also if we replace clique-width by any with measure p such that $p \leq_f \mathsf{cw}$ thanks to Lemma 2.34.

These two meta-theorems are useful in order to know whether a problem is FPT parameterized by tree-width or clique-width because we just have to express the problem in MSO_2 or MSO_1. As their proofs are constructive, given an MSO_2 formula representing some problem Π, we can use them to design an FPT algorithm for Π parameterized by tree-width (or clique-width if ϕ is an MSO_1 formula). However, the running time of this algorithm may be a tower of exponentials. In fact, the height of this tower of exponentials depends on the number of alternating quantifiers in the monadic second order formula. Consequently, we cannot use these meta-theorems to design efficient FPT algorithms and for doing so, you need to "get your hands dirty".

Observe that, there is little hope to extend Theorem 2.52 to problems expressible in MSO_2. Indeed, Courcelle, Makowsky, and Rotics [28] proved that there exist problems expressible in MSO_2, that are not solvable in polynomial time even on cliques unless NEXP = EXP where NEXP (resp. EXP) is the set of decision problems solvable by a non-deterministic (resp. deterministic) Turing machine using time $2^{n^{O(1)}}$ with n the input size. Notice that NEXP $\neq$ EXP is a weaker assumption that P $\neq$ NP.

However, this negative result does not rule out the existence of FPT algorithms parameterized by clique-width for problems that are not expressible in MSO_1. Several classical problems are not expressible in MSO_1 such as Max-Cut, Edge Dominating Set (EDS), Graph Coloring (GC), and Hamiltonian Cycle (HC). These problems are known to be FPT parameterized by tree-width thanks to Courcelle's theorem or some variants of this latter theorem [3, 13, 34]. They are also known to admit XP algorithms parameterized by clique-width (see the following subsection). A natural question is whether these problems are FPT parameterized by clique-width. Fomin, Golovach, Lokshtanov, and Saurabh [58] proved the W[1]-hardness of EDS, GC, and HC with clique-width as parameter, which implies that these problems are not FPT parameterized by clique-width, unless W[1] = FPT. In 2014, the same authors [59] proved that Max-Cut and EDS do not admit $f(k) \cdot n^{o(k)}$-time algorithms unless ETH fails.

There are no such meta-theorems for mim-width. In fact, Vatshelle proves [132, Lemma 5.1.5] that Clique parameterized by mim-width is NP-hard on graphs of mim-width at most 6. As Clique is expressible in MSO_1, it is unlikely that there exists a meta-theorems like Courcelle's one for mim-width.

2.6.2 Parameterized complexity of classical problems

Once we know that a problem is FPT, a natural question that arises is how fast we can solve this problem. In this section, we summarize the asymptotically fastest known FPT algorithms for each parameter defined in this chapter and for some classical problems such as Minimum Vertex Cover, Minimum Dominating Set, Minimum Feedback Vertex Set, Graph Coloring, Hamiltonian Cycle, Max Cut, etc. We also give the best known lower bounds for these problems.

We begin by the algorithms for (σ, ρ)-Dominating Set problems and their connected and acyclic variants. All of these problems are FPT parameterized by clique-width and XP parameterized by mim-width. Finally, we will talk about several classical problems that are known

to be W[1]-hard parameterized by clique-width such as Graph Coloring and Hamiltonian Cycle.

As explained in the introduction, we have a precise idea on the parameterized complexity of the different problems dealt with in this subsection concerning tree-width. In fact, all of these problems admit $2^{O(\mathsf{tw}(G))} \cdot n$ time algorithms [9, 37, 129].

(σ, ρ)-Dominating Set problems and their acyclic and connected variants

We present the asymptotically best algorithms for (σ, ρ)-Dominating Set problems and for the acyclic and connected variants of these problems. We define these problems and give a list of famous (σ, ρ)-Dominating Set problems in Section 2.6.2. Examples of such problems are given in Table 1.1. We will see that, for each parameter defined in this chapter, the best known parameterized algorithms for those problems are obtained by using the d-neighbor-width and the upper bounds of Theorem 2.33 on this latter parameter. This highlights how well the d-neighbor width catch the hardness of this kind of problems.

For each parameter defined in this chapter, the best[1] known algorithms for all NP-hard (σ, ρ)-Dominating Set problems are obtained from the following theorem from [18, Theorem 1].

Theorem 2.53 ([18])**.** *Given an n-vertex graph G and a rooted layout $\mathcal{L}$ of G, we can solve any (σ, ρ)-Dominating Set problem in time $O(\mathsf{s\text{-}nec}_d(\mathcal{L})^3 \cdot \log(\mathsf{s\text{-}nec}_d(\mathcal{L})) \cdot n^3)$ with d a constant that depends only on (σ, ρ).*

It is worth noticing that the running time given in [18, Theorem 1] is $O(\mathsf{s\text{-}nec}_d(\mathcal{L})^3 \cdot n^4)$ because the authors replace the factor $\log(\mathsf{s\text{-}nec}_d(\mathcal{L}))$ by n in their running time proof. Notice that Bui-Xuan, Telle, and Vatshelle extended Theorem [18] to a wider class of problems called LC-VSVP problems. This class generalizes the (σ, ρ)-Dominating Set problems and includes homomorphism problems like H-Coloring and H-homomorphism with H a fixed graph.

The two following theorems summarizes the results we obtained in Section 4.2 concerning the acyclic and connected variant of the (σ, ρ)-Dominating Set problems. More precise running times are given in Section 4.2.

Theorem 2.54 (Subsection 4.2.3)**.** *Given an n-vertex graph G and a rooted layout $\mathcal{L}$ of G, any* Connected (σ, ρ)-Dominating Set *problem and* Connected Co-(σ, ρ)-Dominating Set *problem can be solved in time $\mathsf{s\text{-}nec}_d(\mathcal{L})^3 \cdot \mathsf{s\text{-}nec}_1(\mathcal{L})^{O(1)} \cdot \log(\mathsf{s\text{-}nec}_d(\mathcal{L})) \cdot n^3$ with d a constant that depends only on (σ, ρ).*

Theorem 2.55 (Subsection 4.2.4)**.** *For each $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$, given an n-vertex graph G and a rooted layout $\mathcal{L}$ of G, any* AC-(σ, ρ)-Dominating Set *problem and* Acyclic (σ, ρ)-Dominating Set *problem can be solved in time $\mathsf{s\text{-}nec}_2(\mathcal{L})^3 \cdot \mathsf{s\text{-}nec}_1(\mathcal{L})^{O(1)} \cdot \mathcal{N}_f(\mathcal{L})^{O(1)} \cdot n^3$, with d a constant that depends only on (σ, ρ) and where $\mathcal{N}_f(\mathcal{L})$ is the term defined in Table 2.2.*

Table 2.2 – Values of $\mathcal{N}_f(\mathcal{L})$ for every rooted layout $\mathcal{L}$ and for each $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$.

f	mw	$\mathsf{rw}_{\mathbb{Q}}$	rw	mim
$\mathcal{N}_f(\mathcal{L})$	$2^{\mathsf{mw}(\mathcal{L})} \cdot 2n$	$2^{\mathsf{rw}_{\mathbb{Q}}(\mathcal{L}) \log_2(2\mathsf{rw}_{\mathbb{Q}}(\mathcal{L})+1)} \cdot 2n$	$2^{2\mathsf{rw}(\mathcal{L})^2} \cdot 2n$	$2n^{2\mathsf{mim}(\mathcal{L})+1}$

We deduce the following corollary from Theorems 2.53-2.55 and the upper bounds on the d-neighbor width from Theorem 2.33.

[1] Up to a constant in the exponent.

Corollary 2.56. *Given an n-vertex graph G and a rooted layout $\mathcal{L}$ of G, we can solve any* (σ,ρ)-Dominating Set *problem,* Connected (σ,ρ)-Dominating Set *problem,* AC-(σ,ρ)-Dominating Set *problem, and* Acyclic (σ,ρ)-Dominating Set *problem within the following running times*

- $(d+1)^{O(\mathsf{mw}(\mathcal{L}))} \cdot n^{O(1)}$,
- $(d \cdot \mathsf{rw}_{\mathbb{Q}}(\mathcal{L})+1)^{O(\mathsf{rw}_{\mathbb{Q}}(\mathcal{L}))} \cdot n^{O(1)}$,
- $2^{O(d\cdot\mathsf{rw}(\mathcal{L})^2)} \cdot n^{O(1)}$,
- $2^{O(\mathsf{boolw}(\mathcal{L}))} \cdot n^{O(1)}$ *if* $d=1$ *and* $2^{O(d\cdot\mathsf{boolw}(\mathcal{L})^2)} \cdot n^{O(1)}$ *otherwise.*
- $n^{O(d\cdot\mathsf{mim}(\mathcal{L}))}$.

with d a constant that depends only on (σ,ρ).

Corollary 2.56 reveals the power and the generality of the d-neighbor equivalence when it comes to solve NP-hard problems. Indeed, the running times presented in this corollary are (up to a constant in the exponents) the best known for solving NP-hard problems for each width measure. In the following, we give the lower bounds for two (σ,ρ)-Dominating Set problems: Independent Set (IS for short) and Dominating Set (DS for short). We choose these two problems because they seem to be somehow the easiest problems among the NP-hard (σ,ρ)-Dominating Set problems. These lower bounds hold also for many other NP-hard problems dealt with in Corollary 2.56 such as Feedback Vertex Set, Vertex Cover, Connected Vertex Cover, Connected Dominating Set, etc.

Theorem 2.57 ([82, 83])**.** *Unless* ETH *fails, there is no $2^{o(n)}$ time algorithm for* IS *and* DS *where n is the number of vertices of the input graph.*

In fact, Theorem 2.57 holds for every graph problem Π that has a linear reduction from 3-SAT, i.e., we can transform each instance of 3-SAT with N variables and M clauses into an instance of Π with $O(N+M)$ vertices.

From Theorem 2.57, we deduce the following corollary.

Corollary 2.58. *For each $\mathsf{f} \in \{\mathsf{cw},\mathsf{rw},\mathsf{rw}_{\mathbb{Q}},\mathsf{boolw}\}$, unless* ETH *fails, there is no $2^{o(\mathsf{f}(\mathcal{L}))} \cdot n^{O(1)}$ time algorithm for* IS *and* DS *where n is the number of vertices of the input graph and $\mathcal{L}$ a given rooted layout of the input graph G.*

Proof. For every layout $\mathcal{L}$ of G and for each $\mathsf{f} \in \{\mathsf{cw},\mathsf{rw},\mathsf{rw}_{\mathbb{Q}},\mathsf{boolw},\mathsf{mim}\}$, we have $\mathsf{f}(\mathcal{L}) \leq n$. Thus the existence of a $2^{o(\mathsf{f}(\mathcal{L}))} \cdot n^{O(1)}$ time algorithm for IS (resp. DS) implies that IS (resp. DS) are solvable in time $2^{o(n)}$. By Theorem 2.57, this is not possible unless ETH fails. □

We deduce that, for IS and DS, the running times presented in Corollary 2.56 are asymptotically optimal under ETH for clique-width and boolean-width (because $d=1$ for these two problems).

The optimality of the other running times presented in Corollary 2.56 for IS and DS and the other parameters is open. The following open question concerns rank-width, a similar open question holds also for $\mathbb{Q}$-rank-width.

Open Question 2.59. *Given an n-vertex graph with a rooted layout $\mathcal{L}$, can we solve the problem* IS *(or* DS*) in time $2^{o(\mathsf{rw}(\mathcal{L})^2)} \cdot n^{O(1)}$?*

Concerning mim-width, we have no tight lower bound under ETH but the following W[1]-hardness result from [61, Corollary 4].

Theorem 2.60 ([61]). *The problems that ask, given a graph G and a linear layout $\mathcal{L}$ of G, respectively, for the computation of an independent set of size k and a dominating set of size k, are* W[1]*-hard parameterized by $k + \mathsf{mim}(\mathcal{L})$.*

It is worth mentioning that Jaffke, Kwon, and Telle [84] have proved the same statement for the Maximum Induced Forest problem. Consequently, IS, DS, and Feedback Vertex Set are W[1]-hard parameterized by mim-width even when a rooted layout is given as input.

Classical problems that are W[1]-hard parameterized by clique-width

Now, we present some algorithmic results concerning the following problems: Max-Cut, Edge Dominating Set (EDS), Graph Coloring (GC), and Hamiltonian Cycle (HC). For tree-width and clique-width, we have a precise idea on the parameterized complexity of these problems since, for each problem, we have an upper bound which asymptotically matches the best known lower-bound. For rank-width, we only have some upper bounds thanks to [67]. Some para-NP-hardness results are known for mim-width. However, for $\mathbb{Q}$-rank-width and boolean-width, the only algorithmic results we know are those we can deduce from the results on clique-width or rank-width and the known upper bounds between these parameters.

We start by presenting the results concerning clique-width.

The following theorem provides asymptotically tight lower bounds and upper bounds for Max-Cut, Edge Dominating Set and Hamiltonian Cycle.

Theorem 2.61 (Section 4.4 and [59, 60]). *There exist algorithms that, given an n-vertex graph G together with a k-expression, solve in time $n^{O(k)}$ the problems* Max-Cut*,* Edge Dominating Set *and* Hamiltonian Cycle*. Moreover, even when a k-expression is given as input,* Max-Cut*,* Edge Dominating Set*, and* Hamiltonian Cycle *are not solvable in time $f(k) \cdot n^{o(k)}$, for any function f, unless* ETH *fails.*

Hence, Max-Cut, EDS, and HC have the same behavior towards clique-width. The following theorem reveals that Graph Coloring behave quite differently than the other three problems.

Theorem 2.62 ([74, 100]). *There exists an algorithm that, given an n-vertex graph G and a k-expression of G, solves* Graph Coloring *in time $n^{2^{O(k)}}$. Moreover, unless* ETH *fails,* Graph Coloring *cannot be solved in time $f(k) \cdot n^{2^{o(k)}}$ for any function f even when a k-expression is given as input.*

Theorems 2.61 and 2.62 give us a precise idea on the complexities of the problems Max-Cut, EDS, HC, and GC parameterized by clique-width. The main drawback of these theorems is that they mainly use ad hoc techniques and ideas which works well only for clique-width. Using the same ideas for the other parameters would not give efficient algorithms. For example, we deduce, from Theorem 2.23(b) and Theorems 2.61 and 2.62, that Max-Cut, EDS, and HC can be solved in time $n^{2^{O(\mathsf{rw}(G))}}$ and GC can be solved in time $n^{2^{2^{O(\mathsf{rw}(G))}}}$. Ganian, Hliněný, and Obdržálek [67] showed the following theorem which proves that we can do better than $n^{2^{2^{O(\mathsf{rw}(G))}}}$ for GC.

Theorem 2.63 ([67]). *There exists an algorithm that, given an n-vertex graph G and a rooted layout $\mathcal{L}$ of G as input, solves* Graph Coloring *in time $n^{2^{O(\mathsf{rw}(\mathcal{L})^2)}}$.*

In fact, Ganian, Hliněný, and Obdržálek [67] developed a framework to design XP algorithms for rank-width and bi-rank-width, a natural extension of rank-width to directed graphs introduced by Kanté [92]. However, this framework is quite complex and does not always beat the

naive algorithms we can obtain from the algorithms parameterized by clique-width and Theorem 2.23.

Pinning down the right exponent of n for these problems and for ($\mathbb{Q}$-)rank-width and boolean-width remains open. Concerning mim-width, we know that HC and GC are para NP-hard thanks to the two following theorems.

Theorem 2.64 ([86]). Hamiltonian Cycle *is* NP*-hard on graphs of linear mim-width 1 even when a linear layout of mim-width 1 is given.*

Theorem 2.65 ([5, 69]). Graph Coloring *is* NP*-hard on graphs of linear mim-width 2 even when a linear layout of mim-width at most 2 is given.*

Proof. We know that Graph Coloring is NP-hard on Circular Arc graphs [69]. Belmonte and Vatshelle [5] proved that Circular Arc graphs have linear mim-width at most 2 and that we can compute for these graphs a linear layout of mim-width at most 2. □

To the best of our knowledge, the complexity of Max-Cut and EDS parameterized by mim-width remains open. In fact, we do not even know whether Max-Cut is solvable in polynomial time on Interval graphs (which are known to have linear mim-width 1 thanks to [5]).

We conclude this section with some general open questions concerning the problems Max-Cut, EDS, HC, and GC.

Open Question 2.66. *Can we unify the algorithmic results for clique-width provided by Theorems 2.61 and 2.62? Can we design a framework similar to the one for* (σ, ρ)-Dominating Set *problems for handling the problems which are* W[1]*-hard parameterized by clique-width? Is the notion of d-neighbor equivalence also useful for this kind of problems?*

Let us give some hints about how to solve these questions. The first step would be to define a family of problems that includes Max-Cut, EDS, HC. For doing so, we can generalize the concepts of (σ, ρ)-dominating set as follows. Given a graph G and a pair (σ, ρ) of finite or co-finite subsets of $\mathbb{N}$, we say that a set of edges $F \subseteq E(G)$ is a (σ, ρ) edge dominating set if

- for every edge $f \in F$, the number of edges in F incident to f belongs to σ and
- for every edge $e \in E(G) \setminus F$, the number of edges in F incident to e belongs to ρ.

This notion of (σ, ρ) edge dominating set generalizes both the notion of Hamiltonian cycle and edge dominating set. Indeed, an edge dominating set is simply an $(\mathbb{N}, \mathbb{N}^+)$ edge dominating set and a Hamiltonian cycle is a $(\{2\}, \mathbb{N})$ edge dominating set of maximum size and with $G_{|F}$ connected. However, this family of problems does not seem to include the Max-Cut problem.

Nevertheless, can we use the d-neighbor equivalence for these three problems? As these problems are W[1]-hard parameterized by clique-width and HC is para NP-hard parameterized by mim-width, we cannot expect the d-neighbor equivalence for d a constant to be useful. Maybe, we can avoid this dead-end by using the d-neighbor equivalence with $d = n$.

Recently, Bergougnoux and Kanté [6] proved that, given an n-vertex graph and a rooted layout $\mathcal{L}$, we can solve Max Cut in time $\mathsf{s\text{-}nec}_n(\mathcal{L})^{O(1)} \cdot n^{O(1)}$. For every rooted layout of a graph G, they proved that $\mathsf{s\text{-}nec}_n(\mathcal{L})^{O(1)}$ is upper bounded by $n^{\mathsf{mw}(A)}$, $n^{\mathsf{rw}_{\mathbb{Q}}(A)}$ and $n^{2^{\mathsf{rw}(A)}}$. Consequently, the result of [6] implies that Max Cut is solvable in time $n^{O(\mathsf{mw}(\mathcal{L}))}$, $n^{O(\mathsf{rw}_{\mathbb{Q}}(G))}$ and $n^{2^{O(\mathsf{rw}(G))}}$. It is worth mentioning that contrary to the algorithm for Max Cut given in [59], there is no need to assume that the graph is given with a clique-width expression as the algorithm can in [6] be parameterized by $\mathbb{Q}$-rank-width, which is always smaller than clique-width and for which also a fast FPT $(3k+1)$-approximation algorithm exists [121].

Chapter 3

Two new cousins of clique-width

In this chapter, we introduce and study two new width measures: $\mathcal{B}$-rank-width and $\mathbb{N}$-rank-width. In Section 3.1, we define these two parameters from algebra. In Section 3.2, we prove several properties on these parameters. We talk about their computational hardness in Section 3.3 and we compare them to other width measures in Section 3.4. Finally, we express our deep disappointment concerning these two new width parameters in Section 3.5.

3.1 Definitions

These two parameters are defined from the algebraic structure called *commutative semiring*. This concept is defined from the notion of *monoid*.

A monoid is a pair $(R, *)$ with R a set and $* : R \times R \to R$ verifying the following properties:

- *Associativity:* for all $a, b, c \in R$, we have $(a * b) * c = a * (b * c)$,
- *Identity element:* there exists an element $e \in R$ (called *identity element*), such that for every element $a \in R$, we have $e * a = a * e = a$.

A monoid $(R, *)$ is *commutative* if for every element $a, b \in R$, we have $a * b = b * a$.

A *commutative semiring* is a tuple $(R, \dot{+}, *)$ where $(R, \dot{+})$ and $(R, *)$ are two commutative monoids with identity elements, respectively, 0 and 1, and we have the following properties:

- *Distributivity:* for every $a, b, c \in R$, we have $a \cdot (b \dot{+} c) = (a * b) \dot{+} (a * c)$,
- *Multiplication by 0 annihilates R:* for all $a \in R$, we have $a * 0 = 0$.

Unlike rings, we do not require that each element must have an additive inverse.

In order to define $\mathcal{B}$-rank-width, we use the *Boolean semiring* $\mathcal{B} := (\{0, 1\}, \vee, \wedge)$ where $\vee$ and $\wedge$ corresponds respectively to the logical disjunction and the logical conjunction. Contrary to $GF(2)$, 1 has no inverse for the addition in $\mathcal{B}$ since $1 \vee 1 = 1$ and $1 \vee 0 = 1$.

To define $\mathbb{N}$-rank-width, we use the *natural semiring* $(\mathbb{N}, +, \cdot)$ where $+$ and $\cdot$ are the ordinary addition and multiplication.

Similarly to linear algebra, we can define the span of a vector set for semirings. For doing so, we need the following notions. Let $\mathcal{R} := (R, \dot{+}, *)$ be a commutative semiring and $k \in \mathbb{N}^+$. We denote by R^k the set of all sequences of length k whose elements belong to R. Given $v = (v_1, \ldots, v_k), v' = (v'_1, \ldots, v'_k) \in R^k$, the vector denoted by $v \dot{+} v'$ equals $(v_1 \dot{+} v'_1, \ldots, v_k \dot{+} v'_k)$ and given $a \in R$, the vector denoted by $a * v$ equals $(a * v_1, \ldots, a * v_k)$. Given a subset $\mathcal{S} = \{v_1, \ldots, v_r\}$ of R^k, the span of $\mathcal{S}$ over $\mathcal{R}$, denote by $\langle \mathcal{S}, \mathcal{R} \rangle$, is the set of all vectors v in R^k such that there exist $a_1, \ldots, a_r \in R$ such that $v = (a_1 * v_1) \dot{+} (a_2 * v_2) \dot{+} \cdots \dot{+} (a_r * v_r)$.

Given $\mathcal{S} \subseteq R^k$ and a matrix M with k columns whose entries belong to R, we say that $\mathcal{S}$ is an $\mathcal{R}$-*generator* of M if every row vector of M belongs to $\langle \mathcal{S}, \mathcal{R} \rangle$. Observe that, if $\mathcal{R}$ is a field, then the rank of M over $\mathcal{R}$ corresponds to the size of a minimum $\mathcal{R}$-generator of M.

For example, let M be the matrix with rows $(1,1,0), (0,1,1)$ and $(1,1,1)$. The sum of the vectors $(1,1,0)$ and $(0,1,1)$ over $\mathcal{B}, \mathbb{N}$ and $GF(2)$ equals, respectively, $(1,1,1)$, $(1,2,1)$, and $(1,0,1)$. Consequently, the rank of M over $\mathcal{B}$ is 2 and the rank of M over $\mathbb{N}$ or $GF(2)$ equals 3.

In the following, we give the definition of $\mathcal{B}$-rank-width and $\mathbb{N}$-rank-width.

Definition 3.1 ($\mathcal{B}$-rank-width)**.** *For every graph G, the $\mathcal{B}$-rank-width of a graph G corresponds to the $\mathsf{rw}^G_{\mathcal{B}}$-width of G where $\mathsf{rw}^G_{\mathcal{B}} : 2^{V(G)} \to \mathbb{N}$ is a function such that $\mathsf{rw}^G_{\mathcal{B}}(A)$ is the minimum size of a* $\mathcal{B}$-generator *of $M_{A,\overline{A}}$.*

Definition 3.2 ($\mathbb{N}$-rank-width)**.** *For every graph G, the $\mathbb{N}$-rank-width of a graph G corresponds to the $\mathsf{rw}^G_{\mathbb{N}}$-width of G where $\mathsf{rw}^G_{\mathbb{N}} : 2^{V(G)} \to \mathbb{N}$ is a function such that $\mathsf{rw}^G_{\mathbb{N}}(A)$ is the minimum size of a* $(\mathbb{N}, +, \cdot)$-generator *of $M_{A,\overline{A}}$.*

When the underlying graph G is clear from context, we may simply write $\mathsf{rw}_{\mathcal{B}}$ and $\mathsf{rw}_{\mathbb{N}}$ instead of $\mathsf{rw}^G_{\mathcal{B}}$ and $\mathsf{rw}^G_{\mathbb{N}}$.

3.2 Properties

In this section, we compare the values of $\mathcal{B}$-rank-width and $\mathbb{N}$-rank-width with module-width and rank-width. We start with the following observation.

Observation 3.3. *Let M be a binary matrix with k columns and $\mathcal{S}$ be an $(\mathbb{N}, +, \cdot)$-generator of M. The set $\mathcal{S} \cap \{0,1\}^k$ is an $(\mathbb{N}, +, \cdot)$-generator and a $\mathcal{B}$-generator $\mathcal{S}$ of M. Consequently, for every graph G and $A \subseteq V(G)$, we have $\mathsf{rw}_{\mathcal{B}}(A) \leq \mathsf{rw}_{\mathbb{N}}(A)$ and $\mathsf{rw}_{\mathcal{B}}(G) \leq \mathsf{rw}_{\mathbb{N}}(G)$.*

Proof. Let $v \in \{0,1\}^k$. Observe that if there exist $a_1, \ldots, a_r \in \mathbb{N}$ and $v_1, \ldots, v_r \in \mathbb{N}^k$ such that $v = (a_1 \cdot v_1) + (a_2 \cdot v_2) + \cdots + (ar \cdot v_r)$, then, for every $i \in [r]$, we have the following:

- $a_i \in \{0,1\}$,
- if $v_i \notin \{0,1\}^k$, then $a_i = 0$,
- for every $j \in [r]$ with $j \neq i$ and each $\ell \in [k]$, if the ℓth coordinates of v_i and v_j are 1 then either $a_i = 0$ or $a_j = 0$.

We deduce that $\mathcal{S} \cap \{0,1\}^k$ is an $(\mathbb{N}, +, \cdot)$-generator of M. Since $a_i \in \{0,1\}$ and $a_i \cdot c = a_i \wedge c$ for every $i \in [r]$ and $c \in \{0,1\}$, we conclude that $\mathcal{S} \cap \{0,1\}^k$ is also a $\mathcal{B}$-generator of M. □

Now, we prove that $\mathsf{rw}_{\mathcal{B}}(A)$ and $\mathsf{rw}_{\mathbb{N}}(A)$ both correspond to some *biclique* cover problems. Let us give some definitions. A biclique is a bipartite graph $(A \cup B, E)$ such that A and B are two independent sets and $E = \{ab : a \in A \text{ and } b \in B\}$. Given a graph G, we say that B is a biclique of G if B is a biclique and a subgraph of G. Let G be a graph. Given $A \subseteq V(G)$, a *biclique cover* of $G[A, \overline{A}]$ is a collection $\{B_1, \ldots, B_k\}$ of bicliques of $G[A, \overline{A}]$ such that $E(G[A, \overline{A}]) = \bigcup_{i \in [k]} E(B_i)$. We say that a *biclique cover* $\{B_1, \ldots, B_k\}$ of $G[A, \overline{A}]$ is a *biclique partition* of $G[A, \overline{A}]$ if the bicliques $B_1, \ldots, B_k$ are pairwise edge-disjoint. We define $\mathsf{bic\text{-}cover} : 2^{V(G)} \to \mathbb{N}$ and $\mathsf{bic\text{-}part} : 2^{V(G)} \to \mathbb{N}$ such that $\mathsf{bic\text{-}cover}(A)$ (resp. $\mathsf{bic\text{-}part}(A)$) is the minimum $k \in \mathbb{N}$ such that there exists a biclique cover (resp. biclique partition) of $G[A, \overline{A}]$ of size k.

Lemma 3.4. *For every graph G and every $A \subseteq V(G)$, we have $\mathsf{rw}_{\mathcal{B}}(A) = \mathsf{bic\text{-}cover}(A)$ and $\mathsf{rw}_{\mathbb{N}}(A) = \mathsf{bic\text{-}part}(A)$.*

Proof. Let G be a graph, $A \subseteq V(G)$ and $t = |\overline{A}|$. For each $i \in [t]$, we denote by v_i the vertex in $\overline{A}$ which is associated with the ith column of $M_{A,\overline{A}}$.

We begin by proving that $\mathsf{rw}_{\mathcal{B}}(A) \leq \mathsf{bic\text{-}cover}(A)$ and $\mathsf{rw}_{\mathbb{N}}(A) \leq \mathsf{bic\text{-}part}(A)$. Let $\{B_1, \ldots, B_k\}$ be a biclique cover of $G[A, \overline{A}]$. For every $i \in [k]$, we associate B_i with a row vector $b_i := (b_i^1, \ldots, b_i^t)$ such that, for every $j \in [t]$, if $v_j \in V(B_i)$, then $b_i^j := 1$, otherwise $b_i^j := 0$. We claim that $\{b_1, \ldots, b_k\}$ is a $\mathcal{B}$-generator of $M_{A,\overline{A}}$.

Let $v \in A$ and b_v be the row vector of $M_{A,\overline{A}}$ associated with v. Since $\{B_1, \ldots, B_k\}$ is a biclique cover, we deduce that, for every $u \in N_G(v) \cap \overline{A}$, there exists $i \in [k]$ such that $uv \in E(B_i)$ and $V(B_i) \cap \overline{A} \subseteq N_G(v) \cap \overline{A}$. Hence, there exist $\ell_1, \ldots, \ell_t \in [k]$ such that $N_{G[A,\overline{A}]}(v) = \big(V(B_{\ell_1}) \cup V(B_{\ell_2}) \cup \cdots \cup V(B_{\ell_t})\big) \cap \overline{A}$. Consequently, we have $b_{\ell_1} \vee b_{\ell_2} \vee \cdots \vee b_{\ell_t} = b_v$.

Observe that if $\{B_1, \ldots, B_k\}$ is a biclique partition of $G[A, \overline{A}]$, then for every $i, j \in [t]$ such that $i \neq j$, we have $(V(B_{\ell_i}) \cap V(B_{\ell_j})) \cap \overline{A} = \varnothing$ and thus $b_{\ell_1} + b_{\ell_2} + \cdots + b_{\ell_t} = b_v$.

We conclude that $\{b_1, \ldots, b_k\}$ is a $\mathcal{B}$-generator of $M_{A,\overline{A}}$ and if $\{B_1, \ldots, B_k\}$ is a biclique partition of $G[A, \overline{A}]$, then $\{b_1, \ldots, b_k\}$ is an $(\mathbb{N}, +, \cdot)$-generator of $M_{A,\overline{A}}$. It follows that $\mathsf{rw}_{\mathcal{B}}(A) \leq \mathsf{bic\text{-}cover}(A)$ and $\mathsf{rw}_{\mathbb{N}}(A) \leq \mathsf{bic\text{-}part}(A)$.

Finally, it remains to prove that $\mathsf{bic\text{-}cover}(A) \leq \mathsf{rw}_{\mathcal{B}}(A)$ and $\mathsf{bic\text{-}part}(A) \leq \mathsf{rw}_{\mathbb{N}}(A)$. Let $\mathcal{S} = \{b_1, \ldots, b_k\}$ be a $\mathcal{B}$-generator of $M_{A,\overline{A}}$. For each $i \in [k]$, we write $b_i := (b_i^1, \ldots, b_i^t)$. For every $v \in A$, let $\mathcal{S}(v)$ be a subset of $\mathcal{S}$ such that the row vector of $M_{A,\overline{A}}$ associated with v equals $\bigvee_{b \in \mathcal{S}(v)} b$.

For every $i \in [k]$, we associate the row vector $b_i = (b_i^1, \ldots, b_i^t)$ with a biclique $B_i := (A_i \cup \overline{A}_i, E_i)$ where

- $\overline{A}_i \subseteq \overline{A}$ such that $v_j \in \overline{A}_i$ iff $b_i^j = 1$,
- $A_i := \{v \in A : b_i \in \mathcal{S}(v)\}$,
- $E_i := \{a\overline{a} : a \in A_i \text{ and } \overline{a} \in \overline{A}_i\}$.

We deduce that $B_1, \ldots, B_k$ are subgraphs of $G[A, \overline{A}]$ because $\mathcal{S}$ is a $\mathcal{B}$-generator, for every $v \in A$ and $b \in \mathcal{S}(v)$, if an entry of b associated with a vertex $u \in \overline{A}$ equals 1, then $vu \in E(G[A, \overline{A}])$. It follows that each B_i is a subgraph of $G[A, \overline{A}]$. By definition of $\mathcal{S}(v)$, for every $v \in A$, we have $N(v) = \bigcup_{b_i \in \mathcal{S}} V(B_i) \cap \overline{A}$. Hence, $\{B_1, \ldots, B_k\}$ is a biclique-cover of $G[A, \overline{A}]$ and $\mathsf{rw}_{\mathcal{B}}(A) \leq \mathsf{bic\text{-}cover}(A)$.

If $\mathcal{S}$ is an $(\mathbb{N}, +, \cdot)$-generator of $M[A, \overline{A}]$, then, for every $i, j \in [k]$ with $i \neq j$, we have $E(B_i) \cap E(B_j) = \varnothing$. Indeed, in this case, for every $v \in A$ and $b_i, b_j \in \mathcal{S}(v)$, if $i \neq j$, then, for all $\ell \in [t]$, we cannot have $b_i^\ell = b_j^\ell = 1$ (since $1 + 1 = 2$). We deduce that, for every $i, j \in [k]$ with $i \neq j$, we have either $V(B_i) \cap V(B_j) \cap \overline{A} = \varnothing$ or $V(B_i) \cap V(B_j) \cap A = \varnothing$. We conclude that if $\mathcal{S}$ is an $(\mathbb{N}, +, \cdot)$-generator of $M[A, \overline{A}]$, then $\{B_1, \ldots, B_k\}$ is a biclique partition of $G[A, \overline{A}]$.

By Observation 3.3, we know that for every $(\mathbb{N}, +, \cdot)$-generator $\mathcal{S}'$ of $M_{A,\overline{A}}$, the set $\mathcal{S}' \cap \{0, 1\}^t$ is an $(\mathbb{N}, +, \cdot)$-generator and a $\mathcal{B}$-generator of $M_{A,\overline{A}}$. Thus, we can construct a biclique partition of $G[A, \overline{A}]$ of size $\mathsf{rw}_{\mathbb{N}}(A)$. We conclude that $\mathsf{bic\text{-}part}(A) \leq \mathsf{rw}_{\mathbb{N}}(A)$. □

Lemma 3.4 implies in particular that $\mathsf{rw}_{\mathcal{B}}$ and $\mathsf{rw}_{\mathbb{N}}$ are symmetric set functions since, by definition, bic-cover and bic-part are two symmetric set functions. With these properties, we are now ready to prove some results on $\mathcal{B}$-rank-width and $\mathbb{N}$-rank-width.

3.3 Computation hardness

In this section, we give some complexity lower bounds on the computation of the $\mathcal{B}$-rank-width and the $\mathbb{N}$-rank-width of graphs. For doing so, we use a meta-theorem of Sæther and Vatshelle

[128] reducing the computation of the set function to the width of a graph. This meta-theorem uses the following definition of $\mathcal{C}$-*satisfying cut-function*.

Definition 3.5 ([128])**.** *A set function* f^G *is a* $\mathcal{C}$-satisfying cut-function *if* f^G *satisfies the following constraints for every graph* G *and* $S \subseteq V(G)$*:*

(1) $\mathsf{f}^G(S) = \mathsf{f}^G(\overline{S})$ *and* $\mathsf{f}^G(S)$ *depends only on the graph* $G[S, \overline{S}]$*.*

(2) $\mathsf{f}^G(S) = 0$ *if* $G[S, \overline{S}]$ *has no edges and at least one otherwise.*

(3) Removing a vertex $x \in \overline{S}$ *does not increase* $\mathsf{f}^G(S)$*, and reduce* $\mathsf{f}^G(S)$ *by at most one.*

(4) If $G[S, \overline{S}]$ *is the disjoint union of* $G_1[A_1, B_1]$ *and* $G_2[A_2, B_2]$*, then* $\mathsf{f}^G(S) = \mathsf{f}^{G_1}(A_1) + \mathsf{f}^{G_2}(A_2)$*.*

(5) If $v \in \overline{S}$ *has a twin vertex*[1] *in* $G[S, \overline{S}]$*, then* $\mathsf{f}^G(S) = \mathsf{f}^{G[V(G)\setminus v]}(S)$*.*

Except mw all the set functions we deal with in this thesis are $\mathcal{C}$-satisfying cut-functions. In particular, it is quite easy to check that $\mathsf{rw}^G_{\mathcal{B}}$ and $\mathsf{rw}^G_{\mathbb{N}}$ are two $\mathcal{C}$-satisfying cut-functions. This follows from the fact that for every graph G and $A \subseteq V(G)$, we have $\mathsf{rw}_{\mathcal{B}}(A) = \mathsf{bic\text{-}cover}(A)$ and $\mathsf{rw}_{\mathbb{N}}(A) = \mathsf{bic\text{-}part}(A)$.

Sæther and Vatshelle [128] proved the following theorem on the computation of $\mathcal{C}$-satisfying cut functions.

Theorem 3.6 ([128])**.** *Given a graph* G*, a subset* $A \subseteq V(G)$*,* $k \in \mathbb{N}^+$ *and a* $\mathcal{C}$*-satisfying function* $\mathsf{f}^G : 2^{V(G)} \to \mathbb{R}$*, we can construct in polynomial time, a graph* G^A *such that the* f*-width of* G^A *is at most* $k + \lfloor k \rfloor + 1$ *if and only if* $\mathsf{f}(A) \leq k$*.*

Deciding whether $\mathsf{rw}_{\mathcal{B}}(A) \leq k$ and $\mathsf{rw}_{\mathbb{N}}(A) \leq k$ for some $k \in \mathbb{N}$ is known to be NP-hard from [90, 114]. Moreover, Chandran, Issac, and Karrenbauer [22] have proved that, given a graph G, $A \subseteq V(G)$ and $k \in \mathbb{N}$, there is no $2^{2^{o(k)}} \cdot n^{O(1)}$ time algorithm that decides whether $\mathsf{rw}_{\mathcal{B}}(A) \leq k$ unless ETH fails. From these hardness results and Theorem 3.6, we deduce the following theorem on the problems $\mathsf{rw}_{\mathcal{B}}$-Width and $\mathsf{rw}_{\mathbb{N}}$-Width.

Theorem 3.7. *Both* $\mathsf{rw}_{\mathcal{B}}$-Width *and* $\mathsf{rw}_{\mathbb{N}}$-Width *are* NP*-hard. Moreover, there is no* $2^{2^{o(k)}} \cdot n^{O(1)}$ *time algorithm for* $\mathsf{rw}_{\mathcal{B}}$-Width *unless* ETH *fails.*

3.4 Relation with other width measures

In this section, we compare the values of $\mathcal{B}$-rank-width and $\mathbb{N}$-rank-width with other width measures such as module-width and rank-width.

Theorem 3.8. *For any graph* G *and every* $A \subseteq V(G)$*, we have*

(a) $\mathsf{rw}_{\mathbb{N}}(A) \leq \mathsf{mw}(A)$*,*

(b) $\mathsf{mw}(A) \leq 2^{\mathsf{rw}_{\mathcal{B}}(A)}$*,*

(c) $\mathsf{rw}_{\mathbb{Q}}(A) \leq \mathsf{rw}_{\mathbb{N}}(A)$*,*

(d) $\mathsf{boolw}(A) \leq \mathsf{rw}_{\mathcal{B}}(A)$ *and* $\mathsf{nec}_1(A) \leq 2^{\mathsf{rw}_{\mathcal{B}}(A)}$*,*

(e) $\mathsf{s\text{-}nec}_d(A) \leq 2^{d \cdot \mathsf{rw}_{\mathcal{B}}(A)^2}$ *for every* $d \in \mathbb{N}^+$*,*

[1] A twin vertex of a vertex v is a vertex with the same closed neighborhood as v.

(f) $\mathsf{s\text{-}nec}_d(A) \leq (d+1)^{\mathsf{rw}_{\mathbb{N}}(A)}$ *for every* $d \in \mathbb{N}^+$.

Proof. Let $A \subseteq V(G)$ and $t = |\overline{A}|$. Upper bound (a) follows from the fact that $\mathsf{mw}(A)$ equals the number of different rows in the matrix $M_{A,\overline{A}}$ which is trivially an upper bound on $\mathsf{rw}_{\mathbb{N}}(A)$. Upper bound (b) is due to the fact that the size of the span of a $\mathcal{B}$-generator of size k is at most 2^k. Consequently, the number of different rows of $M_{A,\overline{A}}$ is at most $2^{\mathsf{rw}_{\mathcal{B}}(A)}$ and thus $\mathsf{mw}(A) \leq 2^{\mathsf{rw}_{\mathcal{B}}(A)}$.

Concerning Upper bound (c), observe that, since $\mathbb{N} \subseteq \mathbb{Q}$, for every $t \in \mathbb{N}$ and $\mathcal{S} \subseteq \mathbb{N}^t$, we have $\langle \mathcal{S}, (\mathbb{N}, +, \cdot)\rangle \subseteq \langle \mathcal{S}, (\mathbb{Q}, +, \cdot)\rangle$. Since $\mathsf{rw}_{\mathbb{Q}}(A)$ equals the size of a minimum size of $(\mathbb{Q}, +, \cdot)$-generator of $M_{A,\overline{A}}$, we deduce that $\mathsf{rw}_{\mathbb{Q}}(A) \leq \mathsf{rw}_{\mathbb{N}}(A)$.

For Upper bound (d), notice that any sum of rows of $M_{A,\overline{A}}$ over $\mathcal{B}$ belong to the span of a $\mathcal{B}$-generator of $M_{A,\overline{A}}$. As $\mathsf{nec}_1(A) := |\{N(X) \cap \overline{A} \,:\, X \subseteq A\}|$, we deduce that $\mathsf{nec}_1(A) \leq 2^{\mathsf{rw}_{\mathcal{B}}(A)}$. As $\mathsf{boolw}(A) := \log_2(\mathsf{nec}_1)$, we have that $\mathsf{boolw}(A) \leq \mathsf{rw}_{\mathcal{B}}(A)$.

Upper bound (e) is implied by Upper bound (d) and Theorem 2.33(e), i.e., $\mathsf{s\text{-}nec}_d(A) \leq 2^{d \cdot \mathsf{boolw}(A)^2}$.

The last upper bound follows from the fact that any sum of rows of $M_{A,\overline{A}}$ over $\mathbb{N}$ belong to the span of an $(\mathbb{N}, +, \cdot)$-generator of $M_{A,\overline{A}}$. Let $\mathcal{S}$ be a minimum $(\mathbb{N}, +, \cdot)$-generator of $M_{A,\overline{A}}$. we deduce that $\mathsf{nec}_d(A) \leq (d+1)^{\mathsf{rw}_{\mathbb{N}}(A)}$ from the following inequality

$$\mathsf{nec}_d(A) \leq |\{0, \ldots, d\}^t \cap \langle \mathcal{S}, (\mathbb{N}, +, \cdot)\rangle\,| \leq (d+1)^{|\mathcal{S}|} = (d+1)^{\mathsf{rw}_{\mathbb{N}}(A)}.$$

□

From Theorem 3.8 and the results presented in Section 2.3, we deduce the following upper bounds on the value of $\mathsf{rw}_{\mathcal{B}}$ and $\mathsf{rw}_{\mathbb{N}}$ over vertex cuts.

Corollary 3.9. *For any graph* G *and every* $A \subseteq V(G)$*, we have:*

(a) $\mathsf{rw}(A) \leq \mathsf{rw}_{\mathbb{Q}}(A) \leq \mathsf{rw}_{\mathbb{N}}(A) \leq \mathsf{mw}(A) \leq 2^{\mathsf{rw}_{\mathcal{B}}(A)}$.

(b) $\mathsf{rw}_{\mathcal{B}}(A) \leq \mathsf{rw}_{\mathbb{N}}(A) \leq \mathsf{mw}(A) \leq 2^{\mathsf{rw}(A)}$,

Proof. By Theorem 2.23, we know that for every $A \subseteq V(G)$, we have $\mathsf{mw}(A) \leq 2^{\mathsf{rw}(A)}$. By Observation 3.3, we know that $\mathsf{rw}_{\mathcal{B}}(A) \leq \mathsf{rw}_{\mathbb{N}}(A)$. To deduce (a) we need the fact that $\mathsf{rw}(A) \leq \mathsf{rw}_{\mathbb{Q}}(A)$ from Theorem 2.25 and Upper bounds (b) and (c) of Theorem 3.8. We deduce (b) from these inequalities and from Theorem 3.8(a). □

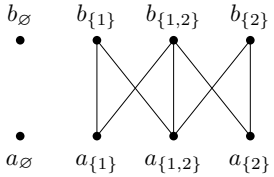
The following theorem proves that Inequality (a) of Corollary 3.9 is tight over vertex cuts.

Theorem 3.10. *For every* $k \in \mathbb{N}$*, there exist a graph* G_k *and* $A \subseteq V(G)$ *such that* $\mathsf{rw}_{\mathcal{B}}(A) = k$ *and* $\mathsf{rw}(A) = 2^k - 1$.

Proof. For every $k \in \mathbb{N}$, we define the set $A_k := \{a_S \,:\, S \subseteq [k]\}$, $B_k := \{b_S \,:\, S \subseteq [k]\}$ and the the graph $G_k = (A_k \cup B_k, \{\{a_S, b_T\} \,:\, S \cap T \neq \varnothing\})$. For instance, Figure 3.1 shows G_2, this graph is known as the *domino* graph.

We claim that $\mathsf{rw}_{\mathcal{B}}(A_k) = k$ and $\mathsf{rw}(A_k) = 2^k$. By definition of G_k, for every $S \subseteq \{1, \ldots, k\}$, we have $N(a_S) = \bigcup_{s \in S} N(a_{\{s\}})$. Consequently, the set of row vectors of $M_{A,B}$ associated with the vertices $a_{\{1\}}, \ldots, a_{\{k\}}$ is a $\mathcal{B}$-generator of $M_{A,B}$, we deduce that $\mathsf{rw}_{\mathcal{B}}(A) = k$.

Now, we prove that $\mathsf{rw}(A) = 2^k$. For a binary matrix M, let us denote by $\mathsf{rw}(M)$ the rank of M over $GF(2)$. For every $k \in \mathbb{N}$, we denote by M_k the matrix M_{A_k,B_k}, We also consider $\overline{M}_k$ the matrix obtained from M_k such that for every $S, T \subseteq [k]$, we have $\overline{M}_k[a_S, b_T] = 1 - M_k[a_S, b_T]$, i.e., $\overline{M}[a_S, b_T] = 1$ if $S \cap T = \varnothing$ and 0 otherwise. Figure 3.1 shows the matrices M_1 and $\overline{M}_1$.



(a) The graph G_2

$$\begin{array}{c} \\ a_\varnothing \\ a_{\{1\}} \end{array} \begin{array}{c} \begin{array}{cc} b_\varnothing & b_{\{1\}} \end{array} \\ \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \end{array}$$

(b) The matrix M_1

$$\begin{array}{c} \\ a_\varnothing \\ a_{\{1\}} \end{array} \begin{array}{c} \begin{array}{cc} b_\varnothing & b_{\{1\}} \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \end{array}$$

(c) The matrix $\overline{M}_1$

Figure 3.1 – Graph G_k with $k = 2$ and the matrices M_1 and $\overline{M}_1$.

We want to prove by recurrence on k that $\mathsf{rw}(M_k) = 2^k - 1$ and $\mathsf{rw}(\overline{M_k}) = 2^k$ for every $k \in \mathbb{N}$. Obviously, this is true for M_0 and $\overline{M_0}$ since $\mathsf{rw}(M_0) = 0$ and $\mathsf{rw}(\overline{M_0}) = 1$.

For applying the recurrence hypothesis, we consider the total order $<$ on the subsets of $\mathbb{N}^+$ where $S < T$ if $\mathsf{max}(S \triangle T) \in T$. For example, we have $\varnothing < \{1\} < \{2\} < \{1,2\} < \{3\} < \{1,3\} < \{2,3\} < \{1,2,3\}$. For every $k \in \mathbb{N}$, we assume that the ith row (resp. column) of M_k and $\overline{M}_k$ are associated, respectively, with a_S (resp. b_S) where S is the ith smallest subset of $[k]$ w.r.t. $<$. This way, for every k, the first $|A_k|/2 = 2^{k-1}$ rows (resp. columns) of M_k and $\overline{M}_k$ are associated with the vertices a_S (resp. b_S) that belongs to A_{k-1} (resp. B_{k-1}), i.e., we have $S \subseteq [k-1]$. On the other hand, the last 2^{k-1} rows (resp. columns) of M_k and $\overline{M}_k$ are associated with the vertices a_S (resp. b_S) with $S = T \cup \{k\}$ with $T \subseteq [k-1]$. By definition, we deduce that, for every $k \in \mathbb{N}^+$, we have

$$M_k = \left(\begin{array}{c|c} M_{k-1} & M_{k-1} \\ \hline M_{k-1} & 1 \end{array}\right) \text{ and } \overline{M}_k = \left(\begin{array}{c|c} \overline{M}_{k-1} & \overline{M}_{k-1} \\ \hline \overline{M}_{k-1} & 0 \end{array}\right).$$

Since rows and columns additions do not change the rank of a matrix and, for every $(a_S, b_T) \in A_k \times B_k$, we have $\overline{M}_k[a_S, b_T] = 1 - M_k[a_S, b_T]$, we deduce the following

$$\mathsf{rw}(M_k) = \mathsf{rw}\left(\begin{array}{c|c} M_{k-1} & M_{k-1} \\ \hline 0 & \overline{M}_{k-1} \end{array}\right) = \mathsf{rw}\left(\begin{array}{c|c} M_{k-1} & 0 \\ \hline 0 & \overline{M}_{k-1} \end{array}\right)$$

Similarly, we deduce the following for $\overline{M}_k$.

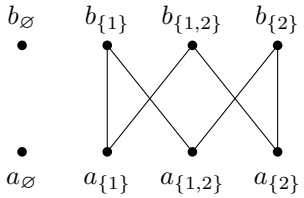
$$\mathsf{rw}(\overline{M}_k) = \mathsf{rw}\left(\begin{array}{c|c} \overline{M}_{k-1} & 0 \\ \hline 0 & \overline{M}_{k-1} \end{array}\right).$$

It follows that the rank of M_k equals the rank of M_{k-1} plus the rank of $\overline{M}_{k-1}$ and the rank of $\overline{M}_k$ equals the double of the rank of $\overline{M}_{k-1}$. By recurrence hypothesis, we deduce that the rank of M_k and $\overline{M}_k$ are, respectively, $2^k - 1$ and 2^k. □

Theorem 3.10 raises the following question which asks whether Inequality (a) of Corollary 3.9 is also tight on graphs.

Open Question 3.11. *For every $k \in \mathbb{N}$, can we construct a graph G with $\mathsf{rw}_{\mathcal{B}}(G) \in O(k)$ and $\mathsf{rw}(G) \in 2^{\Omega(k)}$?*

We suspect that Inequality (b) of Corollary 3.9 is tight (at least over vertex cuts). Unfortunately, we were not able to prove this intuition. However, we think that we can prove it by using the same graphs which have been used in [17] for proving Theorem 2.30. That is, the



(a) The graph H_2

$$\begin{array}{c} \\ a_\varnothing \\ a_{\{1\}} \\ a_{\{2\}} \\ a_{\{1,2\}} \end{array} \begin{array}{c} \begin{array}{cccc} b_\varnothing & b_{\{1\}} & b_{\{2\}} & b_{\{1,2\}} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

(b) The matrix M_{A_2,B_2}

Figure 3.2 – Graph H_k with $k = 2$ and the adjacency matrix between A_2 and B_2.

graphs $\{H_k : k \in \mathbb{N}\}$ where, for every $k \in \mathbb{N}$, we have $V(H_k) = \{a_S, b_S : S \subseteq [k]\}$ and $E(H_k) = \{\{a_S, b_T\} : |S \cap T| \text{ is odd}\}$. Figure 3.2 represents the graph H_2 and the adjacency matrix between A_2 and B_2.

Bui-Xuan, Telle et Vatshelle proved that, for every $k \in \mathbb{N}$, we have $\mathsf{rw}(\{a_S : S \subseteq [k]\}) = k$. To prove that Inequality (b) of Corollary 3.9 is tight, it is sufficient to prove the following conjecture.

Conjecture 3.12. *For every $k \in \mathbb{N}$, we have $\mathsf{rw}_{\mathcal{B}}(A_k) = 2^k - 1$.*

To prove this conjecture, it is sufficient to prove that, for every k and every biclique B of H_k, we have $|E(B)| \leq 2^{k-1}$. Indeed, it is easy to see that $|E(H_k)| = (2^{k-1})^2$, hence if every biclique has at least 2^{k-1} edges, then any biclique cover of H_k contains at most 2^{k-1} bicliques.

3.5 Conclusion

We have explored the properties of $\mathcal{B}$-rank-width and $\mathbb{N}$-rank-width; two new width measures defined from rank-like notions on semirings. They turned out to be equivalent to rank-width and clique-width for $\leq_f$. That is, they have the same FPT power. We compared the value of their associated set functions with the values of $\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{boolw}$ and $\mathsf{s\text{-}nec}_d$. In particular, we proved that $\mathsf{rw}_{\mathcal{B}}(A) \leq 2^{\mathsf{rw}(A)}$ and $\mathsf{rw}(A) \leq 2^{\mathsf{rw}_{\mathcal{B}}(A)}$, for every graph G and $A \subseteq V(G)$.

These properties could make these width measures interesting to obtain efficient running times. But, as we have seen, computing these two width measures is a tough problem. In particular, ETH rules out the existence of an efficient FPT algorithm for computing the $\mathcal{B}$-rank-width.

In front of these results, we must admit that we are quite disappointed by these two new width measures: they do not bring anything new in terms of FPT power and their computations seem to be a tough challenge. It would be interesting to design new matrix decompositions and to check whether we can design some interesting width measures or to prove that all the width measures we can obtain from matrix decompositions are equivalent to clique-width for $\leq_f$.

Chapter 4

Fast algorithms for many connectivity problems

In this chapter, we present our results concerning connectivity problems such as the connected and acyclic variants of (σ, ρ)-Dominating Set problems, and Hamiltonian Cycle. We refer to Section 1.4 for the definitions of the families of problems Connected (σ, ρ)-Dominating Set problems, Acyclic (σ, ρ)-Dominating Set problems, and AC-(σ, ρ)-Dominating Set problems. Some examples of problems that belong to these families are shown in Table 1.1.

In Section 4.1, we develop a framework of the same flavor as the rank-based approach of [9], but suitable for clique-width. We use this framework to design $2^{O(k)} \cdot n$ time algorithms for Connected (σ, ρ)-Dominating Set problems and the Feedback Vertex Set problem given a k-expression.

In Section 4.2, we present a framework that generalizes and simplifies the rank-based approach from [9] and the results developed in Section 4.1. The d-neighbor equivalence relation is at the core of this framework. We use it to solve any problem that belongs to the following families of problems:

- Connected (σ, ρ)-Dominating Set problems,
- Acyclic (σ, ρ)-Dominating Set problems,
- AC-(σ, ρ)-Dominating Set problems.

Through this framework, we obtain the best known algorithms[1] for the problems which belong to these families for the parameters clique-width, ($\mathbb{Q}$-)rank-width, boolean-width, and mim-width.

In Section 4.3, we show that we can use the ideas of Section 4.2 to generalize and simplify the Cut & Count approach of [37]. As a result, we obtain a $\mathsf{s\text{-}nec}_d(\mathcal{L})^{O(1)} \cdot n^{O(1)}$ time algorithm for any Connected (σ, ρ)-Dominating Set problem with d a constant that depends only on (σ, ρ).

In Section 4.4, we present an XP algorithm that solves Hamiltonian Cycle in time $n^{O(k)}$ given a k-expression as input. This algorithm beats the naive algorithm of Espelage et al. [49] that runs in time $n^{\mathcal{O}(k^2)}$. Moreover, the running time of our algorithm matches asymptotically the lower bound of Fomin et al. [59] which states that there is no $f(k) \cdot n^{o(k)}$ time algorithm for Hamiltonian Cycle, unless ETH fails.

[1] Up to a constant in the exponents.

4.1 Connectivity Problems Parameterized by Clique-Width

In this section, we prove that given an n-vertex graph G and a k-expression of G, we can solve any (σ, ρ)-Dominating Set problem and the Feedback Vertex Set problem in time $2^{O(k)} \cdot n$.

Let us explain the ideas of the algorithms with two examples: (1)Connected Dominating Set and (2) Maximum Induced Tree.

Let G be a graph, for which a k-expression ϕ is given, and let H be a k-labeled graph arising in ϕ. It is worth noticing that at the time we are processing the set H, may be not all the edges between the vertices of H are known (those edges may be added in the forthcoming $add_{i,j}$ operations). To facilitate the steps of the dynamic programming algorithm, we first assume that ϕ is an irredundant k-expression.

1. Let D be a connected dominating set of G and let $D_H := D \cap V(H)$. First, D is not necessarily connected, neither a dominating set of H. So, the usual dynamic programming algorithm keeps, for each such D_H, the pair (R, R') of sequences over $\{0,1\}^k$, where $R_i := 1$ if and only if D_H contains a vertex labeled i, and $R'_i := 1$ if and only if $V(H)$ has a vertex labeled i not dominated by D_X. One first observes that if D_H and D'_H have the same pair of sequences (R, R'), then $D_H \cup D_Y$ is a dominating set of G if and only if $D'_H \cup D_Y$ is a dominating set. Therefore, it is sufficient to keep for each pair (R, R') of sequences in $\{0,1\}^k$ the possible partitions of $\{1, \ldots, k\}$ corresponding, informally, to the connected components of the graphs induced by the D_H's, and for each possible partition, the maximum weight among all corresponding D_H's. Notice that the graphs $H[D_H]$ are not necessarily induced subgraphs of G. One easily checks that these tables can be updated without difficulty following the clique-width operations in time $k^{O(k)} \cdot n^{O(1)}$.

   In order to obtain $2^{O(k)} \cdot n$ time algorithms, we modify this algorithm so that we do not guess, via R', the existence of vertices labeled i that are not dominated, but rather the existence of a vertex that will dominate the vertices labeled i (even if they are already dominated in H by D_H). The partition associated with a partial solution D_H takes into account these guesses by considering the i-vertices of D_H as one vertex if $R'_i = 1$ for each $i \in [k]$. With these modifications, the steps of our dynamic programming algorithms can be described in terms of the operations on partitions defined in [9]. We can therefore use the same notion of representativity in order to reduce the time complexity.

2. We consider this example because we reduce the computation of a minimum feedback vertex set to that of a maximum tree. We first observe that we cannot use the same trick as in [9] to ensure the acyclicity, that is counting the number of edges induced by the partial solutions. Indeed, whenever an $add_{i,j}$ operation is used, many edges can be added at the same time. Hence, counting the edges induced by a partial solution would imply to know, for each partial solution, the number of vertices labeled i, for each i. But, this automatically leads to an $n^{O(k)}$ time algorithm. We overcome this difficulty by first defining a binary relation $\mathsf{acyclic}$ on partitions where $\mathsf{acyclic}(p, q)$ holds whenever there are forests E and F, on the same vertex set, such that $E \cup F$ is a forest, and p (resp. q) corresponds to the connected components of E (resp. F). In a second step, we redefine some of the operations on partitions defined in [9] in order to deal with the acyclicity. These operations are used to describe the steps of the algorithm. They informally help updating the partitions after each clique-width operation by detecting partial solutions that may contain cycles. We also define a new notion of representativity, *ac-representativity*, where $\mathcal{S}'$ ac-represents $\mathcal{S}$ if, whenever there is $S \in \mathcal{S}$ that can be completed into a connected acyclic set, there is $S' \in \mathcal{S}'$ that can be completed into a connected acyclic set. We then prove that one can also compute an ac-representative set of size $2^{O(k)}$, assuming the partitions are on $\{1, \ldots, k\}$.

It remains now to describe the tables of the dynamic programming algorithm. First, we are tempted to keep for each forest F of H, the partition induced by the transitive closure of $\sim$ where $i \sim j$ whenever there is a vertex x labeled i that is connected in F to a vertex y labeled j. However, this is not sufficient because we may have in a same connected component two vertices labeled i, and any forthcoming $add_{i,j}$ operation will create a cycle in F if there is a vertex labeled j in F. To overcome this difficulty, we index our dynamic programming tables with functions s that inform, for each i, whether there is a vertex labeled i in the partial solutions, and if yes, in exactly 1 vertex, or in at least two vertices. Indeed, knowing the existence of at least two vertices is sufficient to detect some cycles when we encounter an $add_{i,j}$ operation. We need also to make a difference when we have all the vertices labeled i in different connected components, or when at least two are in a same connected component. This will allow to detect all the partial solutions F that may contain triangles or cycles on four vertices with a forthcoming $add_{i,j}$ operation. Such cycles cannot be detected by the acyclic binary relation on partitions since this latter does not keep track of the number of vertices in each label. However, the other kinds of cycles are detected through the acyclic binary relation. We refer the reader to Subsection 4.1.2 for a more detailed description of the algorithm.

As explained in Section 2.6.2, our algorithms are optimal under ETH. Indeed, unless ETH fails, there is no $2^{o(k)} \cdot n^{O(1)}$ time algorithm for the following problems Feedback Vertex Set, Connected Dominating Set, Connected Vertex Cover and many other NP-hard Connected (σ, ρ)-Dominating Set problems.

The remainder of the section is organized as follows. The notion of *ac-representativity* and the modified operations on partitions are given in Subsection 4.1.1. We also propose the algorithm for computing an ac-representative set of size $2^{O(|L|)}$, for sets of weighted partitions on a finite set L. The algorithms for Feedback Vertex Set and the Connected (σ, ρ)-Dominating Set problems are given in, respectively, Subsections 4.1.2 and 4.1.3.

4.1.1 Representing Sets of Acyclic Weighted Partitions by Matrices

In this section, we manipulate partitions and for doing so, we use the following notations. A partition p of a set S is a collection of non-empty subsets of S that are pairwise non-intersecting and such that $\bigcup_{p_i \in p} p_i = S$; each set in p is called a *block* of p. The set of partitions of a finite set S is denoted by $\Pi(S)$, and $(\Pi(S), \sqsubseteq)$ forms a lattice where $p \sqsubseteq q$ if for each block p_i of p there is a block q_j of q with $p_i \subseteq q_j$. The join operation of this lattice is denoted by $\sqcup$. For example, we have

$$\{\{1,2\},\{3,4\},\{5\}\} \sqcup \{\{1\},\{2,3\},\{4\},\{5\}\} = \{\{1,2,3,4\},\{5\}\}.$$

Let $\#\text{block}(p)$ denote the number of blocks of a partition p. Observe that $\varnothing$ is the only partition of the empty set. A *weighted partition* is an element of $\Pi(S) \times \mathbb{N}$ for some finite set S.

For $p \in \Pi(S)$ and $X \subseteq S$, let $p_{\downarrow X} \in \Pi(X)$ be the partition $\{p_i \cap X : p_i \in p\} \setminus \{\varnothing\}$, and for $Y \supseteq S$, let $p_{\uparrow Y} \in \Pi(Y)$ be the partition $p \cup \left(\bigcup_{y \in Y \setminus S} \{\{y\}\}\right)$.

We recall that a *weighted partition* is an element of $\Pi(L) \times \mathbb{N}$ for some finite set L. Our algorithms compute a set of weighted partitions $\mathcal{A} \subseteq \Pi(L) \times \mathbb{N}$, for each labeled graph H used in the k-expression of the given graph G and for every subset $L \subseteq \{i \in [k] : lab_H^{-1}(i) \neq \varnothing\}$. Each weighted partition $(p, w) \in \mathcal{A} \subseteq \Pi(L) \times \mathbb{N}$ is intended to mean the following: there is a solution $S \subseteq V(H)$ of weight w such that p is the transitive closure of the following equivalence relation $\sim$ on L: $i \sim j$ if there exist an i-vertex and a j-vertex in the same component of $H[S]$. Moreover, for every label i in L, there is at least one i-vertex in S. For each label i in L, we expect the i-vertices of S to have an additional neighbor in any extension of S into an optimum

solution. This way, for each label $i \in L$, we can consider the i-vertices of S as one vertex in terms of connectivity, since they will have a common neighbor in any extension of S. On the other hand, the labels $j \in [k] \setminus L$ such that S contains at least one j-vertex are expected to have no additional neighbor in any extension of S into an optimum solution. Consequently, the vertices in S with a label in $[k] \setminus L$ do no longer play a role in the connectivity. These expectations allow us to represent the connected components of $H[S]$ by p. Our algorithms will guarantee that the weighted partitions computed from (p, w) are computed accordingly to these expectations.

When considering the Feedback Vertex Set problem, as said in the introduction, the trick used in [9] to deal with acyclicity and that consists in counting the number of edges induced by the partial solutions would yield an $n^{O(k)}$ time algorithm in the case of clique-width. Since the partial solutions for the Feedback Vertex Set problem are represented by weighted partitions, we need to certify that whenever we join two weighted partitions and keep it as a partial solution, it does not correspond to a partial solution with cycles. We introduce in the following a notion of acyclicity between two partitions so that we can identify the joins of partitions which do not produce cycles.

Definition 4.1. *Let L be a finite set. We let* acy *be the relation on $\Pi(L) \times \Pi(L)$ where $\mathsf{acy}(p, q)$ holds exactly when $|L| + \#\text{block}(p \sqcup q) - (\#\text{block}(p) + \#\text{block}(q)) = 0$.*

Observe that, if $F_p := (L, E_p)$ and $F_q := (L, E_q)$ are forests, then $\mathsf{acy}(\mathsf{cc}(F_p), \mathsf{cc}(F_q))$ holds if and only if $E_p \cap E_q = \varnothing$ and $(L, E_p \uplus E_q)$ is a forest. The following is then quite easy to deduce.

Fact 4.2. *Let L be a finite set. For all partitions $p, q, r \in \Pi(L)$,*

$$\mathsf{acy}(p, q) \wedge \mathsf{acy}(p \sqcup q, r) \Leftrightarrow \mathsf{acy}(q, r) \wedge \mathsf{acy}(p, q \sqcup r).$$

Proof. For a partition $p \in \Pi(L)$, let $f(p) := |L| - \#\text{block}(p)$. One easily checks that $\mathsf{acy}(p, q)$ holds if and only if $f(p \sqcup q)$ equals $f(p) + f(q)$. One can therefore deduce, by an easy calculation from this equivalence, that $\mathsf{acy}(p \sqcup q, r) \wedge \mathsf{acy}(p, q)$ is equivalent to saying that $f(p \sqcup q \sqcup r)$ equals $f(p) + f(q) + f(r)$. The same statement holds for $\mathsf{acy}(q, r) \wedge \mathsf{acy}(p, q \sqcup r)$. □

By definition of acy and of $\sqcup$, we can also observe the following.

Fact 4.3. *Let L be a finite set. Let $q \in \Pi(L)$ and let $X \subseteq L$ such that no subset of X is a block of q. Then, for each $p \in \Pi(L \setminus X)$, we can observe the following equivalences*

$$p_{\uparrow X} \sqcup q = \{L\} \Longleftrightarrow p \sqcup q_{\downarrow(L \setminus X)} = \{L \setminus X\} \qquad \textit{and} \tag{4.1}$$

$$\mathsf{acy}(p_{\uparrow X}, q) \Longleftrightarrow \mathsf{acy}(p, q_{\downarrow(L \setminus X)}). \tag{4.2}$$

We modify in this section the operators on weighted partitions defined in [9] in order to express our dynamic programming algorithms in terms of these operators, and also to deal with acyclicity. Let L be a finite set. First, for $\mathcal{A} \subseteq \Pi(L) \times \mathbb{N}$, let

$$\mathsf{rmc}(\mathcal{A}) := \{(p, w) \in \mathcal{A} : \forall (p, w') \in \mathcal{A}, w' \leq w)\}.$$

This operator, defined in [9], is used to remove all the partial solutions whose weights are not maximum w.r.t. to their partitions.

Ac-Join. Let L' be a finite set. For $\mathcal{A} \subseteq \Pi(L) \times \mathbb{N}$ and $\mathcal{B} \subseteq \Pi(L') \times \mathbb{N}$, we define $\mathsf{acjoin}(\mathcal{A}, \mathcal{B}) \subseteq \Pi(L \cup L') \times \mathbb{N}$ as

$$\mathsf{acjoin}(\mathcal{A}, \mathcal{B}) := \{(p_{\uparrow L'} \sqcup q_{\uparrow L}, w_1 + w_2) : (p, w_1) \in \mathcal{A}, (q, w_2) \in \mathcal{B} \text{ and } \mathsf{acy}(p_{\uparrow L'}, q_{\uparrow L})\}.$$

This operator is more or less the same as the one in [9], except that we incorporate the acyclicity condition. It is used to construct partial solutions while guaranteeing the acyclicity.

Project. For $X \subseteq L$ and $\mathcal{A} \subseteq \Pi(L) \times \mathbb{N}$, let $\mathsf{proj}(\mathcal{A}, X) \subseteq \Pi(L \setminus X) \times \mathbb{N}$ be

$$\mathsf{proj}(\mathcal{A}, X) := \{(p_{\downarrow(L \setminus X)}, w) \,:\, (p, w) \in \mathcal{A} \text{ and } \forall p_i \in p, (p_i \setminus X) \neq \varnothing\}.$$

This operator considers all the partitions such that no block is completely contained in X, and then removes X from those partitions. We index our dynamic programming tables with functions that inform on the label classes playing a role in the connectivity of partial solutions, and this operator is used to remove from the partitions the label classes that are required to no longer play a role in the connectivity of the partial solutions. If a partition has a block fully contained in X, it means that this block will remain disconnected in the future steps of our dynamic programming algorithm, and that is why we remove such partitions (besides those with cycles).

One needs to perform the above operations efficiently, and this is guaranteed by the following, which assumes that $\log(|\mathcal{A}|) \leq |L|^{O(1)}$ for each $\mathcal{A} \subseteq \Pi(L) \times \mathbb{N}$ (this can be established by applying the operator rmc). The following proposition could be easily proved.

Proposition 4.4. *The operator* acjoin *can be performed in time* $|\mathcal{A}| \cdot |\mathcal{B}| \cdot |L \cup L'|^{O(1)}$ *and the size of its output is upper-bounded by* $|\mathcal{A}| \cdot |\mathcal{B}|$*. The operators* rmc *and* proj *can be performed in time* $|\mathcal{A}| \cdot |L|^{O(1)}$*, and the sizes of their outputs are upper-bounded by* $|\mathcal{A}|$*.*

We now define the notion of representative sets of weighted partitions which is the same as the one in [9], except that we need to incorporate the acyclicity condition as for the acjoin operator above.

Definition 4.5. *Let* L *be a finite set and let* $\mathcal{A} \subseteq \Pi(L) \times \mathbb{N}$*. For* $q \in \Pi(L)$*, let*

$$\textbf{ac-opt}(\mathcal{A}, q) := \mathsf{max}\{w \,:\, (p, w) \in \mathcal{A}, p \sqcup q = \{L\} \text{ and } \mathsf{acy}(p, q)\}.$$

A set of weighted partitions $\mathcal{A}' \subseteq \Pi(L) \times \mathbb{N}$ ac-represents $\mathcal{A}$ *if for each* $q \in \Pi(L)$*, it holds that* $\textbf{ac-opt}(\mathcal{A}, q) = \textbf{ac-opt}(\mathcal{A}', q)$*.*

Let Z *and* L' *be two finite sets. A function* $f : 2^{\Pi(L) \times \mathbb{N}} \times Z \to 2^{\Pi(L') \times \mathbb{N}}$ *is said to* preserve ac-representation *if for each* $\mathcal{A}, \mathcal{A}' \subseteq \Pi(L) \times \mathbb{N}$ *and* $z \in Z$*, it holds that* $f(\mathcal{A}', z)$ *ac-represents* $f(\mathcal{A}, z)$ *whenever* $\mathcal{A}'$ *ac-represents* $\mathcal{A}$*.*

At each step of our algorithm, we will compute a small set $\mathcal{S}'$ that ac-represents the set $\mathcal{S}$ containing all the partial solutions. In order to prove that we compute an ac-representative set of $\mathcal{S}$, we show that $\mathcal{S} = f(\mathcal{R}_1, \ldots, \mathcal{R}_t)$ with f a composition of functions that preserve ac-representation, and $\mathcal{R}_1, \ldots, \mathcal{R}_t$ the sets of partial solutions associated with the previous steps of the algorithm. To compute $\mathcal{S}'$, it is sufficient to compute $f(\mathcal{R}'_1, \ldots, \mathcal{R}'_t)$, where each $\mathcal{R}'_i$ is an ac-representative set of $\mathcal{R}_i$. The following lemma guarantees that the operators we use preserve ac-representation.

Lemma 4.6. *The union of two sets in* $2^{\Pi(L) \times \mathbb{N}}$ *and the operators* rmc*,* proj*, and* acjoin *preserve ac-representation.*

Proof. Let L be a finite set and let $\mathcal{A}$ and $\mathcal{A}'$ be two subsets of $\Pi(L) \times \mathbb{N}$. The proof for the union follows directly from the definition of **ac-opt**.

Rmc. Let $q \in \Pi(L)$. By the definition of rmc, whenever $(p, w) \in \mathcal{A}$ is such that $p \sqcup q = \{L\}$, $\mathsf{acy}(p, q)$ and $\textbf{ac-opt}(\mathcal{A}, q) = w$, then $(p, w) \in \mathsf{rmc}(\mathcal{A})$, otherwise there would exist $(p, w') \in \mathcal{A}$ with $w' > w$ which would contradict $w = \textbf{ac-opt}(\mathcal{A}, q)$. Therefore, $\textbf{ac-opt}(\mathsf{rmc}(\mathcal{A}), q) = \textbf{ac-opt}(\mathcal{A}, q)$. We can then conclude that if $\mathcal{A}'$ ac-represents $\mathcal{A}$, it holds that $\mathsf{rmc}(\mathcal{A}')$ ac-represents $\mathsf{rmc}(\mathcal{A})$.

Projections. Because $\mathsf{proj}(\mathcal{A}, X) = \mathsf{proj}(\mathsf{proj}(\mathcal{A}, x), X \setminus \{x\})$ for all $X \subseteq L$ and $x \in X$, we can assume that $X = \{x\}$. Let $q \in \Pi(L \setminus \{x\})$. For every $(p, w) \in \mathcal{A}$, if $\{x\} \in p$, then $p \sqcup q_{\uparrow x} \neq \{L\}$, and $(p_{\downarrow L \setminus x}, w) \notin \mathsf{proj}(\mathcal{A}, \{x\})$. Otherwise, $(p_{\downarrow L \setminus x}, w) \in \mathsf{proj}(\mathcal{A}, \{x\})$, and by Fact 4.3 we have

$$p_{\downarrow L \setminus x} \sqcup q = \{L \setminus \{x\}\} \iff p \sqcup q_{\uparrow x} = \{L\} \qquad \text{and}$$
$$\mathsf{acy}(p_{\downarrow L \setminus x}, q) \iff \mathsf{acy}(p, q_{\uparrow x}).$$

Therefore, we have $\textbf{ac-opt}(\mathsf{proj}(\mathcal{A}, \{x\}), q) = \textbf{ac-opt}(\mathcal{A}, q_{\uparrow x})$. From this equality, we can conclude that $\mathsf{proj}(\mathcal{A}', \{x\})$ ac-represents $\mathsf{proj}(\mathcal{A}, \{x\})$, for all $\mathcal{A}' \subseteq \mathcal{A}$ such that $\mathcal{A}'$ ac-represents $\mathcal{A}$, that is proj preserves ac-representation.

Ac-Join. Let L' be a finite set and let $\mathcal{B} \subseteq \Pi(L') \times \mathbb{N}$. Let $r \in \Pi(L \cup L')$.

Observe that for all $(q, w_2) \in \mathcal{B}$, if a subset of $L' \setminus L$ is a block of $q_{\uparrow L} \sqcup r$, then for all $p \in \Pi(L)$, we have $p_{\uparrow L'} \sqcup q_{\uparrow L} \sqcup r \neq \{L \cup L'\}$. Therefore, $\textbf{ac-opt}(\mathsf{acjoin}(\mathcal{A}, \mathcal{B}), r) = \textbf{ac-opt}(\mathsf{acjoin}(\mathcal{A}, \mathcal{B}'), r)$ where $\mathcal{B}'$ is the set of all $(q, w) \in \mathcal{B}$ such that no subset of $L' \setminus L$ is a block of $q_{\uparrow L} \sqcup r$.

By definition, $\textbf{ac-opt}(\mathsf{acjoin}(\mathcal{A}, \mathcal{B}'), r)$ equals

$$\mathsf{max}\{w_1 + w_2 \,:\, (p, w_1) \in \mathcal{A} \wedge (q, w_2) \in \mathcal{B}' \wedge (p_{\uparrow L'} \sqcup q_{\uparrow L} \sqcup r) = \{L \cup L'\} \\ \wedge \mathsf{acy}(p_{\uparrow L'}, q_{\uparrow L}) \wedge \mathsf{acy}(p_{\uparrow L'} \sqcup q_{\uparrow L}, r)\}.$$

By Fact 4.2, $\textbf{ac-opt}(\mathsf{acjoin}(\mathcal{A}, \mathcal{B}'), r)$ is then equal to

$$\mathsf{max}\{w_1 + w_2 \,:\, (p, w_1) \in \mathcal{A} \wedge (q, w_2) \in \mathcal{B}' \wedge (p_{\uparrow L'} \sqcup q_{\uparrow L} \sqcup r) = \{L \cup L'\} \\ \wedge \mathsf{acy}(q_{\uparrow L}, r) \wedge \mathsf{acy}(p_{\uparrow L'}, q_{\uparrow L} \sqcup r)\}.$$

We deduce, by Fact 4.3 and the definition of $\mathcal{B}'$, that $\textbf{ac-opt}(\mathsf{acjoin}(\mathcal{A}, \mathcal{B}'), r)$ equals

$$\mathsf{max}\{w_1 + w_2 \,:\, (p, w_1) \in \mathcal{A} \wedge (q, w_2) \in \mathcal{B}' \wedge (p \sqcup (q_{\uparrow L} \sqcup r)_{\downarrow L}) = \{L\} \\ \wedge \mathsf{acy}(q_{\uparrow L}, r) \wedge \mathsf{acy}(p, (q_{\uparrow L} \sqcup r)_{\downarrow L})\}.$$

Therefore, we can conclude that $\textbf{ac-opt}(\mathsf{acjoin}(\mathcal{A}, \mathcal{B}), r)$ equals

$$\mathsf{max}\{w_2 + \textbf{ac-opt}(\mathcal{A}, (q_{\uparrow L} \sqcup r)_{\downarrow L}) \,:\, (q, w_2) \in \mathcal{B}' \wedge \mathsf{acy}(q_{\uparrow L}, r)\}.$$

Therefore, if $\mathcal{A}'$ ac-represents $\mathcal{A}$, then we can conclude that $\textbf{ac-opt}(\mathsf{acjoin}(\mathcal{A}, \mathcal{B}), r)$ equals $\textbf{ac-opt}(\mathsf{acjoin}(\mathcal{A}', \mathcal{B}), r)$. As this statement is true for all $r \in \Pi(L \cup L')$, we can conclude that $\mathsf{acjoin}(\mathcal{A}', \mathcal{B})$ ac-represents $\mathsf{acjoin}(\mathcal{A}, \mathcal{B})$ whenever $\mathcal{A}'$ ac-represents $\mathcal{A}$. Symmetrically, we deduce that $\mathsf{acjoin}(\mathcal{A}, \mathcal{B}^\star)$ ac-represents $\mathsf{acjoin}(\mathcal{A}, \mathcal{B})$ whenever $\mathcal{B}^\star$ ac-represents $\mathcal{B}$. □

In the remaining, we will prove that, for every set $\mathcal{A} \subseteq \Pi(L) \times \mathbb{N}$, we can find, in time $|\mathcal{A}| \cdot 2^{O(|L|)}$, a subset $\mathcal{A}' \subseteq \mathcal{A}$ of size at most $|L| \cdot 2^{|L|}$ that ac-represents $\mathcal{A}$. As in [9], we will encode the ac-representativity by a matrix over $\mathbb{F}_2$ and show that this one has rank at most the desired bound. Next, we show that an optimum basis of this matrix ac-represents $\mathcal{A}$ and such a basis can be computed using the following lemma from [9]. The constant ω denotes the matrix multiplication exponent.

Lemma 4.7 ([9]). *Let M be an $n \times m$-matrix over $\mathbb{F}_2$ with $m \leq n$ and let $w : \{1, \ldots, n\} \to \mathbb{N}$ be a weight function. Then, one can find a basis of maximum weight of the row space of M in time $O(nm^{\omega-1})$.*

Theorem 4.8. *There exists an algorithm* $\mathsf{reduce}^{\mathsf{acy}}$ *that, given a set of weighted partitions* $\mathcal{A} \subseteq \Pi(L) \times \mathbb{N}$*, outputs in time* $|\mathcal{A}| \cdot 2^{(\omega-1)\cdot|L|} \cdot |L|^{O(1)}$ *a subset* $\mathcal{A}'$ *of* $\mathcal{A}$ *that ac-represents* $\mathcal{A}$ *and such that* $|\mathcal{A}'| \leq |L| \cdot 2^{|L|-1}$*.*

Proof. If $L = \varnothing$, then it is enough to return $\mathcal{A}' := \{(\varnothing, w)\}$, where $(\varnothing, w) \in \mathcal{A}$ and w is maximum because $\varnothing$ is the only partition of the empty set.

Assume from now that $L \neq \varnothing$ (this will ensure that the following definitions exist). Let us first define the matrix that encodes the property that the join of two partitions corresponds to a partition arising from a connected solution.

Let v_0 be a fixed element of L and let $\mathsf{cuts}(L) := \{(L_1, L_2) : L_1 \uplus L_2 = L \text{ and } v_0 \in L_1\}$. Let M and C be, respectively, a $(\Pi(L), \Pi(L))$-matrix and a $(\Pi(L), \mathsf{cuts}(L))$-matrix, both over $\mathbb{F}_2$, such that

$$M[p, q] := \begin{cases} 0 & \text{if } p \sqcup q \neq \{L\}, \\ 1 & \text{otherwise.} \end{cases}$$

$$C[p, (L_1, L_2)] := \begin{cases} 0 & \text{if } p \not\sqsubseteq (L_1, L_2), \\ 1 & \text{otherwise.} \end{cases}$$

As in [9, 38], we fix an element v_0 to ensure that for all $p \in \Pi(L)$, the number of cuts (L_1, L_2) such that $p \sqcup q \sqsubseteq (L_1, L_2)$ is odd if and only if $p \sqcup q = \{L\}$. In fact, this number equals $2^{\#\text{block}(p)-1}$. This property is used in [9] to prove that $M = C \cdot C^t$.

Let $\mathcal{A}$ be a set of weighted partitions. In order to compute an ac-representative set of $\mathcal{A}$, we will decompose $\mathcal{A}$ into a small number of sets $\mathcal{A}_i$. Then, for each set $\mathcal{A}_i$, we compute a set $\mathcal{A}'_i \subseteq \mathcal{A}_i$ of $\mathcal{A}_i$ such that the union of the sets $\mathcal{A}'_i$ ac-represents $\mathcal{A}$. To compute $\mathcal{A}'_i$, we use Lemma 4.7 to find a maximum basis of the row space of C restricted to $\mathcal{A}_i$.

For each $0 \leq i \leq |L| - 1$, let $\mathcal{A}_i$ be the set $\{p : (p, w) \in \mathcal{A} \text{ and } |L| - \#\text{block}(p) = i\}$, and let $C^i_{\mathcal{A}}$ be the restriction of C to rows in $\mathcal{A}_i$. Let $\mathcal{B}_i$ be a basis of the row space of $C^i_{\mathcal{A}}$ of maximum weight, where the weights are the weights[2] of the considered weighted partitions in $\mathcal{A}$. Observe that $|\mathcal{B}_i| \leq 2^{|L|-1}$ because the rank of $C^i_{\mathcal{A}}$ is bounded by $|\mathsf{cuts}(L)| = 2^{|L|-1}$. For $p \in \mathcal{A}_i$, let $\mathcal{B}_i(p)$ be the subset of $\mathcal{B}_i$ such that $C[p, (L_1, L_2)] = \sum_{q \in \mathcal{B}_i(p)} C[q, (L_1, L_2)]$ for all $(L_1, L_2) \in \mathsf{cuts}(L)$. Let $\mathcal{A}'_i$ be the subset of $\mathcal{A}$ corresponding to the rows in $\mathcal{B}_i$, and let $\mathcal{A}' := \mathcal{A}'_0 \uplus \cdots \uplus \mathcal{A}'_{|L|-1}$. Notice that $|\mathcal{A}'| \leq |L| \cdot 2^{|L|-1}$.

Since $C^i_{\mathcal{A}}$ has $|\mathcal{A}_i|$ rows and $2^{|L|-1}$ columns, $\mathcal{A}_i$ is computable in time $|\mathcal{A}_i| \cdot 2^{|L|-1} \cdot |L|^{O(1)}$. By Lemma 4.7, we can compute $\mathcal{B}_i$ in time $|\mathcal{A}_i| \cdot 2^{(\omega-1)\cdot|L|} \cdot |L|^{O(1)}$. Hence, we can compute $\mathcal{A}'$ in time $|\mathcal{A}| \cdot 2^{(\omega-1)\cdot|L|} \cdot |L|^{O(1)}$ because $\{\mathcal{A}_0, \ldots, \mathcal{A}_{|L|-1}\}$ is a partition of $\{p : (p, w) \in \mathcal{A}\}$.

It remains now to show that $\mathcal{A}'$ ac-represents $\mathcal{A}$. First, let us show that for all $p \in \mathcal{A}_i$ and $r \in \Pi(L)$, if $M[p, r] = 1$, then there is $q \in \mathcal{B}_i(p)$ such that $M[q, r] = 1$. Now, from the equality

[2] We can assume w.l.o.g. that $\mathcal{A} = \mathsf{rmc}(\mathcal{A})$, and thus for each $p \in \mathcal{A}$, there is a unique $w \in \mathbb{N}$ such that $(p, w) \in \mathcal{A}$.

$M = C \cdot C^t$, we have

$$\begin{aligned} M[p,r] &= \sum_{(L_1,L_2)\in \mathsf{cuts}(L)} C[p,(L_1,L_2)] \cdot C^t[(L_1,L_2),r] \\ &= \sum_{(L_1,L_2)\in \mathsf{cuts}(L)} \left(\sum_{q\in \mathcal{B}_i(p)} C[q,(L_1,L_2)] \right) \cdot C^t[(L_1,L_2),r] \\ &= \sum_{q\in \mathcal{B}_i(p)} \left(\sum_{(L_1,L_2)\in \mathsf{cuts}(L)} C[q,(L_1,L_2)] \cdot C^t[(L_1,L_2),r] \right) \\ &= \sum_{q\in \mathcal{B}_i(p)} M[q,r]. \end{aligned}$$

So, $M[p,r] = 1$ if and only if there is an odd number of $q \in \mathcal{B}_i(p)$ such that $M[q,r] = 1$.

Since by construction $\mathcal{A}' \subseteq \mathcal{A}$, for each $q \in \Pi(L)$, we have $\textbf{ac-opt}(\mathcal{A}, q) \geq \textbf{ac-opt}(\mathcal{A}', q)$. Assume towards a contradiction that $\mathcal{A}'$ does not ac-represent $\mathcal{A}$. Thus, there is $q \in \Pi(L)$ such that $\textbf{ac-opt}(\mathcal{A}, q) > \textbf{ac-opt}(\mathcal{A}', q)$, and hence there is $(p, w) \in \mathcal{A} \setminus \mathcal{A}'$ such that $p \sqcup q = \{L\}$, $\mathsf{acy}(p, q)$ holds and $w > \textbf{ac-opt}(\mathcal{A}', q)$. Let $i := |L| - \#\text{block}(p)$. Hence, $p \in \mathcal{A}_i$ and there exists $p' \in \mathcal{B}_i(p)$ such that $M[p', q] = 1$ that is $p' \sqcup q = \{L\}$. Let $(p^\star, w^\star) \in \mathcal{A}'_i$ such that $p^\star \in \mathcal{B}_i(p)$, $p^\star \sqcup q = \{L\}$ and $w^\star$ is maximum.

Since $(p^\star, w^\star) \in \mathcal{A}'_i$, we have $|L| - \#\text{block}(p^\star) = |L| - \#\text{block}(p) = i$. We can conclude that $\mathsf{acy}(p^\star, q)$ holds because $\mathsf{acy}(p, q)$ holds. Indeed, by definition, $\mathsf{acy}(p, q)$ holds if and only if $|L| + \#\text{block}(p \sqcup q) - (\#\text{block}(p) + \#\text{block}(q)) = 0$. Since $p \sqcup q = \{L\}$, we deduce that $\mathsf{acy}(p, q)$ holds if and only if $i = |L| + 1 - \#\text{block}(q)$. Because $p^\star \sqcup q = \{L\}$ and $|L| - \#\text{block}(p^\star) = i$, we can conclude that $\mathsf{acy}(p^\star, q)$ holds.

Hence, we have $\textbf{ac-opt}(\mathcal{A}', q) \geq w^\star$. Since $w > \textbf{ac-opt}(\mathcal{A}', q)$, it must hold that $w > w^\star$. But, $(\mathcal{B}_i \setminus \{p^\star\}) \cup \{p\}$ is also a basis of $C^i_{\mathcal{A}}$ since the set of independent row sets of a matrix forms a matroid. Since $w > w^\star$, the weight of $(\mathcal{B}_i \setminus \{p^\star\}) \cup \{p\}$ is strictly greater than the weight of $\mathcal{B}_i$, yielding a contradiction. □

4.1.2 Feedback Vertex Set

We will use the weighted partitions defined in the previous section to represent the partial solutions. At each step of our algorithm, we will ensure that the stored weighted partitions correspond to acyclic partial solutions. However, the framework in the previous section deals only with connected acyclic solutions. So, instead of computing a maximum induced forest (the complementary of a minimum feedback vertex set), we will compute a maximum induced tree. As in [9], we introduce a hypothetical new vertex that is universal and denoted by v_0, and we compute a pair (F, E_0) so that F is a maximum induced forest of G, E_0 is a subset of edges incident to v_0, and $(V(F) \cup \{v_0\}, E(F) \cup E_0)$ is a tree. In order now to reduce the sizes of the dynamic programming tables, we will express the steps of the algorithm in terms of the operators on weighted partitions defined in the previous section.

Let us explain the idea of the algorithm before defining the dynamic programming tables and the steps. Let H be a k-labeled graph. We are interested in storing *ac-representative sets* of all induced forests of H that may produce a solution. If F is an induced forest of H, we would like to store the partition p corresponding to the quotient set of the transitive closure of the relation $\sim$ on $V(F)$ where $x \sim y$ if x and y have the same label or are in the same connected component. If $J \subseteq [k]$ is such that $\bigcup_{x\in V(F)} lab_H(x) = J$, then this is equivalent to storing the partition p of J where i and j are in the same block if there are, respectively, an i-vertex x and a j-vertex y in the same connected component of F.

Now, if H is used in a k-expression of a k-labeled graph G, then in the clique-width operations defining G we may add edges between the i-vertices and the j-vertices of H, for some $i, j \in J$. Now, this has no effect if there are exactly one i-vertex and one j-vertex at distance one in $H[F]$, otherwise cycles may be created, e.g., whenever an i-vertex and a j-vertex are non-adjacent and belong to the same connected component, or the number of i-vertices and j-vertices are both at least two. Nevertheless, we are not able to handle all these cases with the operators on weighted partitions. To resolve the situation where an i-vertex and a j-vertex are already adjacent, we consider *irredundant k-expressions*, i.e., whenever an operation $add_{i,j}$ is used there are no edges between i-vertices and j-vertices. For the other cases, we index the dynamic programming tables with functions $s : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$ that tell, for each $i \in [k]$, if the label class $lab_H^{-1}(i)$ does not intersect F ($s(i) = \gamma_0$), or if it does, in one vertex ($s(i) = \gamma_1$), or in at least two ($s(i) \in \{\gamma_2, \gamma_{-2}\}$) vertices. We have two possible values for label classes intersecting $V(F)$ in at least two vertices because whenever two i-vertices belong to the same connected component of F, F does not produce a valid solution once an operation $add_{i,\ell}$ is applied to H with $s(\ell) \neq \gamma_0$. So, if a label class $lab_H^{-1}(i)$ intersects $V(F)$ in at least two vertices, since we do not know whether a clique-width operation $add_{i,\ell}$ with $s(\ell) \neq \gamma_0$ will be applied to H, we guess it in the function s, and whenever $s(i)$ equals γ_{-2}, we throw p when we encounter an $add_{i,\ell}$ operation (with $s(\ell) \neq \gamma_0$), and if $s(i) = \gamma_2$, we force F to not having two i-vertices in the same connected component.

Nonetheless, even though we are able to detect the partitions corresponding to induced subgraphs with cycles, taking p as the transitive closure of the relation $\sim$ on $[k]$ described above may detect false cycles. Indeed, let x_i, x_j and x_ℓ, x'_ℓ be, respectively, an i-vertex, a j-vertex and two ℓ-vertices, such that x_i and x_ℓ belong to the same connected component in F, and similarly, x_j and x'_ℓ to another connected component of F. Now, if we apply an operation $add_{i,j}$ on H, we may detect a cycle with p (through the acjoin operation), which may not exist when for instance there are only one i and one j-vertex in F, both in different connected components. We resolve this case with the functions s indexing the dynamic programming tables by forcing each label i in $s^{-1}(\gamma_2)$ to wait for exactly one clique-width operation $add_{i,t}$ for some $t \in [k]$. We, therefore, translate all the acyclicity tests to the acjoin operation. Indeed, the case explained will be no longer a false cycle as x_i and x_j will be adjacent (with the $add_{i,j}$ operation), and we know that x_ℓ and x'_ℓ will be connected to some other vertex in F, and since x_i is connected to x_ℓ and x_j to x'_ℓ, we have a cycle. The following notion of *certificate graph* formalizes this requirement.

Definition 4.9 (Certificate graph of a solution)**.** *Let G be a k-labeled graph, F an induced forest of G, $s : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$, and E_0 a subset of edges incident to v_0. Let β be a bijection from $s^{-1}(\gamma_2)$ to a set V_s^+ disjoint from $V(G) \cup \{v_0\}$. The* certificate graph of (F, E_0) with respect to s, *denoted by* $\mathbf{CG}(F, E_0, s)$, *is the graph $(V(F) \cup V_s^+ \cup \{v_0\}, E(F) \cup E_0 \cup E_s^+)$ with*

$$E_s^+ := \bigcup_{i \in s^{-1}(\gamma_2)} \{\{v, \beta(i)\} : v \in (V(F) \cap lab_G^{-1}(i))\}.$$

In a certificate graph of (F, E_0), the vertices in V_s^+ represent the expected future neighbors of all the vertices in $V(F) \cap lab_G^{-1}(s^{-1}(\gamma_2))$. For convenience, whenever we refer to a vertex v_i^+ of V_s^+, we mean the vertex of V_s^+ adjacent to the i-vertices in $\mathbf{CG}(F, E_0, s)$. See Figure 4.1 for an example of a certificate graph.

We are now ready to define the sets of weighted partitions whose representatives we manipulate in our dynamic programming tables.

Definition 4.10 (Weighted partitions in $\mathcal{A}_G[s]$)**.** *Let G be a k-labeled graph and let $s : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$ be a total function. The entries of $\mathcal{A}_G[s]$ are all weighted partitions $(p, w) \in$*

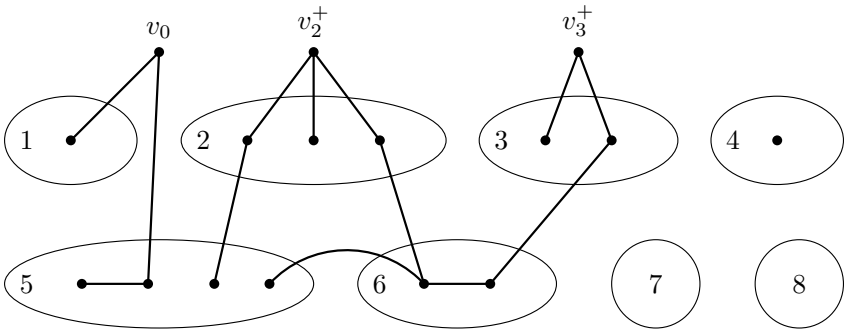


Figure 4.1 – Example of a certificate graph; here $p = \{\{v_0, 1\}, \{2, 3\}, \{4\}\}$, $s^{-1}(\gamma_0) = \{7, 8\}$, $s^{-1}(\gamma_1) = \{1, 4\}$, $s^{-1}(\gamma_2) = \{2, 3\}$ and $s^{-1}(\gamma_{-2}) = \{5, 6\}$. The set V_s^+ is $\{v_2^+, v_3^+\}$ with v_2^+ mapped to 2 and v_3^+ mapped to 3.

$\Pi(s^{-1}(\{\gamma_1, \gamma_2\}) \cup \{v_0\}) \times \mathbb{N}$ *such that there exist an induced forest* F *of* G *and* $E_0 \subseteq \{v_0 v : v \in V(F)\}$ *so that* $\mathsf{w}(V(F)) = w$*, and*

1. *The sets* $s^{-1}(\gamma_0) = \{i \in [k] : |V(F) \cap lab_G^{-1}(i)| = 0\}$ *and* $s^{-1}(\gamma_1) = \{i \in [k] : |V(F) \cap lab_G^{-1}(i)| = 1\}$.
2. *The certificate graph* $\mathbf{CG}(F, E_0, s)$ *is a forest.*
3. *Each connected component* C *of* $\mathbf{CG}(F, E_0, s)$ *has at least one vertex which belongs to* $lab_G^{-1}(s^{-1}(\{\gamma_1, \gamma_2\})) \cup \{v_0\}$.
4. *The partition* p *equals* $(s^{-1}(\{\gamma_1, \gamma_2\}) \cup \{v_0\})/\sim$ *where* $i \sim j$ *if and only if an* i*-vertex is connected to a* j*-vertex in* $\mathbf{CG}(F, E_0, s)$*, we consider* v_0 *as a* v_0*-vertex.*

Conditions (1) and (3) are as explained above, and automatically imply that, for each i in $s^{-1}(\{\gamma_2, \gamma_{-2}\})$, we have $|V(F) \cap lab_G^{-1}(i)| \geq 2$. Conditions (2) and (4) guarantee that $(V(F) \cup \{v_0\}, E(F) \cup E_0)$ can be extended into a tree, if any. They also guarantee that cycles detected through the acjoin operation correspond to cycles, and each cycle can be detected with it.

In the sequel, we call any triplet $(F, E_0, (p, \mathsf{w}(F)))$ a *candidate solution* in $\mathcal{A}_G[s]$ if Condition (4) is satisfied, and if in addition Conditions (1)-(3) are satisfied, we call it a *solution* in $\mathcal{A}_G[s]$.

Given a k-labeled graph G, the size of a maximum induced tree of G corresponds to the maximum, over all $s : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$ with $s^{-1}(\gamma_2) = \varnothing$, of $\mathsf{max}\{w : (\{L\}, w) \in \mathcal{A}_G[s]\}$ with $L = s^{-1}(\gamma_1) \cup \{v_0\}$. Indeed, by definition, if $(\{L\}, w)$ belongs to $\mathcal{A}_G[s]$, then there exist an induced forest F of G with $\mathsf{w}(F) = w$ and a set E_0 of edges incident to v_0 such that $(V(F) \cup \{v_0\}, E(F) \cup E_0)$ is a tree. This follows from the fact that if $s^{-1}(\gamma_2) = \varnothing$, then we have $\mathbf{CG}(F, E_0, s) = (V(F) \cup \{v_0\}, E(F) \cup E_0)$.

Our algorithm will store, for each k-labeled graph G and for every function $s : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$, an ac-representative set $tab_G[s]$ of $\mathcal{A}_G[s]$. We are now ready to give the different steps of the algorithm, depending on the clique-width operations.

Computing tab_G **for** $G = \mathbf{1}(x)$**.** For $s : \{1\} \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$, let

$$tab_G[s] := \begin{cases} \{(\{\{1, v_0\}\}, \mathsf{w}(x)), (\{\{1\}, \{v_0\}\}, \mathsf{w}(x))\} & \text{if } s(1) = \gamma_1, \\ \{(\{\{v_0\}\}, 0)\} & \text{if } s(1) = \gamma_0, \\ \varnothing & \text{if } s(1) \in \{\gamma_2, \gamma_{-2}\}. \end{cases}$$

Since $|V(G)| = 1$, there are no solutions intersecting the label class $lab_G^{-1}(1)$ on at least two vertices, and so the set of weighted partitions satisfying Definition 4.10 equals the empty set if $s(1) \in \{\gamma_2, \gamma_{-2}\}$. If $s(1) = \gamma_1$, there are two possibilities, depending on whether $E_0 = \varnothing$ or $E_0 = \{xv_0\}$. We can thus conclude that $tab_G[s] = \mathcal{A}_G[s]$ is correctly computed.

Computing tab_G for $G = add_{i,j}(H)$. We can suppose that H is k-labeled. Let $s : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$.

(a) If $s(i) = \gamma_0$ or $s(j) = \gamma_0$, then let $tab_G[s] := tab_H[s]$. We just copy all the solutions not intersecting $lab_H^{-1}(i)$ or $lab_H^{-1}(j)$. In this case, we do not need to use the operators $\mathsf{reduce}^{\mathsf{acy}}$ and rmc since we update $tab_G[s]$ with one table from tab_H. In the following cases, we assume that $s(i) \neq \gamma_0$ and $s(j) \neq \gamma_0$.

(b) If $s(i) = \gamma_2$ or $s(j) = \gamma_2$, then we let $tab_G[s] = \varnothing$. In this case $\mathcal{A}_G[s] = \varnothing$. Indeed, for every set $X \subseteq V(G)$ respecting Condition (1) and for all subsets $E_0 \subseteq \{xv_0 : x \in X\}$, the graph $\mathbf{CG}(X, E_0, s)$ contains a cycle. For example, if $s(i) = \gamma_2$ and $s(j) = \gamma_1$, then the two i-vertices in X are adjacent in $\mathbf{CG}(X, \varnothing, s)$ to v_i^+ and to the j-vertex in X, thus $\mathbf{CG}(X, \varnothing, s)$ contains a cycle of length four.

(c) If $s(i) = s(j) = \gamma_{-2}$, then we let $tab_G[s] = \varnothing$. Similarly to Case (b), we have $\mathcal{A}_G[s] = \varnothing$ because every vertex set with two i-vertices and two j-vertices induce a cycle of length four in G.

(d) Otherwise, we let $tab_G[s] := \mathsf{rmc}(\mathcal{A})$ with

$$\mathcal{A} := \mathsf{proj}(s^{-1}(\gamma_{-2}) \cap \{i, j\}, \mathsf{acjoin}(tab_H[s_H], \{(\{\{i, j\}\}, 0)\})$$

where $s_H(\ell) := s(\ell)$, for $\ell \in [k] \setminus \{i, j\}$, and

$$(s_H(i), s_H(j)) := \begin{cases} (\gamma_1, \gamma_1) & \text{if } s(i) = s(j) = \gamma_1, \\ (\gamma_2, \gamma_1) & \text{if } (s(i), s(j)) = (\gamma_{-2}, \gamma_1), \\ (\gamma_1, \gamma_2) & \text{if } (s(i), s(j)) = (\gamma_1, \gamma_{-2}). \end{cases}$$

Observe that this case corresponds to $s(i), s(j) \in \{\gamma_1, \gamma_{-2}\}$ with $s(i) = \gamma_1$ or $s(j) = \gamma_1$. Intuitively, we consider the weighted partitions $(p, w) \in tab_H[s_H]$ such that i and j belong to different blocks of p, we merge the blocks containing i and j, remove the elements in $s^{-1}(\gamma_{-2}) \cap \{i, j\}$ from the resulting block, and add the resulting weighted partition to $\mathcal{A}$. Notice that it is not necessary to call the operator $\mathsf{reduce}^{\mathsf{acy}}$ in this case since we update $tab_G[s]$ with only one table from tab_H.

Lemma 4.11. *Let $G = add_{i,j}(H)$ be a k-labeled graph such that there are no edges between an i-vertex and a j-vertex in H. For each function $s : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$, $tab_G[s]$ ac-represents $\mathcal{A}_G[s]$ assuming that $tab_H[s']$ ac-represents $\mathcal{A}_H[s']$ for all $s' : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$.*

Proof. We first recall that $lab_G(x) = lab_H(x)$ for all $x \in V(G) = V(H)$. If we are in Cases (b)-(c), then we are done since we clearly have $\mathcal{A}_G[s] = \varnothing$. Since the used operators preserve ac-representation, it is enough to prove that in Case (a), we have $\mathcal{A}_G[s] = \mathcal{A}_H[s]$, and in Case (d), we have $\mathcal{A}_G[s] = \mathcal{A}$ if we suppose that $tab_H[s_H] = \mathcal{A}_H[s_H]$. If we are in Case (a), then we are done because one easily checks that $(F, E_0, (p, w))$ is a solution in $\mathcal{A}_H[s]$ if and only if $(F, E_0, (p, w))$ is a solution in $\mathcal{A}_G[s]$, i.e., we have $\mathcal{A}_H[s] = \mathcal{A}_G[s]$.

Now, we assume that we are in Case (d), that is $s(i), s(j) \in \{\gamma_1, \gamma_{-2}\}$ with $s(i) = \gamma_1$ or $s(j) = \gamma_1$. We can assume w.l.o.g. that $s(j) = \gamma_1$. Let $(F, E_0, (p, w))$ be a solution in $\mathcal{A}_G[s]$. We prove that $(p, w) \in \mathcal{A}$. Let $\{x_j\} := V(F) \cap lab_G^{-1}(j)$ (by assumption $s(j) = \gamma_1$), $X_i := V(F) \cap lab_G^{-1}(i)$, $E_{i,j} := \{vx_j : v \in X_i\}$, and $F_H := (V(F), E(F) \setminus E_{i,j})$. Because we assume that there are no edges between an i-vertex and a j-vertex in H, we know that $E_{i,j} \cap E(H) = \varnothing$, i.e., F_H is an induced forest of H. Let p' be the partition on $s_H^{-1}(\{\gamma_1, \gamma_2\}) \cup \{v_0\}$ such that $(F_H, E_0, (p', w))$ is a candidate solution in $\mathcal{A}_H[s_H]$. We claim that (F_H, E_0, s_H) is a solution in $\mathcal{A}_H[s_H]$. By the definition of s_H, Condition (1) is trivially satisfied. If $|X_i| = 1$, then $\mathbf{CG}(F_H, E_0, s_H)$ is a subgraph of $\mathbf{CG}(F, E_0, s)$ and so Condition (2) is satisfied. And if $|X_i| \geq 2$ and $\mathbf{CG}(F_H, E_0, s_H)$ contains a cycle, then the cycle should contain the vertex $v_i^+ \in V_{s_H}^+$, but this vertex may be replaced by x_j in $\mathbf{CG}(F, E_0, s)$, contradicting the fact this latter is acyclic. Condition (3) is also satisfied because $s_H(i), s_H(j) \in \{\gamma_1, \gamma_2\}$ and for every connected component C of $\mathbf{CG}(F_H, E_0, s_H)$, either C is a connected component of $\mathbf{CG}(F, E_0, s)$ or C contains an ℓ-vertex with $\ell \in \{i, j\}$. Therefore, $(F_H, E_0, (p', w))$ is a solution in $\mathcal{A}_H[s_H]$.

It remains to prove that (p, w) is added in $\mathcal{A}$. We claim that $\mathsf{acy}(p', \{\{i, j\}\}_{\uparrow L})$ holds with $L := s^{-1}(\gamma_1, \gamma_2) \cup \{v_0\}$. By definition of acy it is equivalent to prove that i and j cannot belong to a same block of p'. Assume towards a contradiction that i and j belong to a same block of p'. Then, there is a path, in $\mathbf{CG}(F_H, E_0, v_0)$, between an i-vertex x_i and the j-vertex x_j of F. Let us choose this path P to be the smallest one. One first notices that P cannot contain the vertex v_i^+ of $V_{s_H}^+$, if any, because v_i^+ is only adjacent to i-vertices. Because $V(\mathbf{CG}(F_H, E_0, s_H)) \setminus \{v_i^+\} = V(\mathbf{CG}(F, E_0, s))$, we would conclude that $\mathbf{CG}(F, E_0, s)$ contains a cycle as $x_i x_j \in E(F) \setminus E(F_H)$, contradicting that $(F, E_0, (p, w))$ is a solution in $\mathcal{A}_G[s]$. Therefore, $\mathsf{acy}(p', \{\{i, j\}\}_{\uparrow L})$ holds. By assumption $s(j) = \gamma_1$, thus $j \notin s^{-1}(\gamma_{-2}) \cap \{i, j\}$. Since i and j are in the same block of the partition $p \sqcup \{\{i, j\}\}_{\uparrow L}$, we conclude that $(p, w) \in \mathsf{proj}(s^{-1}(\gamma_{-2}) \cap \{i, j\}, \mathsf{acjoin}(\{(p', w)\}, \{(\{\{i, j\}\}, 0)\}))$.

Finally, it remains to prove that each weighted partition $(p, w) \in \mathcal{A}$ belongs to $\mathcal{A}_G[s]$. Let $(F_H, E_0^H, (p', w))$ be a solution in $\mathcal{A}_H[s_H]$ so that

$$(p, w) \in \mathsf{proj}(s^{-1}(\gamma_{-2}) \cap \{i, j\}, \mathsf{acjoin}(\{(p', w)\}, \{(\{\{i, j\}\}, 0)\})).$$

Let $F := G[V(F_H)]$. By assumption $s(j) = \gamma_1$, and thus $s_H(j) = \gamma_1$. Let $\{x_j\} := V(F_H) \cap lab_H^{-1}(j)$, $X_i := V(F_H) \cap lab_H^{-1}(i)$, and $E_{i,j} := \{x_j v : v \in X_i\}$. Notice that $E(F) \setminus E(F_H) = E_{i,j}$. We claim that $(F, E_0^H, (p, w))$ is a solution in $\mathcal{A}_G[s]$.

- First, Condition (1) is trivially satisfied by the definition of s_H.
- Secondly, $\mathbf{CG}(F, E_0^H, s)$ is a forest. Indeed, $\{i, j\}$ cannot be a block of p', otherwise the acjoin operator would discard p'. If $|X_i| = 1$, then we have

  $$\mathbf{CG}(F, E_0^H, s) = (V(\mathbf{CG}(F_H, E_0^H, s_H)), E(\mathbf{CG}(F_H, E_0^H, s_H)) \cup E_{i,j})$$

  and $\mathbf{CG}(F, E_0^H, s)$ is clearly a forest. Otherwise, if $|X_i| \geq 2$, then $\mathbf{CG}(F, E_0^H, s)$ can be obtained from $\mathbf{CG}(F_H, E_0^H, s_H)$ by fusing the vertex v_i^+ and the vertex x_j. Clearly, this operation keeps the graph acyclic since x_j and v_i^+ are not connected in $\mathbf{CG}(F_H, E_0^H, s_H)$. Thus $(F, E_0^H, (p, w))$ satisfies Condition (2).
- Each connected component of $\mathbf{CG}(F_H, E_0^H, s_H)$ is contained in a connected component of $\mathbf{CG}(F, E_0^H, s)$, and the i-vertices are in the same connected component, in $\mathbf{CG}(F, E_0^H, s)$, as x_j. Therefore, Condition (3) is satisfied by $(F, E_0^H, (p, w))$ as $s(j) = \gamma_1$ and $s(\ell) = s_H(\ell)$ for all $\ell \in [k] \setminus \{i, j\}$.

- Also, Condition (4) is satisfied as p is then obtained from p' by merging the blocks of p' which contain i and j, and by removing i if $s(i) = \gamma_{-2}$.

We can therefore conclude that $(F, E_0^H, (p, w))$ is a solution in $\mathcal{A}_G[s]$. □

Computing tab_G for $G = ren_{i\to j}(H)$. We can suppose that H is k-labeled. Let $s : [k] \setminus \{i\} \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$.

(a) Let $\mathcal{A}_1 := tab_H[s_1]$ where $s_1(i) := \gamma_0$ and $s_1(\ell) := s(\ell)$ for all $\ell \in [k] \setminus \{i\}$. This set contains all weighted partitions corresponding to solutions not intersecting $lab_H^{-1}(i)$. They are trivially solutions in $\mathcal{A}_G[s]$.

(b) If $s(j) = \gamma_0$, then let $\mathcal{A}_2 := \varnothing$, otherwise let $s_2 : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$ such that $s_2(j) := \gamma_0$, $s_2(i) := s(j)$ and $s_2(\ell) := s(\ell)$ for all $\ell \in [k] \setminus \{i, j\}$ and let

$$\mathcal{A}_2 := \begin{cases} tab_H[s_2] & \text{if } s(j) = \gamma_{-2}, \\ \mathsf{proj}(\{i\}, \mathsf{acjoin}(tab_H[s_2], \{(\{\{i, j\}\}, 0)\})) & \text{otherwise.} \end{cases}$$

This set contains all weighted partitions corresponding to solutions that intersect $lab_H^{-1}(j)$ but not $lab_H^{-1}(j)$. They are solutions in $\mathcal{A}_G[s]$ by replacing i by j with the acjoin operator, if $s(j) \neq \gamma_{-2}$, in the corresponding weighted partitions. Its worth noticing that if $s(j) = \gamma_0$, then $s_1 = s_2$ and this is why we set $\mathcal{A}_2 = \varnothing$ in this case.

(c) If $s(j) \neq \gamma_{-2}$, then let $\mathcal{A}_3 := \varnothing$, otherwise let

$$\mathcal{A}_3 := \bigcup_{s_3 \in \mathcal{S}_3} \mathsf{proj}(\{i, j\}, tab_H[s_3]),$$

where $\mathcal{S}_3$ is the set of all the functions s_3 with $s_3(i), s_3(j) \in \{\gamma_1, \gamma_{-2}\}$, and $s_3(\ell) = s(\ell)$ for all $\ell \in [k] \setminus \{i, j\}$. Intuitively, $\mathcal{S}_3$ is the set of functions coherent with s if $s(j) = \gamma_{-2}$. The set $\mathcal{A}_3$ corresponds to partial solutions intersecting $lab_H^{-1}(i)$ and $lab_H^{-1}(j)$ when $s(j) = \gamma_{-2}$. In this case, we have to ensure that the partial solutions in $\mathcal{A}_3$ respect Condition (3) of Definition 4.10. We do that by removing all the partitions with a block included in $\{i, j\}$.

(d) We now define the last set considering the other cases. If $s(j) \neq \gamma_2$, then let $\mathcal{A}_4 := \varnothing$, otherwise let

$$\mathcal{A}_4 := \bigcup_{s_4 \in \mathcal{S}_4} \mathsf{proj}(\{i\}, \mathsf{acjoin}(tab_H[s_4], \{(\{\{i, j\}\}, 0)\})),$$

where $\mathcal{S}_4$ is the set of all the functions s_4 with $s_4(i), s_4(j) \in \{\gamma_1, \gamma_2\}$ and $s_4(\ell) = s(\ell)$ for all $\ell \in [k] \setminus \{i, j\}$. Informally, $\mathcal{S}_4$ is the set of functions compatible with s if $s(j) = \gamma_2$. The set $\mathcal{A}_4$ corresponds to partial solutions intersecting $lab_H^{-1}(i)$ and $lab_H^{-1}(j)$ when $s(j) = \gamma_2$. We have to force that i-vertices and j-vertices belong to different connected components. We check this with the function acy in the operator acjoin.

We let $tab_G[s] := \mathsf{reduce}^{\mathsf{acy}}(\mathsf{rmc}(\mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3 \cup \mathcal{A}_4))$.

Lemma 4.12. *Let $G = ren_{i\to j}(H)$ with H a k-labeled graph. For each function $s : [k] \setminus \{i\} \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$, the table $tab_G[s]$ ac-represents $\mathcal{A}_G[s]$ assuming that $tab_H[s']$ ac-represents $\mathcal{A}_H[s']$ for all $s' : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$.*

Proof. Since the used operators preserve ac-representation, it is enough to prove that $\mathcal{A}_G[s] = \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3 \cup \mathcal{A}_4$ if we assume that $tab_H[s'] = \mathcal{A}_H[s']$ for every $s' : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$.

Let $(F, E_0, (p, w))$ be a solution in $\mathcal{A}_G[s]$. We want to prove that $(p, w) \in \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3 \cup \mathcal{A}_4$. If $V(F)$ does not intersect $lab_H^{-1}(i)$, then $(F, E_0, (p, w))$ is a solution in $\mathcal{A}_H[s_1]$. Assume now that $V(F)$ intersects $lab_H^{-1}(i)$. If $V(F) \cap lab_H^{-1}(j) = \varnothing$ and $s(j) = \gamma_{-2}$, $(F, E_0, (p, w))$ is a solution in $\mathcal{A}_H[s_2]$. If $V(F) \cap lab_H^{-1}(j) = \varnothing$ and $s(j) \in \{\gamma_1, \gamma_2\}$, then it is easy to check that $(F, E_0, (p', w))$ is a solution in $\mathcal{A}_H[s_2]$ where p' is obtained from p by replacing j by i.

We may assume now that $V(F) \cap lab_H^{-1}(i) \neq \varnothing$ and $V(F) \cap lab_H^{-1}(j) \neq \varnothing$. Then, we have $s(j) \in \{\gamma_2, \gamma_{-2}\}$ as $|lab_G^{-1}(j) \cap V(F)| \geq 2$. Let $s_\star : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$ be the function such that $s_\star(\ell) := s(\ell)$ for all $\ell \in [k] \setminus \{i, j\}$, and for $t \in \{i, j\}$,

$$s_\star(t) := \begin{cases} \gamma_1 & \text{if } |V(F) \cap lab_H^{-1}(t)| = 1, \\ s(j) & \text{if } |V(F) \cap lab_H^{-1}(t)| \geq 2. \end{cases}$$

By definition, if $s(j) = \gamma_{-2}$, then $s_\star$ belongs to $\mathcal{S}_3$ and if $s(j) = \gamma_2$, then $s_\star$ belongs to $\mathcal{S}_4$. Let p' be the partition on $s_\star^{-1}(\{\gamma_1, \gamma_2\}) \cup \{v_0\}$ such that $(F, E_0, (p', w))$ is a candidate solution in $\mathcal{A}_H[s_\star]$. We claim that $(F, E_0, (p', w))$ is a solution in $\mathcal{A}_H[s_\star]$. By Definition of $s_\star$ and of (F, E_0), Condition (1) is satisfied by $(F, E_0, (p', w))$.

Suppose first that $s(j) = \gamma_{-2}$. Observe that Condition (2) is satisfied because the certificate graphs of (F, E_0) with respect to s and $s_\star$ are the same. Condition (3) is also satisfied by definition of $s_\star$ and because $\mathbf{CG}(F, E_0, s) = \mathbf{CG}(F, E_0, s_\star)$. So, $(F, E_0, (p', w))$ is a solution in $\mathcal{A}_H[s_\star]$.

Assume now that $s(j) = \gamma_2$. Condition (2) is satisfied. Indeed, if $s_\star(i) = \gamma_1$, then the graph $\mathbf{CG}(F, E_0, s)$ is a subgraph of $\mathbf{CG}(F, E_0, s_\star)$. Otherwise, if $s_\star(i) = \gamma_2$, then $\mathbf{CG}(F, E_0, s)$ can be obtained from $\mathbf{CG}(F, E_0, s_\star)$ by fusing v_i^+ with v_j^+. In both cases, it is easy to see that $\mathbf{CG}(F, E_0, s_\star)$ is acyclic as $\mathbf{CG}(F, E_0, s)$ is acyclic. Condition (3) is satisfied because each connected component C of $\mathbf{CG}(F, E_0, s)$ contains at least a vertex in $lab_G^{-1}(s^{-1}(\{\gamma_1, \gamma_2\})) \cup \{v_0\}$, and we have from the definition of $s_\star$

$$lab_H^{-1}(s_\star^{-1}(\{\gamma_1, \gamma_2\})) = lab_G^{-1}(s^{-1}(\{\gamma_1, \gamma_2\})).$$

In both cases, $(F, E_0, (p', w))$ is a solution in $\mathcal{A}_H[s_\star]$, and depending on $s(j)$, we can clearly conclude that (p, w) is obtained from (p', w).

Let us now prove that for any weighted partition $(p, w) \in \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3 \cup \mathcal{A}_4$, there is a pair (F, E_0) such that $(F, E_0, (p, w))$ is a solution in $\mathcal{A}_G[s]$. This is clear if $(p, w) \in \mathcal{A}_1 \cup \mathcal{A}_2$.

Assume that $s(j) = \gamma_{-2}$ and let $(p, w) \in \mathcal{A}_3$. Let $s_3 \in \mathcal{S}_3$ and (p', w) be the weighted partition from $tab_H[s_3]$ from which (p, w) is obtained. Let $(F, E_0, (p', w))$ be a solution in $\mathcal{A}_H[s_3]$. We claim that $(F, E_0, (p, w))$ is a solution in $\mathcal{A}_G[s]$. By definition of $\mathcal{S}_3$, we clearly have $|V(F) \cap lab_H^{-1}(\{i, j\})| = |V(F) \cap lab_G^{-1}(j)| \geq 2$. We deduce that Condition (1) is satisfied. Condition (2) is also satisfied because $\mathbf{CG}(F, E_0, s_3)$ is the same as $\mathbf{CG}(F, E_0, s)$. We claim that Condition (3) is satisfied. Notice that $s^{-1}(\{\gamma_1, \gamma_2\}) = s_3^{-1}(\{\gamma_1, \gamma_2\}) \setminus \{i, j\}$. Moreover, if $i \in s_3^{-1}(\{\gamma_1, \gamma_2\})$, by definition of s_3, we have $s_3(i) = \gamma_1$, that is F has exactly one vertex x such that $lab_H(x) = i$ (the same statement is true for j). Since we use the operator proj with $\{i, j\}$, there are no blocks of p' included in $\{i, j\}$. Hence, if F contains one vertex x such that $lab_H(x) = i$ (or $lab_H(x) = j$), then by Condition (4) this vertex is connected in $\mathbf{CG}(F, E_0, s_3)$ to either v_0 or to an ℓ-vertex with $\ell \in s^{-1}(\{\gamma_1, \gamma_2\})$. We can conclude that each connected component of F must contain a vertex in $lab_G^{-1}(s^{-1}(\{\gamma_1, \gamma_2\})) \cup \{v_0\}$, i.e. Condition (3) is satisfied. Condition (4) is satisfied owing to the fact that p' is obtained from p by doing a projection on $\{i, j\}$.

Suppose now that $s(j) = \gamma_2$ and let $(p, w) \in \mathcal{A}_4$. Let $s_4 \in \mathcal{S}_4$ and (p', w) be the weighted partition from $tab_H[s_4]$ from which (p, w) is obtained, and let $(F, E_0, (p', w))$ be a solution in

$\mathcal{A}_H[s_4]$. By definition of $\mathcal{S}_4$, we deduce that Condition (1) is satisfied (see the case $s(j) = \gamma_{-2}$). If there is a cycle in $\mathbf{CG}(F, E_0, s)$, then it must be between an i-vertex and a j-vertex of H. But then we must have a path between them in $\mathbf{CG}(F, E_0, s_4)$, i.e., i and j belong to a same block of p', contradicting that (p, w) is produced from (p', w) (because the acjoin operator would detect that $\mathsf{acy}(p, \{\{i, j\}\}_{\uparrow[k]})$ does not hold). So, Condition (2) is satisfied. We deduce that Condition (3) is satisfied from the fact that by definition of s_4, we have $lab_G^{-1}(s^{-1}(\{\gamma_1, \gamma_2\})) = lab_H^{-1}(s_4^{-1}(\{\gamma_1, \gamma_2\}))$. Also, as p is obtained from p' by merging the blocks containing i and j, and by removing i, we deduce that Condition (4) is satisfied.

In both cases, we can conclude that $(F, E_0, (p, w))$ is a solution in $\mathcal{A}_G[s]$. □

Computing tab_G for $G = G_a \oplus G_b$. We can suppose w.l.o.g. that G_a and G_b are both k-labeled[3]. Let $s : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$.

We say that $s_a : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$ and $s_b : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$ *u-agree on* s if,

(u1) for each $i \in s_a^{-1}(\gamma_0)$, $s(i) = s_b(i)$. Similarly, for each $i \in s_b^{-1}(\gamma_0)$, $s(i) = s_a(i)$,

(u2) for each $i \in s^{-1}(\gamma_1)$, either $s_a(i) = \gamma_0$ or $s_b(i) = \gamma_0$,

(u3) for each $i \in [k] \setminus (s_a^{-1}(\gamma_0) \cup s_b^{-1}(\gamma_0))$, if $s(i) = \gamma_2$, then $s_a(i), s_b(i) \in \{\gamma_1, \gamma_2\}$,

(u4) for each $i \in [k] \setminus (s_a^{-1}(\gamma_0) \cup s_b^{-1}(\gamma_0))$, if $s(i) = \gamma_{-2}$, then $s_a(i), s_b(i) \in \{\gamma_1, \gamma_{-2}\}$.

The functions s_a and s_b inform about the indices to look at tab_{G_a} and tab_{G_b} in order to construct $tab_G[s]$. Let $(F, E_0, (p, w))$ be a solution in $\mathcal{A}_G[s]$, and assume that it is constructed from solutions $(F_a, E_a^0, (p_a, w_a))$ and $(F_b, E_b^0, (p_b, w_b))$, in respectively, $\mathcal{A}_{G_a}[s_a]$ and $\mathcal{A}_{G_b}[s_b]$. The first condition tells that if F_a (resp. F_b) does not intersect $lab_G^{-1}(i)$, then the intersection of F with $lab_G^{-1}(i)$ depends only on $V(F_b) \cap lab_G^{-1}(i)$ (resp. $V(F_a) \cap lab_G^{-1}(i)$), and so if $V(F)$ does not intersect $lab_G^{-1}(i)$, then F_a and F_b do not intersect $lab_G^{-1}(i)$. The second condition tells that if $|F \cap lab_G^{-1}(i)| = 1$, then either $F \cap lab_G^{-1}(i) = F_a \cap lab_G^{-1}(i)$ or $F \cap lab_G^{-1}(i) = F_b \cap lab_G^{-1}(i)$. The other two conditions tell when F intersects both $lab_{G_a}^{-1}(i)$ and $lab_{G_b}^{-1}(i)$. Notice that we may have $s(i)$ set to γ_{-2} (or γ_2), while F_a and F_b each intersects $lab_G^{-1}(i)$ in exactly one vertex.

We let $tab_G[s] := \mathsf{reduce}^{\mathsf{acy}}(\mathsf{rmc}(\mathcal{A}))$ where,

$$\mathcal{A} := \bigcup_{\substack{s_a, s_b \\ \text{u-agree on } s}} \mathsf{acjoin}\left(\mathsf{proj}(s^{-1}(\gamma_{-2}), tab_{G_a}[s_a]), \mathsf{proj}(s^{-1}(\gamma_{-2}), tab_{G_b}[s_b])\right).$$

The weighted partitions (p, w) added in $tab_G[s]$ are all the weighted partitions that are ac-joins of weighted partitions (p_a, w_a) and (p_b, w_b) from $tab_{G_a}[s_a]$ and $tab_{G_b}[s_b]$, respectively. We need to do the projections before the join because we may have $s_a(i) = s_b(i) = \gamma_1$, $s(i) = \gamma_{-2}$, and there is j such that $s(j) = \gamma_2$ with j in the same block as i in both partitions p_a and p_b. In this setting, if we do the projection after the acjoin operator, this latter will detect that $\mathsf{acy}(p_a, p_b)$ does not hold, and won't construct (p, w), which indeed corresponds to a solution in $\mathcal{A}_G[s]$.

Lemma 4.13. *Let $G = G_a \oplus G_b$ be a k-labeled graph. For each function $s : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$, the table $tab_G[s]$ ac-represents $\mathcal{A}_G[s]$ assuming that $tab_{G_a}[s']$ and $tab_{G_b}[s']$ ac-represent, respectively, $\mathcal{A}_{G_a}[s']$ and $\mathcal{A}_{G_b}[s']$, for each $s' : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$.*

[3]If $J \subset [k]$ is the set of labels of G_a (or G_b), we can extend the domain of any function $s' : J \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$ to $[k]$ by setting $s'(i) := \gamma_0$ for all $i \in [k] \setminus J$.

Proof. Since the used operators preserve ac-representation, it is enough to prove that $\mathcal{A}_G[s] = \mathcal{A}$ if we consider that $tab_{G_t}[s'] = \mathcal{A}_{G_t}[s']$, for every $t \in \{a, b\}$ and $s' : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$.

Let $(F, E_0, (p, w))$ be a solution in $\mathcal{A}_G[s]$. We claim that $(p, w) \in \mathcal{A}$. For $t \in \{a, b\}$, let $F_t := G_t[V(F) \cap V(G_t)]$, $E_0^t := \{v_0 v \in E_0 : v \in V(F_t)\}$, and $w_t := \mathsf{w}(V(F_t))$, and let $s_t : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$ such that

$$s_t(i) := \begin{cases} \gamma_0 & \text{if } V(F_t) \cap lab_{G_t}^{-1}(i) = \varnothing, \\ \gamma_1 & \text{if } |V(F_t) \cap lab_{G_t}^{-1}(i)| = 1, \\ s(i) & \text{if } |V(F_t) \cap lab_{G_t}^{-1}(i)| \geq 2. \end{cases}$$

It is straightforward to verify that s_a and s_b u-agree on s. Observe that, by definition, $s_t^{-1}(\gamma_{-2}) \subseteq s^{-1}(\gamma_{-2})$ and $s_t^{-1}(\gamma_2) \subseteq s^{-1}(\gamma_2)$, for each $t \in \{a, b\}$. Let p_a and p_b be, respectively, partitions on $s_a^{-1}(\{\gamma_1, \gamma_2\}) \cup \{v_0\}$ and $s_b^{-1}(\{\gamma_1, \gamma_2\}) \cup \{v_0\}$ such that $(F_a, E_0^a, (p_a, w_a))$ and $(F_b, E_0^b, (p_b, w_b))$ are, respectively, candidate solutions in $\mathcal{A}_{G_a}[s_a]$ and $\mathcal{A}_{G_b}[s_b]$.

We claim that $(F_a, E_0^a, (p_a, w_a))$ is a solution in $\mathcal{A}_{G_a}[s_a]$. By definition of p_a, s_a, and of (F_a, E_0^a), Conditions (1) and (4) are clearly satisfied. Because $s_a^{-1}(\gamma_2) \subseteq s^{-1}(\gamma_2)$, and $F = F_a \oplus F_b$, we can conclude that $\mathbf{CG}(F_a, E_0^a, s_a)$ is an induced subgraph of $\mathbf{CG}(F, E_0, s)$, and because $\mathbf{CG}(F, E_0, s)$ is acyclic, we can conclude that $\mathbf{CG}(F_a, E_0^a, s_a)$ is acyclic, i.e., Condition (2) is satisfied. If a connected component C of $\mathbf{CG}(F_a, E_0^a, s_a)$ does not intersect $s_a^{-1}(\{\gamma_1, \gamma_2\}) \cup \{v_0\}$, then C is entirely contained in $lab_{G_a}^{-1}(s_a^{-1}(\gamma_{-2}))$. But, this yields a contradiction with $(F, E_0, (p, w))$ satisfying Condition (3) because C is a connected component of $\mathbf{CG}(F, E_0, s)$, and $s_a^{-1}(\gamma_{-2}) \subseteq s^{-1}(\gamma_{-2})$. Therefore Condition (3) is also satisfied. We can thus conclude that $(F_a, E_0^a, (p_a, w_a))$ is a solution in $\mathcal{A}_{G_a}[s_a]$. Similarly, one can check that $(F_b, E_0^b, (p_b, w_b))$ is a solution in $\mathcal{A}_{G_b}[s_b]$.

It remains to prove that

$$(p, w) \in \mathsf{acjoin}(\mathsf{proj}(s^{-1}(\gamma_{-2}), \{(p_a, w_a)\}), \mathsf{proj}(s^{-1}(\gamma_{-2}), \{(p_b, w_b)\})).$$

First, recall that each connected component of F is either a connected component of F_a or of F_b. Then, because $(F, E_0, (p, w))$ satisfies Condition (3), we have that $\mathsf{proj}(s^{-1}(\gamma_{-2}), \{(p_a, w_a)\})) \neq \varnothing$, and similarly $\mathsf{proj}(s^{-1}(\gamma_{-2}), \{(p_b, w_b)\})) \neq \varnothing$. We deduce that

$$(p'_a, w_a) := (p_{a\downarrow(s^{-1}(\gamma_{-2}))}, w_a) \in \mathsf{proj}(s^{-1}(\gamma_{-2}), \{(p_a, w_a)\}),$$
$$(p'_b, w_b) := (p_{b\downarrow(s^{-1}(\gamma_{-2}))}, w_b) \in \mathsf{proj}(s^{-1}(\gamma_{-2}), \{(p_b, w_b)\}).$$

Let $p''_a := p'_{a\uparrow([k]\setminus s^{-1}(\{\gamma_0, \gamma_{-2}\}))}$ and $p''_b := p'_{b\uparrow([k]\setminus s^{-1}(\{\gamma_0, \gamma_{-2}\}))}$. We claim that $\mathsf{acy}(p''_1, p''_2)$ holds. Assume towards a contradiction that it is not the case. We let $\sim_a$ (resp. $\sim_b$) be an equivalence relation on $s^{-1}(\{\gamma_1, \gamma_2\}) \cup \{v_0\}$ where $i \sim_a j$ (resp. $i \sim_b j$) if there is an i-vertex[4] and a j-vertex that are connected in $\mathbf{CG}(F_a, E_0^a, s_a)$ (resp. $\mathbf{CG}(F_b, E_0^b, s_b)$). By the graphical definition of acy, we can easily see that if $\mathsf{acy}(p''_a, p''_b)$ does not hold, then there is a sequence $i_0, \ldots, i_{2r-1}$ of $s^{-1}(\{\gamma_1, \gamma_2\}) \cup \{v_0\}$ such that[5], for all $0 \leq \alpha < r - 1$, we have $i_{2\alpha} \sim_b i_{2\alpha+1}$ and $i_{2\alpha+1} \sim_a i_{2\alpha+2}$. We can thus construct a cycle in $\mathbf{CG}(F, E_0, s)$ from this sequence since $V(F_a) \cap V(F_b) = \varnothing$, $\mathbf{CG}(F_a, E_0^a, s_a)$ and $\mathbf{CG}(F_b, E_0^b, s_b)$ are induced subgraphs of $\mathbf{CG}(F, E_0, s)$, and all the vertices labeled with a label from $s^{-1}(\gamma_2)$ are adjacent to v_i^+ in $\mathbf{CG}(F, E_0, s)$. This yields a contradiction as $\mathbf{CG}(F, E_0, s)$ is acyclic by assumption. Therefore, $\mathsf{acy}(p''_1, p''_2)$ holds.

Finally, $p = p''_a \sqcup p''_b$ because one easily checks that there is an i-vertex x connected to a j-vertex y in $\mathbf{CG}(F, E_0, s)$ if and only if $i \mathcal{R} j$ where $\mathcal{R}$ is the transitive closure of ($i \sim_a j$ or $i \sim_b j$). This follows from the fact that for every $i \in s^{-1}(\{\gamma_1, \gamma_2\})$, either there is exactly one

[4] We consider v_0 as a v_0-vertex.

[5] The indexes are modulo $2r$.

i-vertex in F or the i-vertices of F are all adjacent to v_i^+ in $\mathbf{CG}(F, E_0, s)$. In both cases, the i-vertices of F are in the same connected component of $\mathbf{CG}(F, E_0, s)$. Since the equivalence classes of $\mathcal{R}$ correspond to the blocks of $p_1'' \sqcup p_2''$, and $w = w_a + w_b$, we can conclude that $(p, w) \in \mathsf{acjoin}(\mathsf{proj}(s^{-1}(\gamma_{-2}), \{(p_a, w_a)\}), \mathsf{proj}(s^{-1}(\gamma_{-2}), \{(p_b, w_b)\}))$.

We now prove that if (p, w) is added to $tab_G[s]$ from $(p_a, w_a) \in \mathcal{A}_{G_a}[s_a]$ and $(p_b, w_b) \in \mathcal{A}_{G_b}[s_b]$, then there exists a pair (F, E_0) such that $(F, E_0, (p, w))$ is a solution in $\mathcal{A}_G[s]$. Let (F_a, E_0^a) and (F_b, E_0^b) such that $(F_a, E_0^a, (p_a, w_a))$ and $(F_b, E_0^b, (p_b, w_b))$ are solutions in, respectively, $\mathcal{A}_{G_a}[s_a]$ and $\mathcal{A}_{G_b}[s_b]$ with s_a and s_b u-agreeing on s. We claim that $(F, E_0, (p, w))$ is a solution in $\mathcal{A}_G[s]$ with $F := (V(F_1) \cup V(F_2), E(F_1) \cup E(F_2))$ and $E_0 := E_0^a \cup E_0^b$. Because s_a and s_b u-agree on s, we clearly have that Condition (1) is satisfied.

Let $\sim_a$ and $\sim_b$ as defined above. Assume towards a contradiction that there exists a cycle C in $\mathbf{CG}(F, E_0, s)$. Since both $\mathbf{CG}(F_a, E_0^a, s_a)$ and $\mathbf{CG}(F_b, E_0^b, s_b)$ are acyclic, C must be a cycle alternating between paths in $\mathbf{CG}(F_a, E_0^a, s_a)$ and paths in $\mathbf{CG}(F_b, E_0^b, s_b)$. One can easily check that this implies the existence of a sequence $i_0, \ldots, i_{2r-1}$ of $s^{-1}(\{\gamma_1, \gamma_2\}) \cup \{v_0\}$ such that[5], for all $0 \leq \alpha < r - 1$, we have $i_{2\alpha} \sim_b i_{2\alpha+1}$ and $i_{2\alpha+1} \sim_a i_{2\alpha+2}$. Moreover, it is easy to infer, from this sequence and the graphical definition of acy, that $\mathsf{acy}\left(p'_{a\uparrow L}, p'_{b\uparrow L}\right)$ does not hold with $p'_a := p_{a\downarrow(s^{-1}(\gamma_{-2}))}$, $p'_b := p_{b\downarrow(s^{-1}(\gamma_{-2}))}$ and $L := s^{-1}(\{\gamma_1, \gamma_2\})$, contradicting the fact that $(p, w) = \mathsf{acjoin}(\mathsf{proj}(s^{-1}(\gamma_{-2}), \{(p_a, w_a)\}), \mathsf{proj}(s^{-1}(\gamma_{-2}), \{(p_b, w_b)\}))$. Therefore, $\mathbf{CG}(F, E_0, s)$ is acyclic and so Condition (2) is satisfied.

If we suppose that Condition (3) is not satisfied, then there is a connected component C of $\mathbf{CG}(F, E_0, s)$ that does not intersect $s^{-1}(\{\gamma_1, \gamma_2\}) \cup \{v_0\}$, i.e., C is fully contained in $lab_G^{-1}(s^{-1}(\gamma_{-2}))$. Since $F = F_a \uplus F_b$, C is either a connected component of F_a or of F_b. Suppose *w.l.o.g.* that C is a connected component of F_a. Observe that C intersects $lab_{G_a}^{-1}(s_a^{-1}(\{\gamma_1, \gamma_2\}))$ because $(F_a, E_0^a, (p_a, w_a))$ is a solution in $\mathcal{A}_{G_a}[s_a]$. Moreover, C does not intersect $lab_{G_a}^{-1}(s_a^{-1}(\gamma_2))$, otherwise C would intersect $lab_G^{-1}(s^{-1}(\gamma_2))$ since if $s_a(i) = \gamma_2$, then $s(i) = \gamma_2$, for all $i \in [k]$. Thus C is a connected component of $\mathbf{CG}(F_a, E_0^a, s_a)$ and $b_C := \{i \in s_a^{-1}(\gamma_1) : C \cap lab_{G_a}^{-1}(i) \neq \varnothing\}$ is a block of p_a because $(F_a, E_0^a, (p_a, w_a))$ is a candidate solution in $\mathcal{A}_{G_a}[s_a]$. Thus, by definition of proj, we have $\mathsf{proj}(s^{-1}(\gamma_{-2}), \{(p_a, w_a)\}) = \varnothing$, which contradicts the fact that $(p, w) = \mathsf{acjoin}(\mathsf{proj}(s^{-1}(\gamma_{-2}), \{(p_a, w_a)\}), \mathsf{proj}(s^{-1}(\gamma_{-2}), \{(p_b, w_b)\}))$. So, Condition (3) is also satisfied. We deduce that Condition (4) is satisfied by observing that, for $i, j \in s^{-1}(\{\gamma_1, \gamma_2\}) \cup \{v_0\}$, there is an i-vertex connected to a j-vertex in $\mathbf{CG}(F, E_0, s)$ if and only if $i\mathcal{R}j$ where $\mathcal{R}$ is the transitive closure of $(i \sim_a j$ or $i \sim_b j)$. This concludes the proof that $(F, E_0, (p, w))$ is a solution in $\mathcal{A}_G[s]$. □

Theorem 4.14. *There is an algorithm that, given an n-vertex graph G and an irredundant k-expression of G, computes a minimum feedback vertex set in time $O(15^k \cdot 2^{(\omega+1)\cdot k} \cdot k^{O(1)} \cdot n)$.*

Proof. We do a bottom-up traversal of the k-expression and at each step we update the tables as indicated above. The correctness of the algorithm follows from Lemmas 4.11-4.13. From the definition of $\mathcal{A}_G[s]$, we conclude that the maximum weight of an induced forest is the maximum, over all $s : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$ with $s^{-1}(\gamma_2) = \varnothing$, of $\mathsf{max}\{w : (\{\{v_0\} \cup s^{-1}(\gamma_1)\}\}, w) \in tab_G[s]\}$ because $tab_G[s]$ ac-represents $\mathcal{A}_G[s]$ for all $s : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$.

Let us discuss the time complexity now. If $G = add_{i,j}(H)$ or $G = ren_{i\to j}(H)$, and $s : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$, then we update $tab_G[s]$ from a constant number of tables from tab_H, each identified in constant time from s. Since each table contains at most $2^k \cdot (k + 1)$ entries, we call the function $\mathsf{reduce}^{\mathsf{acy}}$ with a set of size at most $O(2^k \cdot (k + 1))$ as input. By Theorem 4.8 and Proposition 4.4, we can thus update tab_G in time $2^{\omega \cdot k} \cdot k^{O(1)}$. If $G = G_a \oplus G_b$, then we claim that the tables from tab_G are computable in time $O(15^k \cdot 2^{(\omega+1)\cdot k} \cdot k^{O(1)})$. For $s : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$,

we let

$$\mathcal{A}[s] := \bigcup_{\substack{s_a, s_b \\ \text{u-agree on } s}} \mathsf{acjoin}\left(\mathsf{proj}(s^{-1}(\gamma_{-2}), tab_{G_a}[s_a]), \mathsf{proj}(s^{-1}(\gamma_{-2}), tab_{G_b}[s_b])\right).$$

By Theorem 4.8, computing $tab_G[s] := \mathsf{reduce}^{\mathsf{acy}}(\mathsf{rmc}(\mathcal{A}[s]))$ can be done in time $|\mathcal{A}[s]| \cdot 2^{(\omega-1)\cdot(k+1)} \cdot k^{O(1)}$. Therefore, we can compute the tables from tab_G in time

$$\sum_{s:[k]\to\{\gamma_0,\gamma_1,\gamma_2,\gamma_{-2}\}} |\mathcal{A}[s]| \cdot 2^{(\omega-1)\cdot k} \cdot k^{O(1)}.$$

Now, observe that there are at most 15^k functions $s, s_a, s_b : [k] \to \{\gamma_0, \gamma_1, \gamma_2, \gamma_{-2}\}$ such that s_a and s_b u-agree on s. Indeed, for all $i \in [k]$, if s_a and s_b u-agree on s, then the tuple $(s_a(i), s_b(i), s(i))$ can take up to 15 values. See Table 4.1 for all the possible values.

Table 4.1 – Possibles values of $s(i)$ depending on the value of $s_a(i)$ and $s_b(i)$ when s_a and s_b u-agree on s, there are 15 possible values for the tuple $(s_a(i), s_b(i), s(i))$.

	$s_b(i) = \gamma_0$	$s_b(i) = \gamma_1$	$s_b(i) = \gamma_2$	$s_b(i) = \gamma_{-2}$
$s_a(i) = \gamma_0$	γ_0	γ_1	γ_2	γ_{-2}
$s_a(i) = \gamma_1$	γ_1	γ_2, γ_{-2}	γ_2	γ_{-2}
$s_a(i) = \gamma_2$	γ_2	γ_2	γ_2	forbidden
$s_a(i) = \gamma_{-2}$	γ_{-2}	γ_{-2}	forbidden	γ_{-2}

Because each table of tab_{G_a} and tab_{G_b} contains at most $2^k \cdot (k+1)$ values, by Proposition 4.4, we have

$$|\mathsf{acjoin}\left(\mathsf{proj}(s^{-1}(\gamma_{-2}), tab_{G_a}[s_a]), \mathsf{proj}(s^{-1}(\gamma_{-2}), tab_{G_b}[s_b])\right)| \leq 2^{2k} \cdot k^3.$$

It follows that $\sum_{s:[k]\to\{\gamma_0,\gamma_1,\gamma_2,\gamma_{-2}\}} |\mathcal{A}[s]| \leq 15^k \cdot 2^{2k} \cdot k^3$. Hence, we can conclude that the tables from tab_G can be computed in time $O(15^k \cdot 2^{(\omega+1)\cdot k} \cdot k^{O(1)})$.

Because the size of a k-expression is $O(n \cdot k^2)$, we can conclude that a minimum weighted feedback vertex set can be computed in the given time. □

4.1.3 Connected variants of (σ, ρ)-Dominating Set problems

We will show here how to use the operators defined in [9] in order to obtain a $2^{O(d\cdot k)} \cdot n$ time algorithm for computing a minimum or a maximum weighted connected (σ, ρ)-dominating set, given a k-expression, with d a constant that depends only on (σ, ρ). We deduce from this algorithm a $2^{O(k)} \cdot n^{O(1)}$ time algorithm for computing a minimum node-weighted Steiner tree, and a $2^{O(d\cdot k)} \cdot n^{O(1)}$ time algorithm for computing a maximum (or minimum) weighted connected co-(σ, ρ)-dominating set.

We let $opt \in \{\mathsf{min}, \mathsf{max}\}$, i.e., we are interested in computing a connected (σ, ρ)-dominating set of maximum (or minimum) weight if $opt = \mathsf{max}$ (or $opt = \mathsf{min}$). Let us first give some definitions. As defined in Section 4.1.1, rmc works only for the case $opt = \mathsf{max}$, we redefine it as follows in order to take into account minimization problems.

$$\mathsf{rmc}(\mathcal{A}) := \{(p, w) \in \mathcal{A} : \forall (p, w') \in \mathcal{A}, opt(w, w') = w\}.$$

Join. Let L' be a finite set. For $\mathcal{A} \subseteq \Pi(L) \times \mathbb{N}$ and $\mathcal{B} \subseteq \Pi(L') \times \mathbb{N}$, we define $\mathsf{join}(\mathcal{A}, \mathcal{B}) \subseteq \Pi(L \cup L') \times \mathbb{N}$ as

$$\mathsf{join}(\mathcal{A}, \mathcal{B}) := \{(p_{\uparrow L'} \sqcup q_{\uparrow L}, w_1 + w_2) \,:\, (p, w_1) \in \mathcal{A}, (q, w_2) \in \mathcal{B}\}.$$

This operator is the one from [9]. It is used mainly to construct partial solutions of $G \oplus H$ from partial solutions of G and H.

The following proposition could be easily proved. Here also, we assume that $\log(|\mathcal{A}|) \leq |L|^{O(1)}$ for each $\mathcal{A} \subseteq \Pi(L) \times \mathbb{N}$ (this can be established by applying the operator rmc).

Proposition 4.15. *The operator* join *can be performed in time* $|\mathcal{A}| \cdot |\mathcal{B}| \cdot |L \cup L'|^{O(1)}$ *and the size of its output is upper-bounded by* $|\mathcal{A}| \cdot |\mathcal{B}|$.

The following is the same as Definition 4.5, but does not require acyclicity.

Definition 4.16 ([9])**.** *For* $\mathcal{A} \subseteq \Pi(L) \times \mathbb{N}$*, with* L *a finite set, and* $q \in \Pi(L)$*, let*

$$\mathbf{opt}(\mathcal{A}, q) := opt\{w \,:\, (p, w) \in \mathcal{A}, p \sqcup q = \{L\}\}.$$

A set of weighted partitions $\mathcal{B} \subseteq \Pi(L) \times \mathbb{N}$ represents $\mathcal{A}$ *if for each* $q \in \Pi(L)$*, it holds that* $\mathbf{opt}(\mathcal{A}, q) = \mathbf{opt}(\mathcal{B}, q)$.

Let Z *and* L' *be two finite sets. A function* $f : 2^{\Pi(L) \times \mathbb{N}} \times Z \to 2^{\Pi(L') \times \mathbb{N}}$ *is said to* preserve representation *if for each* $\mathcal{A}, \mathcal{B} \subseteq \Pi(L) \times \mathbb{N}$ *and* $z \in Z$*, it holds that* $f(\mathcal{B}, z)$ *represents* $f(\mathcal{A}, z)$ *whenever* $\mathcal{B}$ *represents* $\mathcal{A}$.

Lemma 4.17 ([9])**.** *The operators* rmc*,* proj *and* join *preserve representation.*

Theorem 4.18 ([9])**.** *There exists an algorithm* reduce *that, given a set of weighted partitions* $\mathcal{A} \subseteq \Pi(L) \times \mathbb{N}$*, outputs in time* $|\mathcal{A}| \cdot 2^{(\omega-1)|L|} \cdot |L|^{O(1)}$ *a subset* $\mathcal{B}$ *of* $\mathcal{A}$ *that represents* $\mathcal{A}$*, and such that* $|\mathcal{B}| \leq 2^{|L|-1}$.

We use the following function to upper bound the amount of information we need to store in our dynamic programming tables concerning the (σ, ρ)-domination. For every non-empty finite or co-finite subset $\mu \subseteq \mathbb{N}$, we define $d(\mu)$ such as

$$d(\mu) := \begin{cases} 0 & \text{if } \mu = \mathbb{N}, \\ 1 + \mathsf{min}(\mathsf{max}(\mu), \mathsf{max}(\mathbb{N} \setminus \mu)) & \text{otherwise.} \end{cases}$$

For example, $d(\mathbb{N}^+) = 1$ and for every $c \in \mathbb{N}$, we have $d(\{0, \ldots, c\}) = c + 1$.

The definition of d is motivated by the following observation which is due to the fact that, for all $\mu \subseteq \mathbb{N}$, if $d(\mu) \in \mu$, then μ is co-finite and contains $\mathbb{N} \setminus \{1, \ldots, d(\mu) - 1\}$.

Fact 4.19. *Let* $A \subseteq V(G)$ *and let* (σ, ρ) *be a pair of finite or co-finite subsets of* $\mathbb{N}$*. Let* $d := \mathsf{max}(d(\sigma), d(\rho))$*. For all* $X \subseteq A$ *and* $Y \subseteq \overline{A}$*,* $X \cup Y$ (σ, ρ)*-dominates* A *if and only if* $\mathsf{min}(d, |N(v) \cap X| + |N(v) \cap Y|)$ *belongs to* σ *(resp.* ρ*) if* $v \in X$ *(resp.* $v \notin X$*).*

Let us describe with a concrete example the information we need concerning the (σ, ρ)-domination. We say that a set $D \subseteq V(G)$ is a 2-dominating set if every vertex in $V(G)$ has at least two neighbors in D. It is worth noticing that a 2-dominating set is an $(\mathbb{N} \setminus \{0, 1\}, \mathbb{N} \setminus \{0, 1\})$-dominating set and $d(\mathbb{N} \setminus \{0, 1\}) = 2$. Let H be a k-labeled graph used in an irredundant k-expression of a graph G. Assuming $D_H \subseteq V(H)$ is a subset of a 2-dominating set D of G, we would like to characterize the sets $Y \subseteq V(G) \setminus V(H)$ such that $D_H \cup Y$ is a 2-dominating set of H. One first observes that D_H is not necessarily a 2-dominating set of H. Since we want to

2-dominate $V(H)$, we need to know for each vertex x in $V(H)$ how many neighbors it needs in addition to be 2-dominated . For doing so, we associate D_H with a sequence $R' := (r'_1, \ldots, r'_k)$ over $\{0,1,2\}^k$ such that, for each $i \in [k]$, every vertex in $lab_H^{-1}(i)$ has at least $2 - r'_i$ neighbors in D_H. For example, if $r'_i = 1$, then every i-vertex has at least one neighbor in D_H. Notice that D_H can be associated with several such sequences. This sequence is enough to characterize what we need to 2-dominate $V(H)$ since the vertices with the same label in H have the same neighbors in the graph $(V(G), E(G) \setminus E(H))$ and each vertex needs at most 2 additional neighbors to be 2-dominated.

In order to update the sequence $R' := (r'_1, \ldots, r'_k)$ associated with a set D_H, we associate with D_H another sequence $R := (r_1, \ldots, r_k)$ over $\{0,1,2\}^k$ such that r_i corresponds to the minimum between 2 and the number of i-vertices in D_H, for each $i \in [k]$. This way, when we apply an operation $add_{i,j}$ on H, we know that every i-vertex has at least $2 - r'_i + r_j$ neighbors in D_H in the graph $add_{i,j}(H)$. For example, if $r_j = 1$ and every i-vertex has at least one neighbor in H that belongs to D_H, then we know that every i-vertex is 2-dominated by D_H in the graph $add_{i,j}(H)$.

Let (σ, ρ) be a fixed pair of non-empty finite or co-finite subsets of $\mathbb{N}$. Let's first show how to compute an optimum connected (σ,ρ)-dominating set. We consider node-weighted Steiner tree and connected co-(σ,ρ)-dominating set at the end of the section. Let $d := \mathsf{max}\{d(\sigma), d(\rho)\}$.

The following definitions formalize the intuitions we give for 2-dominating set to the (σ,ρ)-domination.

Definition 4.20 (Certificate graph of a solution)**.** *Let G be a k-labeled graph, and $R' := (r'_1, \ldots, r'_k) \in \{0, \ldots, d\}^k$. Let $V^+ := \{v_1^1, \ldots, v_d^1, v_1^2, \ldots, v_d^2, \ldots, v_1^k, \ldots, v_d^k\}$ be a set disjoint from $V(G)$ and of size $d \cdot k$. Let $V^+(R') := V_1^+(R') \cup \cdots \cup V_k^+(R')$ with*

$$V_i^+(R') := \begin{cases} \varnothing & \text{if } r'_i = 0, \\ \{v_1^i, \ldots, v_{r'_i}^i\} & \text{otherwise.} \end{cases}$$

The certificate graph of G with respect to R', denoted by $\mathbf{CG}(G, R')$, is the graph $(V(G) \cup V^+(R'), E(G) \cup E_1^+ \cup \cdots \cup E_k^+)$ with

$$E_i^+ = \{\{v, v_t^i\} \,:\, v_t^i \in V_i^+(R') \wedge v \in lab_G^{-1}(i)\}.$$

It is worth noticing that E_i^+ is empty if $lab_G^{-1}(i) = \varnothing$ or $V_i^+(R') = \varnothing$.

Definition 4.21. *Let G be a k-labeled graph. For each $D \subseteq V(G)$ and $i \in [k]$, let $r_{i,G}^d(D) := \mathsf{min}(d, |lab_G^{-1}(i) \cap D|)$ and let $r_G^d(D) := (r_{1,G}^d(D), \ldots, r_{k,G}^d(D))$.*

The sequence $r_G^d(D)$ describes how each label class is intersected by D up to d vertices. Moreover, notice that $|\{r_G^d(D) \,:\, D \subseteq V(G)\}| \leq |\{0, \ldots, d\}^k| \leq (d+1)^k$.

The motivation behind these two sequences is that, in order to computing an optimum (σ,ρ)-dominating set, it is enough to compute, for any k labeled graph H used in an irredundant k-expression of a graph G and for each $R, R' \in \{0, \ldots, d\}^k$, the optimum weight of a set $D \subseteq V(H)$ such that

- $r_H(D) = R$,
- $D \cup V^+(R')$ (σ,ρ)-dominates $V(H)$ in the graph $\mathbf{CG}(H, R')$.

It is worth noticing that the sequences $r_H^d(D)$ and R' are similar to the notion of *d-neighbor equivalence* introduced in [18].

We can assume w.l.o.g. that $d \neq 0$, that is $\sigma \neq \mathbb{N}$ or $\rho \neq \mathbb{N}$. Indeed, if $\sigma = \rho = \mathbb{N}$, then the problem of finding a minimum (or maximum) weighted (co-)connected (σ, ρ)-dominating set is trivial. For computing an optimum connected (σ, ρ)-dominating set, we will as in Section 4.1.2 keep partitions of a subset of labels corresponding to the connected components of the sets D (that are candidates for the (σ, ρ)-domination). As $d \neq 0$, we know through $r_H(D)$ the label classes intersected by D. Moreover, we know through $R' = (r'_1, \ldots, r'_k)$ whether the i-vertices in such a D will have a neighbor in any extension D' of D into a (σ, ρ)-dominating set. It is enough to keep the partition of the labels i with $r_{i,H}(D) \neq 0$ and $r'_i \neq 0$ that corresponds to the equivalence classes of the equivalence relation $\sim$ on $\{i \in [k] : r_i \neq 0 \text{ and } r'_i \neq 0\}$ where $i \sim j$ if and only if an i-vertex is connected to a j-vertex in $\mathbf{CG}(G, R')[D \cup V^+]$.

We use the following definition to simplify the notations.

Definition 4.22. *For $R = (r_1, \ldots, r_k), R' = (r'_1, \ldots, r'_k) \in \{0, \ldots, d\}^k$, we define $\mathsf{active}(R, R') = \{i \in [k] : r_i \neq 0$ and $r'_i \neq 0\}$.*

We are now ready to define the sets of weighted partitions whose representative sets we manipulate in our dynamic programming tables.

Definition 4.23 (Weighted partitions in $\mathcal{D}_G[R, R']$)**.** *Let G be a k-labeled graph, and $R, R' \in \{0, \ldots, d\}^k$. The entries of $\mathcal{D}_G[R, R']$ are all the weighted partitions $(p, w) \in \Pi(\mathsf{active}(R, R')) \times \mathbb{N}$ so that there exists a set $D \subseteq V(G)$ such that $\mathsf{w}(D) = w$, and*

1. *$r_G^d(D) = R$,*
2. *$D \cup V^+(R')$ (σ, ρ)-dominates $V(G)$ in $\mathbf{CG}(G, R')$,*
3. *if $\mathsf{active}(R, R') = \varnothing$, then $G[D]$ is connected, otherwise for each connected component C of $G[D]$, we have $C \cap lab_G^{-1}(\mathsf{active}(R, R')) \neq \varnothing$,*
4. *$p = \mathsf{active}(R, R')/\sim$ where $i \sim j$ if and only if an i-vertex is connected to a j-vertex in $\mathbf{CG}(G, R')[D \cup V^+(R')]$.*

Conditions (1) and (2) guarantee that (p, w) corresponds to a set D that is coherent with R and R'. Condition (3) guarantees that each partial solution can be extended into a connected graph. Contrary to Section 4.1.2, the set of labels expected to play a role in the connectivity (i.e. $\mathsf{active}(R, R')$) can be empty. In this case, we have to make sure that the weighted partitions represent connected solutions. It is worth mentioning that $G[\varnothing]$, i.e. the empty graph, is considered as a connected graph. Observe that for each $(p, w) \in \mathcal{D}_G[R, R']$, the partition p has the same meaning as in Section 4.1.2. It is worth noticing that $\sim$ is an equivalence relation because if $i \in \mathsf{active}(R, R')$, then all the vertices in $lab_G^{-1}(i)$ are connected in $\mathbf{CG}(G, R')$ through the vertices in $V_i^+(R')$. In fact, the relation $\sim$ is equivalent to the transitive closure of the relation $\asymp$ where $i \asymp j$ if there exist an i-vertex and a j-vertex in the same connected component of $G[D]$.

In the sequel, we call a pair $(D, (p, \mathsf{w}(D)))$ a *candidate solution* in $\mathcal{D}_G[R, R']$ if the partition $p = \mathsf{active}(R, R')/\sim$ where $i \sim j$ if and only if an i-vertex is connected to a j-vertex in $\mathbf{CG}(G, R')[D \cup V^+(R')]$. If in addition Conditions (1)-(3) are satisfied, we call $(D, (p, \mathsf{w}(D)))$ a *solution* in $\mathcal{D}_G[R, R']$.

It is straightforward to check that the weight of an optimum solution is the optimum over all $R \in \{0, \ldots, d\}^k$ of $opt\{w : (\varnothing, w) \in \mathcal{D}_G[R, \{0\}^k]\}$ for a k-labeled graph G.

Analogously to Section 4.1.2 our dynamic programming algorithm will store a subset of $\mathcal{D}_G[R, R']$ of size 2^{k-1} that represents $\mathcal{D}_G[R, R']$. Recall that we suppose that any graph is given with an irredundant k-expression.

Computing tab_G for $G = \mathbf{1}(x)$. For $(r_1) \in \{0, \dots, d\}, (r'_1) \in \{0, \dots, d\}$, let

$$tab_G[(r_1), (r'_1)] := \begin{cases} \{(\varnothing, 0)\} & \text{if } r_1 = 0 \text{ and } r'_1 \in \rho, \\ \{(\{\{1\}\}, \mathsf{w}(x))\} & \text{if } r_1 = 1 \text{ and } r'_1 \in \sigma, \\ \varnothing & \text{otherwise.} \end{cases}$$

Since there is only one vertex in G labeled 1, $\mathcal{D}_G[(r_1), (r'_1)]$ is empty whenever $r_1 \notin \{0, 1\}$. Also, the possible solutions are either to put x in the solution ($r_1 = 1$) or to discard it ($r_1 = 0$); in both cases we should check that x is (σ, ρ)-dominated by $V_1^+(R')$. We deduce then that $tab_G[(r_1), (r'_1)] = \mathcal{D}_G[(r_1), (r'_1)]$.

Computing tab_G for $G = ren_{i \to j}(H)$. We can suppose that H is k-labeled and that $i = k$. Let $R := (r_1, \dots, r_{k-1}), R' := (r'_1, \dots, r'_{k-1}) \in \{0, \dots, d\}^{k-1}$.

To compute $tab_G[R, R']$, we define $\mathcal{S}$, the set of tuples coherent with respect to R and Condition (1), as follows

$$\mathcal{S} := \{(s_1, \dots, s_k) \in \{0, \dots, d\}^k \,:\, r_j = \mathsf{min}(d, s_k + s_j) \text{ and } \forall \ell \in [k] \setminus \{i, j\}, s_\ell = r_\ell\}.$$

It is worth noticing that we always have $(r_1, \dots, r_{k-1}, 0) \in \mathcal{S}$. Moreover, if $r_j = 0$, then $\mathcal{S} = \{(r_1, \dots, r_{k-1}, 0)\}$. We define also $S' = (s'_1, \dots, s'_k) \in \{0, \dots, d\}^k$ with $s'_k = r'_j$ and $s'_\ell = r'_\ell$ for all $\ell \in [k-1]$. Notice that S' is the only tuple compatible with R' and Condition (2) since for every $v \in lab_G^{-1}(j)$, the number of vertices in $V^+(R')$ adjacent to v in $\mathbf{CG}(G, R')$ is the same as the number of vertices in $V^+(S')$ adjacent to v in $\mathbf{CG}(H, S')$.

(a) If $r_j = 0$ or $r'_j = 0$, then we let

$$tab_G[R, R'] := \mathsf{reduce}\left(\mathsf{rmc}\left(\bigcup_{S \in \mathcal{S}} tab_H[S, S']\right)\right).$$

In this case, the vertices in $lab_G^{-1}(j)$ are not expected to play a role in terms of connectivity anymore as either we expect no additional neighbors for them or they are not intersected by the partial solutions.

(b) Otherwise, we let $tab_G[R, R'] := \mathsf{reduce}(\mathsf{rmc}(\mathcal{A}))$ with

$$\mathcal{A} := \mathsf{proj}\left(\{k\}, \bigcup_{S \in \mathcal{S}} \mathsf{join}(tab_H[S, S'], \{(\{\{j, k\}\}, 0)\})\right).$$

Intuitively, we put in $tab_G[R, R']$ all the weighted partitions (p, w) which belong to the tables $tab_H[S, S']$ with $S \in \mathcal{S}$, after merging the blocks in p containing k and j, and removing k from the resulting partition.

Lemma 4.24. *Let $G = ren_{k \to j}(H)$ with H a k-labeled graph. For all $R := (r_1, \dots, r_{k-1}), R' := (r'_1, \dots, r'_{k-1}) \in \{0, \dots, d\}^{k-1}$, the table $tab_G[R, R']$ is a representative set of $\mathcal{D}_G[R, R']$ assuming that $tab_H[S, S']$ is a representative set of $\mathcal{D}_H[S, S']$ for all $S \in \{0, \dots, d\}^k$.*

Proof. Since the used operators preserve representation, it is enough to prove that each weighted partition added to $tab_G[R, R']$ belongs to $\mathcal{D}_G[R, R']$, and that

- in Case (a), we have $\mathcal{D}_G[R, R'] \subseteq \bigcup_{S \in \mathcal{S}} \mathcal{D}_H[S, S']$, and

- in Case (b), we have $\mathcal{D}_G[R,R'] \subseteq \mathcal{A}$ if we let $tab_H[S,S'] = \mathcal{D}_H[S,S']$ for every $S \in \{0,\ldots,d\}^k$.

Let $(D,(p,w))$ be a solution in $\mathcal{D}_G[R,R']$. We start by proving that we have $(p,w) \in \bigcup_{S\in\mathcal{S}} \mathcal{D}_H[S,S']$ if we are in Case (a), or $(p,w) \in \mathcal{A}$ if we are in Case (b). By the definition of $\mathcal{S}$, we deduce that $r_H(D) \in \mathcal{S}$. Indeed, $r_{j,G}(D) = \mathsf{min}(d, |D \cap lab_G^{-1}(j)|)$ equals $\mathsf{min}(d, r^d_{j,H}(D) + r^d_{k,H}(D))$ because $lab_G^{-1}(j) = lab_H^{-1}(\{j,k\})$.

Let $p' \in \Pi(\mathsf{active}(r^d_H(D), S'))$ such that $(D,(p',w))$ is a candidate solution in $\mathcal{D}_H[r^d_H(D),R']$. We claim that $(D,(p',w))$ is a solution in $\mathcal{D}_H[r^d_H(D),R']$. Condition (1) is trivially satisfied. We deduce from the definition of S' that Condition (2) is satisfied. We claim that Condition (3) is satisfied. If $R' = \{0\}^{k-1}$, then we have $S' = \{0\}^k$ and Condition (3) is satisfied because $H[D] = G[D]$ must be connected since $(D,(p,w))$ is a solution in $\mathcal{D}_G[R,R']$. Otherwise, every connected component C of $G[D] = H[D]$ intersects $lab_G^{-1}(\mathsf{active}(R,R'))$. Let C be a connected component of $H[D]$. If C contains a vertex labeled ℓ in G with $\ell \in \mathsf{active}(R,R') \setminus \{j\}$, then by definition of S', we have $\ell \in \mathsf{active}(r^d_H(D), S')$. Suppose now, C contains a vertex v in $lab_G^{-1}(j)$ and $j \in \mathsf{active}(R,R')$. If v is labeled k in H, then $r^d_{k,H}(D) \neq 0$ and thus k belongs to $\mathsf{active}(r^d_H(D), S')$ because $s'_k = r'_j \neq 0$. Symmetrically, if v is labeled j in H, then $j \in \mathsf{active}(r^d_H(D), S')$. In both cases, C intersects $lab_H^{-1}(\mathsf{active}(r^d_H(D), S'))$. We can conclude that Condition (3) is satisfied. Hence, $(D,(p',w))$ is a solution in $\mathcal{D}_H[r^d_H(D),R']$.

If $r_j = 0$ or $r'_j = 0$ (Case (a)), then $\mathsf{active}(S,S') = \mathsf{active}(R,R')$. Consequently, it is easy to see that $p = p'$ f (3) and thus $(p,w) \in tab_H[S,S']$.

Assume now that $r_j \neq 0$ and $r'_j \neq 0$ (Case (b)). Let $D_k := D \cap lab_H^{-1}(k)$ and $D_j := D \cap lab_H^{-1}(j)$. Observe that the graph $\mathbf{CG}(G,R')[D \cup V^+(R')]$ can be obtained from the graph $\mathbf{CG}(H,S')[D \cup V^+(S')]$ by removing the vertices in $V^+_k(R')$ and by adding the edges between $V^+_j(R')$ and D_k. Hence, p is obtained from p' by merging the blocks containing j and k, and by removing k. Thus, we can conclude that $(p,w) \in \mathcal{A}$.

It remains to prove that each weighted partition (p,w) added to $tab_G[R,R']$ belongs to $\mathcal{D}_G[R,R']$. Let (p,w) be a weighted partition added to $tab_G[R,R']$ from $(p',w) \in tab_H[S,S']$, and let $D \subseteq V(H)$ such that $(D,(p',w))$ is a solution in $\mathcal{D}_H[S,S']$. We want to prove that $(D,(p,w))$ is a solution in $\mathcal{D}_G[R,R']$. From the definitions of $\mathcal{S}$ and S', we deduce that D satisfies Conditions (1) and (2). Since $\mathsf{active}(S,S') \setminus \{k\} = \mathsf{active}(R,R')$ and $G[D] = H[D]$, we deduce that D satisfies also Condition (3) that $lab_G^{-1}(j) = lab_H^{-1}(\{j,k\})$.

If $r_j = 0$ or $r'_j = 0$, then $\mathsf{active}(S,S') = \mathsf{active}(R,R')$. Consequently, we deduce that $p = p'$ and that $(D,(p,w))$ satisfies Condition (4). Otherwise, we deduce from the previous observations concerning the differences between $\mathbf{CG}(G,R')[D \cup V^+(R')]$ and $\mathbf{CG}(H,S')[D \cup V^+(S')]$, that $(D,(p,w))$ satisfies Condition (4). In both cases, we can conclude that $(D,(p,w))$ is a solution in $\mathcal{D}_G[R,R']$. □

Computing tab_G for $G = add_{i,j}(H)$. We suppose w.l.o.g. that H is k-labeled. Let $R := (r_1,\ldots,r_k) \in \{0,\ldots,d\}^k, R' := (r'_1,\ldots,r'_k) \in \{0,\ldots,d\}^k$.

Let $S' := (s'_1,\ldots,s'_k) \in \{0,\ldots,d\}^k$ such that $s'_i := \mathsf{min}(d, r'_i + r_j)$, $s'_j := \mathsf{min}(d, r'_j + r_i)$, and $s'_\ell := r'_\ell$ for all $\ell \in [k] \setminus \{i,j\}$. It is easy to see that S' is the only tuple compatible with R' and Condition (2).

(a) If $\mathsf{active}(R,R') = \varnothing$, then we let

$$tab_G[R,R'] := \mathsf{rmc}\left(\{(\varnothing,w) \,:\, (p,w) \in tab_H[R,S']\}\right).$$

In this case, the partial solutions in $tab_G[R, R']$ are associated with connected solutions by Condition (3). The partial solutions in $tab_H[R, S']$ trivially satisfy this condition in G. Notice that $tab_G[R, R']$ represents $\mathcal{D}_G[R, R']$ because the function $f : 2^{\Pi(\mathsf{active}(R,R'))\times\mathbb{N}} \to 2^{\{\varnothing\}\times\mathbb{N}}$ with $f(\mathcal{A}) := \{(\varnothing, w) : (p, w) \in \mathcal{A}\}$ preserves representation.

(b) If $r_i = 0$ or $r_j = 0$, we let $tab_G[R, R'] := tab_H[R, S']$. We just copy all the solutions not intersecting $lab_G^{-1}(i)$ or $lab_G^{-1}(j)$. In this case, the connectivity of the solutions is not affected by the $add_{i,j}$ operation.

(c) Otherwise, we let $tab_G[R, R'] := \mathsf{rmc}(\mathcal{A})$, where

$$\mathcal{A} := \mathsf{proj}(\{t \in \{i, j\} : r'_t = 0\}, \mathsf{join}(tab_H[R, S'], \{(\{\{i, j\}\}, 0)\})).$$

In this last case, we have $i, j \in \mathsf{active}(R, S')$. We put in $tab_G[R, R']$, the weighted partitions of $tab_H[R, S']$ after merging the blocks containing i and j, and removing i or j if, respectively, $r'_i = 0$ and $r'_j = 0$, i.e., if they don't belong, respectively, to $\mathsf{active}(R, R')$.

It is worth noticing that if $|tab_H[R, S']| \leq 2^{k-1}$, then we have $|tab_G[R, R']| \leq 2^{k-1}$. Thus, we do not have to use the function reduce to compute tab_G.

Lemma 4.25. *Let $G = add_{i,j}(H)$ be a k-labeled graph such that there are no edges between an i-vertex and a j-vertex in H. For all tuples $R := (r_1, \ldots, r_k), R' := (r'_1, \ldots, r'_k) \in \{0, \ldots, d\}^k$, the table $tab_G[R, R']$ is a representative set of $\mathcal{D}_G[R, R']$ assuming that $tab_H[R, S']$ is a representative set of $\mathcal{D}_H[R, S']$.*

Proof. Since the used operators preserve representation, it is enough to prove that every weighted partition added to $tab_G[R, R']$ belongs to $\mathcal{D}_G[R, R']$, and that

- in Case (a), we have $\mathcal{D}_G[R, R'] \subseteq \{(\varnothing, w) : (p, w) \in \mathcal{D}_H[R, S']\}$,
- in Case (b), we have $\mathcal{D}_G[R, R'] \subseteq \mathcal{D}_H[R, S']$, and
- in Case (c), we have $\mathcal{D}_G[R, R'] \subseteq \mathcal{A}$ if we let $tab_H[R, S'] = \mathcal{D}_H[R, S']$.

Let $(D, (p, w))$ be a solution in $\mathcal{D}_G[R, R']$. Let $p' \in \Pi(\mathsf{active}(R, S'))$ such that $(D, (p', w))$ is a candidate solution in $\mathcal{D}_H[R, S']$. We claim that $(D, (p', w))$ is a solution in $\mathcal{D}_H[R, S']$. Condition (1) is trivially satisfied because $lab_G(v) = lab_H(v)$ for all $v \in V(G)$. We claim that Condition (2) is satisfied, that is $D \cup V^+(S')$ (σ, ρ) dominates $V(H)$ in $\mathbf{CG}(H, S')$. It is quite easy to see that $D \cup V^+(S')$ (σ, ρ) dominates $V(H) \setminus lab_H^{-1}(\{i, j\})$. Let $v \in lab_H^{-1}(i)$. We claim that v is (σ, ρ) dominated by $D \cup V^+(S')$. Because we assume that there are no edges between an i-vertex and a j-vertex in H, we have $|N_G(v) \cap D| = |N_H(v) \cap D| + |D \cap lab_G^{-1}(j)|$. Since $D \cup V^+(R')$ (σ, ρ)-dominates $V(G)$, if $v \in D$, then $|N_G(v) \cap D| + r'_i \in \sigma$, otherwise, $|N_G(v) \cap D| + r'_i \in \rho$. By Fact 4.19, we conclude that $\mathsf{min}(d, |N_H(v) \cap D| + |D \cap lab_G^{-1}(j)| + r'_i)$ belongs to σ if $v \in D$, otherwise it belongs to ρ. As $\mathsf{min}(d, r'_i + |D \cap lab_G^{-1}(j)|) = \mathsf{min}(d, r'_i + r_j) = s'_i$, we deduce that $D \cup V^+(S')$ (σ, ρ) dominates v. Thus, every i-vertex is (σ, ρ)-dominated by $D \cup V^+(S')$ in $\mathbf{CG}(H, S')$. Symmetrically, we deduce that every j-vertex is (σ, ρ)-dominated by $D \cup V^+(S')$ in $\mathbf{CG}(H, S')$. Hence, Condition (2) is satisfied.

In order to prove that Condition (3) is satisfied, we distinguish the following cases. First, suppose that $\mathsf{active}(R, R') = \varnothing$. By Condition (3), $G[D]$ is connected. As $G = add_{i,j}(H)$, the graph $H[D]$ is obtained from $G[D]$ by removing all edges between the i-vertices and the j-vertices. We deduce that either $H[D]$ is connected (if $r_i = 0$ or $r_j = 0$), or that every connected component of $H[D]$ contains at least one vertex whose label is i or j (otherwise $G[D]$ would not be connected). In both cases, Condition (3) is satisfied.

Assume now that $\mathsf{active}(R, R') \neq \varnothing$. If $r_i = 0$ (resp. $r_j = 0$), then, by definition of S', we have $s'_j = r'_j$ (resp. $s'_i = r'_i$). Therefore, if $r_i = 0$ or $r_j = 0$, then we have $\mathsf{active}(R, R') = \mathsf{active}(R, S')$, and Condition (3) is trivially satisfied. Otherwise, if $r_i \neq 0$ and $r_j \neq 0$, then, by definition of S', we have $s'_i \neq 0$, $s'_j \neq 0$, and thus $i, j \in \mathsf{active}(R, S')$. In this case, we conclude that Condition (3) is satisfied because, for every connected component C of $H[D]$, either C is a connected component of $G[D]$, or C contains at least one vertex whose label is i or j.

Hence, $(D, (p', w))$ is a solution in $\mathcal{D}_H[R, S']$. If $\mathsf{active}(R, R') = \varnothing$, then $p = \varnothing$ and (p, w) is added in $tab_G[R, R']$. Else if $r_i = 0$ or $r_j = 0$, then $p' = p$ and (p, w) is also added to $tab_G[R, R']$. Otherwise, it is easy to see that p is obtained from p' by merging the blocks of p' containing i and j, and by removing them if they belong to $\{\ell \in [k] : r'_\ell = 0\}$. Thus, we can conclude that $(p, w) \in \{(\varnothing, w) : (p, w) \in \mathcal{D}_H[R, S']\})$ in Case (a), $(p, w) \in \mathcal{D}_H[R, S']$ in Case (b), and $(p, w) \in \mathcal{A}$ in Case (c).

It remains to prove that every weighted partition added to $tab_G[R, R']$ belongs to $\mathcal{D}_G[R, R']$. Let (p, w) be a weighted partition added to $tab_G[R, R']$ from $(p', w) \in tab_H[R, S']$ and let $D \subseteq V(H)$ such that $(D, (p', w))$ is a solution in $\mathcal{D}_H[R, S']$. We claim that $(D, (p, w))$ is a solution in $\mathcal{D}_G[R, R']$. Obviously, Condition (1) is satisfied. We deduce that Condition (2) is satisfied from the definition of S' and the previous arguments. We claim that Condition (3) is satisfied. Suppose first that $R' = \{0\}^k$. Then either $S' = \{0\}^k$ or $\{i, j\} = \mathsf{active}(R, S')$. Indeed, we have $\mathsf{active}(R, S') \subseteq \{i, j\}$. Now, $i \in \mathsf{active}(R, S')$ if and only if $r_i \neq 0$ and $s'_i \neq 0$, and then, by definition of S', $s'_j = r_i \neq 0$ and $s'_i = r_j \neq 0$. Thus, $i \in \mathsf{active}(R, S')$ if and only if $j \in \mathsf{active}(R, S')$. Therefore, we conclude that either $H[D]$ is connected or every connected component C of $H[D]$ contains at least a vertex whose label is i or j. As $G[D]$ is obtained from $H[D]$ by adding all the edges between the i-vertices and the j-vertices, we conclude that, in both cases, $G[D]$ is connected.

Otherwise, if $R' \neq \{0\}^k$, then every connected component of $H[D]$ contains at least one vertex whose label is in $\mathsf{active}(R, S')$. If $r_i = 0$ or $r_j = 0$, then we are done because $\mathsf{active}(R, R') = \mathsf{active}(R, S')$ by definition of S'. Suppose now that $r_i \neq 0$ and $r_j \neq 0$. In this case, we have $i, j \in \mathsf{active}(R, S')$ from the definition of S'. Assume towards a contradiction that there exists a connected component C in $G[D]$ such that C does not intersect $lab_G^{-1}(\mathsf{active}(R, R'))$. Notice that C intersects $lab_G^{-1}(\mathsf{active}(R, S'))$ because C is a union of connected components of $H[D]$. As $\mathsf{active}(R, S') \setminus \mathsf{active}(R, R') \subseteq \{i, j\}$, we deduce that C contains at least one vertex whose label is i or j. Since the i-vertices and the j-vertices of D are in the same connected component of $G[D]$, we have $lab_G^{-1}(\{i, j\}) \cap D \subseteq C$. Therefore, we conclude that $\mathsf{active}(R, S') \setminus \mathsf{active}(R, R') = \{i, j\}$. It follows, that all the connected components of $H[D]$ that intersect $lab_G^{-1}(\{i, j\})$ do not intersect $lab_G^{-1}(\mathsf{active}(R, S') \setminus \{i, j\})$ because they are contained in C. We can conclude that $\{i, j\}$ is a block of $p'' := p' \sqcup \{\{i, j\}\}_{\uparrow \mathsf{active}(R, S')}$. Hence, we have $\mathsf{proj}(\{t \in \{i, j\} : r'_t = 0\}, \{(p'', w)\}) = \varnothing$ because $\{t \in \{i, j\} : r'_t = 0\} = \{i, j\}$. This contradicts the fact that (p, w) is obtained from (p', w'). Thus, Condition (3) is satisfied.

We deduce from the previous observations concerning Condition (4) that this condition is also satisfied. Thus, every solution (p, w) added to $tab_G[R, R']$ belongs to $\mathcal{D}_G[R, R']$. □

Computing tab_G for $G = G_a \oplus G_b$. Assume that G is k-labeled and let $R := (r_1, \ldots, r_k), R' := (r'_1, \ldots, r'_k) \in \{0, \ldots, d\}^k$. The following notion characterizes the pairs (A, B) compatible with R with respect to Condition (1). We say that $((a_1, \ldots, a_k), (b_1, \ldots, b_k))$ is *R-compatible* if and only if for all $i \in [k]$, we have $r_i = \mathsf{min}(d, a_i + b_i)$. We can suppose w.l.o.g. that G_a and G_b are k-labeled[6].

[6]If, for example, $lab_{G_a} : V(G_a) \to [k] \setminus \{1\}$ and $lab_{G_b} : V(G_b) \to [k]$, we consider that $((a_2, \ldots, a_k), (b_1, \ldots, b_k))$ is R-compatible if $b_1 = r_1$ and for all $i \in [k] \setminus \{1\}$, we have $r_i = min(d, a_i + b_i)$.

(a) If $R' = \{0\}^k$, then we let $tab_G[R, R'] := \mathsf{reduce}(\mathsf{rmc}(tab_{G_a}[R, R'] \cup tab_{G_b}[R, R']))$ if $0 \in \rho$, otherwise we let $tab_G[R, R'] = \varnothing$. Condition (3) implies that the partial solutions in $\mathcal{D}_G[R, R']$ are either fully contained in $V(G_a)$ or in $V(G_b)$ since there are no edges between these vertex sets in G. Moreover, in order to satisfy Condition (2), we must have $0 \in \rho$.

(b) Otherwise, we let $tab_G[R, R'] := \mathsf{reduce}(\mathsf{rmc}(\mathcal{A}))$ where

$$\mathcal{A} := \bigcup_{(A,B) \text{ is } R\text{-compatible}} \mathsf{join}(tab_{G_a}[A, R'], tab_{G_b}[B, R']).$$

Lemma 4.26. *Let $G = G_a \oplus G_b$ be a k-labeled graph. For all $R = (r_1, \ldots, r_k), R' = (r'_1, \ldots, r'_k) \in \{0, \ldots, d\}^k$, the table $tab_G[R, R']$ is a representative set of $\mathcal{D}_G[R, R']$ assuming that $tab_{G_a}[A, R']$ and $tab_{G_b}[B, R']$ are representative sets of $\mathcal{D}_{G_a}[A, R']$ and $\mathcal{D}_{G_b}[B, R']$, respectively, for all $A, B \in \{0, \ldots, d\}^k$.*

Proof. Since the used operators preserve representation, it is easy to see that if $R' = \{0\}^k$, then we are done as $\mathcal{D}_G[R, R'] = \mathcal{D}_{G_a}[R, R'] \cup \mathcal{D}_{G_b}[R, R']$ if $0 \in \rho$, otherwise $\mathcal{D}_G[R, R'] = \varnothing$. Indeed, by Condition (3), for all solutions $(D, (p, w))$ in $\mathcal{D}_G[R, R']$, the graph $G[D]$ must be connected. Since $G = G_a \oplus G_b$, there are no edges between $V(G_a)$ and $V(G_b)$ in $G[D]$. Thus, D is either included in $V(G_a)$ or in $V(G_b)$. Since $V(G_a) \neq \varnothing$ and $V(G_b) \neq \varnothing$, we have $V(G) \setminus D \neq \varnothing$. This implies that $0 \in \rho$, otherwise the vertices in $V(G) \setminus D$ will not be (σ, ρ)-dominated by $D \cup V^+(R') = D$ in $\mathbf{CG}(G, R')$ as $V^+(R') = \varnothing$.

In the following, we assume that $R' \neq \{0\}^k$. Since the used operators preserve representation, it is enough to prove that $\mathcal{D}_G[R, R'] = \mathcal{A}$ if we let $tab_{G_t}[S, R'] = \mathcal{D}_{G_t}[S, R']$ for all $S \in \{0, \ldots, d\}^k$ and $t \in \{a, b\}$.

Let $(D, (p, w))$ be a solution in $\mathcal{D}_G[R, R']$. We start by proving that $(p, w) \in \mathcal{A}$. Let $D_a = D \cap V(G_a)$ and $D_b = D \cap V(G_b)$. From the definition of R-compatibility, we deduce that $r^d_{G_a}(D_a)$ and $r^d_{G_b}(D_b)$ are R-compatible. Indeed, we have $\mathsf{min}(d, |lab_G^{-1}(i) \cap D|) = \mathsf{min}(d, r^d_{G_a}(D_a) + r^d_{G_b}(D_b))$ for all $i \in [k]$.

Let $p_a \in \Pi(\mathsf{active}(r^d_{G_a}(D_a), R'))$ such that $(D_a, (p_a, \mathsf{w}(D_a)))$ is a candidate solution in the table $\mathcal{D}_{G_a}[r^d_{G_a}(D_a), R']$. We claim that the pair $(D_a, (p_a, \mathsf{w}(D_a)))$ is a solution in $\mathcal{D}_{G_a}[r^d_{G_a}(D_a), R']$. Condition (1) is trivially satisfied. By assumption, $D \cup V^+(R')$ (σ, ρ) dominates $V(G)$ in the graph $\mathbf{CG}(G, R')$. Since there are no edges between the vertices in $V(G_a)$ and those in $V(G_b)$, we conclude that $D_a \cup V^+(R')$ (σ, ρ)-dominates $V(G_a)$ in $\mathbf{CG}(G_a, R')$. That is Condition (2) is satisfied. Observe that every connected component of $G[D]$ is either included in D_a or in D_b. Therefore, every connected component of $G_a[D_a]$ contains a vertex v with a label $j \in \mathsf{active}(R, R')$. Since $v \in D_a$, we conclude that $r^d_{j,G_a}(D_a) \neq 0$, thus $j \in \mathsf{active}(r^d_{G_a}(D_a), R')$. We can conclude that Condition (3) is satisfied. Thus, $(D_a, (p_a, \mathsf{w}(D_a)))$ is a solution in $\mathcal{D}_{G_a}[r^d_{G_a}(D_a), R']$. Symmetrically, we deduce that there exits $p_b \in \mathsf{active}(R^d_{G_b}(D_b), R')$ such that $(D_b, (p_b, \mathsf{w}(D_b)))$ is a solution in $\mathcal{D}_{G_b}[r^d_{G_b}(D_b), R']$.

It remains to prove that $p = p_{a\uparrow L} \sqcup p_{b\uparrow L}$ with $L = \mathsf{active}(R, R')$. First, observe that $\mathsf{active}(r^d_{G_a}(D_a), R') \cup \mathsf{active}(r^d_{G_b}(D_b, R')) = \mathsf{active}(R, R')$ and thus $p_{a\uparrow L} \sqcup p_{b\uparrow L}$ is a partition of $\mathsf{active}(R, R')$. Let $\sim_a$ (resp. $\sim_b$) be the equivalence relation such that $i \sim_a j$ (resp. $i \sim_b j$) if an i-vertex is connected to a j-vertex in the graph $\mathbf{CG}(G_a, R')[D_a \cup V^+(R')]$ (resp. $\mathbf{CG}(G_b, R')[D_b \cup V^+(R')]$). By Condition (4), two labels i, j are in the same block of p if and only if an i-vertex and a j-vertex are connected in $\mathbf{CG}(G, R')[D \cup V^+(R')]$. On the other hand, i and j are in the same block of $p_{a\uparrow L} \sqcup p_{b\uparrow L}$ if and only if $i \mathcal{R} j$ where $\mathcal{R}$ is the transitive closure of the relation ($i \sim_a j$ or $i \sim_b j$). By definition of $\mathbf{CG}(G, R')$, for every label $i \in \mathsf{active}(R, R')$, we have $V_i^+(R') \neq \varnothing$ and the vertices in $lab_G^{-1}(i) \cap D$ are all adjacent to the vertices in $V_i^+(R')$. One can easily deduce from these observations that $p = p_{a\uparrow L} \sqcup p_{b\uparrow L}$. Hence, $(p, w) \in \mathcal{A}$.

We now prove that every weighted partition in $\mathcal{A}$ belongs to $\mathcal{D}_G[R, R']$. Let (p, w) be a weighted partition added to $tab_G[R, R']$ from a solution $(D_a, (p_a, w_a))$ in $\mathcal{D}_{G_a}[A, R']$ and a solution $(D_b, (p_b, w_b))$ in $\mathcal{D}_{G_b}[B, R']$. We claim that $(D_a \cup D_b, (p_{a\uparrow L} \sqcup p_{b\uparrow L}, w_a + w_b))$ is a solution in $\mathcal{D}_G[R, R']$ with $L = \mathsf{active}(R, R')$. We deduce that Condition (1) is satisfied from the definition of R-compatibility and because $\mathsf{min}(d, |lab_G^{-1}(i) \cap D|) = \mathsf{min}(d, r^d_{G_a}(D_a) + r^d_{G_b}(D_b))$ for all $i \in [k]$. With the same arguments given previously, one easily deduces that Conditions (2)-(4) are also satisfied. We conclude that $(D_a \cup D_b, (p_{a\uparrow L} \sqcup p_{b\uparrow L}, w_a + w_b))$ is a solution in $\mathcal{D}_G[R, R']$. □

Theorem 4.27. *There is an algorithm that, given an n-vertex graph G and an irredundant k-expression of G, computes a maximum (or a minimum) connected (σ, ρ)-dominating set in time $(d+1)^{3k} \cdot 2^{(\omega+1)\cdot k} \cdot k^{O(1)} \cdot n$ with $d := \mathsf{max}(d(\sigma), d(\rho))$.*

Proof. We do a bottom-up traversal of the k-expression and at each step we update the tables as indicated above. The correctness of the algorithm follows from Lemmas 4.24-4.26. From the definition of $\mathcal{D}_G[R, R']$, we deduce that the weight of an optimum connected (σ, ρ)-dominating set corresponds to the optimum over all $R \in \{0, \ldots, d\}^k$ of $\mathbf{opt}\{w : (\varnothing, w) \in tab_G[R, \{0\}^k]\}$ because $tab_G[R, \{0\}^k]$ represents $\mathcal{D}_G[R, \{0\}^k]$.

Let us discuss the time complexity now. We claim that the tables of tab_G can be computed in time $(d+1)^{3k} \cdot 2^{(\omega+1)\cdot k} \cdot k^{O(1)}$. We distinguish the following cases:

- If $G = \mathbf{1}(x)$, then it is easy to see that tab_G is computable in time $O(d)$.
- If $G = add_{i,j}(H)$, then we update $tab_G[R, R']$ from one entry $tab_H[R, S']$ for some fixed S' computable in constant time. The used join operation runs in time $2^{k-1} \cdot k^{O(1)}$ (from Proposition 4.15). Thus, tab_G is computable in time $(d+1)^{2k} \cdot 2^{k-1} \cdot k^{O(1)}$.
- Now, if $G = ren_{i\to j}(H)$, then we update $tab_G[R, R']$ from at most $|\mathcal{S}| = (d+1)^2$ tables from tab_H, each identified in constant time from (R, R'). Since each table of tab_H contains at most 2^{k-1} entries, computing the call at the function reduce take $(d+1)^2 \cdot 2^{\omega\cdot k} \cdot k^{O(1)}$. Thus, we can compute tab_G in time $(d+1)^{2k+2} \cdot 2^{\omega\cdot k} \cdot k^{O(1)}$.
- If $G = G_a \oplus G_b$, then the bottleneck is when $R' \neq \{0\}^k$. Indeed, if $R' = \{0\}^k$, then $tab_G[R, R']$ can be computed in time $O(2^{\omega\cdot k})$ since $tab_G[R, R']$ is computed from two tables, each containing at most 2^{k-1} entries. Let $R' \neq \{0\}^k$. By Theorem 4.18, we can compute the set of tables $tab_G[R, R']$ with $R \in \{0, 1, \ldots, d\}^k\}$ in time

$$\sum_{R\in\{0,\ldots,d\}^k} \left(\sum_{\substack{(A,B) \text{ is} \\ R\text{-compatible}}} |\mathsf{join}(tab_{G_a}[A, R'], tab_{G_b}[B, R'])| \cdot 2^{(\omega-1)\cdot k} \cdot k^{O(1)} \right).$$

  Observe that for all $A, B \in \{0, \ldots, d\}^k$:

  1. There is only one $R \in \{0, \ldots, d\}^k$ such that (A, B) is R-compatible. This follows from the definition of R-compatibility. Hence, there are at most $(d+1)^{2k}$ tuples (A, B, R) such that (A, B) is R-compatible.
  2. From Proposition 4.15, the size of $|\mathsf{join}(tab_{G_a}[A, R'], tab_{G_b}[B, R'])|$ is bounded by $2^{2(k-1)}$ and this set can be computed in time $2^{2(k-1)} \cdot k^{O(1)}$.

  Since $2^{2(k-1)} \cdot 2^{(\omega-1)\cdot k} \leq 2^{(\omega+1)\cdot k}$, we conclude from the observations (1)-(2) above that we can compute the tables $tab_G[R, R']$, for every $\{0\}^k \neq R \in \{0, 1, \ldots, d\}^k$, in time $(d+1)^{2k} \cdot 2^{(\omega+1)\cdot k} \cdot k^{O(1)}$.

  Hence, we can update tab_G in time $(d+1)^{3k} \cdot 2^{(\omega+1)\cdot k} \cdot k^{O(1)}$.

Therefore, in the worst case, the tables of tab_G takes $(d+1)^{3k} \cdot 2^{(\omega+1)\cdot k} \cdot k^{O(1)}$ time to be computed. Because the size of a k-expression is $O(n \cdot k^2)$, we can conclude that a maximum (or minimum) weighted (σ, ρ)-dominating set can be computed in the given time. □

As a consequence of Theorem 4.27, we have the following corollary.

Corollary 4.28. *There is an algorithm that, given an n-vertex graph G, a subset $K \subseteq V(G)$ and an irredundant k-expression of G, computes a minimum node-weighted Steiner tree for (G, K) in time $2^{(\omega+4)\cdot k} \cdot k^{O(1)} \cdot n$.*

Proof. We can assume w.l.o.g. that $|K| \geq 2$. We can reduce the problem Node-weighted Steiner Tree to a variant of a (σ, ρ)-Dominating Set problem where $\sigma = \mathbb{N}^+$ and $\rho = \mathbb{N}$. This variant requires K to be included in the (σ, ρ)-dominating set. We can add this constraint, by modifying how we compute the table tab_G, when $G = \mathbf{1}(x)$ and $x \in K$. For $(r_1), (r_1') \in \{0, 1\}$, we let

$$tab_G[R, R'] := \begin{cases} \{(\{\{1\}\}, \mathsf{w}(x))\} & \text{if } r_1 = 1 \text{ and } r_1' = 1, \\ \varnothing & \text{otherwise.} \end{cases}$$

It is straightforward to check that this modification implements this constraint and our algorithm with this modification computes a minimum node-weighted Steiner tree. The running time follows from the running time of Theorem 4.27 with $d = 1$. □

More modifications are needed in order to compute a maximum (or minimum) weighted connected co-(σ, ρ)-dominating set.

Corollary 4.29. *There is an algorithm that, given an n-vertex graph G and an irredundant k-expression of G, computes a maximum (or minimum) weighted co-(σ, ρ)-dominating set in time $(d+2)^{3k} \cdot 2^{(\omega+1)\cdot k} \cdot k^{O(1)} \cdot n$.*

Proof. First, we need to modify the definition of the tables $\mathcal{D}_G$. Let H be a k-labeled graph, $R, R' \in \{0, \ldots, d\}^k$, and $\overline{R}, \overline{R}' \in \{0, 1\}^k$. The entries of $\mathcal{D}_H[R, R', \overline{R}, \overline{R}']$ are all the weighted partitions $(p, w) \in \Pi(\mathsf{active}(\overline{R}, \overline{R}')) \times \mathbb{N}$ such that there exists a set $X \subseteq V(G)$ so that $w = \mathsf{w}(X)$ and

1. $r_H^d(V(H) \setminus X) = R$ and $r_H^1(X) = \overline{R}$,
2. $(V(H) \setminus X) \cup V^+(R')$ (σ, ρ)-dominates $V(H)$ in $\mathbf{CG}(H, R')$,
3. if $\mathsf{active}(\overline{R}, \overline{R}') = \varnothing$, then $H[X]$ is connected, otherwise every connected component of $H[X]$ intersects $lab_H^{-1}(\mathsf{active}(\overline{R}, \overline{R}'))$,
4. $p = \mathsf{active}(\overline{R}, \overline{R}')/\sim$ where $i \sim j$ if and only if an i-vertex is connected to a j-vertex in $\mathbf{CG}(H, \overline{R}')[X \cup V^+(\overline{R}')]$.

As a solution is a set X such that $V(G) \setminus X$ is a (σ, ρ)-dominating set and $G[X]$ is a connected graph, we need information about $X \cap V(H)$ and $(V(H) \setminus X)$. Intuitively, R, R' are the information we need to guarantee the (σ, ρ)-domination, and $\overline{R}, \overline{R}'$ are the information we need to guarantee the connectedness. In particular, $\overline{R}$ specifies which label classes are intersected and $\overline{R}'$ tells which label classes are expected to have at least one additional neighbor in the future.

These modifications imply in particular to change the notion of R-compatibility. For each $t \in \{a, b, c\}$, let $R_t = (r_1^t, \ldots, r_k^t) \in \{0, \ldots, d\}^k$, and $\overline{R_t} = (\overline{r}_1^t, \ldots, \overline{r}_k^t) \in \{0, 1\}^k$, we say that $(R_a, \overline{R_a}, R_b, \overline{R_b})$ is $(R_c, \overline{R_c})$-compatible if for all $i \in [k]$, we have $r_i^c = \mathsf{min}(d, r_i^a + r_i^b)$ and $\overline{r}_i^c = \mathsf{min}(1, \overline{r}_i^a + \overline{r}_i^b)$.

It is now an exercise to modify the algorithm of Theorem 4.27 in order to update the tables tab_H through the different clique-width operations. The weight of an optimum solution corresponds to the optimum over all $R \in \{0,\dots,d\}^k, \overline{R} \in \{0,1\}^k$ of $\mathbf{opt}\{w : (\varnothing, w) \in tab_G[R,\{0\}^k,\overline{R},\{0\}^k]\}$, since $tab_G[R,\{0\}^k,\overline{R},\{0\}^k]$ represents $\mathcal{D}_G[R,\{0\}^k,\overline{R},\{0\}^k]$.

Let us discuss the time complexity now. Let H be a k-labeled graph that is used in the k-expression of G. First, observe that we do not need to compute $tab_H[R,R',\overline{R},\overline{R}']$ for all $R,R' \in \{0,\dots,d\}^k$, and $\overline{R},\overline{R}' \in \{0,1\}^k$. Indeed, for all $X \subseteq V(H)$ and $i \in [k]$, if we have $r^d_{i,H}(V(H)\setminus X) < d$, then

$$r^1_{i,H}(X) = \begin{cases} 0 & \text{if } |lab_H^{-1}(i)| = r^d_{i,H}(V(H)\setminus X), \\ 1 & \text{otherwise.} \end{cases}$$

Hence, we deduce that there are at most $(d+2)^k$ pairs $(R,\overline{R}) \in \{0,\dots,d\}^k \times \{0,1\}^k$ such that $R = r^d_H(V(H)\setminus X)$ and $\overline{R} = r^1_H(X)$ for some $X \subseteq V(H)$. Indeed, whenever $r^d_{i,H}(V(H)\setminus X) < d$, there is only one possible value for $r^1_{i,H}(X)$, and when $r^d_{i,H}(V(H)\setminus X) = d$, there are at most 2 possible values for $r^1_{i,H}(X)$. With the same arguments used to prove the running time of Theorem 4.27, one easily deduces that there are at most $(d+2)^{2k}$ tuples $(R_a,\overline{R_a},R_b,\overline{R_b},R_c,\overline{R_c})$ such that $(R_a,\overline{R_a},R_b,\overline{R_b})$ is $(R_c,\overline{R_c})$-compatible.

Moreover, it is sufficient to consider $(d+2)^k$ pairs $(R',\overline{R}') \in \{0,\dots,d\}^k \times \{0,1\}^k$ when we update tab_H. For every $i \in [k]$, we let $c^i_H := |N_G(lab_H^{-1}(i)) \setminus N_H(lab_H^{-1}(i))|$. Notice that for every vertex $v \in lab_H^{-1}(i)$, we have $|N_G(v)| = |N_H(v)| + c^i_H$. Informally, we cannot expect more than c^i_H neighbors in the future for the i-vertices of H. Hence, it is enough to consider the pairs $(R',\overline{R}') \in \{0,\dots,d\}^k \times \{0,1\}^k$, with $R' = (r'_1,\dots,r'_k)$ and $\overline{R}' = (\overline{r}'_1,\dots,\overline{r}'_k)$, such that for all $i \in [k]$, if $r'_i < d$, then

$$\overline{r}'_i = \begin{cases} 0 & \text{if } r'_i \geq c^H_i, \\ 1 & \text{otherwise.} \end{cases}$$

That is, for every $i \in [k]$, if $r'_i < d$, then there is one possible value for $\overline{r}'_i$ because if we expect $r'_i < d$ neighbors for the i-vertices in $V(H)\setminus X$, then we must expect $\mathsf{min}(0, c^i_H - r'_i)$ neighbors for the i-vertices in X. If $r'_i = d$, then there are no restrictions on the value of $\overline{r}'_i$. Thus, the pairs $(r'_i,\overline{r}'_i)$ can take up to $(d+2)$ values. We conclude that there are at most $(d+2)^k$ pairs $(R',\overline{R}') \in \{0,\dots,d\}^k \times \{0,1\}^k$ worth to looking at. With these observations and the arguments used in the running time proof of Theorem 4.27, we conclude that we can compute a maximum (or minimum) weighted co-(σ,ρ)-dominating set in the given time. □

4.2 The d-neighbor equivalence versus acyclicity and connectivity constraints

In this section, we design a framework based on the 1-neighbor equivalence (see Definition 1.1) and using some ideas of the rank-based approach of [9] to design efficient algorithms for many problems involving a connectivity constraint. This framework provides tools to reduce the size of the set of partial solutions we compute at each step of a dynamic programming algorithm. We prove that many ad-hoc algorithms for these problems can be unified into a single algorithm that is almost the same as the one from [18] computing a dominating set.

In Subsection 4.2.3, we use our framework to design an algorithm that, given a rooted layout $\mathcal{L}$, solves any Connected (σ,ρ)-Dominating Set problem. This includes some well-known problems such as Connected Dominating Set, Connected Vertex Cover or

Node Weighted Steiner Tree. The running time of our algorithm is polynomial in n and $\mathsf{s\text{-}nec}_d(\mathcal{L})$, with d a constant that depends on σ and ρ. Consequently, each Connected (σ, ρ)-Dominating Set problem admits algorithms with the running times given in Table 4.2.

Table 4.2 – Running times of our algorithms for the different parameters, where n is the number of vertices of the given graph.

Clique-width	Rank-width	$\mathbb{Q}$-rank-width	Mim-width
$2^{O(k)} \cdot n^{O(1)}$	$2^{O(k^2)} \cdot n^{O(1)}$	$2^{O(k \log(k))} \cdot n^{O(1)}$	$n^{O(k)}$

In Subsection 4.2.4, we introduce some new concepts to deal with acyclicity. We use these concepts to deal with the AC-(σ, ρ)-Dominating Set problems. Both Maximum Induced Tree and Longest Induced Path are the AC variants of (σ, ρ)-Dominating Set problems. We prove that there exist algorithms that solve these AC variants in the running times given in Table 4.2. To obtain these results, we rely heavily on the d-neighbor equivalence. However, we were not able to provide an algorithm whose running time is polynomial in n and $\mathsf{s\text{-}nec}_d(\mathcal{L})$ for some constant d. Instead, we provide an algorithm whose behavior depends slightly on each width measure considered in Table 4.2.

We moreover prove that we can modify slightly this algorithm to solve any Acyclic (σ, ρ)-Dominating Set problem. In particular, this shows that we can use the algorithm for Maximum Induced Tree to solve the Feedback Vertex Set problem.

Our approach

Let us explain our approach with the connected and AC variants of the Dominating Set problem. To solve these problems, our algorithms do a bottom-up traversal of a given layout $\mathcal{L}$ of the input graph G and at each step we compute a set of partial solutions. In our case, the steps of our algorithms are associated with the vertex bipartitions $(A, \overline{A})$ induced by the edges of a layout and the partial solutions are subsets of A. At each step, our algorithms compute, for each pair (R, R') where R (resp. R') is a 1-neighbor equivalence class of A (resp. $\overline{A}$), a set of partial solutions $\mathcal{A}_{R,R'} \subseteq R$. The way we compute these sets guarantees that the partial solutions in $\mathcal{A}_{R,R'}$ will be completed with sets in R'. Consequently, we have information about how we will complete our partial solutions since every $Y \in R'$ has the same neighborhood in A.

To deal with the local constraint of these problems, namely the domination constraint, we use the ideas of Bui-Xuan et al. [18]. For each pair (R, R'), let us say that $X \subseteq A$ is *coherent* with (R, R') if: (1) $X \in R$ and (2) $X \cup Y$ dominates A in the graph G for every $Y \in R'$. To compute a minimum dominating set, Bui-Xuan et al. proved that it is enough to keep, for each pair (R, R'), a partial solution X of minimum weight that is coherent with (R, R'). Intuitively, if a partial solution X that is coherent with (R, R') could be completed into a dominating set of G, then it is the case for every partial solution coherent with (R, R'). This is due to the fact that any pair of sets in R (resp. R') dominate the same vertices in $\overline{A}$ (resp. A).

To solve the connectivity variant, we compute, for each (R, R'), a set $\mathcal{A}_{R,R'}$ of partial solutions coherent with (R, R'). Informally, $\mathcal{A}_{R,R'}$ has to be as small as possible, but if a partial solution coherent with (R, R') leads to a minimum connected dominating set, then $\mathcal{A}_{R,R'}$ must contain such a partial solution. To deal with this intuition, we introduce the relation of R'-*representativity* between sets of partial solutions. We say that $\mathcal{A}^\star$ R'-*represents* a set $\mathcal{A}$, if, for all sets $Y \in R'$, we have $\mathsf{best}(\mathcal{A}, Y) = \mathsf{best}(\mathcal{A}^\star, Y)$ where $\mathsf{best}(\mathcal{B}, Y)$ is the minimum weight of a set $X \in \mathcal{B}$ such

that $G[X \cup Y]$ is connected. The main tool of our framework is a function reduce that, given a set of partial solutions $\mathcal{A}$ and a 1-neighbor equivalence class R' of $\overline{A}$, outputs a subset of $\mathcal{A}$ that R'-represents $\mathcal{A}$ and whose size is upper bounded by $\mathsf{s\text{-}nec}_1(\mathcal{L})^2$. To design this function, we use ideas from the rank-based approach of [9]. That is, we define a small matrix $\mathcal{C}$ with $|\mathcal{A}|$ rows and $\mathsf{s\text{-}nec}_1(\mathcal{L})^2$ columns. Then, we show that a basis of maximum weight of the row space of $\mathcal{C}$ corresponds to an R'-representative set of $\mathcal{A}$. Since $\mathcal{A}$ has $\mathsf{s\text{-}nec}_1(\mathcal{L})^2$ columns, the size of a basis of $\mathcal{A}$ is smaller than $\mathsf{s\text{-}nec}_1(\mathcal{L})^2$. By calling reduce after each computational step, we keep the sizes of the sets of partial solutions polynomial in $\mathsf{s\text{-}nec}_1(\mathcal{L})$. Besides, the definition of R'-representativity guarantees that the set of partial solutions computed for the root of $\mathcal{L}$ contains a minimum connected dominating set.

For the AC variant of dominating set, we need more information in order to deal with the acyclicity. We obtain this extra information by considering that R (resp. R') is a 2-neighbor equivalence class over A (resp. $\overline{A}$). This way, for all sets $X \subseteq A$, the vertices in X that have at least 2 neighbors in R', have at least 2 neighbors in Y, for all $Y \in R'$. These vertices play a major role in the acyclicity constraint because they may create cycles when X is joined with a partial solution Y in $\overline{A}$; and thus they are important in our algorithm. We need also a new notion of representativity. We say that $\mathcal{A}^\star$ R'-*ac-represents* a set $\mathcal{A}$, if, for all sets $Y \in R'$, we have $\mathsf{best}^{\mathsf{acy}}(\mathcal{A}, Y) = \mathsf{best}^{\mathsf{acy}}(\mathcal{A}^\star, Y)$ where $\mathsf{best}^{\mathsf{acy}}(\mathcal{B}, Y)$ is the minimum weight of a set $X \in \mathcal{B}$ such that $G[X \cup Y]$ is a tree. As for the R'-representativity, we provide a function that, given a set of partial solutions $\mathcal{A}$ and a 2-neighbor equivalence class R' of $\overline{A}$, outputs a small subset $\mathcal{A}^\star$ of $\mathcal{A}$ that R'-ac-represents $\mathcal{A}$. Unfortunately, we were not able to upper bound the size of $\mathcal{A}^\star$ by a polynomial in n and $\mathsf{s\text{-}nec}_d(\mathcal{L})$ (for some constant d). Instead, we prove that, for clique-width, rank-width, $\mathbb{Q}$-rank-width, and mim-width, the size of $\mathcal{A}^\star$ can be upper bounded by, respectively, $2^{O(k)} \cdot n$, $2^{O(k^2)} \cdot n$, $2^{O(k \log(k))} \cdot n$, and $n^{O(k)}$. The key to compute $\mathcal{A}^\star$ is to decompose $\mathcal{A}$ into a small number of sets $\mathcal{A}_1 \ldots, \mathcal{A}_\ell$, said R'-*consistent*, where the notion of R'-ac-representativity matches the notion of R'-representativity. More precisely, any R'-representative set of an R'-consistent set $\mathcal{A}$ is also an R'-ac-representative set of $\mathcal{A}$. To compute an R'-ac-representative set of $\mathcal{A}$ it is then enough to compute an R'-representative set for each R'-consistent set in the decomposition of $\mathcal{A}$. The union of these R'-representative sets is an R'-ac-representative set of $\mathcal{A}$. Besides the notion of representativity, the algorithm for the AC variant of Dominating Set is very similar to the one for finding a minimum connected dominating set.

As explained for clique-width in Section 4.1, we can not use the same trick as in [9] to ensure the acyclicity, that is counting the number of edges induced by the partial solutions. Indeed, we would need to differentiate at least n^k partial solutions (for any parameter k considered in Table 4.2) in order to update this information. We give more explanation on this statement at the beginning of Subsection 4.2.4.

4.2.1 Preliminaries

A *consistent cut* of X is an ordered bipartition (X_1, X_2) of X such that $N(X_1) \cap X_2 = \varnothing$. We denote by $\mathsf{cuts}(X)$ the set of all consistent cuts of X. In our proofs, we use the following facts.

Fact 4.30. *Let $X \subseteq V(G)$. For every $C \in \mathsf{cc}(G[X])$ and every $(X_1, X_2) \in \mathsf{cuts}(X)$, we have either $C \subseteq X_1$ or $C \subseteq X_2$.*

We deduce from the above fact that $|\mathsf{cuts}(X)| = 2^{|\mathsf{cc}(G[X])|}$.

Fact 4.31. *Let X and Y be two disjoint subsets of $V(G)$. We have $(W_1, W_2) \in \mathsf{cuts}(X \cup Y)$ if and only if the following conditions are satisfied*

1. $(W_1 \cap X, W_2 \cap X) \in \mathsf{cuts}(X)$,

2. $(W_1 \cap Y, W_2 \cap Y) \in \mathsf{cuts}(Y)$, *and*

3. $N(W_1 \cap X) \cap (W_2 \cap Y) = \varnothing$ *and* $N(W_2 \cap X) \cap (W_1 \cap Y) = \varnothing$.

We refer to Section 1.3 for a definition of the d-neighbor equivalence relation. The following fact follows directly from the definition of the d-neighbor equivalence relation. We use it several times in our proofs.

Fact 4.32. *Let $A, B \subseteq V(G)$ such that $A \subseteq B$, and let $d \in \mathbb{N}^+$. For all $X, Y \subseteq A$, if $X \equiv_A^d Y$, then $X \equiv_B^d Y$.*

In order to manipulate the equivalence classes of $\equiv_A^d$, one needs to compute a representative for each equivalence class in polynomial time. This is achieved with the following notion of a representative. Let G be a graph with an arbitrary ordering of $V(G)$ and let $A \subseteq V(G)$. For each $X \subseteq A$, let us denote by $\mathsf{rep}_A^d(X)$ the lexicographically smallest set $R \subseteq A$ such that $|R|$ is minimized and $R \equiv_A^d X$. Moreover, we denote by $\mathcal{R}_A^d$ the set $\{\mathsf{rep}_A^d(X) : X \subseteq A\}$. It is worth noticing that the empty set always belongs to $\mathcal{R}_A^d$, for all $A \subseteq V(G)$ and $d \in \mathbb{N}^+$. Moreover, we have $\mathcal{R}_{V(G)}^d = \mathcal{R}_\varnothing^d = \{\varnothing\}$ for all $d \in \mathbb{N}^+$. In order to compute $\mathcal{R}_A^d$, we use the following lemma.

Lemma 4.33 ([18])**.** *For every $A \subseteq V(G)$ and $d \in \mathbb{N}^+$, one can compute in time $O(\mathsf{nec}_d(A) \cdot \log(\mathsf{nec}_d(A)) \cdot |V(G)|^2)$, the sets $\mathcal{R}_A^d$ and a data structure, that given a set $X \subseteq A$, computes $\mathsf{rep}_A^d(X)$ in time $O(\log(\mathsf{nec}_d(A)) \cdot |A| \cdot |V(G)|)$.*

In the following, we fix G an n-vertex graph, (T, δ) a rooted layout of G, and $\mathsf{w} : V(G) \to \mathbb{Q}$ a weight function over the vertices of G. We also assume that $V(G)$ is ordered.

4.2.2 Representative sets

In this section, we define a notion of representativity between sets of partial solutions w.r.t. the connectivity. Our notion of representativity is defined w.r.t. some node x of T and the 1-neighbor equivalence class of some set $R' \subseteq \overline{V_x}$. In our algorithms, R' will always belong to $\mathcal{R}_{\overline{V_x}}^d$ for some $d \in \mathbb{N}^+$. Our algorithms compute a set of partial solutions for each $R' \in \mathcal{R}_{\overline{V_x}}^d$. The partial solutions computed for R' will be completed with sets equivalent to R' w.r.t. $\equiv_{\overline{V_x}}^d$. Intuitively, the R''s represent some expectation about how we will complete our sets of partial solutions. For the connectivity and the domination, $d = 1$ is enough but if we need more information for some reasons (for example the (σ, ρ)-domination or the acyclicity), we may take $d > 1$. This is not a problem as the d-neighbor equivalence class of R' is included in the 1-neighbor equivalence class of R'. Hence, in this section, we fix a node x of T and a set $R' \subseteq \overline{V_x}$ to avoid to overload the statements by the sentence "let x be a node of T and $R' \subseteq \overline{V_x}$". We let $\mathbf{opt} \in \{\mathsf{min}, \mathsf{max}\}$; if we want to solve a maximization (or minimization) problem, we use $\mathbf{opt} = \mathsf{max}$ (or $\mathbf{opt} = \mathsf{min}$). We use it also, as here, in the next sections.

Definition 4.34 ((x, R')-representativity)**.** *For every $\mathcal{A} \subseteq 2^{V(G)}$ and $Y \subseteq V(G)$, we define*

$$\mathsf{best}(\mathcal{A}, Y) := \mathbf{opt}\{\mathsf{w}(X) : X \in \mathcal{A} \text{ and } G[X \cup Y] \text{ is connected }\}.$$

Let $\mathcal{A}, \mathcal{B} \subseteq 2^{V_x}$. We say that $\mathcal{B}$ (x, R')-represents $\mathcal{A}$ if, for every $Y \subseteq \overline{V_x}$ such that $Y \equiv_{\overline{V_x}}^1 R'$, we have $\mathsf{best}(\mathcal{A}, Y) = \mathsf{best}(\mathcal{B}, Y)$.

Notice that the (x, R')-representativity is an equivalence relation. The set $\mathcal{A}$ is meant to represent a set of partial solutions of $G[V_x]$ which have been computed. We expect to complete these partial solutions with partial solutions of $G[\overline{V_x}]$ which are equivalent to R' w.r.t. $\equiv_{\overline{V_x}}^1$. If

$\mathcal{B}$ (x, R')-represents $\mathcal{A}$, then we can safely substitute $\mathcal{B}$ to $\mathcal{A}$ because the quality of the output of the dynamic programming algorithm will remain the same. Indeed, for every subset Y of $\overline{V_x}$ such that $Y \equiv^1_{V_x} R'$, the optimum solutions obtained by the union of a partial solution in $\mathcal{A}$ and Y will have the same weight as the optimum solution obtained from the union of a set in $\mathcal{B}$ and Y.

The following theorem presents the main tool of our framework: a function reduce that, given a set of partial solutions $\mathcal{A}$, outputs a subset of $\mathcal{A}$ that (x, R')-represents $\mathcal{A}$ and whose size is upper bounded by $\mathsf{s\text{-}nec}_1(\mathcal{L})^2$. To design this function, we use ideas from the rank-based approach of [9]. That is, we define a small matrix $\mathcal{C}$ with $|\mathcal{A}|$ rows and $\mathsf{s\text{-}nec}_1(V_x)^2$ columns. Then, we show that a basis of maximum weight of the row space of $\mathcal{C}$ corresponds to an (x, R')-representative set of $\mathcal{A}$. Since $\mathcal{A}$ has $\mathsf{s\text{-}nec}_1(\mathcal{L})^2$ columns, the size of a basis of $\mathcal{A}$ is smaller than $\mathsf{s\text{-}nec}_1(\mathcal{L})^2$.

In order to compute a small (x, R')-representative set of a set $\mathcal{A} \subseteq 2^{V_x}$, the following theorem requires that the sets in $\mathcal{A}$ are pairwise equivalent w.r.t. $\equiv^1_{V_x}$. This is useful since in our algorithm we classify our sets of partial solutions with respect to this property. We need this to guarantee that the partial solutions computed for R' will be completed with sets equivalent to R' w.r.t. $\equiv^d_{V_x}$. However, if one wants to compute a small (x, R')-representative set of a set $\mathcal{A}$ that does not respect this property, then it is enough to compute an (x, R')-representative set for each 1-neighbor equivalence class of $\mathcal{A}$. The union of these (x, R')-representative sets is an (x, R')-representative set of $\mathcal{A}$.

Theorem 4.35. *Let $R \in \mathcal{R}^1_{V_x}$. Then, there exists an algorithm* reduce *that, given $\mathcal{A} \subseteq 2^{V_x}$ such that $X \equiv^1_{V_x} R$ for all $X \in \mathcal{A}$, outputs in time $O(|\mathcal{A}| \cdot \mathsf{nec}_1(V_x)^{2(\omega-1)} \cdot n^2)$ a subset $\mathcal{B} \subseteq \mathcal{A}$ such that $\mathcal{B}$ (x, R')-represents $\mathcal{A}$ and $|\mathcal{B}| \leq \mathsf{nec}_1(V_x)^2$.*

Proof. We assume w.l.o.g. that $\mathbf{opt} = \mathsf{max}$, the proof is symmetric for $\mathbf{opt} = \mathsf{min}$. First, we suppose that $R' \equiv^1_{V_x} \varnothing$. Observe that, for every $Y \equiv^1_{V_x} \varnothing$, we have $N(Y) \cap V_x = N(\varnothing) \cap V_x = \varnothing$. It follows that, for every $Y \subseteq \overline{V_x}$ such that $Y \equiv^1_{V_x} \varnothing$ and $Y \neq \varnothing$, we have $\mathsf{best}(\mathcal{A}, Y) = -\infty$. Moreover, by definition of best, we have $\mathsf{best}(\mathcal{A}, \varnothing) = \mathsf{max}\{\mathsf{w}(X) : X \in \mathcal{A} \text{ and } G[X] \text{ is connected}\}$. Hence, if $R' \equiv^1_{V_x} \varnothing$, then it is sufficient to return $\mathcal{B} = \{X\}$, where X is an element of $\mathcal{A}$ of maximum weight that induces a connected graph.

Assume from now that R' is not equivalent to $\varnothing$ w.r.t. $\equiv^1_{V_x}$. Let $X \in \mathcal{A}$. If there exists $C \in \mathsf{cc}(G[X])$ such that $N(C) \cap R' = \varnothing$, then, for all $Y \equiv^1_{V_x} R'$, we have $N(C) \cap Y = \varnothing$. Moreover, as R' is not equivalent to $\varnothing$ w.r.t. $\equiv^1_{V_x}$, we have $Y \neq \varnothing$. Consequently, for every $Y \equiv^1_{V_x} R'$, the graph $G[X \cup Y]$ is not connected. We can conclude that $\mathcal{A} \setminus \{X\}$ (x, R') represents $\mathcal{A}$. Thus, we can safely remove from $\mathcal{A}$ all such sets, and this can be done in time $|\mathcal{A}| \cdot n^2$. From now on, we may assume that, for all $X \in \mathcal{A}$ and for all $C \in \mathsf{cc}(G[X])$, we have $N(C) \cap R' \neq \varnothing$. It is worth noticing that if $R = \varnothing$ or more generally $N(R) \cap R' = \varnothing$, then by assumption, $\mathcal{A} = \varnothing$.

Indeed, if $N(R) \cap R' = \varnothing$, then, for every $X \in \mathcal{A}$, we have $N(X) \cap R' = N(R) \cap R' = \varnothing$ and in particular, for every $C \in \mathsf{cc}(G[X])$, we have $N(C) \cap R' = \varnothing$ (and we have assumed that no such set exists in $\mathcal{A}$).

Symmetrically, if, for some $Y \subseteq \overline{V_x}$ there exists $C \in \mathsf{cc}(G[Y])$ such that $N(C) \cap R = \varnothing$, then, for every $X \in \mathcal{A}$, the graph $G[X \cup Y]$ is not connected. Let $\mathcal{D}$ be the set of all subsets Y of $\overline{V_x}$ such that $Y \equiv^1_{V_x} R'$ and, for all $C \in \mathsf{cc}(G[Y])$, we have $N(C) \cap R \neq \varnothing$. Notice that the sets in $2^{\overline{V_x}} \setminus \mathcal{D}$ do not matter for the (x, R')-representativity.

For every $Y \in \mathcal{D}$, we let v_Y be one fixed vertex of Y. In the following, we denote by $\mathfrak{R}$ the set $\{(R'_1, R'_2) \in \mathcal{R}^1_{V_x} \times \mathcal{R}^1_{V_x}\}$. Let $\mathcal{M}$, $\mathcal{C}$, and $\overline{\mathcal{C}}$ be, respectively, an $(\mathcal{A}, \mathcal{D})$-matrix, an $(\mathcal{A}, \mathfrak{R})$-matrix,

and an $(\mathfrak{R}, \mathcal{D})$-matrix such that

$$\mathcal{M}[X,Y] := \begin{cases} 1 & \text{if } G[X \cup Y] \text{ is connected,} \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathcal{C}[X,(R'_1,R'_2)] := \begin{cases} 1 & \text{if } \exists (X_1,X_2) \in \mathsf{cuts}(X) \text{ such that } N(X_1) \cap R'_2 = \varnothing \text{ and } N(X_2) \cap R'_1 = \varnothing, \\ 0 & \text{otherwise.} \end{cases}$$

$$\overline{\mathcal{C}}[(R'_1,R'_2),Y] := \begin{cases} 1 & \text{if } \exists (Y_1,Y_2) \in \mathsf{cuts}(Y) \text{ such that } v_Y \in Y_1,\ Y_1 \equiv^1_{V_x} R'_1, \text{ and } Y_2 \equiv^1_{V_x} R'_2, \\ 0 & \text{otherwise.} \end{cases}$$

Intuitively, $\mathcal{M}$ contains all the information we need. Indeed, it is easy to see that a basis of maximum weight of the row space of $\mathcal{M}$ in $GF(2)$ is an (x,R')-representative set of $\mathcal{A}$. But, $\mathcal{M}$ is too big to be computable efficiently. Instead, we prove that a basis of maximum weight of the row space of $\mathcal{C}$ is an (x,R')-representative set of $\mathcal{A}$. This follows from the fact that $(\mathcal{C} \cdot \overline{\mathcal{C}})[X,Y]$ equals the number of consistent cuts (W_1,W_2) in $\mathsf{cuts}(X \cup Y)$ such that $v_Y \in W_1$. That is $(\mathcal{C} \cdot \overline{\mathcal{C}})[X,Y] = 2^{|\mathsf{cc}(G[X \cup Y])|-1}$. Consequently, $\mathcal{M} =_2 \mathcal{C} \cdot \overline{\mathcal{C}}$, where $=_2$ denotes the equality in $GF(2)$, i.e., $(\mathcal{C} \cdot \overline{\mathcal{C}})[X,Y]$ is odd if and only if $G[X \cup Y]$ is connected. We deduce the running time of reduce and the size of $\mathsf{reduce}(\mathcal{A})$ from the size of $\mathcal{C}$ (i.e. $|\mathcal{A}| \cdot \mathsf{nec}_1(V_x)^2$) and the fact that $\mathcal{C}$ is easy to compute.

We start by proving that $\mathcal{M} =_2 \mathcal{C} \cdot \overline{\mathcal{C}}$. Let $X \in \mathcal{A}$ and $Y \in \mathcal{D}$. We want to prove the following equality

$$(\mathcal{C} \cdot \overline{\mathcal{C}})[X,Y] = \sum_{(R'_1,R'_2) \in \mathfrak{R}} \mathcal{C}[X,(R'_1,R'_2)] \cdot \overline{\mathcal{C}}[(R'_1,R'_2),Y] = 2^{|\mathsf{cc}(G[X \cup Y])|-1}.$$

We prove this equality with the following two claims.

Claim 4.35.1. *We have $\mathcal{C}[X,(R'_1,R'_2)] \cdot \overline{\mathcal{C}}[(R'_1,R'_2),Y] = 1$ if and only if there exists $(W_1,W_2) \in \mathsf{cuts}(X \cup Y)$ such that $v_Y \in W_1$, $W_1 \cap Y \equiv^1_{V_x} R'_1$ and $W_2 \cap Y \equiv^1_{V_x} R'_2$.*

Proof. By definition, we have $\mathcal{C}[X,(R'_1,R'_2)] \cdot \overline{\mathcal{C}}[(R'_1,R'_2),Y] = 1$, if and only if

(a) $\exists (Y_1,Y_2) \in \mathsf{cuts}(Y)$ such that $v_Y \in Y_1$, $Y_1 \equiv^1_{V_x} R'_1$, $Y_2 \equiv^1_{V_x} R'_2$, and

(b) $\exists (X_1,X_2) \in \mathsf{cuts}(X)$ such that $N(X_1) \cap R'_2 = \varnothing$ and $N(X_2) \cap R'_1 = \varnothing$.

Let $(Y_1,Y_2) \in \mathsf{cuts}(Y)$ and $(X_1,X_2) \in \mathsf{cuts}(X)$ that satisfy, respectively, Properties (a) and (b). By definition of $\equiv^1_{V_x}$, we have $N(X_1) \cap Y_2 = \varnothing$ because $N(X_1) \cap R'_2 = \varnothing$ and $Y_2 \equiv^1_{V_x} R'_2$. Symmetrically, we have $N(X_2) \cap Y_1 = \varnothing$. By Fact 4.31, we deduce that $(X_1 \cup Y_1, X_2 \cup Y_2) \in \mathsf{cuts}(X \cup Y)$. This proves the claim. □

Claim 4.35.2. *Let (W_1,W_2) and $(W'_1,W'_2) \in \mathsf{cuts}(X \cup Y)$. We have $W_1 \cap Y \equiv^1_{V_x} W'_1 \cap Y$ and $W_2 \cap Y \equiv^1_{V_x} W'_2 \cap Y$ if and only if $W_1 = W'_1$ and $W_2 = W'_2$.*

Proof. We start by an observation about the connected components of $X \cup Y$. As $Y \in \mathcal{D}$, for all $C \in \mathsf{cc}(G[Y])$, we have $N(C) \cap R \neq \varnothing$. Moreover, by assumption, for all $C \in \mathsf{cc}(G[X])$, we have $N(C) \cap R' \neq \varnothing$. Since $X \equiv^1_{V_x} R$ and $Y \equiv^1_{V_x} R'$, every connected component of $G[X \cup Y]$ contains at least one vertex of X and one vertex of Y.

Suppose that $W_1 \cap Y \equiv^1_{V_x} W'_1 \cap Y$ and $W_2 \cap Y \equiv^1_{V_x} W'_2 \cap Y$. Assume towards a contradiction that $W_1 \neq W'_1$ and $W_2 \neq W'_2$. Since $W_1 \neq W'_1$, by Fact 4.30, we deduce that there exists $C \in \mathsf{cc}(G[X \cup Y])$ such that either (1) $C \subseteq W_1$ and $C \subseteq W'_2$ or (2) $C \subseteq W'_1$ and $C \subseteq W_2$. We can assume w.l.o.g. that there exits $C \in \mathsf{cc}(G[X \cup Y])$ such that $C \subseteq W_1$ and $C \subseteq W'_2$.

From the above observation, C contains at least one vertex of X and one of Y, and we have $N(C \cap X) \cap (W_1 \cap Y) \neq \varnothing$ and $N(C \cap X) \cap (W_2' \cap Y) \neq \varnothing$. But, since $W_2 \cap Y \equiv^1_{V_x} W_2' \cap Y$, we have $N(C \cap X) \cap (W_2 \cap Y) \neq \varnothing$. This implies in particular that $N(W_1) \cap W_2 \neq \varnothing$. It is a contradiction with the fact that $(W_1, W_2) \in \mathsf{cuts}(X \cup Y)$. □

Notice that Claim 4.35.2 implies that, for every $(R_1', R_2') \in \mathfrak{R}$, there exists at most one consistent cut $(W_1, W_2) \in \mathsf{cuts}(X \cup Y)$ such that $v_Y \in W_1$, $W_1 \cap Y \equiv^1_{V_x} R_1'$, and $W_2 \cap Y \equiv^1_{V_x} R_2'$. We can thus conclude from these two claims that

$$(\mathcal{C} \cdot \overline{\mathcal{C}})[X, Y] = |\{(W_1, W_2) \in \mathsf{cuts}(X \cup Y) \,:\, v_Y \in W_1\}|.$$

By Fact 4.30, we deduce that $(\mathcal{C} \cdot \overline{\mathcal{C}})[X, Y] = 2^{|\mathsf{cc}(G[X \cup Y])|-1}$ since every connected component of $G[X \cup Y]$ can be in both sides of a consistent cut at the exception of the connected component containing v_Y. Hence, $(\mathcal{C} \cdot \overline{\mathcal{C}})[X, Y]$ is odd if and only if $|\mathsf{cc}(G[X \cup Y])| = 1$. We conclude that $\mathcal{M} =_2 \mathcal{C} \cdot \overline{\mathcal{C}}$.

Let $\mathcal{B} \subseteq \mathcal{A}$ be a basis of maximum weight of the row space of $\mathcal{C}$ over $GF(2)$. We claim that $\mathcal{B}$ (x, R')-represents $\mathcal{A}$.

Let $Y \subseteq \overline{V_x}$ such that $Y \equiv^1_{V_x} R'$. Observe that, by definition of $\mathcal{D}$, if $Y \notin \mathcal{D}$, then $\mathsf{best}(\mathcal{A}, Y) = \mathsf{best}(\mathcal{B}, Y) = -\infty$. Thus it is sufficient to prove that, for every $Y \in \mathcal{D}$, we have $\mathsf{best}(\mathcal{A}, Y) = \mathsf{best}(\mathcal{B}, Y)$.

Let $X \in \mathcal{A}$ and $Y \in \mathcal{D}$. Recall that we have proved that $M[X, Y] =_2 (\mathcal{C} \cdot \overline{\mathcal{C}})[X, Y]$. Since $\mathcal{B}$ is a basis of $\mathcal{C}$, there exists $\mathcal{B}' \subseteq \mathcal{B}$ such that, for each $(R_1', R_2') \in \mathfrak{R}$, we have $\mathcal{C}[X, (R_1', R_2')] =_2 \sum_{W \in \mathcal{B}'} \mathcal{C}[W, (R_1', R_2')]$. Thus, we have the following equality

$$\begin{aligned}
\mathcal{M}[X, Y] &=_2 \sum_{(R_1', R_2') \in \mathfrak{R}} \mathcal{C}[X, (R_1', R_2')] \cdot \overline{\mathcal{C}}[(R_1', R_2'), Y] \\
&=_2 \sum_{(R_1', R_2') \in \mathfrak{R}} \left(\sum_{W \in \mathcal{B}'} \mathcal{C}[W, (R_1', R_2')] \right) \cdot \overline{\mathcal{C}}[(R_1', R_2'), Y] \\
&=_2 \sum_{W \in \mathcal{B}'} \left(\sum_{(R_1', R_2') \in \mathfrak{R}} \mathcal{C}[W, (R_1', R_2')] \cdot \overline{\mathcal{C}}[(R_1', R_2'), Y] \right) \\
&=_2 \sum_{W \in \mathcal{B}'} (\mathcal{C} \cdot \overline{\mathcal{C}})[W, Y] =_2 \sum_{W \in \mathcal{B}'} \mathcal{M}[W, Y].
\end{aligned}$$

If $\mathcal{M}[X, Y] = 1$ (i.e. $G[X \cup Y]$ is connected), then there is an odd number of sets W in $\mathcal{B}'$ such that $\mathcal{M}[W, Y] = 1$ (i.e. $G[W \cup Y]$ is connected). Hence, there exists at least one $W \in \mathcal{B}'$ such that $G[W \cup Y]$ is connected. Let $W \in \mathcal{B}'$ such that $\mathcal{M}[W, Y] = 1$ and $\mathsf{w}(W)$ is maximum. Assume towards a contradiction that $\mathsf{w}(W) < \mathsf{w}(X)$. Notice that $(\mathcal{B} \setminus \{W\}) \cup \{X\}$ is also a basis of $\mathcal{C}$ since the set of independent row sets of a matrix forms a matroid. Since $\mathsf{w}(W) < \mathsf{w}(X)$, the weight of the basis $(\mathcal{B} \setminus \{W\}) \cup \{X\}$ is strictly greater than the weight of the basis $\mathcal{B}$, yielding a contradiction. Thus $\mathsf{w}(X) \leq \mathsf{w}(W)$. Hence, for all $Y \in \mathcal{D}$ and all $X \in \mathcal{A}$, if $G[X \cup Y]$ is connected, then there exists $W \in \mathcal{B}$ such that $G[W \cup Y]$ is connected and $\mathsf{w}(X) \leq \mathsf{w}(W)$. This is sufficient to prove that $\mathcal{B}$ (x, R')-represents $\mathcal{A}$. Since $\mathcal{B}$ is a basis, the size of $\mathcal{B}$ is at most the number of columns of $\mathcal{C}$, thus, $|\mathcal{B}| \leq \mathsf{nec}_1(V_x)^2$.

It remains to prove the running time. We claim that $\mathcal{C}$ is easy to compute.

By Fact 4.30, $\mathcal{C}[X, (R_1', R_2')] = 1$ if and only if, for each $C \in \mathsf{cc}(G[X])$, we have either $N(C) \cap R_1' = \varnothing$ or $N(C) \cap R_2' = \varnothing$. Thus, each entry of $\mathcal{C}$ is computable in time $O(n^2)$. Since $\mathcal{C}$ has $|\mathcal{A}| \cdot |\mathcal{R}^1_{V_x}|^2 = |\mathcal{A}| \cdot \mathsf{nec}_1(V_x)^2$ entries, we can compute $\mathcal{C}$ in time $O(|\mathcal{A}| \cdot \mathsf{nec}_1(V_x)^2 \cdot n^2)$. Furthermore,

by Lemma 4.7, a basis of maximum weight of $\mathcal{C}$ can be computed in time $O(|\mathcal{A}| \cdot \mathsf{nec}_1(V_x)^{2(\omega-1)})$. We conclude that $\mathcal{B}$ can be computed in time $O(|\mathcal{A}| \cdot \mathsf{nec}_1(V_x)^{2(\omega-1)} \cdot n^2)$. □

Now to boost up a dynamic programming algorithm P on some rooted layout (T, δ) of G, we can use the function reduce to keep the size of the sets of partial solutions bounded by $\mathsf{s\text{-}nec}_1(T, \delta)^2$. We call P' the algorithm obtained from P by calling the function reduce at every step of computation. We can assume that the set of partial solutions $\mathcal{A}_r$ computed by P and associated with the root r of (T, δ) contains an optimal solution (this will be the cases in our algorithms). To prove the correctness of P', we need to prove that $\mathcal{A}'_r$ $(r, \varnothing)$-represents $\mathcal{A}_r$ where $\mathcal{A}'_r$ is the set of partial solutions computed by P' and associated with r. For doing so, we need to prove that at each step of the algorithm the operations we use preserve the (x, R')-representativity. The following fact states that we can use the union without restriction, it follows directly from Definition 4.34 of (x, R')-representativity.

Fact 4.36. *If $\mathcal{B}$ and $\mathcal{D}$ (x, R')-represents, respectively, $\mathcal{A}$ and $\mathcal{C}$, then $\mathcal{B} \cup \mathcal{D}$ (x, R')-represents $\mathcal{A} \cup \mathcal{C}$.*

The second operation we use in our dynamic programming algorithms is the merging operator $\bigotimes$. In order to safely use it, we need the following notion of compatibility that just tells which partial solutions from V_a and V_b can be joined to possibly form a partial solution in V_x. (It was already used in [18] without naming it.)

Definition 4.37 (d-(R, R')-compatibility)**.** *Suppose that x is an internal node of T with a and b as children. Let $d \in \mathbb{N}^+$ and $R \in \mathcal{R}^d_{V_x}$. We say that $(A, A') \in \mathcal{R}^d_{V_a} \times \mathcal{R}^d_{\overline{V_a}}$ and $(B, B') \in \mathcal{R}^d_{V_b} \times \mathcal{R}^d_{\overline{V_b}}$ are d-(R, R')-compatible if we have:*

- $A \cup B \equiv^d_{V_x} R$,
- $A' \equiv^d_{\overline{V_a}} B \cup R'$, *and*
- $B' \equiv^d_{\overline{V_b}} A \cup R'$.

The d-(R, R')-compatibility just tells which partial solutions from V_a and V_b can be joined to possibly form a partial solution in V_x.

Lemma 4.38. *Suppose that x is an internal node of T with a and b as children. Let $d \in \mathbb{N}^+$ and $R \in \mathcal{R}^d_{V_x}$. Let $(A, A') \in \mathcal{R}^d_{V_a} \times \mathcal{R}^d_{\overline{V_a}}$ and $(B, B') \in \mathcal{R}^d_{V_b} \times \mathcal{R}^d_{\overline{V_b}}$ that are d-(R, R')-compatible. Let $\mathcal{A} \subseteq 2^{V_a}$ such that, for all $X \in \mathcal{A}$, we have $X \equiv^d_{V_a} A$, and let $\mathcal{B} \subseteq 2^{V_b}$ such that, for all $W \in \mathcal{B}$, we have $W \equiv^d_{V_b} B$. If $\mathcal{A}' \subseteq \mathcal{A}$ (a, A')-represents $\mathcal{A}$ and $\mathcal{B}' \subseteq \mathcal{B}$ (b, B')-represents $\mathcal{B}$, then $\mathcal{A}' \bigotimes \mathcal{B}'$ (x, R')-represents $\mathcal{A} \bigotimes \mathcal{B}$.*

Proof. We assume w.l.o.g. that $\mathbf{opt} = \mathsf{max}$, the proof is symmetric for $\mathbf{opt} = \mathsf{min}$. Suppose that $\mathcal{A}' \subseteq \mathcal{A}$ (a, A')-represents $\mathcal{A}$ and $\mathcal{B}' \subseteq \mathcal{B}$ (b, B')-represents $\mathcal{B}$. To prove the lemma, it is sufficient to prove that $\mathsf{best}(\mathcal{A}' \bigotimes \mathcal{B}', Y) = \mathsf{best}(\mathcal{A} \bigotimes \mathcal{B}, Y)$ for every $Y \equiv^1_{V_x} R'$.

Let $Y \subseteq \overline{V_x}$ such that $Y \equiv^1_{V_x} R'$. We start by proving the following facts

(a) for every $W \in \mathcal{B}$, we have $W \cup Y \equiv^1_{\overline{V_a}} A'$,

(b) for every $X \in \mathcal{A}$, we have $X \cup Y \equiv^1_{\overline{V_b}} B'$.

Let $W \in \mathcal{B}$. Owing to the d-(R, R')-compatibility, we have $B \cup R' \equiv^d_{V_a} A'$. Since $W \equiv^d_{V_b} B$ and $V_b \subseteq \overline{V_a}$, by Fact 4.32, we deduce that $W \equiv^d_{V_a} B$ and thus $W \cup R' \equiv^d_{V_a} A'$. In particular, we have $W \cup R' \equiv^1_{V_a} A'$. Similarly, we have from Fact 4.32 that $W \cup Y \equiv^1_{V_a} A'$ because $Y \equiv^1_{V_x} R'$ and $\overline{V_x} \subseteq \overline{V_a}$. This proves Fact (a). The proof for Fact (b) is symmetric.

Now observe that, by the definitions of best and of the merging operator $\bigotimes$, we have (even if $\mathcal{A} = \varnothing$ or $\mathcal{B} = \varnothing$)

$$\mathsf{best}\big(\mathcal{A} \bigotimes \mathcal{B}, Y\big) = \mathsf{max}\{\mathsf{w}(X) + \mathsf{w}(W) \,:\, X \in \mathcal{A} \wedge W \in \mathcal{B} \wedge G[X \cup W \cup Y] \text{ is connected}\}.$$

Since $\mathsf{best}(\mathcal{A}, W \cup Y) = \mathsf{max}\{\mathsf{w}(X) \,:\, X \in \mathcal{A} \wedge G[X \cup W \cup Y] \text{ is connected}\}$, we deduce that

$$\mathsf{best}\big(\mathcal{A} \bigotimes \mathcal{B}, Y\big) = \mathsf{max}\{\mathsf{best}(\mathcal{A}, W \cup Y) + \mathsf{w}(W) \,:\, W \in \mathcal{B}\}.$$

Since $\mathcal{A}'$ (a, A')-represents $\mathcal{A}$, by Fact (a), we have

$$\begin{aligned}\mathsf{best}\big(\mathcal{A} \bigotimes \mathcal{B}, Y\big) &= \mathsf{max}\{\mathsf{best}(\mathcal{A}', W \cup Y) + \mathsf{w}(W) \,:\, W \in \mathcal{B}\} \\ &= \mathsf{best}\big(\mathcal{A}' \bigotimes \mathcal{B}, Y\big).\end{aligned}$$

Symmetrically, we deduce from Fact (b) that $\mathsf{best}\big(\mathcal{A}' \bigotimes \mathcal{B}, Y\big) = \mathsf{best}\big(\mathcal{A}' \bigotimes \mathcal{B}', Y\big)$. This stands for every $Y \subseteq \overline{V_x}$ such that $Y \equiv^1_{V_x} R'$. Thus, we conclude that $\mathcal{A}' \bigotimes \mathcal{B}'$ (x, R')-represents $\mathcal{A} \bigotimes \mathcal{B}$. □

4.2.3 Connected variant of (σ, ρ)-Dominating Set problems

In this subsection, we use the function d defined in Section 4.1. We remind that $d(\mathbb{N}) := 0$, and for a finite or co-finite subset μ of $\mathbb{N}$, let

$$d(\mu) := 1 + \mathsf{min}(\mathsf{max}(\mu), \mathsf{max}(\mathbb{N} \setminus \mu)).$$

Let $d := \mathsf{max}\{1, d(\sigma), d(\rho)\}$. As explained in Subsection 4.1.3 to characterize the (σ, ρ)-domination it is enough to count up to d neighbors (see Fact 4.19). As in [18], we use the *d-neighbor equivalence* relation to characterize the (σ, ρ)-domination of the partial solutions. We will need the following lemma in our proof.

Lemma 4.39 ([18])**.** *Let $A \subseteq V(G)$. Let $X \subseteq A$ and $Y, Y' \subseteq \overline{A}$ such that $Y \equiv^d_A Y'$. Then $(X \cup Y)$ (σ, ρ)-dominates A if and only if $(X \cup Y')$ (σ, ρ)-dominates A.*

In this subsection, we present an algorithm that computes a maximum (or minimum) connected (σ, ρ)-dominating set with a graph G and a layout (T, δ) as inputs. Its running time is $O(\mathsf{s\text{-}nec}_d(T, \delta)^{O(1)} \cdot n^3)$. The same algorithm, with some little modifications, will be able to find a minimum Steiner tree or a maximum (or minimum) connected co-(σ, ρ)-dominating set as well.

For each node x of T and for each pair $(R, R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}$, we will compute a set of partial solutions $\mathcal{D}_x[R, R']$ *coherent* with (R, R') that (x, R')-represents the set of all partial solutions coherent with (R, R'). We say that a set $X \subseteq V_x$ is coherent with (R, R') if $X \equiv^d_{V_x} R$ and $X \cup R'$ (σ, ρ) dominates V_x. Observe that by Lemma 4.39, we have $X \cup Y$ (σ, ρ)-dominates V_x, for all $Y \equiv^d_{V_x} R'$ and for all $X \subseteq V_x$ coherent with (R, R'). We compute these sets by a bottom-up dynamic programming algorithm, starting at the leaves of T. The computational steps are trivial for the leaves. For the internal nodes of T, we simply use the notion of d-(R, R')-compatibility and the merging operator.

By calling the function reduce defined in Section 4.2.2, each set $\mathcal{D}_x[R, R']$ contains at most $\mathsf{s\text{-}nec}_1(T, \delta)^2$ partial solutions. If we want to compute a maximum (resp. minimum) connected

(σ, ρ)-dominating set, we use the framework of Section 4.2.2 with $\mathbf{opt} = \mathsf{max}$ (resp. $\mathbf{opt} = \mathsf{min}$). If G admits a connected (σ, ρ)-dominating set, then a maximum (or minimum) connected (σ, ρ)-dominating set can be found by looking at the entry $\mathcal{D}_r[\varnothing, \varnothing]$ with r the root of T.

We begin by defining the sets of partial solutions for which we will compute representative sets.

Definition 4.40. *Let $x \in V(T)$. For all pairs $(R, R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}$, we let $\mathcal{A}_x[R, R'] := \{X \subseteq V_x : X \equiv^d_{V_x} R$ and $X \cup R'$ (σ, ρ)-dominates $V_x\}$.*

For each node x of $V(T)$, our algorithm will compute a table $\mathcal{D}_x$ that satisfies the following invariant.

Invariant. For every $(R, R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}$, the set $\mathcal{D}_x[R, R']$ is a subset of $\mathcal{A}_x[R, R']$ of size at most $\mathsf{s\text{-}nec}_1(T, \delta)^2$ that (x, R')-represents $\mathcal{A}_x[R, R']$.

Notice that, by the definition of $\mathcal{A}_r[\varnothing, \varnothing]$ (r being the root of T) and the definition of (x, R')-representativity, if G admits a connected (σ, ρ)-dominating set, then $\mathcal{D}_r[\varnothing, \varnothing]$ must contain a maximum (or minimum) connected (σ, ρ)-dominating set.

The following lemma provides an equality between the entries of the table $\mathcal{A}_x$ and the entries of the tables $\mathcal{A}_a$ and $\mathcal{A}_b$ for each internal node $x \in V(T)$ with a and b as children. We use this lemma to prove, by induction, that the entry $\mathcal{D}_x[R, R']$ (x, R')-represents $\mathcal{A}_x[R, R']$ for every $(R, R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}$. Note that this lemma can be deduced from [18].

Lemma 4.41. *For all $(R, R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}$, we have*

$$\mathcal{A}_x[R, R'] = \bigcup_{(A,A'),\ (B,B')\ d\text{-}(R,R')\text{-compatible}} \mathcal{A}_a[A, A'] \bigotimes \mathcal{A}_b[B, B'].$$

Proof. The lemma is implied by the two following claims.

Claim 4.41.1. *For all $X \in \mathcal{A}_x[R, R']$, there exist d-(R, R')-compatible pairs (A, A') and (B, B') such that $X \cap V_a \in \mathcal{A}_a[A, A']$ and $X \cap V_b \in \mathcal{A}_b[B, B']$.*

Proof. Let $X \in \mathcal{A}_x[R, R']$, $X_a := X \cap V_a$ and $X_b := X \cap V_b$. Let $A := \mathsf{rep}^d_{V_a}(X_a)$ and $A' := \mathsf{rep}^d_{\overline{V_a}}(X_b \cup R')$. Symmetrically, we define $B := \mathsf{rep}^d_{V_b}(X_b)$ and $B' := \mathsf{rep}^d_{\overline{V_b}}(X_a \cup R')$.

We claim that $X_a \in \mathcal{A}_a[A, A']$. As $X \in \mathcal{A}_x[R, R']$, we know, by Definition 4.40, that $X \cup R' = X_a \cup X_b \cup R'$ is a (σ, ρ)-dominating set of V_x. In particular, $X_a \cup (X_b \cup R')$ (σ, ρ)-dominates V_a. Since $A' \equiv^d_{\overline{V_a}} X_b \cup R'$, by Lemma 4.39, we conclude that $X_a \cup A'$ (σ, ρ)-dominates V_a. As $A \equiv^d_{V_a} X_a$, we have $X_a \in \mathcal{A}_a[A, A']$. By symmetry, we deduce $B \in \mathcal{A}_b[B, B']$.

It remains to prove that (A, A') and (B, B') are d-(R, R')-compatible.

- By construction, we have $X_a \cup X_b = X \equiv^d_{V_x} R$. As $A \equiv^d_{V_a} X_a$ and from Fact 4.32, we have $A \cup X_b \equiv^d_{V_x} R$. Since $B \equiv^d_{V_b} X_b$, we deduce that $A \cup B \equiv^d_{V_x} R$.
- By definition, we have $A' \equiv^d_{\overline{V_a}} X_b \cup R'$. As $B \equiv^d_{V_b} X_b$ and by Fact 4.32, we have $A' \equiv^d_{\overline{V_a}} B \cup R'$. Symmetrically, we deduce that $B' \equiv^d_{\overline{V_b}} R' \cup A$.

Thus, (A, A') and (B, B') are d-(R, R')-compatible. □

Claim 4.41.2. *For every $X_a \in \mathcal{A}_a[A, A']$ and $X_b \in \mathcal{A}_b[B, B']$ such that (A, A') and (B, B') are d-(R, R')-compatible, we have $X_a \cup X_b \in \mathcal{A}_x[R, R']$.*

Proof. Since $X_a \equiv^d_{V_a} A$ and $X_b \equiv^d_{V_b} B$, by Fact 4.32, we deduce that $X_a \cup X_b \equiv^d_{V_x} A \cup B$. Thus, by the definition of d-(R, R')-compatibility, we have $X_a \cup X_b \equiv^d_{V_x} R$.

It remains to prove that $X_a \cup X_b \cup R'$ (σ, ρ)-dominates V_x. As before, one can check that Fact 4.32 implies that $X_b \cup R' \equiv^d_{V_a} B \cup R'$. From Lemma 4.39, we conclude that $X_a \cup X_b \cup R'$ (σ, ρ)-dominates V_a. Symmetrically, we prove that $X_a \cup X_b \cup R'$ (σ, ρ)-dominates V_b. As $V_x = V_a \cup V_b$, we deduce that $X_a \cup X_b \cup R'$ (σ, ρ)-dominates V_x. Hence, we have $X_a \cup X_b \in \mathcal{A}_x[R, R']$. □

□

We are now ready to prove the main theorem of this subsection.

Theorem 4.42. *There exists an algorithm that, given an n-vertex graph G and a rooted layout (T, δ) of G, computes a maximum (or minimum) connected (σ, ρ)-dominating set in time $O(\mathsf{s\text{-}nec}_d(T,\delta)^3 \cdot \mathsf{s\text{-}nec}_1(T,\delta)^{2(\omega+1)} \cdot \log(\mathsf{s\text{-}nec}_d(T,\delta)) \cdot n^3)$ with $d := \mathsf{max}\{1, d(\sigma), d(\rho)\}$.*

Proof. The algorithm is a usual bottom-up dynamic programming algorithm and computes for each node x of T the table $\mathcal{D}_x$.

The first step of our algorithm is to compute, for each $x \in V(T)$, the sets $\mathcal{R}^d_{V_x}$, $\mathcal{R}^d_{\overline{V_x}}$ and a data structure to compute $\mathsf{rep}^d_{V_x}(X)$ and $\mathsf{rep}^d_{\overline{V_x}}(Y)$, for any $X \subseteq V_x$ and any $Y \subseteq \overline{V_x}$, in time $O(\log(\mathsf{s\text{-}nec}_d(T,\delta)) \cdot n^2)$. As T has $2n-1$ nodes, by Lemma 4.33, we can compute these sets and data structures in time $O(\mathsf{s\text{-}nec}_d(T,\delta) \cdot \log(\mathsf{s\text{-}nec}_d(T,\delta)) \cdot n^3)$.

Let x be a leaf of T with $V_x = \{v\}$. Observe that, for all $(R, R') \in \mathcal{R}^{V_x}_d \times \mathcal{R}^{\overline{V_x}}_d$, we have $\mathcal{A}_x[R, R'] \subseteq 2^{V_x} = \{\varnothing, \{v\}\}$. Thus, our algorithm can directly compute $\mathcal{A}_x[R, R']$ and set $\mathcal{D}_x[R, R'] := \mathcal{A}_x[R, R']$. In this case, the invariant trivially holds.

Now let x be an internal node with a and b as children such that the invariant holds for a and b. For each $(R, R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}$, the algorithm computes $\mathcal{D}_x[R, R'] := \mathsf{reduce}(\mathcal{B}_x[R, R'])$, where the set $\mathcal{B}_x[R, R']$ is defined as follows

$$\mathcal{B}_x[R, R'] := \bigcup_{(A,A'),\ (B,B')\ d\text{-}(R,R')\text{-compatible}} \mathcal{D}_a[A, A'] \bigotimes \mathcal{D}_b[B, B'].$$

We claim that the invariant holds for x. Let $(R, R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}$.

We start by proving that the set $\mathcal{B}_x[R, R']$ is an (x, R')-representative set of $\mathcal{A}_x[R, R']$. By Lemma 4.38, for all d-(R, R')-compatible pairs (A, A') and (B, B'), we have

$$\mathcal{D}_a[A, A'] \bigotimes \mathcal{D}_b[B, B'] \ (x, R')\text{-represents } \mathcal{A}_a[A, A'] \bigotimes \mathcal{A}_b[B, B'].$$

By Lemma 4.41 and by construction of $\mathcal{D}_x[R, R']$ and from Fact 4.36, we conclude that $\mathcal{B}_x[R, R']$ (x, R')-represents $\mathcal{A}_x[R, R']$.

From the invariant, we have $\mathcal{D}_a[A, A'] \subseteq \mathcal{A}_a[A, A']$ and $\mathcal{D}_b[B, B'] \subseteq \mathcal{A}_b[B, B']$, for all d-(R, R')-compatible pairs (A, A') and (B, B'). Thus, from Lemma 4.41, it is clear that by construction, we have $\mathcal{B}_x[R, R'] \subseteq \mathcal{A}_x[R, R']$. Hence, $\mathcal{B}_x[R, R']$ is a subset and an (x, R')-representative set of $\mathcal{A}_x[R, R']$.

Notice that, for each $X \in \mathcal{B}_x[R, R']$, we have $X \equiv^d_{V_x} R$. Thus, we can apply Theorem 4.35 and the function reduce on $\mathcal{B}_x[R, R']$. By Theorem 4.35, $\mathcal{D}_x[R, R']$ is a subset and an (x, R')-representative set of $\mathcal{B}_x[R, R']$. Thus $\mathcal{D}_x[R, R']$ is a subset of $\mathcal{A}_x[R, R']$. Notice that the (x, R')-representativity is an equivalence relation and in particular it is transitive. Consequently, $\mathcal{D}_x[R, R']$ (x, R')-represents $\mathcal{A}_x[R, R']$. From Theorem 4.35, the size of $\mathcal{D}_x[R, R']$ is at most $\mathsf{nec}_1(V_x)^2$ and that $\mathcal{D}_x[R, R'] \subseteq \mathcal{B}_x[R, R']$. As $\mathsf{nec}_1(V_x) \leq \mathsf{s\text{-}nec}_1(T, \delta)$ and $\mathcal{B}_x[R, R'] \subseteq \mathcal{A}_x[R, R']$, we conclude that the invariant holds for x.

By induction, the invariant holds for all nodes of T. The correctness of the algorithm follows from the fact that $\mathcal{D}_r[\varnothing, \varnothing]$ $(r, \varnothing)$-represents $\mathcal{A}_r[\varnothing, \varnothing]$.

Running Time. Let x be a node of T. Suppose first that x is a leaf of T. Then $|\mathcal{R}^d_{V_x}| \leq 2$ and $|\mathcal{R}^d_{\overline{V_x}}| \leq d$. Thus, $\mathcal{D}_x$ can be computed in time $O(d \cdot n)$.

Assume now that x is an internal node of T with a and b as children.

Notice that, by Definition 4.37, for every $(A, B, R') \in \mathcal{R}^d_{V_a} \times \mathcal{R}^d_{V_b} \times \mathcal{R}^d_{\overline{V_x}}$, there exists only one tuple $(A', B', R) \in \mathcal{R}^d_{\overline{V_a}} \times \mathcal{R}^d_{\overline{V_b}} \times \mathcal{R}^d_{V_x}$ such that (A, A') and (B, B') are d-(R, R')-compatible. More precisely, you have to take $R = \mathsf{rep}^d_{V_x}(A \cup B)$, $A' = \mathsf{rep}^d_{\overline{V_a}}(R' \cup B)$, and $B' = \mathsf{rep}^d_{\overline{V_b}}(R' \cup A)$. Thus, there are at most $\mathsf{s\text{-}nec}_d(T, \delta)^3$ tuples (A, A', B, B', R, R') such that (A, A') and (B, B') are d-(R, R')-compatible. It follows that we can compute the intermediary table $\mathcal{B}_x$ by doing the following.

- Initialize each entry of $\mathcal{B}_x$ to $\varnothing$.
- For each $(A, B, R') \in \mathcal{R}^d_{V_a} \times \mathcal{R}^d_{V_b} \times \mathcal{R}^d_{\overline{V_x}}$, compute $R' := \mathsf{rep}^d_{V_x}(A \cup B)$, $A' = \mathsf{rep}^d_{\overline{V_a}}(R' \cup B)$, and $B' = \mathsf{rep}^d_{\overline{V_b}}(R' \cup A)$. Then, update $\mathcal{B}_x[R, R'] := \mathcal{B}_x[R, R'] \cup (\mathcal{D}_a[A, A'] \bigotimes \mathcal{D}_b[B, B'])$.

Each call to the functions $\mathsf{rep}^d_{V_x}$, $\mathsf{rep}^d_{\overline{V_a}}$, and $\mathsf{rep}^d_{\overline{V_b}}$ takes $O(\log(\mathsf{s\text{-}nec}_d(T, \delta)) \cdot n^2)$ time. We deduce that the running time to compute the entries of $\mathcal{B}_x$ is

$$O\left(\mathsf{s\text{-}nec}_d(T, \delta)^3 \log(\mathsf{s\text{-}nec}_d(T, \delta)) \cdot n^2 + \sum_{(R,R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}} |\mathcal{B}_x[R, R']| \cdot n^2\right).$$

Observe that, for each $(R, R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}$, by Theorem 4.35, the running time to compute $\mathsf{reduce}(\mathcal{B}_x[R, R'])$ from $\mathcal{B}_x[R, R']$ is $O(|\mathcal{B}_x[R, R']| \cdot \mathsf{s\text{-}nec}_1(T, \delta)^{2(\omega-1)} \cdot n^2)$. Thus, the total running time to compute the table $\mathcal{D}_x$ from the table $\mathcal{B}_x$ is

$$O\left(\sum_{(R,R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}} |\mathcal{B}_x[R, R']| \cdot \log(\mathsf{s\text{-}nec}_d(T, \delta)) \cdot \mathsf{s\text{-}nec}_1(T, \delta)^{2(\omega-1)} \cdot n^2\right). \tag{4.3}$$

For each (A, A') and (B, B'), the size of $\mathcal{D}_a[A, A'] \bigotimes \mathcal{D}_b[B, B']$ is at most $|\mathcal{D}_a[A, A']| \cdot |\mathcal{D}_b[B, B']| \leq \mathsf{s\text{-}nec}_1(T, \delta)^4$. Since there are at most $\mathsf{s\text{-}nec}_d(T, \delta)^3$ pairs d-(R, R')-compatible, we can conclude that

$$\sum_{(R,R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}} |\mathcal{B}_x[R, R']| \leq \mathsf{s\text{-}nec}_d(T, \delta)^3 \cdot \mathsf{s\text{-}nec}_1(T, \delta)^4.$$

From Equation (1), we deduce that the entries of $\mathcal{D}_x$ are computable in time

$$O(\mathsf{s\text{-}nec}_d(T, \delta)^3 \cdot \mathsf{s\text{-}nec}_1(T, \delta)^{2(\omega+1)} \cdot \log(\mathsf{s\text{-}nec}_d(T, \delta)) \cdot n^2).$$

Since T has $2n-1$ nodes, the running time of our algorithm is $O(\mathsf{s\text{-}nec}_d(T, \delta)^3 \cdot \mathsf{s\text{-}nec}_1(T, \delta)^{2(\omega+1)} \cdot \log(\mathsf{s\text{-}nec}_d(T, \delta)) \cdot n^3)$. □

As a corollary, we can solve in time $\mathsf{s\text{-}nec}_1(T, \delta)^{(2\omega+5)} \cdot \log(\mathsf{s\text{-}nec}_1(T, \delta)) \cdot n^3$ the Node-Weighted Steiner Tree problem that asks, given a subset of vertices $K \subseteq V(G)$ called *terminals*, a subset T of minimal weight such that $K \subseteq T \subseteq V(G)$ and $G[T]$ is connected.

Corollary 4.43. *There exists an algorithm that, given an n-vertex graph G, a subset $K \subseteq V(G)$, and a rooted layout (T, δ) of G, computes a minimum node-weighted Steiner tree for (G, K) in time $O(\mathsf{s\text{-}nec}_1(T, \delta)^{(2\omega+5)} \cdot \log(\mathsf{s\text{-}nec}_1(T, \delta)) \cdot n^3)$.*

Proof. Observe that a Steiner tree is a minimum connected $(\mathbb{N},\mathbb{N})$-dominating set of G that contains K. Thus, it is sufficient to change the definition of the table $\mathcal{A}_x$ as follows. Let $x \in V(T)$. For all $(R,R') \in \mathcal{R}^1_{V_x} \times \mathcal{R}^1_{\overline{V_x}}$, we define $\mathcal{A}_x[R,R'] \subseteq V_x$ as follows

$$\mathcal{A}_x[R,R'] := \{X \subseteq V_x \,:\, X \equiv^d_{V_x} R,\ K \cap V_x \subseteq X, \text{ and } X \cup R' \ (\mathbb{N},\mathbb{N})\text{-dominates } V_x\}.$$

Notice that this modification will just modify the way we compute the table $\mathcal{D}_x$ when x is a leaf of T associated with a vertex in K. With this definition of $\mathcal{A}_x$ and by Definition 4.34 of (x,R')-representativity, if G contains an optimal solution, then $\mathcal{D}_r[\varnothing,\varnothing]$ contains an optimal solution of G. The running time comes from the running time of Theorem 4.42 with $d=1$. □

Because the incidence graph of a graph of tree-width k has tree-width at most $k+1$, and one can reduce the computation of a weighted Steiner tree on a graph to the computation of a node-weighted Steiner tree on its incidence graph, we simplify and generalise the algorithm from [9]. With few modifications, we can easily deduce an algorithm to compute a maximum (or minimum) connected co-(σ,ρ)-dominating set.

Corollary 4.44. *There exists an algorithm that, given an n-vertex graph G and a rooted layout (T,δ) of G, computes a maximum (or minimum) connected co-(σ,ρ)-dominating set in time $O(\mathsf{s\text{-}nec}_d(T,\delta)^3 \cdot \mathsf{s\text{-}nec}_1(T,\delta)^{(2\omega+5)} \cdot \log(\mathsf{s\text{-}nec}_d(T,\delta)) \cdot n^3)$ with $d := \mathsf{max}\{1, d(\sigma), d(\rho)\}$.*

Proof. To find a maximum (or minimum) co-(σ,ρ)-dominating set, we need to modify the definition of the table $\mathcal{A}_x$, the invariant and the computational steps of the algorithm from Theorem 4.42. For each vertex $x \in V(T)$, we define the set of indices of our table $\mathcal{D}_x$ as $\mathbb{I}_x := \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}} \times \mathcal{R}^1_{V_x} \times \mathcal{R}^1_{\overline{V_x}}$.

For all $(R,R',\overline{R},\overline{R}') \in \mathbb{I}_x$, we define $\mathcal{A}_x[R,R',\overline{R},\overline{R}'] \subseteq 2^{V_x}$ as the following set

$$\{X \subseteq V_x \,:\, X \equiv^1_{V_x} \overline{R},\ (V_x \setminus X) \equiv^d_{V_x} R, \text{ and } (V_x \setminus X) \cup R' \ (\sigma,\rho)\text{-dominates } V_x\}.$$

It is worth noticing that the definition of $\mathcal{A}_x$ does not depend on $\overline{R}'$, it is just more convenient to write the proof this way in order to obtain an algorithm similar to the one from Theorem 4.42.

Similarly to Theorem 4.42, for each node x of $V(T)$, our algorithm will compute a table $\mathcal{D}_x$ that satisfies the following invariant.

Invariant. For every $(R,R',\overline{R},\overline{R}') \in \mathbb{I}_x$, the set $\mathcal{D}_x[R,R',\overline{R},\overline{R}']$ is a subset of $\mathcal{A}_x[R,R',\overline{R},\overline{R}']$ of size at most $\mathsf{s\text{-}nec}_1(T,\delta)^2$ that $(x,\overline{R}')$-represents $\mathcal{A}_x[R,R',\overline{R},\overline{R}']$.

Intuitively, we use $\overline{R}$ and $\overline{R}'$ to deal with the connectivity constraint of the co-(σ,ρ)-dominating set and R and R' for the (σ,ρ)-domination.

The following claim adapts Lemma 4.41 to the co-(σ,ρ)-dominating set case.

Claim 4.44.1. *Let x be an internal node of T with a and b as children. For all $(R,R',\overline{R},\overline{R}') \in \mathbb{I}_x$, we have*

$$\mathcal{A}_x[R,R',\overline{R},\overline{R}'] := \bigcup_{\substack{(A,A'),\ (B,B')\ d\text{-}(R,R')\text{-compatible} \\ (\overline{A},\overline{A}'),\ (\overline{B},\overline{B}')\ 1\text{-}(\overline{R},\overline{R}')\text{-compatible}}} \mathcal{A}_a[A,A',\overline{A},\overline{A}'] \bigotimes \mathcal{A}_b[B,B',\overline{A},\overline{A}'].$$

The proof of this claim follows from the proof of Lemma 4.41. With these modifications, it is straightforward to check that the algorithm of Theorem 4.42 can be adapted to compute a minimum or maximum connected co-(σ,ρ)-dominating set of $V(G)$. With the same analysis as in Theorem 4.42, one easily deduces that the running time of this modified algorithm is $O(\mathsf{s\text{-}nec}_d(T,\delta)^3 \cdot \mathsf{s\text{-}nec}_1(T,\delta)^{(2\omega+5)} \cdot \log(\mathsf{s\text{-}nec}_d(T,\delta)) \cdot n^3)$. □

4.2.4 Acyclic and AC variants of (σ, ρ)-Dominating Set problems

In this subsection, we present an algorithm that solves any AC-(σ, ρ)-Dominating Set problem. Examples of such problems are given in Table 1.1. Unfortunately, we were not able to obtain an algorithm whose running time is polynomial in n and the d-neighbor-width of the given layout (for some constant d). But, for the other parameters, by using their respective properties, we get the running time presented in Table 4.3 which are roughly the same as those in the previous subsection. Moreover, we show, via a polynomial reduction, that we can use our algorithm for AC-(σ, ρ)-Dominating Set problems (with some modifications) to solve any Acyclic (σ, ρ)-Dominating Set problem.

Table 4.3 – Upper bounds on the running time of our algorithms for an AC-(σ, ρ)-Dominating Set problem with $\mathcal{L} = (T, \delta)$ and $d := \mathsf{max}\{2, d(\sigma), d(\rho)\}$.

Parameter	Running time
Mim-width	$O(n^{(2\omega+3d+4)\mathsf{mim}(\mathcal{L})+4} \cdot \mathsf{mim}(\mathcal{L}))$
Module-width	$O((d+1)^{3\mathsf{mw}(\mathcal{L})} \cdot 2^{(2\omega+3)\mathsf{mw}(\mathcal{L})} \cdot \mathsf{mw}(\mathcal{L}) \cdot n^4)$
Rank-width	$O(2^{(2\omega+3d+4)\mathsf{rw}(\mathcal{L})^2} \cdot \mathsf{rw}(\mathcal{L}) \cdot n^4)$
$\mathbb{Q}$-rank-width	$O(2^{(2\omega+5)\mathsf{rw}_{\mathbb{Q}}(\mathcal{L}) \log_2(d \cdot \mathsf{rw}_{\mathbb{Q}}(\mathcal{L}))} \cdot \mathsf{rw}_{\mathbb{Q}}(\mathcal{L}) \cdot n^4)$

Let us first explain why we cannot use the same trick as in [9] on the algorithms of Section 4.1.3 to ensure the acyclicity, that is classifying the partial solutions X – associated with a node $x \in V(T)$ – with respect to $|X|$ and $|E(G[X])|$. Indeed, for two sets $X, W \subseteq V_x$ with $|X| = |W|$ and $|E(G[X])| = |E(G[W])|$, we have $|E(G[X \cup Y])| = |E(G[W \cup Y])|$, for all $Y \subseteq \overline{V_x}$, if and only if $X \equiv^n_{V_x} W$. Hence, the trick used in [9] would imply to classify the partial solutions with respect to their n-neighbor equivalence class. But, the upper bounds we have on $\mathsf{nec}_n(V_x)$ with respect to module-width, ($\mathbb{Q}$-)rank-width would lead to an XP algorithm. In fact, for every $k \in \mathbb{N}$ and every $n \geq 2k$, one can construct an n-vertex bipartite graph $H_k[A, \overline{A}]$ where $\mathsf{mw}(A) = k$ and $\mathsf{nec}_n(A) = (n/\mathsf{mw}(A))^{\mathsf{mw}(A)}$ (see Figure 4.2). Since both $\mathsf{rw}(A)$ and $\mathsf{rw}_{\mathbb{Q}}(A)$ are upper-bounded by $\mathsf{mw}(A)$, we deduce that using the trick of [9] would give, for each $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}\}$, an $n^{\Omega(\mathsf{f}(T,\delta))}$ time algorithm.

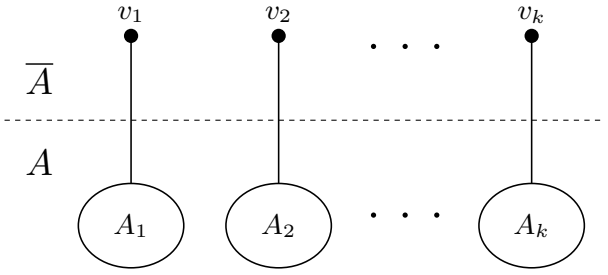


Figure 4.2 – Bipartite graph $H_k[A, \overline{A}]$ where $\mathsf{mw}(A) = k$ and $\mathsf{nec}_n(A) = (n/\mathsf{mw}(A))^{\mathsf{mw}(A)}$. Each A_i's contains $n - k/k$ vertices whose neighborhoods are $\{v_i\}$.

In the following, we introduce some new concepts that extends the framework designed in Section 4.2.2 in order to manage acyclicity. All along, we give intuitions on these concepts through a concrete example: Maximum Induced Tree. Finally, we present the algorithms for the AC-(σ, ρ)-Dominating Set problems and the algorithms for Acyclic (σ, ρ)-Dominating Set problems.

We start by defining a new notion of representativity to deal with the acyclicity constraint. This new notion of representativity is defined w.r.t. to the 2-neighbor equivalence class of a set $R' \subseteq \overline{V_x}$. We consider 2-neighbor equivalence classes instead of 1-neighbor equivalence classes in order to manage the acyclicity (see the following explanations). Similarly to Section 4.2.2, every concept introduced in this section is defined with respect to a node x of T and a set $R' \subseteq \overline{V_x}$. To simplify this section, we fix a node x of T and $R' \subseteq \overline{V_x}$. In our algorithm, R' will always belong to $\mathcal{R}^d_{\overline{V_x}}$ for some $d \in \mathbb{N}^+$ with $d \geq 2$. For Maximum Induced Tree $d = 2$ is enough and in general, we use $d := \mathsf{max}\{2, d(\sigma), d(\rho)\}$.

The following definition extends Definition 4.34 of Section 4.2.2 to deal with the acyclicity. We let $\mathbf{opt} \in \{\mathsf{min}, \mathsf{max}\}$; if we want to solve a maximization (or minimization) problem, we use $\mathbf{opt} = \mathsf{max}$ (or $\mathbf{opt} = \mathsf{min}$).

Definition 4.45 ($(x, R')^{\mathsf{acy}}$-representativity)**.** *For every $\mathcal{A} \subseteq 2^{V(G)}$ and $Y \subseteq V(G)$, we define*

$$\mathsf{best}(\mathcal{A}, Y)^{\mathsf{acy}} := \mathbf{opt}\{\mathsf{w}(X) : X \in \mathcal{A} \text{ and } G[X \cup Y] \text{ is a tree}\}.$$

Let $\mathcal{A}, \mathcal{B} \subseteq 2^{V_x}$. We say that $\mathcal{B}$ $(x, R')^{\mathsf{acy}}$-represents $\mathcal{A}$ if, for every $Y \subseteq \overline{V_x}$ such that $Y \equiv^2_{\overline{V_x}} R'$, we have $\mathsf{best}^{\mathsf{acy}}(\mathcal{A}, Y) = \mathsf{best}^{\mathsf{acy}}(\mathcal{B}, Y)$.

As for the (x, R')-representativity, we need to prove that the operations we use in our algorithm preserve the $(x, R')^{\mathsf{acy}}$-representativity. The following fact follows from Definition 4.45 of $(x, R')^{\mathsf{acy}}$-representativity.

Fact 4.46. *If $\mathcal{B}$ and $\mathcal{D}$ $(x, R')^{\mathsf{acy}}$-represents, respectively, $\mathcal{A}$ and $\mathcal{C}$, then $\mathcal{B} \cup \mathcal{D}$ $(x, R')^{\mathsf{acy}}$-represents $\mathcal{A} \cup \mathcal{C}$.*

The following lemma is an analog of Lemma 4.38 for the notion of $(x, R')^{\mathsf{acy}}$-representativity. The proof is almost the same as the one of Lemma 4.38. We refer to Definition 4.37 for the notion of d-(R, R')-compatibility.

Lemma 4.47. *Let $d \in \mathbb{N}^+$ such that $d \geq 2$. Suppose that x is an internal node of T with a and b as children. Let $R \in \mathcal{R}^d_{V_x}$. Let $(A, A') \in \mathcal{R}^d_{V_a} \times \mathcal{R}^d_{\overline{V_a}}$ and $(B, B') \in \mathcal{R}^d_{V_b} \times \mathcal{R}^d_{\overline{V_b}}$ that are d-(R, R')-compatible. Let $\mathcal{A} \subseteq 2^{V_a}$ such that, for all $X \in \mathcal{A}$, we have $X \equiv^d_{V_a} A$, and let $\mathcal{B} \subseteq 2^{V_b}$ such that, for all $W \in \mathcal{B}$, we have $W \equiv^d_{V_b} B$.*

If $\mathcal{A}' \subseteq \mathcal{A}$ $(a, A')^{\mathsf{acy}}$-represents $\mathcal{A}$ and $\mathcal{B}' \subseteq \mathcal{B}$ $(b, B')^{\mathsf{acy}}$-represents $\mathcal{B}$, then

$$\mathcal{A}' \bigotimes \mathcal{B}' \ (x, R')^{\mathsf{acy}}\text{-represents } \mathcal{A} \bigotimes \mathcal{B}.$$

Proof. Suppose that $\mathcal{A}' \subseteq \mathcal{A}$ $(a, A')^{\mathsf{acy}}$-represents $\mathcal{A}$ and that $\mathcal{B}' \subseteq \mathcal{B}$ $(b, B')^{\mathsf{acy}}$-represents $\mathcal{B}$. In order to prove this lemma, it is sufficient to prove that, for each $Y \equiv^2_{\overline{V_x}} R'$, we have $\mathsf{best}^{\mathsf{acy}}(\mathcal{A}' \bigotimes \mathcal{B}', Y) = \mathsf{best}^{\mathsf{acy}}(\mathcal{A} \bigotimes \mathcal{B}, Y)$.

Let $Y \subseteq \overline{V_x}$ such that $Y \equiv^2_{\overline{V_x}} R'$. We claim the following facts

(a) for every $W \in \mathcal{B}$, we have $W \cup Y \equiv^2_{\overline{V_a}} A'$,

(b) for every $X \in \mathcal{A}$, we have $X \cup Y \equiv^2_{\overline{V_b}} B'$.

Let $W \in \mathcal{B}$. By Fact 4.32, we have that $W \equiv^d_{\overline{V_a}} B$ because $V_b \subseteq \overline{V_a}$ and $W \equiv^d_{V_b} B$. Since $d \geq 2$, we have $W \equiv^2_{\overline{V_a}} B$. By Fact 4.32, we deduce also that $Y \equiv^2_{\overline{V_a}} R'$. Since (A, A') and (B, B') are d-(R, R')-compatible, we have $A' \equiv^d_{\overline{V_a}} R' \cup B$. In particular, we have $A' \equiv^2_{\overline{V_a}} R' \cup B$ because $d \geq 2$. We can conclude that $W \cup Y \equiv^2_{\overline{V_a}} A'$ for every $W \in \mathcal{B}$. The proof for Fact (b) is symmetric.

Now observe that, by the definitions of $\mathsf{best}^{\mathsf{acy}}$ and of the merging operator $\bigotimes$, we have

$$\mathsf{best}^{\mathsf{acy}}\big(\mathcal{A}\bigotimes\mathcal{B},Y\big)=\mathbf{opt}\{\mathsf{w}(X)+\mathsf{w}(W)\,:\,X\in\mathcal{A}\wedge W\in\mathcal{B}\wedge G[X\cup W\cup Y]\text{ is a tree}\}.$$

Since $\mathsf{best}^{\mathsf{acy}}(\mathcal{A},W\cup Y)=\mathbf{opt}\{\mathsf{w}(X)\,:\,X\in\mathcal{A}\wedge G[X\cup W\cup Y]\text{ is a tree}\}$, we deduce that

$$\mathsf{best}^{\mathsf{acy}}\big(\mathcal{A}\bigotimes\mathcal{B},Y\big)=\mathbf{opt}\{\mathsf{best}^{\mathsf{acy}}(\mathcal{A},W\cup Y)+\mathsf{w}(W)\,:\,W\in\mathcal{B}\}.$$

Since $\mathcal{A}'$ (a,A')-represents $\mathcal{A}$ and by Fact (a), we have

$$\begin{aligned}\mathsf{best}^{\mathsf{acy}}\big(\mathcal{A}\bigotimes\mathcal{B},Y\big)&=\mathbf{opt}\{\mathsf{best}^{\mathsf{acy}}(\mathcal{A}',W\cup Y)+\mathsf{w}(W)\,:\,W\in\mathcal{B}\}\\&=\mathsf{best}^{\mathsf{acy}}\big(\mathcal{A}'\bigotimes\mathcal{B},Y\big).\end{aligned}$$

Symmetrically, we deduce from Fact (b) that $\mathsf{best}^{\mathsf{acy}}\big(\mathcal{A}'\bigotimes\mathcal{B},Y\big)=\mathsf{best}^{\mathsf{acy}}\big(\mathcal{A}'\bigotimes\mathcal{B}',Y\big)$. This stands for every $Y\subseteq\overline{V_x}$ such that $Y\equiv^2_{V_x}R'$. Thus, we conclude that $\mathcal{A}'\bigotimes\mathcal{B}'$ $(x,R')^{\mathsf{acy}}$-represents $\mathcal{A}\bigotimes\mathcal{B}$. □

In order to compute a maximum induced tree, we design an algorithm similar to those of Section 4.1.3. That is, for each $(R,R')\in\mathcal{R}^2_{V_x}\times\mathcal{R}^2_{\overline{V_x}}$, our algorithm will compute a set $\mathcal{D}_x[R,R']\subseteq 2^{V_x}$ that is an $(x,R')^{\mathsf{acy}}$-representative set of small size of the set $\mathcal{A}_x[R]:=\{X\subseteq V_x$ such that $X\equiv^2_{V_x}R\}$. This is sufficient to compute a maximum induced tree of G since we have $\mathcal{A}_r[\varnothing]=2^{V(G)}$ with r the root of T. Thus, by Definition 4.45, any $(r,\varnothing)^{\mathsf{acy}}$-representative set of $\mathcal{A}_r[\varnothing]$ contains a maximum induced tree.

The key to compute the tables of our algorithm is a function that, given $\mathcal{A}\subseteq 2^{V_x}$, computes a small subset of $\mathcal{A}$ that $(x,R')^{\mathsf{acy}}$-represents $\mathcal{A}$. This function starts by removing from $\mathcal{A}$ some sets that will never give a tree with a set $Y\equiv^2_{\overline{V_x}}R'$. For doing so, we characterize the sets $X\in\mathcal{A}$ such that $G[X\cup Y]$ is a tree for some $Y\equiv^2_{\overline{V_x}}R'$. We call these sets *$R'$-important*. The following gives a formal definition of these important and unimportant partial solutions.

Definition 4.48 (R'-important)**.** *We say that $X\subseteq V_x$ is R'-important if there exists $Y\subseteq\overline{V_x}$ such that $Y\equiv^2_{\overline{V_x}}R'$ and $G[X\cup Y]$ is a tree, otherwise, we say that X is R'-unimportant.*

By definition, any set obtained from a set $\mathcal{A}$ by removing R'-unimportant sets is an $(x,R')^{\mathsf{acy}}$-representative set of $\mathcal{A}$. The following lemma gives some necessary conditions on R'-important sets. It follows that any set which does not respect one of these conditions can safely be removed from $\mathcal{A}$. These conditions are the key to obtain the running times of Table 4.2. At this point, we need to introduce the following notations. For every $X\subseteq V_x$, we define $X^0:=\{v\in X\,:\,N(v)\cap R'=\varnothing\}$, $X^1:=\{v\in X\,:\,|N(v)\cap R'|=1\}$, and $X^{2+}:=\{v\in X\,:\,|N(v)\cap R'|\geq 2\}$. Notice that, for every $Y\equiv^2_{\overline{V_x}}R'$ and $X\subseteq V_x$, the vertices in X^0 have no neighbor in Y, those in X^1 have exactly one neighbor in Y and those in X^{2+} have at least 2 neighbors in Y.

In order to prove the lemma, we need the properties of the 2-neighbor equivalence relation. More precisely, we use the fact that, for all $X\subseteq V_x$ and $Y\equiv^2_{\overline{V_x}}R'$, the vertices in X having at least two neighbors in Y correspond exactly to those having at least two neighbors in R'. These vertices play a major role in the acyclicity and the computation of representatives in the following sense. By removing from $\mathcal{A}$ the sets that do not respect the two above properties, we are able to decompose $\mathcal{A}$ into a small number of sets $\mathcal{A}_1,\ldots,\mathcal{A}_t$ such that an (x,R')-representative set of $\mathcal{A}_i$ is an $(x,R')^{\mathsf{acy}}$-representative set of $\mathcal{A}_i$ for each $i\in\{1,\ldots,t\}$. We find an $(x,R')^{\mathsf{acy}}$-representative set of $\mathcal{A}$, by computing an (x,R')-representative set $\mathcal{B}_i$ for each $\mathcal{A}_i$ with the function reduce. This is sufficient because $\mathcal{B}_1\cup\cdots\cup\mathcal{B}_t$ is an $(x,R')^{\mathsf{acy}}$-representative set of $\mathcal{A}$.

Lemma 4.49. *If $X \subseteq V_x$ is R'-important, then $G[X]$ is a forest and the following properties are satisfied:*

1. *for every pair of distinct vertices a and b in X^{2+}, we have $N(a) \cap \overline{V_x} \neq N(b) \cap \overline{V_x}$,*
2. *$|X^{2+}|$ is upper bounded by $2\mathsf{mim}(V_x)$, $2\mathsf{rw}(V_x)$, $2\mathsf{rw}_{\mathbb{Q}}(V_x)$, and $2\log_2(\mathsf{nec}_1(V_x))$.*

Proof. Obviously, any R'-important set must induce a forest.

Let $X \subseteq V_x$ be an R'-important set. Since X is R'-important, there exists $Y \subseteq \overline{V_x}$ such that $Y \equiv^2_{V_x} R'$ and $G[X \cup Y]$ is a tree.

Assume towards a contradiction that there exist two distinct vertices a and b in X^{2+} such that $N(a) \cap \overline{V_x} = N(b) \cap \overline{V_x}$. Since a and b belong to X^{2+} and $Y \equiv^2_{V_x} R'$, both a and b have at least two neighbors in Y. Thus, a and b have at least two common neighbors in Y. We conclude that $G[X \cup Y]$ admits a cycle of length four, yielding a contradiction. We conclude that Property (1) holds for every R'-important set.

Now, we prove that Property (2) holds for X. Observe that, by Theorem 2.32, $\mathsf{mim}(V_x)$ is upper bounded by $\mathsf{rw}(V_x)$, $\mathsf{rw}_{\mathbb{Q}}(V_x)$, and $\log_2(\mathsf{nec}_1(V_x))$. Thus, in order to prove Property (2), it is sufficient to prove that $|X^{2+}| \leq 2\mathsf{mim}(V_x)$.

We claim that $|X^{2+}| \leq 2k$ where k is the size of a maximum induced matching of $F := G[X^{2+}, Y]$. Since F is an induced subgraph of $G[V_x, \overline{V_x}]$, we have $k \leq \mathsf{mim}(V_x)$ and this is enough to prove Property (2). Notice that F is a forest because F is a subgraph of $G[X \cup Y]$, which is a tree.

In the following, we prove that F admits a *good bipartition*, that is a bipartition $\{X_1, X_2\}$ of $X^{2+} \cap V(F)$ such that, for each $i \in \{1, 2\}$ and, for each $v \in X_i$, there exists $y_v \in Y \cap V(F)$ such that $N_F(y_v) \cap X_i = \{v\}$. Observe that this is enough to prove Property (2) since if F admits a good bipartition $\{X_1, X_2\}$, then $|X_1| \leq k$ and $|X_2| \leq k$. Indeed, if F admits a good bipartition $\{X_1, X_2\}$, then, for each $i \in \{1, 2\}$, the set of edges $M_i = \{vy_v : v \in X_i\}$ is an induced matching of F. In order to prove that F admits a good bipartition it is sufficient to prove that each connected component of F admits a good bipartition.

Let $C \in \mathsf{cc}(F)$ and $u \in C \cap X^{2+}$. As F is a forest, $F[C]$ is a tree. Observe that the distance in F between each vertex $v \in C \cap X^{2+}$ and u is even because $F := G[X^{2+}, Y]$. Let C_1 (resp. C_2) be the set of all vertices $v \in C \cap X^{2+}$ such that there exists an odd (resp. even) integer $\ell \in \mathbb{N}$ so that the distance between v and u in F is $2i$. We claim that $\{C_1, C_2\}$ is a good bipartition of $F[C]$.

Let $i \in \{1, 2\}$, $v \in C_i$ and $\ell \in \mathbb{N}$ such that the distance between v and u in F is 2ℓ. Let P be the set of vertices in $V(F) \setminus \{v\}$ that share a common neighbor with v in F. We want to prove that there exists $y \in Y$ such that $N_F(y) \cap C_i = \{v\}$. For doing so, it is sufficient to prove that $N_F(v) \setminus N_F(C_i \setminus \{v\}) = N_F(v) \setminus N_F(P \cap C_i) \neq \varnothing$. Observe that, for every $v' \in P$, the distance between v' and u in F is either $2\ell - 2$, 2ℓ or $2\ell + 2$ because $F[C]$ is a tree and the distance between v and u is 2ℓ. By construction of $\{C_1, C_2\}$, every vertex at distance $2\ell - 2$ and $2\ell + 2$ from u must belong to C_{3-i}. Thus, every vertex in $P \cap C_i$ is at distance 2ℓ from u. If $\ell = 0$, then we are done because $v = u$ and $P \cap C_i = \varnothing$. Assume that $\ell \neq 0$. As $F[C]$ is a tree, v has only one neighbor w at distance $2\ell - 1$ from u in F. Because $F[C]$ is a tree, we deduce that $N_F(v) \cap N_F(P \cap C_i) = \{w\}$. Since $v \in X^{2+}$, v has at least two neighbors in $F = G[X^{2+}, Y]$ (because $Y \equiv^2_{V_x} R'$), we conclude that $N_F(v) \setminus N_F(P \cap C_i) \neq \varnothing$. Hence, we deduce that $\{C_1, C_2\}$ is a good bipartition of $F[C]$.

We deduce that every connected component of F admits a good bipartition and thus F admits a good bipartition. Thus, $|X^{2+}| \leq 2\mathsf{mim}(V_x)$. □

The following definition characterizes the sets $\mathcal{A} \subseteq 2^{V_x}$ for which an (x, R')-representative set is also an $(x, R')^{\mathsf{acy}}$-representative set.

Definition 4.50. *We say that $\mathcal{A} \subseteq 2^{V_x}$ is R'-consistent if, for each $Y \subseteq \overline{V_x}$ such that $Y \equiv^2_{V_x} R'$, if there exists $W \in \mathcal{A}$ such that $G[W \cup Y]$ is a tree, then, for each $X \in \mathcal{A}$, either $G[X \cup Y]$ is a tree or $G[X \cup Y]$ is not connected.*

The following lemma proves that an (x, R')-representative set of an R'-consistent set is also an $(x, R')^{\mathsf{acy}}$-representative set of this later.

Lemma 4.51. *Let $\mathcal{A} \subseteq 2^{V_x}$. For all $\mathcal{D} \subseteq \mathcal{A}$, if $\mathcal{A}$ is R'-consistent and $\mathcal{D}$ (x, R')-represents $\mathcal{A}$, then $\mathcal{D}$ $(x, R')^{\mathsf{acy}}$-represents $\mathcal{A}$.*

Proof. We assume that $\mathbf{opt} = \mathsf{max}$, the proof for $\mathbf{opt} = \mathsf{min}$ is similar. Let $Y \equiv^2_{V_x} R'$. If $\mathsf{best}^{\mathsf{acy}}(\mathcal{A}, Y) = -\infty$, then we also have $\mathsf{best}^{\mathsf{acy}}(\mathcal{D}, Y) = -\infty$ because $\mathcal{D} \subseteq \mathcal{A}$.

Assume now that $\mathsf{best}^{\mathsf{acy}}(\mathcal{A}, Y) \neq -\infty$. Thus, there exists $W \in \mathcal{A}$ such that $G[W \cup Y]$ is a tree. Since $\mathcal{A}$ is R'-consistent, for all $X \in \mathcal{A}$, the graph $G[X \cup Y]$ is either a tree or it is not connected. Thus, by Definition 4.34 of best, we have $\mathsf{best}^{\mathsf{acy}}(\mathcal{A}, Y) = \mathsf{best}(\mathcal{A}, Y)$. As $\mathcal{D} \subseteq \mathcal{A}$, we have also $\mathsf{best}^{\mathsf{acy}}(\mathcal{D}, Y) = \mathsf{best}(\mathcal{D}, Y)$. We conclude by observing that if $\mathcal{D}$ (x, R')-represents $\mathcal{A}$, then $\mathsf{best}^{\mathsf{acy}}(\mathcal{D}, Y) = \mathsf{best}^{\mathsf{acy}}(\mathcal{A}, Y)$. □

The next lemma proves that, for each $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$, we can decompose a set $\mathcal{A} \subseteq 2^{V_x}$ into a small collection $\{\mathcal{A}_1, \ldots, \mathcal{A}_t\}$ of pairwise disjoint subsets of $\mathcal{A}$ such that each $\mathcal{A}_i$ is R'-consistent. Even though some parts of the proof are specific to each parameter, the ideas are roughly the same. First, we remove the sets X in $\mathcal{A}$ that do not induce a forest. If $\mathsf{f} = \mathsf{mw}$, we remove the sets in $\mathcal{A}$ that do not respect Condition (1) of Lemma 4.49, otherwise, we remove the sets that do not respect the upper bound associated with f from Condition (2) of Lemma 4.49. These sets can be safely removed as, by Lemma 4.49, they are R'-unimportant. After removing these sets, we obtain the decomposition of $\mathcal{A}$ by taking the equivalence classes of some equivalence relation that is roughly the n-neighbor equivalence relation. Owing to the set of R'-unimportant sets we have removed from $\mathcal{A}$, we prove that the number of equivalence classes of this latter equivalence relation respects the upper bound associated with f that is described in Table 4.4.

Lemma 4.52. *Let $\mathcal{A} \subseteq 2^{V_x}$. For each $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$, there exists a collection of pairwise disjoint subsets $\mathcal{A}_1, \ldots, \mathcal{A}_t$ of $\mathcal{A}$ computable in time $O(|\mathcal{A}| \cdot \mathcal{N}_{\mathsf{f}}(T, \delta) \cdot n^2)$ such that*

- *$\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_t$ $(x, R')^{\mathsf{acy}}$-represents $\mathcal{A}$,*
- *$\mathcal{A}_i$ is R'-consistent for each $i \in \{1, \ldots, t\}$ and*
- *$t \leq \mathcal{N}_{\mathsf{f}}(T, \delta)$,*

where $\mathcal{N}_{\mathsf{f}}(T, \delta)$ is the term defined in Table 4.4.

Table 4.4 – Upper bounds $\mathcal{N}_{\mathsf{f}}(T, \delta)$ on the size of the decomposition of Lemma 4.52 for each $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$.

f	mw	$\mathsf{rw}_{\mathbb{Q}}$	rw	mim
$\mathcal{N}_{\mathsf{f}}(T, \delta)$	$2^{\mathsf{mw}(T,\delta)} \cdot 2n$	$(2\mathsf{rw}_{\mathbb{Q}}(V_x) + 1)^{\mathsf{rw}_{\mathbb{Q}}(V_x)} \cdot 2n$	$2^{2\mathsf{rw}(T,\delta)^2} \cdot 2n$	$2n^{2\mathsf{mim}(T,\delta)+1}$

Proof. We begin by defining an equivalence relation $\sim$ on 2^{V_x} such that each equivalence class of $\sim$ over 2^{V_x} is an R'-consistent set.

For $X \subseteq V_x$, let $\sigma(X)$ be the vector corresponding to the sum (over $\mathbb{Q}$) of the row vectors of $M_{V_x,\overline{V_x}}$ corresponding to X. Notice that if $\sigma(X) = \sigma(X')$, then $X \equiv^d_{V_x} X'$, for all $d \in \mathbb{N}^+$, because the entries of $\sigma(X)$ represent the number of neighbors in X for each vertex in $\overline{V_x}$. Moreover, if $\sigma(X) = \sigma(X')$, then $|E(G[X,Y])| = |E(G[X',Y])|$ for every $Y \subseteq \overline{V_x}$.

We define the equivalence relation $\sim$ on 2^{V_x} such that $X \sim W$ if we have $\sigma(X^{2+}) = \sigma(W^{2+})$ and $|E(G[X])| - |X \setminus X^1| = |E(G[W])| - |W \setminus W^1|$.

Intuitively, if $X \sim W$, then, for all $Y \equiv^2_{V_x} R'$, we have $|E(G[X \cup Y])| = |X \cup Y| - 1$ if and only if $|E(G[W \cup Y])| = |W \cup Y| - 1$. Thus, if $X \sim W$ and both sets induce with Y a connected graph, then both sets induce with Y a tree (because a graph F is a tree if and only if F is connected and $|V(F)| = |E(F)| - 1$). Consequently, an equivalence class of $\sim$ is an R'-consistent set.

Claim 4.52.1. *Let $\mathcal{A}' \subseteq \mathcal{A}$. If, for all $X, W \in \mathcal{A}'$, we have $X \sim W$, then $\mathcal{A}'$ is R'-consistent.*

Proof. Suppose that $X \sim W$ for all $X, W \in \mathcal{A}'$. In order to prove that $\mathcal{A}'$ is R'-consistent, it is enough to prove that, for each $X, W \in \mathcal{A}'$ and $Y \equiv^2_{V_x} R'$, if $G[X \cup Y]$ is a tree and $G[W \cup Y]$ is connected, then $G[W \cup Y]$ is a tree.

Let $Y \equiv^2_{V_x} R'$ and $X, W \in \mathcal{A}'$. Assume that $G[X \cup Y]$ is a tree and that $G[W \cup Y]$ is connected. We want to prove that $G[W \cup Y]$ is a tree.

Since $G[X \cup Y]$ is a tree, we have $|E(G[X \cup Y])| = |X \cup Y| - 1$. Since the vertices in X^0 have no neighbors in Y, we can decompose $|E(G[X \cup Y])| = |X \cup Y| - 1$ to obtain the following equality

$$|E(G[Y])| + |E(G[X^{2+}, Y])| + |E(G[X^1, Y])| + |E(G[X])| = |X \cup Y| - 1. \tag{4.4}$$

Since every vertex in X^1 has exactly one neighbor in Y (because $Y \equiv^2_{V_x} R'$) , we have $|E(G[X^1, Y])| = |X^1|$. Thus, Equation (1) is equivalent to

$$|E(G[Y])| + |E(G[X^{2+}, Y])| + |E(G[X])| = |X \setminus X^1| + |Y| - 1. \tag{4.5}$$

Since $X \sim W$, we have $|E(G[X])| - |X \setminus X^1| = |E(G[W])| - |W \setminus W^1|$. Moreover, owing to $\sigma(X^{2+}) = \sigma(W^{2+})$, we have $|E(G(X^{2+}, Y))| = |E(G(W^{2+}, Y))|$. We conclude that Equation (2) is equivalent to

$$|E(G[Y])| + |E(G[W^{2+}, Y])| + |E(G[W])| = |W \setminus W^1| + |Y| - 1. \tag{4.6}$$

With the same arguments to prove that (3) is equivalent to $|E(G[X \cup Y])| = |X \cup Y| - 1$, we can show that (3) is equivalent to $|E(G(W \cup Y))| = |W \cup Y| - 1$. By assumption, $G[W \cup Y]$ is connected and thus we conclude that $G[W \cup Y]$ is a tree. □

We are now ready to decompose $\mathcal{A}$. We start by removing from $\mathcal{A}$ all the sets that do not induce a forest. Trivially, this can be done in time $O(|\mathcal{A}| \cdot n)$. Moreover, these sets are R'-unimportant and thus we keep an $(x, R')^{\mathsf{acy}}$-representative set of $\mathcal{A}$. Before explaining how we proceed separately for each parameter, we need the following observation which follows from the removal of all the sets in $\mathcal{A}$ that do not induce a forest.

Observation 4.53. *For all $X \in \mathcal{A}$, we have $-n \leq |E(G[X])| - |X \setminus X^1| < n$.*

Concerning module-width. We remove all the sets X in $\mathcal{A}$ that do not respect Condition (1) of Lemma 4.49. By Lemma 4.49, these sets are R'-unimportant and thus we keep an $(x, R')^{\mathsf{acy}}$-representative set of $\mathcal{A}$. After removing these sets, for each $X \in \mathcal{A}$, every pair (a, b) of distinct vertices in X^{2+} have a different neighborhood in $\overline{V_x}$. Observe that, by definition of module-width, we have

- $\mathsf{mw}(V_x) = |\{N(v) \cap \overline{V_x} \,:\, v \in V_x\}|$ and
- for every $a, b \in V_x$, if $N(a) \cap \overline{V_x} = N(b) \cap \overline{V_x}$, then $\sigma(\{a\}) = \sigma(\{b\})$.

We deduce from these observations that $|\{\sigma(X^{2+}) \,:\, X \in \mathcal{A}\}| \leq 2^{\mathsf{mw}(V_x)}$. Thus, the number of equivalence classes of $\sim$ over $\mathcal{A}$ is at most $2^{\mathsf{mw}(V_x)} \cdot 2n \leq \mathcal{N}_{\mathsf{mw}}(T, \delta)$. The factor $2n$ comes from Observation 4.53 and appears also in all subsequent upper-bounds.

Concerning mim-width. We remove from $\mathcal{A}$ all the sets X such that $|X^{2+}| > 2\mathsf{mim}(V_x)$. By Lemma 4.49, these sets are R'-unimportant and thus we keep an $(x, R')^{\mathsf{acy}}$-representative set of $\mathcal{A}$. Observe that this can be done in time $O(n^{\mathsf{mim}(V_x)+1} + |\mathcal{A}| \cdot n^2)$ because $\mathsf{mim}(V_x)$ can be computed in time $O(n^{\mathsf{mim}(V_x)} + 1)$. Since $|X^{2+}| \leq 2\mathsf{mim}(V_x)$, for every $X \in \mathcal{A}$, we have $|\{\sigma(X^{2+}) \,:\, X \in \mathcal{A}\}| \leq n^{2\mathsf{mim}(V_x)}$.

Hence, the number of equivalence classes of $\sim$ over $\mathcal{A}$ is at most $2n^{2\mathsf{mim}(V_x)+1} \leq \mathcal{N}_{\mathsf{mim}}(T, \delta)$.

Concerning rank-width. We remove from $\mathcal{A}$ all the sets X such that $|X^{2+}| > 2\mathsf{rw}(V_x)$ because they are R'-unimportant by Lemma 4.49. We know that there are at most $2^{\mathsf{rw}(V_x)}$ different rows in M. Thus, we have $|\{\sigma(X^{2+}) \,:\, X \in \mathcal{A}\}| \leq (2^{\mathsf{rw}(V_x)})^{2\mathsf{rw}(V_x)}$.

We can therefore conclude that the number of equivalence classes of $\sim$ over $\mathcal{A}$ is at most $2^{2\mathsf{rw}(V_x)^2} \cdot 2n \leq \mathcal{N}_{\mathsf{rw}}(T, \delta)$.

Concerning $\mathbb{Q}$-rank-width. We remove all the sets $X \in \mathcal{A}$ such that $|X^{2+}| > 2\mathsf{rw}_{\mathbb{Q}}(V_x)$. By Lemma 4.49, we keep an $(x, R')^{\mathsf{acy}}$-representative set of $\mathcal{A}$. We claim that $|\{\sigma(X^{2+}) \,:\, X \in \mathcal{A}\}| \leq 2^{\mathsf{rw}_{\mathbb{Q}}(V_x) \cdot \log_2(2\mathsf{rw}_{\mathbb{Q}}(V_x)+1)}$. Notice that the proof can be deduced from [120, Theorem 4.2].

Let C be a set of $\mathsf{rw}_{\mathbb{Q}}(A)$ linearly independent columns of $M_{A,\overline{A}}$. Since the rank over $\mathbb{Q}$ of $M_{V_x,\overline{V_x}}$ is $\mathsf{rw}_{\mathbb{Q}}(V_x)$, every linear combination of row vectors of $M_{V_x,\overline{V_x}}$ is completely determined by its entries in C. Since $|X^{2+}| \leq 2\mathsf{rw}_{\mathbb{Q}}(V_x)$ for every $X \in \mathcal{A}$, the values in $\sigma(X^{2+})$ are between 0 and $2\mathsf{rw}_{\mathbb{Q}}(V_x)$. Hence, the number of possible values for $\sigma(X^{2+})$ is at most $(2\mathsf{rw}_{\mathbb{Q}}(V_x)+1)^{\mathsf{rw}_{\mathbb{Q}}(V_x)}$.

We conclude that the number of equivalence classes of $\sim$ over $\mathcal{A}$ is at most $(2\mathsf{rw}_{\mathbb{Q}}(V_x) + 1)^{\mathsf{rw}_{\mathbb{Q}}(V_x)} \cdot 2n \leq \mathcal{N}_{\mathsf{rw}_{\mathbb{Q}}}(T, \delta)$.

It remains to prove the running time. Observe that, for module-width, ($\mathbb{Q}$-)rank-width and 1-neighbor-width, the removal of R'-unimportant sets can be done in time $O(|\mathcal{A}| \cdot n^2)$. Indeed, $\mathsf{mw}(V_x), \mathsf{rw}(V_x)$ and $\mathsf{rw}_{\mathbb{Q}}(V_x)$ can be computed in time $O(n^2)$. Notice that we can decide whether $X \sim W$ in time $O(n^2)$. Therefore, for each $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$, we can therefore compute the equivalence classes of $\mathcal{A}$ in time $O(|\mathcal{A}| \cdot \mathcal{N}_{\mathsf{f}}(T, \delta) \cdot n^2)$. □

We are now ready to give an analog of Theorem 4.8 for the $(x, R')^{\mathsf{acy}}$-representativity.

Theorem 4.54. *Let $R \in \mathcal{R}^2_{V_x}$. For each $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$, there exists an algorithm $\mathsf{reduce}^{\mathsf{acy}}_{\mathsf{f}}$ that, given a set $\mathcal{A} \subseteq 2^{V_x}$ such that $X \equiv^2_{V_x} R$ for every $X \in \mathcal{A}$, outputs in time $O((\mathsf{nec}_1(V_x)^{2(\omega-1)} + \mathcal{N}_{\mathsf{f}}(T, \delta)) \cdot |\mathcal{A}| \cdot n^2)$, a subset $\mathcal{B} \subseteq \mathcal{A}$ such that $\mathcal{B}$ $(x, R')^{\mathsf{acy}}$-represents $\mathcal{A}$ and $|\mathcal{B}| \leq \mathcal{N}_{\mathsf{f}}(T, \delta) \cdot \mathsf{nec}_1(V_x)^2$.*

Proof. Let $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$. By Lemma 4.52, we can compute in time $O(|\mathcal{A}| \cdot \mathcal{N}_{\mathsf{f}}(T, \delta) \cdot n^2)$ a collection $\{\mathcal{A}_1, \ldots, \mathcal{A}_t\}$ of pairwise disjoint subsets of $\mathcal{A}$ such that

- $\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_t$ $(x, R')^{\mathsf{acy}}$-represents $\mathcal{A}$,
- $\mathcal{A}_i$ is R'-consistent for each $i \in \{1, \ldots, t\}$,

- $t \leq \mathcal{N}_{\mathsf{f}}(T, \delta)$.

For each $X \in \mathcal{A}$, we have $X \equiv^1_{V_x} R$ because $X \equiv^2_{V_x} R$. Since $\mathcal{A}_1, \ldots, \mathcal{A}_t \subseteq \mathcal{A}$, we can apply Theorem 4.8 to compute, for each $i \in \{1, \ldots, t\}$, the set $\mathcal{B}_i := \mathsf{reduce}(\mathcal{A}_i)$. By Theorem 4.8, for each $i \in \{1, \ldots, t\}$, the set $\mathcal{B}_i$ is a subset and an (x, R')-representative set of $\mathcal{A}_i$ whose size is bounded by $\mathsf{nec}_1(V_x)^2$. Moreover, as $\mathcal{A}_i$ is R'-consistent, we have $\mathcal{B}_i$ $(x, R')^{\mathsf{acy}}$-represents $\mathcal{A}_i$ by Lemma 4.51.

Let $\mathcal{B} := \mathcal{B}_1 \cup \cdots \cup \mathcal{B}_t$. Since $\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_t$ $(x, R')^{\mathsf{acy}}$-represents $\mathcal{A}$, we deduce from Fact 4.46 that $\mathcal{B}$ $(x, R')^{\mathsf{acy}}$-represents $\mathcal{A}$. Furthermore, we have $|\mathcal{B}| \leq \mathcal{N}_{\mathsf{f}}(T, \delta) \cdot \mathsf{nec}_1(V_x)^2$ owing to $t \leq \mathcal{N}_{\mathsf{f}}(T, \delta)$ and $|\mathcal{B}_i| \leq \mathsf{nec}_1(V_x)^2$ for all $i \in \{1, \ldots, t\}$.

It remains to prove the running time. By Theorem 4.8, we can compute $\mathcal{B}_1, \ldots, \mathcal{B}_t$ in time $O(|\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_t| \cdot \mathsf{nec}_1(V_x)^{2(\omega-1)} \cdot n^2)$. Since the sets $\mathcal{A}_1, \ldots, \mathcal{A}_t$ are subsets of $\mathcal{A}$ and pairwise disjoint, we have $|\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_t| \leq |\mathcal{A}|$. That proves the running time and concludes the theorem. □

We are now ready to present an algorithm that solves any AC-(σ, ρ)-Dominating Set problem. This algorithm follows the same ideas as the algorithm from Theorem 4.27, except that we use $\mathsf{reduce}^{\mathsf{acy}}_{\mathsf{f}}$ instead of reduce.

Theorem 4.55. *For each $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$, there exists an algorithm that, given an n-vertex graph G and a rooted layout (T, δ) of G, solves any* AC-(σ, ρ)-Dominating Set *problem, in time*

$$O(\mathsf{s\text{-}nec}_d(T, \delta)^3 \cdot \mathsf{s\text{-}nec}_1(T, \delta)^{2(\omega+1)} \cdot \mathcal{N}_{\mathsf{f}}(T, \delta)^2 \cdot \log(\mathsf{s\text{-}nec}_d(T, \delta)) \cdot n^3),$$

with $d := \mathsf{max}\{2, d(\sigma), d(\rho)\}$.

Proof. Let $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$. If we want to compute a solution of maximum (resp. minimum) weight, then we use the framework of Section 4.2.2 with $\mathbf{opt} = \mathsf{max}$ (resp. $\mathbf{opt} = \mathsf{min}$).

The first step of our algorithm is to compute, for each $x \in V(T)$, the sets $\mathcal{R}^d_{V_x}$, $\mathcal{R}^d_{\overline{V_x}}$ and a data structure to compute $\mathsf{rep}^d_{V_x}(X)$ and $\mathsf{rep}^d_{\overline{V_x}}(Y)$, for any $X \subseteq V_x$ and any $Y \subseteq \overline{V_x}$, in time $O(\log(\mathsf{s\text{-}nec}_d(T, \delta)) \cdot n^2)$. As T has $2n-1$ nodes, by Lemma 4.33, we can compute these sets and data structures in time $O(\mathsf{s\text{-}nec}_d(T, \delta) \cdot \log(\mathsf{s\text{-}nec}_d(T, \delta)) \cdot n^3)$.

For each node $x \in T$ and, for each $(R, R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}$, we define $\mathcal{A}_x[R, R'] \subseteq 2^{V_x}$ as follows

$$\mathcal{A}_x[R, R'] := \{X \subseteq V_x \,:\, X \equiv^d_{V_x} R \text{ and } X \cup R' \ (\sigma, \rho)\text{-dominates } V_x\}.$$

We deduce the following claim from the proof of Claim 4.41.

Claim 4.55.1. *For every internal node $x \in V(T)$ with a and b as children and $(R, R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}$, we have*

$$\mathcal{A}_x[R, R'] = \bigcup_{(A,A'),\ (B,B')\ d\text{-}(R,R')\text{-}compatible} \mathcal{A}_a[A, A'] \bigotimes \mathcal{A}_b[B, B'].$$

For each node x of $V(T)$, our algorithm will compute a table $\mathcal{D}_x$ that satisfies the following invariant.

Invariant. For every $(R, R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}$, the set $\mathcal{D}_x[R, R']$ is a subset of $\mathcal{A}_x[R, R']$ of size at most $\mathcal{N}_{\mathsf{f}}(T, \delta) \cdot \mathsf{nec}_1(V_x)^2$ that $(x, R')^{\mathsf{acy}}$-represents $\mathcal{A}_x[R, R']$.

Notice that by Definition of $(x, R')^{\mathsf{acy}}$-representativity, if the invariant holds for r, then $\mathcal{D}_r[\varnothing, \varnothing]$ contains a set X of maximum (or minimum) weight such that X is a (σ, ρ)-dominating set of G and $G[X]$ is a tree.

The algorithm is a usual bottom-up dynamic programming algorithm and computes for each node x of T the table $\mathcal{D}_x$.

Let x be a leaf of T with $V_x = \{v\}$. Observe that $\mathcal{A}_x[R, R'] \subseteq 2^{V_x} = \{\varnothing, \{v\}\}$. Thus, our algorithm can directly compute $\mathcal{A}_x[R, R']$ and set $\mathcal{D}_x[R, R'] := \mathcal{A}_x[R, R']$. In this case, the invariant trivially holds.

Now, take x an internal node of T with a and b as children such that the invariant holds for a and b. For each $(R, R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}$, the algorithm computes $\mathcal{D}_x[R, R'] := \mathsf{reduce}^{\mathsf{acy}}_{\mathsf{f}}(\mathcal{B}_x[R, R'])$, where the set $\mathcal{B}_x[R, R']$ is defined as follows

$$\mathcal{B}_x[R, R'] := \bigcup_{(A,A'),\ (B,B')\ d\text{-}(R,R')\text{-compatible}} \mathcal{D}_a[A, A'] \bigotimes \mathcal{D}_b[B, B'].$$

Similarly to the proof of Theorem 4.27, we deduce from Fact 4.46, Lemma 4.47, Claim 4.55.1 and Theorem 4.54, that $\mathcal{D}_x[R, R']$ is a subset and an $(x, R')^{\mathsf{acy}}$-representative set of $\mathcal{A}_x[R, R']$. By Theorem 4.54, we have $|\mathcal{D}_x[R, R']| \leq \mathcal{N}_{\mathsf{f}}(T, \delta) \cdot \mathsf{s\text{-}nec}_1(T, \delta)^2$.

Consequently, the invariant holds for x and by induction, it holds for all the nodes of T. The correctness of the algorithm follows.

Running Time. The running time of our algorithm is almost the same as the running time given in Theorem 4.55. The only difference is the factor $\mathcal{N}_{\mathsf{f}}(T, \delta)^2$ which is due to the following fact: by the invariant condition, for each (A, A') and (B, B'), the size of $\mathcal{D}_a[A, A'] \bigotimes \mathcal{D}_b[B, B']$ is at most $\mathcal{N}_{\mathsf{f}}(T, \delta)^2 \cdot \mathsf{s\text{-}nec}_1(T, \delta)^4$. □

By constructing for any graph G a graph G' such that the width measure of G' is linear in the width measure of G, and any optimum acyclic (σ, ρ)-dominating set of G corresponds to an optimum AC-(σ, ρ)-dominating set of G' and vice-versa, we obtain the following which allows for instance to compute a feedback vertex set in time $n^{O(c)}$, c the mim-width.

Theorem 4.56. *For each $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$, there exists an algorithm that, given an n-vertex graph G and a rooted layout (T, δ) of G, solves any* Acyclic (σ, ρ)-Dominating Set *problem in time*

$$O(\mathsf{s\text{-}nec}_d(T, \delta)^3 \cdot \mathsf{s\text{-}nec}_1(T, \delta)^{2(\omega+1)} \cdot \mathcal{N}_{\mathsf{f}}(T, \delta)^{O(1)} \cdot n^3),$$

with $d := \mathsf{max}\{2, d(\sigma), d(\rho)\}$.

Proof. Let $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$. Suppose that we want to compute a maximum acyclic (σ, ρ)-dominating set. The proof for computing a minimum acyclic (σ, ρ)-dominating set is symmetric.

The first step of this proof is to construct a $2n + 1$-vertex graph G' from G and a layout $(T^\star, \delta^\star)$ of G' from (T, δ) in time $O(n^2)$ such that $(T^\star, \delta^\star)$ respect the following inequalities:

1. for every $d \in \mathbb{N}^+$, $\mathsf{s\text{-}nec}_d(T^\star, \delta^\star) \leq (d + 1) \cdot \mathsf{s\text{-}nec}_d(T, \delta)$,
2. for every $\mathsf{f} \in \{\mathsf{mim}, \mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}\}$, $\mathsf{f}(T^\star, \delta^\star) \leq \mathsf{f}(T, \delta) + 1$.

The second step of this proof consists in showing how the algorithm of Theorem 4.55 can be modified to find a maximum acyclic (σ, ρ)-dominating set of G by running this modified algorithm on G' and $(T^\star, \delta^\star)$.

We construct G' as follows. Let β be a bijection from $V(G)$ to a set V^+ disjoint from $V(G)$. The vertex set of G' is $V(G) \cup V^+ \cup \{v_0\}$ with v_0 a vertex distinct from those in $V(G) \cup V^+$. We extend the weight function w of G to G' such that the vertices of $V(G)$ have the same weight as in G and the weight of the vertices in $V^+ \cup \{v_0\}$ is 0. Finally, the edge set of G' is defined as follows

$$E(G') := E(G) \cup \{\{v, \beta(v)\}, \{v_0, \beta(v)\} \ : \ v \in V(G)\}.$$

We now construct $\mathscr{L} = (T^\star, \delta^\star)$ from $\mathcal{L} := (T, \delta)$. We obtain $T^\star$ and $\delta^\star$ by doing the following transformations on T and δ:

- For each leaf ℓ of T with $\delta(\ell) = \{v\}$, we transform ℓ into an internal node by adding two new nodes a_ℓ and b_ℓ as its children such that $\delta^\star(a_\ell) = v$ and $\delta^\star(b_\ell) = \beta(v)$.
- The root of $T^\star$ is a new node r whose children are the root of T and a new node a_r with $\delta^\star(a_r) = v_0$.

In order to simplify the proof, we use the following notations.

For each node $x \in V(T^\star)$, we let $\overline{V_x^{\mathscr{L}}} := V(G') \setminus V_x^{\mathscr{L}}$ and, for each node $x \in V(T)$, we let $\overline{V_x^{\mathcal{L}}} := V(G) \setminus V_x^{\mathcal{L}}$.

Now, we prove that $(T^\star, \delta^\star)$ respects Inequalities (1) and (2). Let x be a node of $T^\star$. Observe that if $x \in V(T^\star) \setminus V(T)$, then the set $V_x^{\mathscr{L}}$ either contains one vertex or equals $V(G')$. Hence, in this case, the inequalities hold because $\mathsf{nec}_d(V_x^{\mathscr{L}}) \leq d$ for each $d \in \mathbb{N}^+$ and $\mathsf{f}(V_x^{\mathscr{L}}) \leq 1$ for each $\mathsf{f} \in \{\mathsf{mim}, \mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}\}$.

Now, assume that x is also a node of T. Hence, by construction, we have

$$V_x^{\mathscr{L}} = V_x^{\mathcal{L}} \cup \{\beta(v) \,:\, v \in V_x^{\mathcal{L}}\}.$$
$$\overline{V_x^{\mathscr{L}}} = \overline{V_x^{\mathcal{L}}} \cup \{\beta(v) \,:\, v \in \overline{V_x^{\mathcal{L}}}\} \cup \{v_0\}$$

Now, we prove Inequality (1). Let $d \in \mathbb{N}^+$. By construction of G' and $\mathscr{L}$, for each vertex $v \in V_x^{\mathcal{L}}$, we have $\beta(v) \in V_x^{\mathscr{L}}$ and

$$N_{G'}(v) \cap \overline{V_x^{\mathscr{L}}} = N_G(v) \cap \overline{V_x^{\mathcal{L}}}, \tag{4.7}$$
$$N_{G'}(\beta(v)) \cap \overline{V_x^{\mathscr{L}}} = \{v_0\}. \tag{4.8}$$

We deduce that, for every $X, Y \subseteq V_x^{\mathscr{L}}$, we have $X \equiv^d_{V_x^{\mathscr{L}}} Y$ if and only

- $X \cap V(G) \equiv^d_{V_x^{\mathcal{L}}} Y \cap V(G)$ and
- $\mathsf{min}(d, |N(v_0) \cap X|) = \mathsf{min}(d, |N(v_0) \cap Y|)$.

Similarly, we deduce that, for every $X, Y \subseteq \overline{V_x^{\mathscr{L}}}$, we have $X \equiv^d_{\overline{V_x^{\mathscr{L}}}} Y$ if and only if

- $X \cap V(G) \equiv^d_{\overline{V_x^{\mathcal{L}}}} Y \cap V(G)$ and
- $X \cap \{v_0\} = Y \cap \{v_0\}$.

Thus, we can conclude that $\mathsf{s\text{-}nec}_d(V_x^{\mathscr{L}}) \leq (d+1) \cdot \mathsf{s\text{-}nec}_d(V_x^{\mathcal{L}})$. Consequently, Inequality (1) holds.

We deduce Inequality (2) from Figure 4.3 describing the adjacency matrix between $V_x^{\mathscr{L}}$ and $\overline{V_x^{\mathscr{L}}}$.

Now, we explain how we modify the algorithm of Theorem 4.55 in order to find a maximum acyclic (σ, ρ)-dominating set of G by calling this algorithm on G'. For doing so, we modify the definition of the table $\mathcal{A}_x$, the invariant, and the computational steps of the algorithm of Theorem 4.55. The purpose of these modifications is to restrict the (σ, ρ)-domination to the vertices of $V(G)$. For doing so, we consider the set of nodes $S := V(T) \cup \{r, a_r\}$. Observe that, for every node x in S, there are no edges in $G[V_x^{\mathscr{L}}, \overline{V_x^{\mathscr{L}}}]$ between a vertex in $V(G)$ and a vertex in $V(G') \setminus V(G)$. This is not true for the nodes of $V(T^\star) \setminus S$. For this reason, our algorithm ignores the nodes in $V(T^\star) \setminus S$ and computes a table only for the nodes in S.

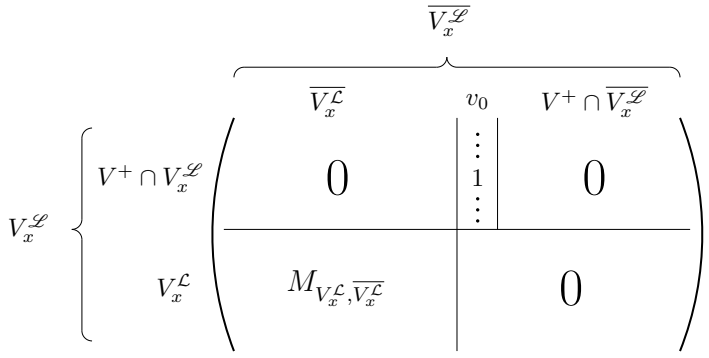


Figure 4.3 – The adjacency matrix between $V_x^{\mathscr{L}}$ and $\overline{V_x^{\mathscr{L}}}$.

For every $x \in S$ and every $(R, R') \in \mathcal{R}^d_{V_x^{\mathscr{L}}} \times \mathcal{R}^d_{\overline{V_x^{\mathscr{L}}}}$ we define $\mathcal{A}_x[R, R'] \subseteq 2^{V_x^{\mathscr{L}}}$ as follows

$$\mathcal{A}_x[R, R'] := \{X \subseteq V_x^{\mathscr{L}} \,:\, X \equiv^d_{V_x^{\mathscr{L}}} R \text{ and } (X \cup R') \cap V(G) \ (\sigma, \rho)\text{-dominates } V_x^{\mathscr{L}} \cap V(G)\}.$$

We claim that if G admits an acyclic (σ, ρ)-dominating set D, then there exists $D' \in \mathcal{A}_r[\varnothing, \varnothing]$ such that $D' \cap V(G) = D$ and $G'[D']$ is a tree. Let D be an acyclic (σ, ρ)-dominating set of G with $\mathsf{cc}(G[D]) = \{C_1, \ldots, C_t\}$. For each $i \in \{1, \ldots, t\}$, let v_i be a vertex in C_i. One easily checks that $G'[D \cup \{\beta(v_i) \,:\, 1 \leq i \leq t\} \cup v_0]$ is a tree. Moreover, by definition of $\mathcal{A}_r[\varnothing, \varnothing]$, for every $X \in \mathcal{A}_r[\varnothing, \varnothing]$, if $G[X]$ is a tree, then $X \cap V(G)$ is an acyclic (σ, ρ)-dominating set of G. Hence, if G admits an acyclic (σ, ρ)-dominating set, any $(r, \varnothing)^{\mathsf{acy}}$-representative set of $\mathcal{A}_r[\varnothing, \varnothing]$ contains a set X such that $X \cap V(G)$ is a maximum acyclic (σ, ρ)-dominating set of G.

For every node $x \in S$, we compute a table $\mathcal{D}_x$ satisfying the following invariant.

Invariant. For each node $x \in S$ and each $(R, R') \in \mathcal{R}^d_{V_x^{\mathscr{L}}} \times \mathcal{R}^d_{\overline{V_x^{\mathscr{L}}}}$, the set $\mathcal{D}_x[R, R']$ is a subset of $\mathcal{A}_x[R, R']$ of size at most $\mathcal{N}_{\mathsf{f}}(T^\star, \delta^\star) \cdot \mathsf{nec}_1(V_x^{\mathscr{L}})^2$ that $(x, R')^{\mathsf{acy}}$-represents $\mathcal{A}_x[R, R']$.

Before we explain how to compute the table $\mathcal{D}_x$, for each $x \in S$, we need the following fact and claim. We deduce the following fact from Lemma 4.39 and the fact that, for every node x in S, there are no edges in $G[V_x^{\mathscr{L}}, \overline{V_x^{\mathscr{L}}}]$ between a vertex in $V(G)$ and a vertex in $V(G') \setminus V(G)$.

Fact 4.57. *Let $x \in S$.*

Let $X \subseteq V_x^{\mathscr{L}}$ and $Y, R' \subseteq \overline{V_x^{\mathscr{L}}}$ such that $Y \equiv^d_{\overline{V_x^{\mathscr{L}}}} R'$. Then $(X \cup R') \cap V(G)$ (σ, ρ)-dominates $V_x^{\mathscr{L}} \cap V(G)$ if and only if $(X \cup Y) \cap V(G)$ (σ, ρ)-dominates $V_x^{\mathscr{L}} \cap V(G)$.

We deduce the following claim from Fact 4.57 and Lemma 4.41.

Claim 4.57.1. *Let $x \in S \setminus \{a_r\}$ such that x is not a leaf in T. Let a and b be the children of x in $T^\star$. For every $(R, R') \in \mathcal{R}^d_{V_x^{\mathscr{L}}} \times \mathcal{R}^d_{\overline{V_x^{\mathscr{L}}}}$, we have*

$$\mathcal{A}_x[R, R'] = \bigcup_{(A,A'),\ (B,B') \ d\text{-}(R,R')\text{-compatible}} \mathcal{A}_a[A, A'] \bigotimes \mathcal{A}_b[B, B'].$$

The algorithm starts by computing the table $\mathcal{D}_x$ for each node $x \in S$ such that $x = a_r$ or x is a leaf of T. Since $|V_x^{\mathscr{L}}| \leq 2$, our algorithm directly computes $\mathcal{A}_x[R, R']$ and set $\mathcal{D}_x[R, R'] := \mathcal{A}_x[R, R']$ for every $(R, R') \in \mathcal{R}^d_{V_x^{\mathscr{L}}} \times \mathcal{R}^d_{\overline{V_x^{\mathscr{L}}}}$.

For the other nodes our algorithm computes the table $\mathcal{D}_x$ exactly as the algorithm of Theorem 4.55.

The correctness of this algorithm follows from Theorem 4.55 and Claim 4.57.1. By Theorem 4.55, the running time of this algorithm is

$$O(\mathsf{s\text{-}nec}_2(\mathscr{L})^3 \cdot \mathsf{s\text{-}nec}_1(\mathscr{L})^{2(\omega+1)} \cdot \mathcal{N}_{\mathsf{f}}(\mathscr{L})^2 \cdot n^3).$$

We deduce the running time in function of $\mathcal{L}$ from Inequalities (1) and (2).

□

4.2.5 Conclusion

We have simplified and generalized the rank-based approach of [9] to work with the d-neighbor-equivalence relation of [18]. As a result, we provide deterministic algorithms for a wide range of connectivity problems running in time $\mathsf{s\text{-}nec}_d(T,\delta)^{O(1)} \cdot n^{O(1)}$ where (T,δ) is a given layout and d is a constant which depends on the problem. From the upper-bounds presented in Theorem 2.33, we obtain parameterized algorithms with parameters and running times given in Table 4.2.

We have extended our framework in order to solve any AC-(σ,ρ) Dominating Set problem and any Acyclic (σ,ρ) Dominating Set problem. This includes well-known problems such as Maximum Induced Tree, Longest Induced Path, and Maximum Induced Forest. We obtain algorithms whose running times match those described in Table 4.2. But, even if our algorithms rely heavily on the d-neighbor-width on the input rooted layout, some specific parts of the running time analysis use the properties of other parameters. We leave open the following question.

Open Question 4.58. *Does there exists a constant $c \in \mathbb{N}^+$ and an algorithm, that given an n-vertex graph G and a rooted layout $\mathcal{L}$ of G, finds a maximum induced tree (or a longest induced path) in time $\mathsf{s\text{-}nec}_c(T,\delta)^{O(1)} \cdot n^{O(1)}$?*

Concerning mim-width, we provide unified polynomial-time algorithms for the considered problems and the graph classes of bounded mim-width presented in Section 2.4. Notice that we also generalize one of the results from [109] proving that the Connected Vertex Cover problem is solvable in polynomial time for circular arc graphs.

As explained in Section 2.6.2, the algorithmic results we obtain for clique-width are asymptotically optimal under the Exponential Time Hypothesis (ETH). However, this is not the case for the other parameters.

Finally, Fomin et al. [62] have shown that we can use fast computation of representative sets in matroids to obtain deterministic $2^{O(k)} \cdot n^{O(1)}$ time algorithms parameterized by tree-width for many connectivity problems. Let us briefly explain the ideas of [62] (we refer to [37] for an introduction on matroid and their use in theoretical computer science). Given a matroid $\mathcal{M}$ and a family $\mathcal{A}$ of independent sets of size p, we say that $\mathcal{A}' \subseteq \mathcal{A}$ *q-represents* $\mathcal{A}$ if for every set B of size q, if there exists $A \in \mathcal{A}$ such that $A \cup B$ is an independent set of $\mathcal{M}$, then there exits $A' \in \mathcal{A}'$ such that $A' \cup B$ is also an independent set. Fomin et al. proved the following theorem on the computation of q-representative sets.

Theorem 4.59 ([62])**.** *Let $\mathcal{M}$ be a linear matroid of rank $p+q=k$ given with its representation matrix $A_{\mathcal{M}}$ over a field $\mathbb{F}$. Given a set $\mathcal{A}$ of independent sets of size p, we can compute a q-representative set $\mathcal{B} \subseteq \mathcal{A}$ of $\mathcal{A}$ with $|\mathcal{B}| \leq \binom{p+q}{p}$ in $O(\binom{p+q}{p} \cdot |\mathcal{A}| \cdot p^\omega + |\mathcal{A}| \cdot \binom{p+q}{p}^{\omega-1})$ operations over $\mathbb{F}$.*

Fomin et al. used this theorem in [62] to design efficient parameterized algorithms and in particular $2^{O(\mathsf{tw}(G))} \cdot n$ time algorithms for Steiner Tree. Is this approach also generalizable to d-neighbor-width? Can it be of any help for answering Question 2.59?

4.3 Cut & Count Approach on Graphs with Structured Neighborhood

In this section, we combine the *Cut & Count* approach of [38] with the d-neighborhood equivalence relation (defined in Section 1.3). We obtain a one-sided error Monte Carlo algorithm with false negatives[7] that, given a graph G and an integer k, decides whether G admits a connected (σ, ρ)-dominating set of size k in time $O(\mathsf{s\text{-}nec}_d(\mathcal{L})^{O(1)} \cdot n^{O(1)})$ with $\mathcal{L}$ a given rooted layout.

Let us explain, briefly, the Cut & Count approach with a concrete example: the unweighted decision variant of Node-Weighted Steiner Tree. Let G be a graph and $K \subseteq V(G)$ be a set of *terminals*. We want to decide if G admits a *solution* of size k for some $k \in [n]$. In this case, a solution is a set $X \subseteq V(G)$ such that $K \subseteq X$ and $G[X]$ is connected. For doing so, the Cut & Count approach reduces this decision problem to its counting modulo 2 variant, i.e., finds the parity of the number of solutions of size k. A priori, this variant does not seem easier to solve, but working modulo 2 allows to use cancellation tricks. Let us explain this reduction and how we solve this counting modulo 2 variant.

The reduction. The idea behind the reduction is to assign to each vertex v of G a weight $\mathsf{w}(v) \in [N]$ uniformly and independently at random for some integer $N \in \mathbb{N}$. Intuitively, if N is large, then there is a good probability p that there exists $W \in [k \cdot N]$ such that there exists a unique solution X of size k and weight W; this implies that the number of solutions of size k and weight W is odd. In fact, setting $N = 2n$ is enough to have $p \geq 1/2$ (see Lemma 4.60).

Counting modulo 2 variant. Given the weight function w from the reduction, what we have to do now is to compute, for each $W \in \{1, \ldots, n \cdot N\}$, the parity of the set $\mathcal{S}_W$ of solutions of size k and weight W. In order to compute the parity of $\mathcal{S}_W$, we proceed into two parts:

- *The Cut part:* relax the connectivity constraint and consider the set $\mathcal{R}_W$ of possibly disconnected candidate solutions, that is consider the set $\mathcal{R}_W := \{X \subseteq V(G) : |X| = k, \mathsf{w}(X) = W \text{ and } K \subseteq X\}$. Furthermore, fix a vertex in $v_0 \in K$ and consider the set $\mathcal{C}_W$ of all pairs $(X, (X_1, X_2))$ where $X \in \mathcal{R}$ and (X_1, X_2) is a consistent cut[8] of X with $v_0 \in X_1$.
- *The Count part:* prove that $\mathcal{C}_W$ and $\mathcal{S}_W$ have the same parity and compute the parity of $\mathcal{C}_W$ on a rooted layout by a dynamic programming algorithm (in [38] they do it on a tree-decomposition in time $2^{O(\mathsf{tw}(G))} \cdot n^{O(1)}$). In this case, proving that $\mathcal{C}_W$ and $\mathcal{S}_W$ have the same parity is not hard. Indeed, for every $X \in \mathcal{R}_W$, the number of consistent cuts C of X such that $(X, C) \in \mathcal{C}_W$ equals $2^{|\mathsf{cc}(G[X])|-1}$ since every connected component of $G[X]$ can be in both sides of a consistent cut at the exception of the connected component containing v_0. Hence, the number of consistent cuts C of X such that $(X, C) \in \mathcal{C}_W$ is odd if and only if $X \in \mathcal{S}_W$.

In the following, we will prove that, given an n-vertex graph G with a rooted layout $\mathcal{L}$ and $k \in \mathbb{N}$, we can use this approach to design a Monte Carlo algorithm to decide whether G admits a connected (σ, ρ)-dominating set of size k in time $\mathsf{s\text{-}nec}_d(\mathcal{L})^{O(1)} \cdot n^{O(1)}$ with $d = \mathsf{max}\{1, d(\sigma), d(\rho)\}$. We will only consider connected (σ, ρ)-dominating sets but, by using the same ideas of Section 4.2, the same algorithm works for deciding whether a graph has a connected co-(σ, ρ)-dominating set of size k or for the unweighted decision variant of Node-Weighted Steiner Tree.

Let G be an n-vertex graph, $k \in \mathbb{N}$, and let (T, δ) be a fixed rooted layout of G with the root of T denoted by r. Moreover, let (σ, ρ) be a pair of non-empty finite or co-finite subsets

[7] An algorithm for decision problems that is always correct when it outputs **yes**.
[8] Defined in Subsection 4.2.1.

of $\mathbb{N}$ and let $d := \mathsf{max}\{1, d(\sigma), d(\rho)\}$. To use the modulo 2 cancellation trick, we need to guess one vertex $v_0 \in V(G)$ that belongs to a solution of size k. We can do this by going through the n possibilities. Let v_0 be a fixed vertex in $V(G)$. In the following, we explain how to design a Monte Carlo algorithm that decides whether G admits a connected (σ, ρ)-dominating set of size k that contains v_0.

As in [38], we use the Isolation Lemma from [111] for the reduction.

Lemma 4.60 ([111]). *Let $\mathcal{S} \subseteq 2^U$ be a set family over a universe U with $|\mathcal{S}| > 1$. For each $u \in U$, choose a weight $\mathsf{w}(u) \in \{1, \ldots, N\}$ uniformly and independently at random. Then, the probability that there exists a unique $S \in \mathcal{S}$ with $\mathsf{w}(S) = \mathsf{min}_{S' \in \mathcal{S}} \mathsf{w}(S')$ is at least $1 - \frac{|U|}{N}$.*

To use this lemma, we choose, for each vertex v of G, a weight $\mathsf{w}(v) \in \{1, \ldots, 2n\}$ uniformly at random. Then we consider, for each $W \in [2n^2]$, the set $\mathcal{S}_{v_0,W}$ of all connected (σ, ρ)-dominating sets of size k and weight W that contains v_0. By Lemma 4.60, if there exists a connected (σ, ρ)-dominating set of size k that contains v_0, then the probability that there exists $W \in [2n^2]$ such that $|\mathcal{S}_{v_0,W}|$ is odd is at least $\frac{1}{2}$. Consequently, from an algorithm that computes the parity of $|\mathcal{S}_{v_0,W}|$ for each W, one can design a one sided Monte Carlo algorithm that decides whether G admits a connected (σ, ρ)-dominating set of size k that contains v_0. This Monte Carlo algorithm checks whether there exists W such that $|\mathcal{S}_{v_0,W}|$ is odd, if such W exists, then it returns **yes**, otherwise, it returns **no**. This algorithm cannot give false positives and may give false negatives with probability at most $1/2$.

In the following, we explain how we compute, for each W, the parity of $|\mathcal{S}_{v_0,W}|$. For all $W \in [2n \cdot k]$, we consider $\mathcal{R}_{v_0,W} \supseteq \mathcal{S}_{v_0,W}$ the set of all the (σ, ρ) dominating sets of cardinality k and weight W that contain v_0. Moreover, we consider also

$$\mathcal{C}_{v_0,W} := \{(X, (X_1, X_2)) \in R_{v_0,W} \times \mathsf{cuts}(X) \,:\, v_0 \in X_1\}.$$

By Fact 4.30, for every $X \in \mathcal{R}_{v_0,W}$, the number of consistent cuts C of X such that $(X, C) \in \mathcal{C}_{v_0,W}$ is $2^{|\mathsf{cc}(G[X])|-1}$. Consequently, we have the following fact.

Fact 4.61. *For every W, $|\mathcal{C}_{v_0,W}|$ and $|\mathcal{S}_{v_0,W}|$ have the same parity.*

Hence, if $|\mathcal{C}_{v_0,W}|$ is odd for some W, we can confirm that G admits a connected (σ, ρ)-dominating set of cardinality k.

We now explain how to compute $|\mathcal{C}_{v_0,W}|$ by a bottom-up dynamic programming algorithm on (T, δ). At each node x of T, we compute a table $\#\mathcal{A}_{x,v_0}$ whose set of indices, denoted by $\mathbb{I}_x$, is $\mathbb{I}_x := \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}} \times \mathcal{R}^1_{V_x} \times \mathcal{R}^1_{V_x} \times [k] \times [2n \cdot k]$. For each $I \in \mathbb{I}_x$, the entry $\#\mathcal{A}_{x,v_0}[I]$ corresponds to the cardinality of the set $\mathcal{A}_{x,v_0}[I]$ whose definition is the following.

Definition 4.62. *Let $x \in V(T)$, for all $(R, R', R_1, R_2, i, W) \in \mathbb{I}_x$.*

We define $\mathcal{A}_{x,v_0}[R, R', R_1, R_2, i, W]$ as the set of all $(X, (X_1, X_2)) \in 2^{V_x} \times \mathsf{cuts}(X)$ satisfying the following conditions:

1. *$|X| = i$ and $\mathsf{w}(X) = W$,*
2. *$X \equiv^d_{V_x} R$,*
3. *if $v_0 \in V_x$, then $v_0 \in X_1$,*
4. *$(X \cup R')$ (σ, ρ)-dominates V_x,*
5. *$X_1 \equiv^1_{V_x} R_1$ and $X_2 \equiv^1_{V_x} R_2$.*

It is straightforward to check that, for every $W \in [2n \cdot k]$, we have $\#\mathcal{A}_r[\varnothing, \varnothing, \varnothing, \varnothing, k, W] = |\mathcal{C}_{v_0,W}|$. Let us explain how we compute the entries of $\#\mathcal{A}_{x,v_0}$.

Computing $\#\mathcal{A}_{x,v_0}$ when x is an internal node of T with children a, b. We start by defining a notion of compatibility between the indices in $\mathbb{I}_x, \mathbb{I}_a$ and $\mathbb{I}_b$.

Definition 4.63. *Let $I_x = (R, R', R_1, R_2, i, W) \in \mathbb{I}_x$. For $I_a = (A, A', A_1, A_2, i_a, W_a) \in \mathbb{I}_a$ and $I_b = (B, B', B_1, B_2, i_b, W_b) \in \mathbb{I}_b$, we say that (I_a, I_b) is I_x-compatible if*

(a) $W = W_a + W_b$ *and* $i = i_a + i_b$,

(b) $R \equiv^d_{V_x} A \cup B$,

(c) $A' \equiv^d_{V_a} R' \cup B'$ *and* $B' \equiv^d_{V_b} R' \cup A'$,

(d) $A_1 \cup B_1 \equiv^1_{V_x} R_1$ *and* $A_2 \cup B_2 \equiv^1_{V_x} R_2$,

(e) $N(A_1) \cap B_2 = \varnothing$ *and* $N(A_2) \cap B_1 = \varnothing$.

The following lemma provides an equality between the entries of $\#\mathcal{A}_{x,v_0}$ and those of $\#\mathcal{A}_{a,v_0}$ and $\#\mathcal{A}_{b,v_0}$.

Lemma 4.64. *For all $I_x = (R, R', R_1, R_2, i, w) \in \mathbb{I}_x$, we have*

$$\#\mathcal{A}_{x,v_0}[I_x] = \sum_{(I_a, I_b) \text{ is } I_x\text{-compatible}} \#\mathcal{A}_{a,v_0}[I_a] \cdot \#\mathcal{A}_{b,v_0}[I_b]$$

Proof. This lemma is implied by the following two claims.

Claim 4.64.1. *Let $I_x = (R, R', R_1, R_2, i, w) \in \mathbb{I}_x$, $I_a = (A, A', A_1, A_2, i_a, W_a) \in \mathbb{I}_a$ and $I_b = (B, B', B_1, B_2, i_b, W_b) \in \mathbb{I}_b$ such that (I_a, I_b) is I_x-compatible. For all $(X, (X_1, X_2)) \in \mathcal{A}_{a,v_0}[I_a]$ and $(Y, (Y_1, Y_2)) \in \mathcal{A}_{b,v_0}[I_b]$, we have $(X \cup Y, (X_1 \cup Y_1, X_2 \cup Y_2)) \in \mathcal{A}_{x,v_0}[I_x]$.*

Proof. Let $(X, (X_1, X_2)) \in \mathcal{A}_{a,v_0}[I_a]$ and $(Y, (Y_1, Y_2)) \in \mathcal{A}_{b,v_0}[I_b]$. We start by proving that $(X \cup Y, (X_1 \cup Y_1, X_2 \cup Y_2))$ checks all Conditions (1)-(5) of Definition 4.62. For doing so, we use Conditions (a)-(e) of Definition 4.63.

First, observe that thanks to Fact 4.32 and the facts that $V(G) = V_a \uplus V_b \uplus \overline{V_x}$, we have (e1) $X \equiv^d_{V_x} A$, (e2) $Y \equiv^d_{V_x} B$, (e3) $Y \equiv^d_{V_a} B$, (e4) $X_1 \equiv^1_{V_x} A_1$, and (e5) $Y_1 \equiv^1_{V_x} B_1$.

Condition (1) is trivially satisfied since we have, by Condition (a), $|X| + |Y| = i_a + i_b = i$ and $\mathsf{w}(X) + \mathsf{w}(Y) = W_a + W_b = W = \mathsf{w}(X \cup Y)$.

Condition (2) is also satisfied. Indeed, from Condition (b), we have $R \equiv^d_{V_x} A \cup B$. Hence, we deduce from (e1) and (e2) that $X \cup Y \equiv^d_{V_x} R$, i.e., Condition (2) is satisfied.

Condition (3) is due to the fact that if $v_0 \in V_x$, then either $v_0 \in V_a$ or $v_0 \in V_b$. Thus, in both cases, we have $v_0 \in X_1 \cup Y_1$ by definition of $\mathcal{A}_{a,v_0}[I_a]$ and $\mathcal{A}_{b,v_0}[I_b]$.

In order to prove that Condition (4) is satisfied, we have to prove that $X \cup Y \cup R'$ (σ, ρ)-dominates V_x. Thanks to Condition (c), we have $A' \equiv^d_{V_a} R' \cup B$. We deduce from (e3) that $A' \equiv^d_{V_a} R' \cup Y$. As $(X, (X_1, X_2)) \in \mathcal{A}_{a,v_0}[I_a]$, we have $X \cup A'$ (σ, ρ)-dominates V_a. Since $A' \equiv^d_{V_a} R' \cup Y$, by Lemma 4.39, we conclude that $X \cup Y \cup R'$ (σ, ρ)-dominates V_a. Symmetrically, we can prove that $X \cup Y \cup R'$ (σ, ρ)-dominates V_b. Consequently, $X \cup Y \cup R'$ (σ, ρ)-dominates V_x, i.e., Condition (4) is satisfied.

In order to prove Condition (5), we have to prove that $X_1 \cup Y_1 \equiv^1_{V_x} R_1$ and $X_2 \cup Y_2 \equiv^1_{V_x} R_2$. By Condition (d), we have $A_1 \cup B_1 \equiv^1_{V_x}$. From (e4) and (e5), we can conclude that $X_1 \cup Y_1 \equiv^1_{V_x} R_1$. Symmetrically, we can prove that $X_2 \cup Y_2 \equiv^1_{V_x} R_2$ and thus Condition (5) is satisfied.

It remains to prove $(X_1 \cup Y_1, X_2 \cup Y_2) \in \mathsf{cuts}(X \cup Y)$. By definition of $\mathcal{A}_{a,v_0}[I_a]$ and $\mathcal{A}_{b,v_0}[I_b]$, we have $(X_1, X_2) \in \mathsf{cuts}(X)$ and $(Y_1, Y_2) \in \mathsf{cuts}(Y)$. Thus, by Fact 4.31, in order to prove that $(X_1 \cup Y_1, X_2 \cup Y_2) \in \mathsf{cuts}(X \cup Y)$, it is sufficient to prove that $N(X_1) \cap Y_2 = \varnothing$ and

$N(X_2) \cap Y_1 = \varnothing$. We know that $X_1 \equiv^1_{V_a} A_1$ and thus $N(X_1) \cap \overline{V_a} = N(A_1) \cap \overline{V_a}$. Symmetrically, we have $N(Y_2) \cap \overline{V_b} = N(B_2) \cap \overline{V_b}$. By Condition (e), we have $N(A_1) \cap B_2 = \varnothing$. Since $B_2 \subseteq \overline{V_a}$ and $X_1 \subseteq \overline{V_b}$, we conclude that $N(A_1) \cap B_2 = N(X_1) \cap Y_2 = \varnothing$. It follows that $(X_1 \cup Y_1, X_2 \cup Y_2) \in \mathsf{cuts}(X \cup Y)$.

□

Claim 4.64.2. *Let $I_x \in \mathbb{I}_x$ and $(X, (X_1, X_2)) \in \mathcal{A}_{x,v_0}[I_x]$. There exist an unique $I_a \in \mathbb{I}_a$ and an unique $I_b \in \mathbb{I}_b$ such that (I_a, I_b) is I_x-compatible and for all $i \in \{a, b\}$, we have*

$$(X \cap V_i, (X_1 \cap V_i, X_2 \cap V_i)) \in \mathcal{A}_i[I_i]$$

Proof. Let $I_a := (A, A', A_1, A_2, i_a, W_a) \in \mathbb{I}_a$ such that:

- $i_a = |X \cap V_a|$, $W_a = \mathsf{w}(X \cap V_a)$,
- $A = \mathsf{rep}^d_{V_a}(X \cap V_a)$, $A' = \mathsf{rep}^d_{V_a}(R' \cup X \cap V_b)$,
- $A_1 = \mathsf{rep}^1_{V_a}(X_1 \cap V_a)$, and $A_2 = \mathsf{rep}^1_{V_a}(X_2 \cap V_a)$.

We claim that $(X \cap V_a, (X_1 \cap V_a, X_2 \cap V_a)) \in \mathcal{A}_{a,v_0}[I_a]$.

By Fact 4.31, $(X_1, X_2) \in \mathsf{cuts}(X)$ implies that $(X_1 \cap V_a, X_2 \cap V_a) \in \mathsf{cuts}(X \cap V_a)$.

By construction of I_a, $(X \cap V_a, (X_1 \cap V_a, X_2 \cap V_a))$ satisfies Conditions (1), (2), (5) of Definition 4.62 w.r.t. $\mathcal{A}_{a,v_0}[I_a]$.

Since $(X, (X_1, X_2)) \in \mathcal{A}_{x,v_0}[I_x]$, if $v_0 \in V_a$, then $v_0 \in V_x$ and we have $v_0 \in X_1 \cap V_a$, that is Condition (3) is satisfied.

It remains to prove Condition (4), i.e., $(X \cap V_a) \cup A'$ (σ, ρ)-dominates V_a. We know that $X \cup R'$ (σ, ρ)-dominates V_x. By construction, we have $A' \equiv^d_{V_a} R' \cup X \cap V_b$. Thus, by Lemma 4.39, we have $(X \cap V_a) \cup A'$ (σ, ρ)-dominates V_a. We can conclude that $(X \cap V_a, (X_1 \cap V_a, X_2 \cap V_a)) \in \mathcal{A}_{a,v_0}[I_a]$.

By Definition 4.62, it is straightforward to check that I_a is the only index of $\mathbb{I}_a$ such that $(X \cap V_a, (X_1 \cap V_a, X_2 \cap V_a)) \in \mathcal{A}_{a,v_0}[I_a]$. Symmetrically, we can construct $I_b := (B, B', B_1, B_2, i_b, w_b) \in \mathbb{I}_b$ such that I_b is the unique element of $\mathbb{I}_b$ so that $(X \cap V_b, (X_1 \cap V_b, X_2 \cap V_b)) \in \mathcal{A}_{b,v_0}[I_b]$.

It remains to prove that (I_a, I_b) is I_x-compatible. By the definitions of I_a and I_b, Conditions (a) and (c) are satisfied. Now, we prove Condition (b), i.e., $A \cup B \equiv^d_{V_x} X$. By construction of I_a, we have $A \equiv^d_{V_a} X \cap V_a$. Since $V_a \subseteq V_x$, by Fact 4.32, we deduce that $A \equiv^d_{V_x} X \cap V_a$. Symmetrically, we can prove that $B \equiv^d_{V_x} X \cap V_b$. We can conclude that $A \cup B \equiv^d_{V_x} X$, i.e., Condition (b) is satisfied. With the same arguments, we can prove that Condition (d) is also satisfied.

It remains to prove Condition (e), i.e., $N(A_1) \cap B_2 = \varnothing$ and $N(A_2) \cap B_1 = \varnothing$. By symmetry, it is enough to prove that $N(A_1) \cap B_2 = \varnothing$. Since $(X_1, X_2) \in \mathsf{cuts}(X)$, we know that $X_1 \cap N(X_2) = \varnothing$ and in particular $X_1 \cap V_a \cap N(X_2 \cap V_b) = \varnothing$. As $A_1 \equiv^1_{V_a} X_1 \cap V_a$, we have $N(A_1) \cap \overline{V_a} = N(X_1 \cap V_a) \cap \overline{V_a}$. Symmetrically, we have $N(B_2) \cap \overline{V_b} = N(X_2 \cap V_b) \cap \overline{V_b}$. Thus, since $A_1 \subseteq \overline{V_b}$ and $B_2 \subseteq \overline{V_a}$, we can conclude that $N(X_1 \cap V_a) \cap (X_2 \cap V_b) = N(A_1) \cap B_2 = \varnothing$. □

□

We are now ready to prove the main theorem of this section.

Theorem 4.65. *There exists a Monte-Carlo algorithm that, given an n-vertex graph G, a rooted layout (T, δ) of G and $k \in [k]$, decides whether G admits a connected (σ, ρ)-dominating of size k in time $O(\mathsf{s\text{-}nec}_d(T, \delta)^3 \cdot \mathsf{s\text{-}nec}_1(T, \delta)^4 \cdot k^2 \cdot n^4)$ with $d = \mathsf{max}\{1, d(\sigma), d(\rho)\}$. The algorithm cannot give false positives and may give false negatives with probability at most 1/2.*

Proof. The first step of our algorithm is to compute, for each $x \in V(T)$, the sets $\mathcal{R}^d_{V_x}$, $\mathcal{R}^1_{V_x}$, $\mathcal{R}^d_{\overline{V_x}}$, $\mathcal{R}^1_{\overline{V_x}}$ and a data structure to compute $\mathsf{rep}^d_{V_x}(X)$, $\mathsf{rep}^1_{V_x}$, $\mathsf{rep}^d_{\overline{V_x}}(X)$, and $\mathsf{rep}^1_{\overline{V_x}}$ for each $X \subseteq V_x$ in time $O(\log(\mathsf{s\text{-}nec}_d(T,\delta)) \cdot n^2)$. As T has $2n-1$ nodes, by Lemma 4.33, we can compute these sets and data structures in time $O(\mathsf{s\text{-}nec}_d(T,\delta) \cdot \log(\mathsf{s\text{-}nec}_d(T,\delta)) \cdot n^3)$.

Then our algorithm choose, for each vertex v of G, a weight $\mathsf{w}(v) \in \{1,\ldots,2n\}$ uniformly at random. For every choice of vertex v_0, our algorithm do the following. We do a bottom-up traversal of T and at each node we compute $\#\mathcal{A}_{x,v_0}[I_x]$ for each $I_x \in \mathbb{I}_x$. For the leaves of T, we can directly compute $\mathcal{A}_{x,v_0}[I_x]$ and $\#\mathcal{A}_{x,v_0}[I_x] = |\mathcal{A}_{x,v_0}[I_x]|$ for each $I_x \in \mathbb{I}_x$. When x is an internal node of T with children a and b, we can compute the entries of $\#\mathcal{A}_{x,v_0}$ by doing the following:

- initialize each entry of $\#\mathcal{A}_{x,v_0}$ to 0,
- for each $(I_x, I_a, I_b) \in \mathbb{I}_x \times \mathbb{I}_a \times \mathbb{I}_b$, such that (I_a, I_b) is I_x-compatible, update $\#\mathcal{A}_{x,v_0}[I_x] := (\#\mathcal{A}_{a,v_0}[I_a] \cdot \#\mathcal{A}_{b,v_0}[I_b]) + \#\mathcal{A}_{x,v_0}[I_x]$.

We claim that we can compute the entries of $\#\mathcal{A}_{x,v_0}$ in time $O(\mathsf{s\text{-}nec}_d(T,\delta)^3 \cdot \mathsf{s\text{-}nec}_1(T,\delta)^4 \cdot k^4 \cdot n^2)$. In order to prove that it is sufficient to prove that there are at most $N := \mathsf{s\text{-}nec}_d(T,\delta)^3 \cdot \mathsf{s\text{-}nec}_1(T,\delta)^4 \cdot k^4 \cdot n^2$ tuples (I_x, I_a, I_b) such that (I_a, I_b) is I_x-compatible and that we enumerate these N tuples in time $O(N)$. First, observe that, for every $(A, B, R') \in \mathcal{R}^d_{V_a} \times \mathcal{R}^d_{V_b} \times \mathcal{R}^d_{\overline{V_x}}$, there is only one tuple $(A', B', R) \in \mathcal{R}^d_{\overline{V_a}} \times \mathcal{R}^d_{\overline{V_b}} \times \mathcal{R}^d_{V_x}$ such that Conditions (b) and (c) of Definition 4.63 are satisfied (see the proof of Theorem 4.42). Moreover, for $(A_1, A_2) \in \mathcal{R}^1_{V_a} \times \mathcal{R}^1_{V_a}$ and $(B_1, B_2) \in \mathcal{R}^1_{V_b} \times \mathcal{R}^1_{V_b}$, there exists only one $(R_1, R_2) \in \mathcal{R}^1_{V_x} \times \mathcal{R}^1_{V_x}$ such that Condition (d) of Definition 4.63 is satisfied. It follows that there are at most N tuples (I_x, I_a, I_b) such that (I_a, I_b) is I_x-compatible and it is easy to check that we can enumerate these N tuples in time $O(N)$.

Assume now that $\#\mathcal{A}_{x,v_0}$ has been computed for every $x \in V(T)$ and every $v_0 \in V(G)$. If for every v_0 and $W \in [2n \cdot k]$, $\#\mathcal{A}_{r,v_0}[\varnothing, \varnothing, \varnothing, \varnothing, k, W]$ is even with r the root of T, then our algorithm returns **no**, otherwise it returns **yes**.

We claim that the algorithm we have presented is a Monte Carlo algorithm that decides whether G admits a connected (σ,ρ)-dominating set of size k. Moreover, by Lemma 4.60, this algorithm cannot give false positives and may give false negatives with probability at most $\frac{1}{2}$.

Assume that G admits a (σ,ρ)-dominating set X of size k. Let v_0 be a vertex of X. Thus, we have $\bigcup_{W \in [2n\cdot k]} \mathcal{S}_{v_0,W} \neq \varnothing$. By construction of w and Lemma 4.60, the probability that there exists $W \in [2n \cdot k]$ such that $|\mathcal{S}_{v_0,W}|$ is odd is at least $\frac{1}{2}$. Moreover, by Definition 4.62, for each $W \in [2n \cdot k]$, we have $\mathcal{C}_{v_0,W} = \mathcal{A}_{r,v_0}[\varnothing, \varnothing, \varnothing, \varnothing, k, W]$. Furthermore, by Fact 4.61, $|\mathcal{S}_{v_0,W}|$ and $|\mathcal{C}_{v_0,W}|$ have the same parity for every $W \in [2n \cdot k]$. Consequently, the probability that our algorithm returns **yes** is at least $\frac{1}{2}$. On the other hand, if G does not admit a (σ,ρ)-dominating set X of size k, then $\mathcal{S}_{v_0,W} = \varnothing$ for every $v_0 \in V(G)$ and $W \in [2n \cdot k]$. In this case, our algorithm always returns **no**. Since T has $2n-1$ nodes, we conclude that the running time of this algorithm is $O(\mathsf{s\text{-}nec}_d(T,\delta)^3 \cdot \mathsf{s\text{-}nec}_1(T,\delta)^4 \cdot k^4 \cdot n^4)$. □

4.4 An Optimal XP Algorithm for Hamiltonian Cycle parameterized by Clique-width

In this section, we prove that there exists an algorithm solving Hamiltonian Cycle in time $n^{O(k)}$, when a clique-width k-expression is given. Specifically, we prove the following.

Theorem 4.66. *There exists an algorithm that, given an n-vertex graph G and a k-expression of G, solves* Hamiltonian Cycle *in time $O(n^{2k+5} \cdot 2^{2k(\log_2(k)+1)} \cdot k^3 \log_2(nk))$.*

Our algorithm is a dynamic programming one whose steps are based on the k-labeled graphs H arising in the k-expression of G. Observe that the edges of a Hamiltonian cycle of G which belong to $E(H)$ induce either a Hamiltonian cycle or a collection of vertex-disjoint paths in G covering H. Consequently, we define a partial solution as a set of edges of H which induces a collection of paths (potentially empty) covering H. As in [49], with each partial solution $\mathcal{P}$, we associate an auxiliary multigraph such that its vertices correspond to the labels of H and each edge $\{i, j\}$ corresponds to a maximal path induced by $\mathcal{P}$ with end-vertices labeled i and j.

Since H is a k-labeled graph arising in a k-expression of G, we have that two vertices x and y with the same label in H have the same neighborhood in $G - E(H)$ (the graph G without the edges of H). It follows that the endpoints of a path in a partial solution are not important and what matters are the labels of these endpoints. As a result, two partial solutions with the same auxiliary multigraph are equivalent, i.e., if one is contained in a Hamiltonian cycle, then the other also. From these observations, one easily deduces the $n^{O(k^2)}$-time algorithm, due to Espelage, Gurski, and Wanke [49], because there are at most n possible paths between two label classes and thus there are at most $n^{\mathcal{O}(k^2)}$ possible auxiliary graphs.

To obtain our $n^{O(k)}$-time algorithm, we refine the above equivalence relation. We define that two partial solutions are equivalent if their auxiliary graphs have the same connected components, and the paths they induce have the same number of endpoints labeled i, for all labels i. The motivation is the following. When we have a partial solution $\mathcal{P}$ and a set of edges $\mathcal{Q} \subseteq E(G) \setminus E(H)$ so that $\mathcal{P} \cup \mathcal{Q}$ forms a Hamiltonian cycle, we consider to make an auxiliary graph of $\mathcal{Q}$, and we identify with the one for $\mathcal{P}$. To distinguish edges obtained from $\mathcal{P}$ or $\mathcal{Q}$, we color edges by red if one comes from $\mathcal{P}$ and by blue otherwise. Then following the Hamiltonian cycle, we can find an Eulerian trail of this merged auxiliary graph that uses edges of distinct colors alternately. But then if $\mathcal{P}'$ is equivalent to $\mathcal{P}$, then one can observe that if we replace the red part with the auxiliary graph of $\mathcal{P}'$, then it also has such an Eulerian trail, and we can show that $\mathcal{P}'$ can also be completed into a Hamiltonian cycle. So, in the algorithm, for each equivalence class, we store one partial solution. We define this equivalence relation formally in Section 4.4.2.

Since, the number of partitions of a k-size set is at most k^k and the number of paths induced by a partial solution is always bounded by n, the number of non-equivalent partial solutions is then bounded by $(2n)^k \cdot k^k$ (the maximum degree of an auxiliary multigraph is at most $2n$ because a loop is counted as two edges). The running time of our algorithm follows from the maximum number of non-equivalent partial solutions. The main effort in the algorithm consists then in updating the equivalence classes of this equivalence relation in terms of operations based on the clique-width operations.

In Subsection 4.4.1, we give basic definitions and notations concerning multigraphs. Our notions of partial solutions and of auxiliary multigraphs are given in Subsection 4.4.2. In Subsection 4.4.3, we prove the equivalence between the existence of Hamiltonian cycles in the input graph and the existence of red-blue alternating Eulerian trails in auxiliary multigraphs, and deduce that it is enough to store $(2n)^k \cdot k^k$ partial solutions at each step of our dynamic programming algorithm. In Subsection 4.4.4, we show how to obtain from the results of Subsection 4.4.3 an $n^{O(k)}$-time algorithm for Hamiltonian Cycle. In Subsection 4.4.5, we give some intuitions for solving in time $n^{O(k)}$ the problems Directed Hamiltonian Cycle given a k-expression. We conclude this section by some open question concerning a generalization of Hamiltonian Cycle.

4.4.1 Preliminaries

A *multigraph* is essentially a graph, but we allow multiple edges, i.e., edges incident with the same set of vertices. Formally, a *multigraph* G is a pair $(V(G), E(G))$ of disjoint sets, also called sets of vertices and edges, respectively, together with a map $\mathsf{mult}_G : E(G) \to V(G) \cup [V(G)]^2$, which maps every edge to one or two vertices, still called its endpoints. The degree of a vertex x in a multigraph G, is defined analogously as in graphs, except that each loop is counted twice, and similarly for other notions. If there are exactly k edges e such that $\mathsf{mult}_G(e) = \{x, y\}$ (or $\mathsf{mult}_G(e) = \{x\}$), then we denote these distinct edges by $\{x, y\}_1, \ldots, \{x, y\}_k$ (or $\{x\}_1, \ldots, \{x\}_k$).

For two multigraphs G and H on the same vertex set $\{v_1, \ldots, v_k\}$ and with disjoint edge sets, we denote by $G \uplus H$ the multigraph with vertex set $\{v_1, \ldots, v_k\}$, edge set $E(G) \cup E(H)$, and

$$\mathsf{mult}_{G \uplus H}(e) := \begin{cases} \mathsf{mult}_G(e) & \text{if } e \in E(G), \\ \mathsf{mult}_H(e) & \text{otherwise.} \end{cases}$$

If the edges of G and H are colored, then this operation preserves the colors of the edges.

We use the notations introduced for graphs in Section 1.2 concerning walks and trails. An *Eulerian trail* in a graph is a closed trail containing all edges. When the edges of a multigraph are colored in red or blue, we say that a walk $W = (v_1, e_1, \ldots, v_{r-1}, e_{r-1}, v_r)$ is a *red-blue walk*, if for every $i \in \{1, \ldots, r-2\}$, the colors of e_i and e_{i+1} are different and if the walk is closed, then the colors of e_1 and e_{r-1} are different. In particular, if the edges of a multigraph are colored red or blue, then a red-blue Eulerian trail is an Eulerian trail that is a red-blue walk. For two walks $W_1 = (v_1, e_1, \ldots, e_{\ell-1}, v_\ell)$ and $W_2 = (v'_1, e'_1, \ldots, e'_{r-1}, v'_r)$ such that $v_\ell = v'_1$, the *concatenation* of W_1 and W_2, denoted by $W_1 - W_2$, is the walk $(v_1, e_1, \ldots, e_{\ell-1}, v_\ell, e'_1, \ldots, e'_{r-1}, v'_r)$.

4.4.2 Partial solutions and auxiliary graphs

Let G be a graph and (H, lab_H) be a k-labeled graph such that H is a subgraph of G.

A *partial solution* of H is a set of edges $\mathcal{P} \subseteq E(H)$ such that $H_{|\mathcal{P}}$ is a union of vertex-disjoint paths, i.e., $H_{|\mathcal{P}}$ is acyclic and, for every vertex $v \in V(H)$, the degree of v in $H_{|\mathcal{P}}$ is at most two. We denote by $\Pi(H)$ the set of all partial solutions of H. We say that a path P in $H_{|\mathcal{P}}$ is *maximal* if the degree of its end-vertices in $H_{|\mathcal{P}}$ is at most one; in other words, there are no paths P' in $H_{|\mathcal{P}}$ such that $V(P) \subsetneq V(P')$. Observe that an isolated vertex in $H_{|\mathcal{P}}$ is considered as a maximal path.

A *complement solution* of H is a subset $\mathcal{Q}$ of $E(G) \setminus E(H)$ such that $G_{|\mathcal{Q}}$ is a union of vertex-disjoint paths with end-vertices in $V(H)$; in particular, for every vertex v in $V(G) \setminus V(H)$, the degree of v in $G_{|\mathcal{Q}}$ is two. We denote by $\overline{\Pi}(H)$ the set of all complement solutions of H. A path P in G with at least 2 vertices is an *H-path* if the end-vertices of P are in $V(H)$ and the internal vertices of P are in $V(G) \setminus V(H)$. By definition, isolated vertices in $V(H)$ are not H-paths. Observe that, for a complement solution $\mathcal{Q}$, we can decompose each maximal path Q of $G_{|\mathcal{Q}}$ with at least 2 vertices into H-paths (not necessarily one).

Examples of a partial solution and a complement solution are given in Figure 4.4. Note that if G has a Hamiltonian cycle C and $E(C) \not\subseteq E(G)$, then $E(C) \cap E(H)$ is a partial solution and $E(C) \cap (E(G) \setminus E(H))$ is a complement solution. We say that a partial solution $\mathcal{P}$ and a complement solution $\mathcal{Q}$ form a Hamiltonian cycle if $(V(G), \mathcal{P} \cup \mathcal{Q})$ is a cycle containing all the vertices of G.

Auxiliary Multigraph. For $\mathcal{P} \in \Pi(H) \cup \overline{\Pi}(H)$ and $i, j \in [k]$, we define ℓ_{ij} and ℓ_i as follows.

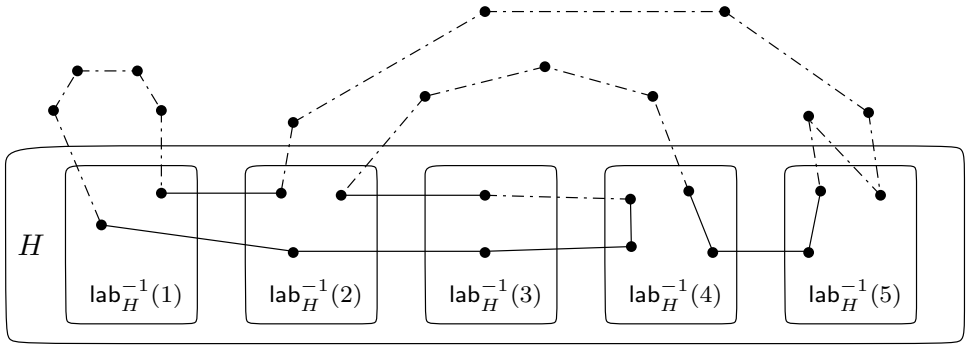


Figure 4.4 – Examples of a partial solution $\mathcal{P}$ (solid lines) and a complement solution $\mathcal{Q}$ (dashed lines) forming a Hamiltonian cycle. Observe that $H_{|\mathcal{P}}$ contains 5 maximal paths and $G_{|\mathcal{Q}}$ contains 5 H-paths (and only 4 maximal paths).

- If $\mathcal{P}$ is a partial solution of H, then ℓ_{ij} is the number of maximal paths in $H_{|\mathcal{P}}$ with end-vertices labeled i and j, and ℓ_i is the number of maximal paths in $H_{|\mathcal{P}}$ with both end-vertices labeled i.
- If $\mathcal{P}$ is a complement solution of H, then ℓ_{ij} is the number of H-paths in $G_{|\mathcal{P}}$ with end-vertices labeled i and j, and ℓ_i is the number of H-paths in $G_{|\mathcal{P}}$ with both end-vertices labeled i.

Now, we define the auxiliary multigraph of $\mathcal{P}$, denoted by $\mathsf{aux}_H(\mathcal{P})$, as the multigraph with vertex set $\{v_1, \ldots, v_k\}$ and edge set

$$\bigcup_{\substack{i,j\in[k]\\ i\neq j}} \{\{v_i, v_j\}_1, \ldots, \{v_i, v_j\}_{\ell_{ij}}\} \cup \bigcup_{i\in[k]} \{\{v_i\}_1, \ldots, \{v_i\}_{\ell_i}\}.$$

Moreover, if $\mathcal{P}$ is a partial solution of H, then we color all the edges of $\mathsf{aux}_H(\mathcal{P})$ in red, and if $\mathcal{P}$ is a complement solution, then we color the edges of $\mathsf{aux}_H(\mathcal{P})$ in blue. An example of an auxiliary multigraph is given in Figure 4.5.

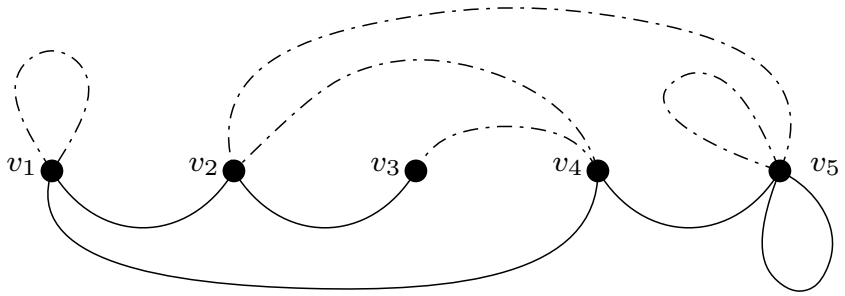


Figure 4.5 – The union $G_1 \uplus G_2$ of auxiliary multigraphs G_1 and G_2 associated with the partial solution (solid lines) and the complement solution (dashed lines) represented in Figure 4.4.

4.4.3 Relations between Hamiltonian cycles and Eulerian trails

Let G be an n-vertex graph and ϕ be an irredundant k-expression of G. Let H be a k-labeled graph arising in the k-expression ϕ. Observe that H is a subgraph of G (disregarding the labels). This section is dedicated to prove the properties of the following relation between partial solutions of H based on the degree sequence and the connected components of their auxiliary multigraphs.

Definition 4.67. *Let $\mathcal{P}_1, \mathcal{P}_2 \in \Pi(H)$. We write $\mathcal{P}_1 \simeq \mathcal{P}_2$ if $\mathsf{aux}_H(\mathcal{P}_1)$ and $\mathsf{aux}_H(\mathcal{P}_2)$ have the same set of connected components and for each vertex v in $\{v_1, \ldots, v_k\}$, $\mathsf{deg}_{\mathsf{aux}_H(\mathcal{P}_1)}(v) = \mathsf{deg}_{\mathsf{aux}_H(\mathcal{P}_2)}(v)$.*

Observe that $\simeq$ is an equivalence relation. For a set $\mathcal{A} \subseteq \Pi(H)$, we define $\mathsf{reduce}_H(\mathcal{A})$ as the operation which returns a set containing one element of each equivalence class of $\mathcal{A}/\simeq$.

The main idea of our algorithm is to call reduce_H at each step of our dynamic programming algorithm in order to keep the size of a set of partial solutions manipulated small, i.e., bounded by $n^{O(k)}$. The running time of our algorithm follows mostly from the following lemma.

Lemma 4.68. *For every $\mathcal{A} \subseteq \Pi(H)$, we have $|\mathsf{reduce}_H(\mathcal{A})| \leq n^k \cdot 2^{k(\log_2(k)+1)}$ and we can moreover compute $\mathsf{reduce}_H(\mathcal{A})$ in time $O(|\mathcal{A}| \cdot nk^2 \log_2(nk))$.*

Proof. To prove that $\mathsf{reduce}_H(\mathcal{A}) \leq n^k \cdot 2^{k(\log_2(k)+1)}$, it is enough to bound the number of equivalence classes of $\Pi(H)/\simeq$.

We claim that, for every $\mathcal{P} \in \Pi(H)$, we have $\sum_{i\in[k]} \mathsf{deg}_{\mathsf{aux}_H(\mathcal{P})}(v_i) \leq 2|V(H)|$. First observe that $\sum_{i\in[k]} \mathsf{deg}_{\mathsf{aux}_H(\mathcal{P})}(v_i) = 2|V(H)|$ when $\mathcal{P} = \varnothing$, since each isolated vertex in $H_{|\mathcal{P}}$ gives a loop in $\mathsf{aux}_H(\mathcal{P})$. Moreover, when $\mathcal{P}$ contains an edge, removing an edge from a partial solution $\mathcal{P}$ of H increases $\sum_{i\in[k]} \mathsf{deg}_{\mathsf{aux}_H(\mathcal{P})}(v_i)$ by two; indeed, this edge removal splits a maximal path of $H_{|\mathcal{P}}$ into two maximal paths. Therefore, any partial solution $\mathcal{P}$ satisfies that $\sum_{i\in[k]} \mathsf{deg}_{\mathsf{aux}_H(\mathcal{P})}(v_i) \leq 2|V(H)|$; in particular each vertex of $\mathsf{aux}_H(\mathcal{P})$ has degree at most $2|V(H)|$. As $\mathsf{aux}_H(\mathcal{P})$ contains k vertices, we deduce that there are at most $(2|V(H)|)^k \leq (2n)^k$ possible degree sequences for $\mathsf{aux}_H(\mathcal{P})$.

Since the number of partitions of $\{v_1, \ldots, v_k\}$ is bounded by $2^{k \log_2 k}$. We conclude that $\simeq$ partitions $\Pi(H)$ into at most $n^k \cdot 2^{k(\log_2 k+1)}$ equivalences classes.

It remains to prove that we can compute $\mathsf{reduce}_H(\mathcal{A})$ in time $O(|\mathcal{A}| \cdot nk \log_2(nk))$. First observe that, for every $\mathcal{P} \in \Pi(H)$, we can compute $\mathsf{aux}_H(\mathcal{P})$ in time $O(nk)$. Moreover, we can also compute the degree sequence of $\mathsf{aux}_H(\mathcal{P})$ and the connected components of $\mathsf{aux}_H(\mathcal{P})$ in time $O(nk)$. Thus, by using the right data structures, we can decide whether $\mathcal{P}_1 \simeq \mathcal{P}_2$ in time $O(nk)$. Furthermore, by using a self-balancing binary search tree, we can compute $\mathsf{reduce}_H(\mathcal{A})$ in time $O(|\mathcal{A}| \cdot nk \log_2(|\mathsf{reduce}_H(\mathcal{A})|))$. Since $\log_2(|\mathsf{reduce}_H(\mathcal{A})|) \leq k \log_2(2nk)$, we conclude that $\mathsf{reduce}_H(\mathcal{A})$ is computable in time $O(|\mathcal{A}| \cdot nk^2 \log_2(nk))$. □

The rest of this section is dedicated to prove that, for a set of partial solutions $\mathcal{A}$ of H, the set $\mathsf{reduce}_H(\mathcal{A})$ is equivalent to $\mathcal{A}$, i.e., if $\mathcal{A}$ contains a partial solution that forms a Hamiltonian cycle with a complement solution, then $\mathsf{reduce}_H(\mathcal{A})$ also. Our results are based on a kind of equivalence between Hamiltonian cycles and red-blue Eulerian trails. The following observation is one direction of this equivalence.

Lemma 4.69. *If $\mathcal{P} \in \Pi(H)$ and $\mathcal{Q} \in \overline{\Pi}(H)$ form a Hamiltonian cycle, then the multigraph $\mathsf{aux}_H(\mathcal{P}) \uplus \mathsf{aux}_H(\mathcal{Q})$ admits a red-blue Eulerian trail.*

Proof. Suppose that $\mathcal{P} \in \Pi(H)$ and $\mathcal{Q} \in \overline{\Pi}(H)$ form a Hamiltonian cycle C. Let $M := \mathsf{aux}_H(\mathcal{P}) \uplus \mathsf{aux}_H(\mathcal{Q})$. From the definitions of a partial solution and of a complement solution, there is a sequence $(P_1, Q_1, \ldots, P_\ell, Q_\ell)$ of paths in $\mathcal{P}$ and $\mathcal{Q}$ such that

- $P_1, P_2, \ldots, P_\ell$ are all the maximal paths in $H_{|\mathcal{P}}$,
- $Q_1, Q_2, \ldots, Q_\ell$ are all the H-paths in $G_{|\mathcal{Q}}$,
- $P_1, Q_1, \ldots, P_\ell, Q_\ell$ appear in C in this order,
- for each $x \in [\ell]$, the first end-vertices of P_x is the last end-vertex of Q_{x-1} and the last end-vertex of P_x is the first end-vertex of Q_x (indices are considered modulo ℓ).

Observe that each maximal path P_x in $H_{|\mathcal{P}}$ with end-vertices labeled i and j is associated with a red edge in M, say e_x with $\mathsf{mult}_M(e_x) = \{v_i, v_j\}$ if $i \neq j$ or $\mathsf{mult}_M(e_x) = \{v_i\}$ if $i = j$ such that the edges $e_1, \ldots, e_\ell$ are pairwise distinct and $E(\mathsf{aux}_H(\mathcal{P})) = \{e_1, \ldots, e_\ell\}$. Similarly, each H-path Q_y of $G_{|\mathcal{Q}}$ with end-vertices labeled i and j is associated with a blue edge f_y in M with $\mathsf{mult}_M(f_y) = \{v_i, v_j\}$ if $i \neq j$ or $\mathsf{mult}_M(f_y) = \{v_i\}$ if $i = j$ such that the edges $f_1, \ldots, f_\ell$ are pairwise distinct and $E(\mathsf{aux}_H(\mathcal{Q})) = \{f_1, \ldots, f_\ell\}$. It is not difficult to see that $(e_1, f_1, \ldots, e_\ell, f_\ell)$ is a red-blue Eulerian trail of $\mathsf{aux}_H(\mathcal{P}) \uplus \mathsf{aux}_H(\mathcal{Q})$. □

Next, we prove the other direction. We use the properties of an irredundant k-expression described in Lemma 2.20.

Lemma 4.70. *Let $\mathcal{P} \in \Pi(H)$. If there exists a complement solution $\mathcal{Q}$ of H such that $\mathsf{aux}_H(\mathcal{P}) \uplus \mathsf{aux}_H(\mathcal{Q})$ admits a red-blue Eulerian trail, then there exists $\mathcal{Q}^\star \in \overline{\Pi}(H)$ such that $\mathcal{P}$ and $\mathcal{Q}^\star$ form a Hamiltonian cycle.*

Proof. Let $T = (v_{a_1}, r_1, v_{c_1}, b_1, v_{a_2}, r_2, v_{c_2}, \cdots, v_{a_\ell}, r_\ell, v_{c_\ell}, b_\ell, v_{a_1})$ be a red-blue Eulerian trail of $\mathsf{aux}_H(\mathcal{P}) \uplus \mathsf{aux}_H(\mathcal{Q})$ with $r_1, \ldots, r_\ell \in E(\mathsf{aux}_H(\mathcal{P}))$ and $b_1, \ldots, b_\ell \in E(\mathsf{aux}_H(\mathcal{Q}))$. In the following, the indexes are modulo ℓ.

For each $i \in [\ell]$, we associate r_i with a maximal path P_i of $H_{|\mathcal{P}}$ with end-vertices labeled a_i and c_i and we associate b_i with an H-path Q_i of $G_{|\mathcal{Q}}$ with end-vertices labeled c_i and a_{i+1}, such that $P_1, \ldots, P_\ell, Q_1, \ldots, Q_\ell$ are all pairwise distinct.

For every $i \in [\ell]$, we construct from Q_i an H-path $Q_i^\star$ of G by doing the following. Let u, v be respectively the last end-vertex of P_i and the first end-vertex of P_{i+1}. Observe that u and the first vertex of Q_i are labeled c_i, and v and the last vertex of Q_i are labeled a_{i+1}. We distinguish two cases:

- Suppose that $Q_i = (x, xy, y)$, i.e., Q_i uses only one edge. Since $\mathcal{Q}$ is a complement solution of H, we have $xy \in E(G) \setminus E(H)$. By Lemma 2.20, we have $uv \in E(G) \setminus E(H)$. In this case, we define $Q_i^\star = (u, uv, v)$.
- Assume now that $Q_i = (x, xy, y, \ldots, w, wz, z)$ with $w, y \in V(G) \setminus V(H)$ (possibly, $w = y$). Since x and u have the same label in H, by Lemma 2.20, we have $N_G(x) \setminus V(H) = N_G(u) \setminus V(H)$. Hence, u is also adjacent to y. Symmetrically, we have v is adjacent to w. In this case, we define $Q_i^\star = (u, uy, y, \ldots, w, wv, v)$, i.e., the path with the same internal vertices as Q_i and with end-vertices u and v.

In both cases, we end up with a path $Q_i^\star$ that uses the same internal vertices as Q_i and whose end-vertices are the last vertex of P_i and the first vertex of P_{i+1}. We conclude that

$$P_1 - Q_1^\star - \cdots - P_\ell - Q_\ell^\star$$

is a Hamiltonian cycle.

Let $\mathcal{Q}^\star$ be the set of edges used by the paths $Q_1^\star, \ldots, Q_\ell^\star$. By construction, we have $\mathcal{Q}^\star \subseteq E(G) \setminus E(H)$, and thus $\mathcal{Q}^\star \in \overline{\Pi}(H)$. Observe that, for every $i \in [\ell]$, the labels of the end-vertices of $Q_i^\star$ are the same as those of Q_i. Consequently, we have $\mathsf{aux}_H(\mathcal{Q}^\star) = \mathsf{aux}_H(\mathcal{Q})$. □

It is well known that a connected multigraph contains an Eulerian trail if and only if every vertex has even degree. As an extension, Kotzig [101] proved that a connected two-edge colored graph (without loops and multiple edges) contains a red-blue Eulerian trail if and only if each vertex is incident with the same number of edges for both colors. This result can be easily generalized to multigraphs by replacing red edge with a path of length 3 whose colors are red, blue, red in the order, and replacing blue edge with a path of length 3 whose colors are blue, red, blue in the order. For the completeness of this manuscript, we add its proof.

Let G be a multigraph whose edges are colored red or blue, and let R and B be respectively the set of red and blue edges. For a vertex $v \in V(G)$, we let $\mathsf{rdeg}_G(v)$ and $\mathsf{bdeg}_G(v)$ be respectively the degree of v in $G_{|R}$ and $G_{|B}$. Recall that a loop is counted twice in the degree of a vertex.

Lemma 4.71 (Kotzig [101])**.** *Let G be a connected multigraph whose edges are colored red or blue. Then G has a red-blue Eulerian trail if and only if, for every vertex v, $\mathsf{bdeg}_G(v) = \mathsf{rdeg}_G(v)$.*

Proof. One can easily check that if G has a red-blue Eulerian trail, then for every vertex v, $\mathsf{bdeg}_G(v) = \mathsf{rdeg}_G(v)$. Indeed, if $T = (v_1, e_1, v_2, \ldots, v_\ell, e_\ell, v_1)$ is a red-blue Eulerian trail, then, for $2 \leq i \leq \ell$, e_{i-1} and e_i have different colors, and e_1 and e_ℓ have different colors, we can conclude that the blue edges incident with a vertex v are in 1-to-1 correspondence with the red edges incident with v (by counting twice the loops).

Let us now prove the other direction. Let $T := (v_1, e_1, v_2, e_2, \cdots, v_i, e_i, v_{i+1})$ be a longest red-blue trail. We may assume that e_1 is colored red. First observe that $v_1 = v_{i+1}$. Otherwise, $\mathsf{bdeg}_T(v_1) + 1 = \mathsf{rdeg}_T(v_1)$ and thus, there is a blue edge in $E(G) \setminus E(T)$ incident with v_1. So, we can extend T by adding this edge, a contradiction. Thus, $v_1 = v_{i+1}$.

Next we show that e_i is colored blue. Suppose e_i is colored red. Then $\mathsf{bdeg}_T(v_1) + 2 = \mathsf{rdeg}_T(v_1)$ and thus, there is a blue edge in $E(G) \setminus E(T)$ incident with v_1. So, we can extend T by adding this edge, a contradiction. Thus, e_i is colored red, and it implies that T is a closed red-blue trail. It means that T can be considered as a closed red-blue trail starting from any vertex of T and following the same order or the reverse order of T.

Now, we show that $V(G) = V(T)$. Suppose $V(G) \setminus V(T)$ is non-empty. Since G is connected, there is an edge vw with $v \in V(T)$ and $w \in V(G) \setminus V(T)$. If vw is a red edge, then starting from this edge and following T from a blue edge incident with v, we can find a red-blue trail longer than T, a contradiction. The same argument holds when vw is a blue edge. Therefore, we have that $V(G) = V(T)$. By a similar argument, one can show that $E(G) = E(T)$; if there is an edge vw in $E(G) \setminus E(T)$, we can extend T by putting vw at the beginning. So, $E(G) = E(T)$.

We conclude that T is a red-blue Eulerian trail, as required. □

In order to prove the correctness of our algorithm, we need the following relation between subsets of partial solutions.

Definition 4.72. *Let $\mathcal{A}$ and $\mathcal{B}$ be two subsets of $\Pi(H)$. We write $\mathcal{A} \lesssim_H \mathcal{B}$ if, for every multigraph $\mathcal{M}$ whose edges are colored blue, whenever there exists $\mathcal{P}_1 \in \mathcal{B}$ such that $\mathsf{aux}_H(\mathcal{P}_1) \uplus \mathcal{M}$ admits a red-blue Eulerian trail, there exists $\mathcal{P}_2 \in \mathcal{A}$ such that $\mathsf{aux}_H(\mathcal{P}_2) \uplus \mathcal{M}$ admits a red-blue Eulerian trail.*

The main idea of our algorithm for Hamiltonian Cycle, is to compute, for every k-labeled graph H arising in ϕ, a set $\mathcal{A} \subseteq \Pi(H)$ of small size such that $\mathcal{A} \lesssim_H \Pi(H)$. Indeed, by Lemmas 4.69 and 4.70, $\mathcal{A} \lesssim_H \Pi(H)$ implies that if there exist $\mathcal{P} \in \Pi(H)$ and $\mathcal{Q} \in \overline{\Pi}(H)$ such that $\mathcal{P}$ and $\mathcal{Q}$ form a Hamiltonian cycle, then there exist $\mathcal{P}^\star \in \mathcal{A}$ and $\mathcal{Q}^\star \in \overline{\Pi}(H)$ such that $\mathcal{P}^\star$ and $\mathcal{Q}^\star$ form a Hamiltonian cycle. The following lemma is the key of our algorithm.

Lemma 4.73. *Let $\mathcal{A} \subseteq \Pi(H)$. Then $\mathsf{reduce}_H(\mathcal{A}) \lesssim_H \mathcal{A}$.*

Proof. Let $\mathcal{P} \in \mathcal{A}$ and $\mathcal{M}$ be a multigraph whose edges are colored blue such that $\mathsf{aux}_H(\mathcal{P}) \uplus \mathcal{M}$ admits a red-blue Eulerian trail. By definition, $\mathsf{reduce}_H(\mathcal{A})$ contains a partial solution $\mathcal{P}^\star$ such that $\mathcal{P} \simeq \mathcal{P}^\star$. As $\mathsf{aux}_H(\mathcal{P}) \uplus \mathcal{M}$ contains a red-blue Eulerian trail, by Lemma 4.71, we have that

- $\mathsf{aux}_H(\mathcal{P}) \uplus \mathcal{M}$ is connected and
- for every $i \in [k]$, $\mathsf{deg}_{\mathsf{aux}_H(\mathcal{P})}(v_i) = \mathsf{deg}_{\mathcal{M}}(v_i)$.

Since $\mathsf{aux}_H(\mathcal{P})$ has the same set of connected components as $\mathsf{aux}_H(\mathcal{P}^\star)$, we know that $\mathsf{aux}_H(\mathcal{P}^\star) \uplus \mathcal{M}$ is also connected. Moreover, for every $i \in [k]$, we have

$$\mathsf{deg}_{\mathsf{aux}_H(\mathcal{P})}(v_i) = \mathsf{deg}_{\mathsf{aux}_H(\mathcal{P}^\star)}(v_i) = \mathsf{deg}_{\mathcal{M}}(v_i).$$

By Lemma 4.71, we conclude that $\mathsf{aux}_H(\mathcal{P}^\star) \uplus \mathcal{M}$ admits a red-blue Eulerian trail.

Thus, for every $\mathcal{P} \in \mathcal{A}$ and multigraph $\mathcal{M}$ with blue edges such that $\mathsf{aux}_H(\mathcal{P}) \uplus \mathcal{M}$ admits a red-blue Eulerian trail, there exists $\mathcal{P}^\star \in \mathsf{reduce}_H(\mathcal{A})$ such that $\mathsf{aux}_H(\mathcal{P}^\star) \uplus \mathcal{M}$ admits a red-blue Eulerian trail. Hence, we have $\mathsf{reduce}_H(\mathcal{A}) \lesssim_H \mathcal{A}$. □

Lemma 4.74. *Let $\mathcal{A}, \mathcal{B} \subseteq \Pi(H)$. If $\mathcal{A} \lesssim_H \mathcal{B}$, then $\mathsf{reduce}_H(\mathcal{A}) \lesssim_H \mathcal{B}$.*

Proof. One easily checks that $\lesssim_H$ is a transitive relation. Now, assuming that $\mathcal{A} \lesssim_H \mathcal{B}$, we have $\mathsf{reduce}_H(\mathcal{A}) \lesssim \mathcal{B}$ because $\mathsf{reduce}_H(\mathcal{A}) \lesssim_H \mathcal{A}$ by Lemma 4.73. □

4.4.4 Hamiltonian Cycle problem

In this subsection, we present our algorithm solving Hamiltonian Cycle. Our algorithm computes recursively, for every k-labeled graph H arising in the k-expression of G, a set $\mathcal{A}_H$ such that $\mathcal{A}_H \lesssim_H \Pi(H)$ and $|\mathcal{A}_H| \leq n^k \cdot 2^{k(\log_2(k)+1)}$. In order to prove the correctness of our algorithm, we need the following lemmas which prove that the operations we use to compute sets of partial solutions preserve the relation $\lesssim_H$.

Lemma 4.75. *Let $H = \rho_{i \to j}(D)$. If $\mathcal{A}_D \lesssim_D \Pi(D)$, then $\mathcal{A}_D \lesssim_H \Pi(H)$.*

Proof. First, observe that H has the same set of vertices and edges as D. Thus, we have $\Pi(H) = \Pi(D)$ and $\overline{\Pi}(H) = \overline{\Pi}(D)$. Suppose that $\mathcal{A}_D \lesssim_D \Pi(D)$.

Let $\mathcal{P} \in \Pi(H)$ and $\mathcal{M}$ be a multigraph whose edges are colored blue such that $\mathsf{aux}_H(\mathcal{P}) \uplus \mathcal{M}$ contains a red-blue Eulerian trail T. To prove the lemma, it is sufficient to prove that there exists $\mathcal{P}^\star \in \mathcal{A}_D$ such that $\mathsf{aux}_H(\mathcal{P}^\star) \uplus \mathcal{M}$ contains a red-blue Eulerian trail.

Let f be a bijective function such that

- for every edge e of $\mathsf{aux}_D(\mathcal{P})$ with endpoints v_ℓ and v_i, for some ℓ, $f(e)$ is an edge of $\mathsf{aux}_H(\mathcal{P})$ with endpoints v_ℓ and v_j, and
- for every loop e with endpoint v_i, $f(e)$ is a loop of $\mathsf{aux}_H(\mathcal{P})$ with endpoint v_j.

By construction of $\mathsf{aux}_D(\mathcal{P})$ and $\mathsf{aux}_H(\mathcal{P})$, such a function exists.

We construct the multigraph $\mathcal{M}'$ from $\mathcal{M}$ and T by successively doing the following:

- For every edge e of the multigraph $\mathsf{aux}_D(\mathcal{P})$ with endpoints v_ℓ and v_i, take the subwalk $W = (v_\ell, f(e), v_j, e_a, v_a)$ of T. Replace e_a in $\mathcal{M}$ by an edge e'_a with endpoints v_i and v_a.
- For every loop e with endpoint v_i in the multigraph $\mathsf{aux}_D(\mathcal{P})$, take the subwalk $W = (v_a, e_a, v_j, f(e), v_j, e_b, v_b)$ of T. Replace e_a (respectively e_b) in $\mathcal{M}$ by an edge e'_a (resp. e'_b) with endpoints v_a and v_i (resp. v_i and v_b).

By construction, one can construct from T a red-blue Eulerian trail of $\mathsf{aux}_D(\mathcal{P}) \uplus \mathcal{M}'$. Since $\mathcal{A}_D \lesssim_D \Pi(D)$, there exists $\mathcal{P}^\star \in \mathcal{A}_D$ such that $\mathsf{aux}_D(\mathcal{P}^\star) \uplus \mathcal{M}'$ contains a red-blue Eulerian trail. Observe that $\mathsf{aux}_H(\mathcal{P})$ (respectively $\mathcal{M}$) is obtained from $\mathsf{aux}_D(\mathcal{P}^\star)$ (resp. $\mathcal{M}'$) by replacing each edge associated with $\{v_i, v_k\}$ or $\{v_i\}$ in $\mathsf{aux}_D(\mathcal{P}^\star)$ (resp. $\mathcal{M}'$) with an edge associated with $\{v_j, v_k\}$ or $\{v_j\}$ respectively. We conclude that $\mathsf{aux}_H(\mathcal{P}^\star) \uplus \mathcal{M}$ admits a red-blue Eulerian trail. □

Lemma 4.76. *Let $H = D \oplus F$. If $\mathcal{A}_D \lesssim_D \Pi(D)$ and $\mathcal{A}_F \lesssim_F \Pi(F)$, then $(\mathcal{A}_D \otimes \mathcal{A}_F) \lesssim_H \Pi(H)$.*

Proof. Observe that $V(D)$ and $V(F)$ are disjoint. Consequently, we have $\Pi(H) = \Pi(D) \otimes \Pi(F)$, and, for all $\mathcal{P}_D \in \Pi(D)$ and $\mathcal{P}_F \in \Pi(F)$, we have $\mathsf{aux}_H(\mathcal{P}_D \cup \mathcal{P}_F) = \mathsf{aux}_H(\mathcal{P}_D) \uplus \mathsf{aux}_H(\mathcal{P}_F)$. Suppose that $\mathcal{A}_D \lesssim_D \Pi(D)$ and $\mathcal{A}_F \lesssim_F \Pi(F)$.

Let $\mathcal{P}_D \in \Pi(D)$ and $\mathcal{P}_F \in \Pi(F)$, and let $\mathcal{M}$ be a multigraph whose edges are colored blue such that there exists a red-blue Eulerian trail T in $\mathsf{aux}_H(\mathcal{P}_D \cup \mathcal{P}_F) \uplus \mathcal{M}$. It is sufficient to prove that there exist $\mathcal{P}_D^\star \in \mathcal{A}_D$ and $\mathcal{P}_F^\star \in \mathcal{A}_F$ such that $\mathsf{aux}_H(\mathcal{P}_D^\star \cup \mathcal{P}_F^\star) \uplus \mathcal{M}$ admits a red-blue Eulerian trail.

We begin by proving that there exists $\mathcal{P}_D^\star \in \mathcal{A}_D$ such that $\mathsf{aux}_H(\mathcal{P}_D^\star \cup \mathcal{P}_F) \uplus \mathcal{M}$ contains a red-blue Eulerian trail. In order to do so, we construct from $\mathsf{aux}_H(\mathcal{P}_D \cup \mathcal{P}_F) \uplus \mathcal{M}$ a multigraph $\mathcal{M}'$ by successively repeating the following: take a maximal sub-walk W of T which uses alternately blue edges and red edges from $\mathsf{aux}_H(\mathcal{P}_F) \uplus \mathcal{M}$, remove these edges and add a blue edge between the two end-vertices of W.

By construction of $\mathcal{M}'$, for every $\mathcal{P}'_D \in \Pi(D)$, if $\mathsf{aux}_D(\mathcal{P}'_D) \uplus \mathcal{M}'$ admits a red-blue Eulerian trail, then $\mathsf{aux}_H(\mathcal{P}'_D \cup \mathcal{P}_F) \uplus \mathcal{M}$ contains a red-blue Eulerian trail also. We also deduce from this construction that the multigraph $\mathsf{aux}_D(\mathcal{P}_D) \uplus \mathcal{M}'$ contains a red-blue Eulerian trail. As $\mathcal{A}_D \lesssim_D \Pi(D)$, there exists $\mathcal{P}_D^\star$ such that $\mathsf{aux}_D(\mathcal{P}_D^\star) \uplus \mathcal{M}'$ contains a red-blue Eulerian trail. We conclude that $\mathsf{aux}_H(\mathcal{P}_D^\star \cup \mathcal{P}_F) \uplus \mathcal{M}$ contains also a red-blue Eulerian trail.

Symmetrically, we can prove that there exists $\mathcal{P}_F^\star \in \mathcal{A}_F$ such that $\mathsf{aux}_H(\mathcal{P}_D^\star \cup \mathcal{P}_F^\star) \uplus \mathcal{M}$ contains a red-blue Eulerian trail. This proves the lemma. □

For two k-labeled subgraphs H and D arising in the k-expression of G such that $H = \eta_{i,j}(D)$, we denote by $\mathcal{E}_{i,j}^H$ the set of edges whose endpoints are labeled i and j, i.e., $\{uv \in E(H) : \mathsf{lab}_H(v) = i \ \wedge \ \mathsf{lab}_H(v) = j\}$. For $\mathcal{P} \in \Pi(H)$, we denote by $\mathcal{P} + (i,j)$ the set of all partial solutions $\mathcal{P}'$ of $\Pi(H)$ obtained by the union of $\mathcal{P}$ and an edge uv in $\mathcal{E}_{i,j}^H$. Observe that u and v must be the endpoints of two distinct maximal paths of $H_{|\mathcal{P}}$. We extend this notation to sets of partial solutions; for every $\mathcal{A} \subseteq \Pi(H)$, we denote by $\mathcal{A} + (i,j)$, the set $\bigcup_{\mathcal{P} \in \mathcal{A}} (\mathcal{P} + (i,j))$. It is worth noting that $\Pi(D) \subseteq \Pi(H)$ and thus the operator $+(i,j)$ is well defined for a partial solution in $\Pi(D)$.

Moreover, for every $\mathcal{A} \subseteq \Pi(D)$ and integer $t \geq 0$, we define the set $\mathcal{A}^t$ as follows

$$\mathcal{A}^t := \begin{cases} \mathcal{A} & \text{if } t = 0, \\ \mathsf{reduce}_H(\mathcal{A}^{t-1} + (i,j)) & \text{otherwise.} \end{cases}$$

Observe that each set $\mathcal{P}$ in $\mathcal{A}^t$ is a partial solution of H and $|\mathcal{P} \cap \mathcal{E}_{i,j}^H| = t$.

Lemma 4.77. *Let $H = \eta_{i,j}(D)$ such that $E(D) \cap \mathcal{E}_{i,j}^H = \varnothing$. If $\mathcal{A}_D \lesssim_D \Pi(D)$, then we have $\mathcal{A}_D^0 \cup \cdots \cup \mathcal{A}_D^n \lesssim_H \Pi(H)$.*

Proof. Suppose that $\mathcal{A}_D \lesssim_D \Pi(D)$. We begin by proving the following claim.

Claim 4.77.1. *Let $\mathcal{A}, \mathcal{B} \subseteq \Pi(H)$. If $\mathcal{A} \lesssim_H \mathcal{B}$, then $\mathcal{A} + (i,j) \lesssim_H \mathcal{B} + (i,j)$.*

Proof. Suppose that $\mathcal{A} \lesssim_H \mathcal{B}$. Let $\mathcal{P} \in \mathcal{B} + (i,j)$ and $\mathcal{M}$ be a multigraph with blue edges such that $\mathsf{aux}_H(\mathcal{P}) \uplus \mathcal{M}$ admits a red-blue Eulerian trail. Take $vw \in \mathcal{E}_{i,j}^H$ such that $\mathcal{P}' := \mathcal{P} - uw$ belongs to $\mathcal{B}$ and $v \in \mathsf{lab}_H^{-1}(i)$ and $w \in \mathsf{lab}_H^{-1}(j)$. Let $\mathcal{M}'$ be the multigraph obtained by adding a blue edge f with endpoints v_i and v_j to $\mathcal{M}$.

We claim that the multigraph $\mathsf{aux}_H(\mathcal{P}') \uplus \mathcal{M}'$ admits a red-blue Eulerian trail. Note that there is a path $P \in \mathcal{P}$ containing the edge uw, and when we remove uw from $\mathcal{P}$, we divide P into two maximal subpaths, say P_1 and P_2. Without loss of generality, we may assume that P_1 contains v and P_2 contains w. Let v' and w' be the other end-vertices of P_1 and P_2, respectively, and let $i' := \mathsf{lab}_H(v')$ and $j' := \mathsf{lab}_H(w')$. Note that $\mathsf{aux}_H(\mathcal{P}')$ can be obtained from $\mathsf{aux}_H(\mathcal{P})$ by removing

an edge e associated with $\{v_{i'}, v_{j'}\}$ and adding two edges e_1 and e_2 associated with $\{v_{i'}, v_i\}$ and $\{v_j, v_{j'}\}$ respectively. So, we can obtain a red-blue Eulerian trail of $\mathsf{aux}_H(\mathcal{P}') \uplus \mathcal{M}'$ from a red-blue Eulerian trail of $\mathsf{aux}_H(\mathcal{P}) \uplus \mathcal{M}$ by replacing $(v_{i'}, e, v_{j'})$ with the sequence $(v_{i'}, e_1, v_i, f, v_j, e_2, v_{j'})$ where f is the blue edge we add to $\mathcal{M}$ to obtain $\mathcal{M}'$. It implies the claim.

Now, since $\mathcal{A} \lesssim_H \mathcal{B}$, there exists $\mathcal{P}^\star \in \mathcal{A}$ such that $\mathsf{aux}_H(\mathcal{P}^\star) \uplus \mathcal{M}'$ admits a red-blue Eulerian trail T. Let W be the subwalk of T such that $W = (v_a, e_a, v_i, f, v_j, e_b, v_b)$. Take two maximal paths P_1 and P_2 in $H_{|\mathcal{P}^\star}$ such that the end-vertices of P_1 (respectively P_2) are labeled a and i (resp. b and j). Let $\widehat{\mathcal{P}}$ be the partial solution of H obtained from $\mathcal{P}^\star$ by adding the edge in $\mathcal{E}^H_{i,j}$ between the end-vertex of P_1 labeled i and the end-vertex of P_2 labeled j. By construction, we have $\widehat{\mathcal{P}} \in \mathcal{A} + (i,j)$ and $\mathsf{aux}_H(\widehat{\mathcal{P}}) \uplus \mathcal{M}$ admits a red-blue Eulerian trail. We conclude that $\mathcal{A} + (i,j) \lesssim_H \mathcal{B} + (i,j)$. □

Let $\Pi(D) + t(i,j)$ be the set of partial solutions of H obtained by applying t times the operation $+(i,j)$ on $\Pi(D)$. Since every partial solution of H is obtained from the union of a partial solution of D and a subset of $\mathcal{E}^H_{i,j}$ of size at most n, we have $\Pi(H) = \bigcup_{0 \leq t \leq n} (\Pi(D) + t(i,j))$.

Since $V(D) = V(H)$ and $E(D) \subseteq E(H)$, we have $\mathcal{A}^0_D = \mathcal{A}_D \lesssim_H \Pi(D) + 0(i,j)$. Let $\ell \in \{1, \ldots, n\}$ and suppose that $\mathcal{A}^{\ell-1}_D \lesssim_H \Pi(D) + (\ell-1)(i,j)$. From Claim 4.77.1, we have $\mathcal{A}^{\ell-1}_D + (i,j) \lesssim_H \Pi(D) + \ell(i,j)$. By Lemma 4.74, we deduce that $\mathcal{A}^\ell_D = \mathsf{reduce}(\mathcal{A}^{\ell-1}_D + (i,j)) \lesssim_H \Pi(D) + \ell(i,j)$.

Thus, by recurrence, for every $0 \leq \ell \leq n$, we have $\mathcal{A}^\ell_D \lesssim_H \Pi(D) + \ell(i,j)$. We conclude that $\mathcal{A}^0_D \cup \cdots \cup \mathcal{A}^n_D \lesssim_H \Pi(H)$. □

We are now ready to prove the main theorem of this section.

Theorem 4.78. *There exists an algorithm that, given an n-vertex graph G and a k-expression ϕ of G, solves* Hamiltonian Cycle *in time $O(n^{2k+5} \cdot 2^{2k(\log_2(k)+1)} \cdot k^3 \cdot \log_2(nk))$.*

Proof. Since every k-expression can be transformed into an irredundant k-expression in linear time, we may assume that ϕ is an irredundant k-expression.

We do a bottom-up traversal of the k-expression and at each k-labeled graph H arising in ϕ, we compute a set $\mathcal{A}_H \subseteq \Pi(H)$ such that $|\mathcal{A}_H| \leq n^k 2^{k(\log(k)+1)}$ and $\mathcal{A}_H \lesssim_H \Pi(H)$, by doing the following:

- If $H = i(v)$, then we have $\Pi(H) = \{\varnothing\}$ because $E(H) = \varnothing$. In this case, we set $\mathcal{A}_H := \{\varnothing\}$. Obviously, we have $\mathcal{A}_H \lesssim_H \Pi(H)$.
- If $H = \rho_{i,j}(D)$, then we set $\mathcal{A}_H := \mathcal{A}_D$.
- If $H = D \oplus F$, then we set $\mathcal{A}_H := \mathsf{reduce}_H(\mathcal{A}_D \otimes \mathcal{A}_F)$.
- If $H = \eta_{i,j}(D)$, then we set $\mathcal{A}_H := \mathsf{reduce}_H(\mathcal{A}^0_D \cup \cdots \cup \mathcal{A}^n_D)$.

For the last three cases, we deduce, by induction, from Lemma 4.74 and Lemmas 4.75–4.77 that $\mathcal{A}_H \lesssim_H \Pi(H)$. Moreover by the use of the function reduce_H, by Lemma 4.68, we have $|\mathcal{A}_H| \leq n^k \cdot 2^{k(\log(k)+1)}$.

We now explain how our algorithm decides whether G admits a Hamiltonian cycle. We claim that G has a Hamiltonian cycle if and only if there exist two k-labeled graphs H and D arising in ϕ with $V(H) = V(G)$ and $H = \eta_{i,j}(D)$, and there exists $\mathcal{P} \in \mathcal{A}_D$ such that, for every $\ell \in [k] \setminus \{i,j\}$, we have $\mathsf{deg}_{\mathsf{aux}_D(\mathcal{P})}(v_\ell) = 0$ and $\mathsf{deg}_{\mathsf{aux}_H(\mathcal{P})}(v_i) = \mathsf{deg}_{\mathsf{aux}_H(\mathcal{P})}(v_j) > 0$.

First suppose that G contains a Hamiltonian cycle C and take the k-labeled graph H arising in ϕ such that $E(C) \subseteq E(H)$ and $|E(H)|$ is minimal. Note that the operations of taking the

disjoint union of two graphs or relabeling cannot create a Hamiltonian cycle. Thus, by minimality, we have $H = \eta_{i,j}(D)$ such that

- D is a k-labeled graph arising in ϕ and $i, j \in [k]$,
- $E(C) \not\subseteq E(D)$.

It follows that $E(C) \setminus E(D) \subseteq \mathcal{E}^H_{i,j}$. Let $\mathcal{P} := E(C) \cap E(D)$ and $\mathcal{Q} := E(C) \cap \mathcal{E}^H_{i,j}$. Observe that $\mathcal{P} \in \Pi(D)$ and $\mathcal{Q} \in \overline{\Pi}(D)$. By Lemma 4.69, the multigraph $\mathsf{aux}_D(\mathcal{P}) \uplus \mathsf{aux}_D(\mathcal{Q})$ contains a red-blue Eulerian trail. Since $\mathcal{A}_D \lesssim_D \Pi(D)$, there exists $\mathcal{P}^\star \in \mathcal{A}_D$ such that $\mathsf{aux}_D(\mathcal{P}^\star) \uplus \mathsf{aux}_D(\mathcal{Q})$ contains a red-blue Eulerian trail. As $\mathcal{Q} \subseteq \mathcal{E}^H_{i,j}$, we deduce that, for every $\ell \in [k] \setminus \{i, j\}$, we have $\mathsf{deg}_{\mathsf{aux}_D(\mathcal{P}^\star)}(v_\ell) = 0$ and $\mathsf{deg}_{\mathsf{aux}_H(\mathcal{P}^\star)}(v_i) = \mathsf{deg}_{\mathsf{aux}_H(\mathcal{P}^\star)}(v_j)$.

For the other direction, suppose that the latter condition holds. Let $\mathcal{Q}$ be the graph on the vertex set $V(G)$ such that it contains exactly $\mathsf{deg}_{\mathsf{aux}_H(\mathcal{P})}(v_i)$ many edges between the set of vertices labeled i and the set of vertices labeled j. Observe that $\mathsf{aux}_H(\mathcal{Q})$ consists of $\mathsf{deg}_{\mathsf{aux}_H(\mathcal{P})}(v_i)$ many edges between v_i and v_j. Therefore, by Lemma 4.71, $\mathsf{aux}_H(\mathcal{P}) \uplus \mathsf{aux}_H(\mathcal{Q})$ admits a red-blue Eulerian trail. Then, by Lemma 4.70, there exists $\mathcal{Q}^\star \in \overline{\Pi}(H)$ such that $\mathcal{P}$ and $\mathcal{Q}^\star$ form a Hamiltonian cycle, as required.

Running time. Let H be a k-labeled graph arising in ϕ. Observe that if $H = i(v)$ or $H = \rho_{i \to j}(D)$, then we compute $\mathcal{A}_H$ in time $O(1)$. By Lemma 4.68, for every $\mathcal{A} \subseteq \Pi(H)$, we can compute $\mathsf{reduce}_H(\mathcal{A})$ in time $O(|\mathcal{A}| \cdot nk^2 \log_2(nk))$. Observe that, for every k-labeled graph D arising in ϕ and such that $\mathcal{A}_D$ is computed before $\mathcal{A}_H$, we have $|\mathcal{A}_D| \leq n^k \cdot 2^{k(\log_2(k)+1)}$. It follows that:

- If $H = D \oplus F$, then we have $|\mathcal{A}_D \otimes \mathcal{A}_F| \leq n^{2k} \cdot 2^{2k(\log_2(k)+1)}$. Thus, we can compute $\mathcal{A}_H := \mathsf{reduce}_H(\mathcal{A}_D \otimes \mathcal{A}_F)$ in time
$$O(n^{2k+1} \cdot 2^{2k(\log_2(k)+1)} \cdot k^2 \log_2(nk)).$$
- If $H = \eta_{i,j}(D)$, then we can compute $\mathcal{A}_H$ in time
$$O(n^{k+4} \cdot 2^{k(\log_2(k)+1)} \cdot k^2 \log_2(nk)).$$
First observe that, for every partial solution $\mathcal{P}$ of H, we have $|\mathcal{P} + (i, j)| \leq n^2$ and we can compute the set $\mathcal{P} + (i, j)$ in time $O(n^2)$. Moreover, by Lemma 4.68, for every $\ell \in \{0, \ldots, n-1\}$, we have $|\mathcal{A}^\ell_D| \leq n^k \cdot 2^{k(\log_2(k)+1)}$ and thus, we deduce that $|\mathcal{A}^\ell_D + (i, j)| \leq n^{k+2} \cdot 2^{k(\log_2(k)+1)}$ and that $\mathcal{A}^{\ell+1}_D$ can be computed in time $O(n^{k+3} \cdot 2^{k(\log_2(k)+1)} \cdot k^2 \log_2(nk))$. Thus, we can compute the sets $\mathcal{A}^1_D, \ldots, \mathcal{A}^n_D$ in time $O(n^{k+4} \cdot 2^{k(\log(k)+1)} \cdot k^2 \log_2(nk))$. The running time to compute $\mathcal{A}_H$ from $\mathcal{A}_D$ follows from $|\mathcal{A}^0_D \cup \cdots \cup \mathcal{A}^n_D| \leq n^{k+1} \cdot 2^{2k(\log_2(k)+1)}$.

Since ϕ uses at most $O(n)$ disjoint union operations and $O(nk^2)$ unary operations, we deduce that the total running time of our algorithm is
$$O(n^{2k+5} \cdot 2^{2k(\log_2(k)+1)} \cdot k^4 \log_2(nk)).$$

□

4.4.5 Directed Hamitonian cycle

In this subsection, we explain how to adapt our approach for directed graphs. A *k-labeled digraph* is a pair (G, lab_G) of a digraph G and a function lab_G from $V(G)$ to $[k]$. Directed clique-width is also defined in [32], and it is based on the four operations, where the three operations of creating a digraph, taking the disjoint union of two labeled digraphs, and relabeling a digraph are the same, and the operation of adding edges is replaced with the following:

- For a labeled digraph G and distinct labels $i, j \in [k]$, add all the non-existent arcs from vertices with label i to vertices with label j (so we do not add arcs of both directions altogether).

A *directed clique-width k-expression* and *directed clique-width* are defined in the same way. A *directed Hamiltonian cycle* of a digraph G is a directed cycle containing all the vertices of G. The Directed Hamiltonian Cycle problem asks, for a given digraph G, whether G contains a directed Hamiltonian cycle or not.

With a similar approach, we can show the following.

Theorem 4.79. *There exists an algorithm that, given an n-vertex digraph G and a directed clique-width k-expression of G, solves* Directed Hamiltonian Cycle *in time $n^{O(k)}$.*

For Directed Hamiltonian Cycle, auxiliary graphs should be directed graphs. We define partial solutions and auxiliary multigraphs similarly, at the exception that a directed maximal path (resp. H-path) from a vertex labeled i to a vertex labeled j in a partial solution (resp. complement solution) corresponds to an arc (v_i, v_j) in its auxiliary multigraph.

Let G be an n-vertex directed graph and ϕ be a directed irredundant k-expression of G. Similarly to the proof of Theorem 4.78, for every k-labeled graph H arising in ϕ, we recursively compute a set $\mathcal{A}_H$ of small size such that $\mathcal{A}_H$ represents $\Pi(H)$ which is the set of all partial solutions of H.

It is not hard to see that Lemmas 4.69 and 4.70 hold also in the directed case. That is, we have an equivalence between directed Hamiltonian cycles in graphs and directed red-blue Eulerian trails in directed multigraphs. Thus, to adapt our ideas for undirected graphs, we only need to characterize the directed multigraphs which admit a red-blue Eulerian trail. Fleischner [56, Theorem VI.17] gave such a characterization for directed graphs without loops and multiple arcs, but the proof can easily be adapted for directed multigraphs.

Let M be a directed multigraph whose arcs are colored red or blue, and let R and B be respectively its set of red and blue arcs. We denote by M^* the digraph derived from M with $V(M^*) := \{v_1, v_2 : v \in V(M)\}$ and $E(M^*) := \{(v_1, w_2) : (v, w) \in B\} \cup \{(v_2, w_1) : (v, w) \in R\}$. For a digraph G and a vertex v of G, we denote by $\mathsf{deg}^+_G(v)$ and $\mathsf{deg}^-_G(v)$, respectively, the outdegree and indegree of v in G.

Lemma 4.80 (Fleischner [56])**.** *Let M be a directed multigraph whose arcs are colored red or blue. The following are equivalent.*

1. *M has a red-blue Eulerian trail.*
2. *M^* has an Eulerian trail.*
3. *The underlying undirected graph of M^* has at most one connected component containing an edge, and, for each vertex v of M^*, $\mathsf{deg}^+_{M^*}(v) = \mathsf{deg}^-_{M^*}(v)$.*

In (3), the condition that "for each vertex v of M^*, $\mathsf{deg}^+_{M^*}(v) = \mathsf{deg}^-_{M^*}(v)$" can be translated to that, for each vertex v of M, the number of blue incoming arcs is the same as the number of red

outgoing arcs, and the number of red incoming arcs is the same as the number of blue outgoing arcs. However, an important point is that instead of having that the underlying undirected graph of M^* has at most one connected component containing an edge, the condition that "the underlying graph of M is connected" or "M is strongly connected" is not sufficient, because the connectedness of M^* depends on the colors of arcs. We give an example[9]. Let G be a graph on $\{x, y, z\}$ with red arcs $(x, y), (y, z)$ and blue arcs $(z, y), (y, x)$. Even though G is strongly connected, it does not have a red-blue Eulerian trail, and one can check that M^* has two connected components containing an edge.

To decide whether the underlying undirected graph of M^* has at most one connected component containing an edge, multiple arcs are useless. So, it is enough to keep one partial solution $\mathcal{P}$ for each degree sequence in $\mathsf{aux}_H(\mathcal{P})$ and for each set $\{\mathsf{mult}_{\mathsf{aux}_H(\mathcal{P})}(e) : e \in E(\mathsf{aux}_H(\mathcal{P}))\}$.

Let $\simeq$ be the equivalence relation on $\Pi(H)$ such that $\mathcal{P}_1 \simeq \mathcal{P}_2$ if the following are satisfied:

- for every pair (v, w) of vertices in $\{v_1, \ldots, v_k\}$, there exists an arc from v to w in $\mathsf{aux}_H(\mathcal{P}_1)$ if and only if there exists one in $\mathsf{aux}_H(\mathcal{P}_2)$,
- for every vertex v in $\{v_1, \ldots, v_k\}$, $\mathsf{deg}^+_{\mathsf{aux}_H(\mathcal{P}_1)}(v) = \mathsf{deg}^+_{\mathsf{aux}_H(\mathcal{P}_2)}(v)$ and $\mathsf{deg}^-_{\mathsf{aux}_H(\mathcal{P}_1)}(v) = \mathsf{deg}^-_{\mathsf{aux}_H(\mathcal{P}_2)}(v)$.

From Lemma 4.80 and the definition of M^*, we deduce the following fact.

Fact 4.81. *Let $\mathcal{P}_1, \mathcal{P}_2 \in \Pi(H)$. If $\mathcal{P}_1 \simeq \mathcal{P}_2$, then $\mathcal{P}_1$ is part of a directed Hamiltonian cycle in G if and only if $\mathcal{P}_2$ is part of a directed Hamiltonian cycle in G.*

From the definition of $\simeq$, one can show that $|\Pi(H)/\simeq| \leq n^{2k} \cdot 2^{k^2}$. Thus we can follow the lines of the proof for undirected graphs, and easily deduce that one can solve Directed Hamiltonian Cycle in time $n^{O(k)}$, where k is the directed clique-width of the given digraph.

4.4.6 Conclusion

In Subsection 2.6.2, we have already raised some open problems concerning problems which are W[1]-hard parameterized by clique-width (see Open question 2.66).

We conclude with one explicit question. A digraph D is an *out-tree* if D is an oriented tree (an undirected tree with orientations on edges) with only one vertex of indegree zero (called the root). The vertices of out-degree zero are called leaves of D. The *Min Leaf Out-Branching* problem asks for a given digraph D and an integer ℓ, whether there is a spanning out-tree of D with at most ℓ leaves. This problem generalizes Hamiltonian Path (and also Hamiltonian Cycle) by taking $\ell = 1$. Ganian, Hliněný, and Obdržálek [66] showed that there is an $n^{2^{O(k)}}$-time algorithm for solving Min Leaf Out-Branching problem, when a clique-width k-expression of a digraph D is given. This raises naturally the following question.

Open Question 4.82. *Is the problem* Minimum Leaf Spanning Tree *solvable in time $n^{O(k)}$ when a clique-width k-expression of the input digraph is given?*

[9] This example was suggested by an anonymous reviewer.

Chapter 5

Counting problems

In this chapter, we present our results on the counting problems. We start by an introduction concerning the complexity of counting problems parameterized by width measures. In Section 5.2, we prove that one can compute in polynomial time the number of minimal transversals of β-acyclic hypergraphs. In Subsection 5.2.3, we show that, as a corollary, we can count in polynomial time the minimal dominating sets of strongly chordal graphs and we discuss about the possible extensions of this corollary. In Subsection 5.2.4, we conclude this chapter by some open questions concerning the counting of minimal transversals and in particular, we discuss about how some existing parameters on hypergraphs could be used to extend our result.

5.1 Introduction

Even through the parameterized complexity of counting problems are less studied than those of decision problems, some results are known concerning width measures. Arnborg, Lagergren, and Seese [3] extended Courcelle's theorem by showing that every counting problem expressible in MSO_2 are FPT parameterized by the tree-width of the input graph. Makowsky [105] proved that evaluating the Tutte polynomial is FPT when parameterized by tree-width. Moreover, Courcelle, Makowsky, and Rotics [33] generalized Theorem 2.52 by showing that every counting problem expressible in MSO_1 is FPT parameterized by the clique-width of the input graph.

Efficient algorithms are known for the counting variants of NP-hard problems parameterized by tree-width. For example, Bodlaender et al. [9] designed a framework called *determinant approach* which provides $2^{O(\mathsf{tw}(G))} \cdot n$ time algorithms for the counting variants of many connectivity problems such as Hamiltonian Cycle and Steiner Tree.

Recently, Golovach et al. [73, Theorem 36] proved the following theorem about counting the *1-minimal* and *1-maximal* (σ, ρ)-dominating sets with the d-neighbor-width as parameter. For a graph G, a (σ, ρ)-dominating set $D \subseteq V(G)$ of G is *1-maximal* (resp. *1-minimal*) if for all $u \in V(G) \setminus D$ (resp. $u \in D$), the set $D \cup u$ (resp. $D \setminus u$) is not a (σ, ρ)-dominating set.

Theorem 5.1 ([73]). *Let (σ, ρ) be a pair of finite or co-finite subsets of $\mathbb{N}$. Given an n-vertex graph G and $\mathcal{L}$ a rooted layout of G, we can count the 1-maximal (or 1-minimal) (σ, ρ)-dominating sets of G in time $O(\mathsf{nec}_d(\mathcal{L})^8 \cdot n^2)$ with d a constant depending on (σ, ρ).*

In particular, Theorem 5.1 implies that we can count the minimal dominating sets (or maximal independent sets) in time $n^{O(\mathsf{mim}(\mathcal{L}))}$ given a rooted layout $\mathcal{L}$ of the input graph.

Concerning intractability results, it is worth mentioning that Flum and Grohe [57] introduced the #W-hierarchy, an analog of the W-hierarchy for parameterized counting problems. They proved that counting paths (or cycles) of length k parameterized by k is #W[1]-complete. Consequently, it is unlikely that there exits an $f(k) \cdot n^{O(1)}$ time algorithm for counting the paths

(or the cycles) of length k of an n-vertex graph, for any function f, even through there is a $2^{O(k)} \cdot n^{O(1)}$ time algorithm for finding a path (or cycle) of length k [2].

5.2 Counting Minimal Transversals of β-acyclic Hypergraph

In this section, we show that we can count in polynomial time the minimal transversals of β-acyclic hypergraphs. The Transversal Enumeration problem (enumerating the minimal transversals) was already known to be tractable for this class [46] and computing the number of transversals is polynomial [15]. However, the complexity of computing the number of minimal transversals was still unknown. More precisely, we prove the following theorem:

Theorem 5.2. *The number of minimal transversals of a β-acyclic hypergraph can be computed in polynomial time.*

A direct consequence of our result is the following corollary concerning the counting of *minimal dominating sets* in a subclass of chordal graphs, called *strongly chordal* graphs.

Corollary 5.3. *The number of minimal dominating sets of a strongly chordal graph is computable in polynomial time.*

Notice that strongly chordal graphs have unbounded mim-width [108], thus Corollary 5.3 is not implied by Theorem 5.1.

Besides the polynomial time algorithm, the main contribution of this chapter is the modification of the framework considered in [21] in order to count minimal models. The techniques used in [73, 94] are based on structural restrictions and as shown in [21] cannot work for β-acyclic hypergraphs. Instead, Capelli showed in [21] how to construct, from the elimination ordering of a β-acyclic hypergraph associated with a boolean formula, a circuit whose satisfying assignments correspond to the models of the boolean formula. Such circuits are known as *decision Decomposable Negation Normal Form* in knowledge compilation. While the technique allows to count the models of non-monotone formulas, it cannot be used to count the minimal models. Indeed, the branchings of the constructed circuit do not allow to control the minimality. We overcome this difficulty by introducing the notion of *blocked transversals*, which corresponds roughly to the minimal transversals of a sub-hypergraph that are transversals of the whole hypergraph. We then show that blocked transversals can be used to control the minimality in the construction of the circuit. However, this control is only possible in the case of monotone Boolean formulas, corresponding to counting the minimal transversals of β-acyclic hypergraphs.

Because of technical definitions, we postpone the details of the algorithm in Subsection 5.2.2. The chapter is organized as follows. In Subsection 5.2.1, we present the concepts related to hypergraphs and transversals, then we define the concept of *blocked transversals* and we give some intermediate lemmas concerning this concept. In Subsection 5.2.2, we start by refining the decomposition of β-acyclic hypergraphs proposed in [21] to take into account *blocked transversals*. In Subsection 5.2.3, we show how to obtain Corollary 5.3 and we discuss about how it partially answers a conjecture raised by Kanté and Uno [96]. We conclude by discussing about the possible extensions of our result and in particular, we present two width measures on hypergraphs which could potentially be used to extend our result.

5.2.1 Definitions and notations

Hypergraphs

A hypergraph $\mathcal{H}$ is a collection of subsets of a finite ground set. The elements of $\mathcal{H}$ are called the *hyperedges* of $\mathcal{H}$ and the *vertex set* of $\mathcal{H}$ is $V(\mathcal{H}) := \bigcup_{e \in \mathcal{H}} e$.

Given $S \subseteq V(\mathcal{H})$, we denote by $\mathcal{H}[S]$ the *hypergraph induced by* S, that is, $\mathcal{H}[S] := \{e \cap S : e \in \mathcal{H}\}$. Any subset $\mathcal{H}'$ of $\mathcal{H}$ is called a *sub-hypergraph* of $\mathcal{H}$. Observe that if there exists $e \in \mathcal{H}$ such that $e \subseteq V(\mathcal{H}) \setminus S$, then $\varnothing \in \mathcal{H}[S]$. We do not enforce hypergraphs to have non-empty edges or to be non-empty. However, a hypergraph with an empty edge may behave counter-intuitively. In the following definitions, we explicitly explain the extremal cases where $\varnothing \in \mathcal{H}$ or $\mathcal{H} = \varnothing$.

Given a hypergraph $\mathcal{H}$ and a subset $S \subseteq V(\mathcal{H})$ of its vertex set, we denote by $\mathcal{H}(S)$ the set of hyperedges of $\mathcal{H}$ containing at least one vertex in S, that is, $\mathcal{H}(S) := \{e \in \mathcal{H} : S \cap e \neq \varnothing\}$; to ease notations, we write $\mathcal{H}(x)$ instead of $\mathcal{H}(\{x\})$ for $x \in V(\mathcal{H})$. Observe that $\mathcal{H}(x)$ is the set of hyperedges containing x. We use this notation on sub-hypergraphs of $\mathcal{H}$, but also on sets of (minimal) transversals of $\mathcal{H}$. For the hypergraph $\mathcal{H}$ shown in Figure 5.1, $\mathcal{H}(\{d, x\})$ is the hypergraph $\{\{b, x\}, \{c, x\}, \{c, d\}\}$.

Given a hypergraph $\mathcal{H}$, a walk between two distinct edges e_1 and e_k is a sequence

$$(e_1, x_1, e_2, x_2, \ldots, e_{k-1}, x_{k-1}, e_k)$$

such that $x_i \in e_i \cap e_{i+1}$ for all $1 \leq i \leq k-1$. Notice that, $(e_k, x_{k-1}, e_{k-1}, \ldots, x_2, e_2, x_1, e_1)$ is also a walk between e_k and e_1. A maximal set of edges of $\mathcal{H}$ that are pairwise connected by a walk is called a *connected component* of $\mathcal{H}$. It is worth noticing that if $C_1, \ldots, C_k$ are the connected components of $\mathcal{H}$, then $V(C_i) \cap V(C_j) = \varnothing$ for distinct i, j in $\{1, \ldots, k\}$.

A hypergraph $\mathcal{H}$ is said *β-acyclic* if there exists an ordering $x_1, \ldots, x_n$ of $V(\mathcal{H})$ such that for each $1 \leq i \leq n$, the set $\{e \cap \{x_i, \ldots, x_n\} : e \in \mathcal{H},\ x_i \in e\}$ is linearly ordered by inclusion. Such an ordering is called a *β-elimination ordering*. It is well-known that every sub-hypergraph $\mathcal{H}'$ of a β-acyclic hypergraph $\mathcal{H}$ is β-acyclic as well (see for instance [45]).

Transversals

Let $\mathcal{H}$ be a hypergraph. A transversal for $\mathcal{H}$ is a subset $T \subseteq V(\mathcal{H})$ such that for every $e \in \mathcal{H}$, $T \cap e \neq \varnothing$. We denote by $\mathsf{tr}(\mathcal{H})$ the set of transversals of $\mathcal{H}$. Observe that if $\varnothing \in \mathcal{H}$, then $\mathsf{tr}(\mathcal{H}) = \varnothing$ as for every $T \subseteq V(\mathcal{H})$, $\varnothing \cap T = \varnothing$, so T cannot be a transversal of $\mathcal{H}$. Finally, observe that if $\mathcal{H} = \varnothing$, then $\mathsf{tr}(\mathcal{H}) = \{\varnothing\}$.

A transversal T of $\mathcal{H}$ is *minimal* if and only if for every $x \in T$, it holds that $T \setminus \{x\} \notin \mathsf{tr}(\mathcal{H})$. A hyperedge e such that $e \cap T = \{x\}$ is said to be *private for x w.r.t. T*. When T is clear from the context, we may refer to such hyperedges as simply privates for x. The following fact follows directly from the definitions:

Fact 5.4. *T is a minimal transversal of a hypergraph $\mathcal{H}$ if and only if T is a transversal of $\mathcal{H}$ and each vertex $x \in T$ has a private.*

We denote by $\mathsf{mtr}(\mathcal{H})$ the set of minimal transversals of $\mathcal{H}$. Again, observe that if $\mathcal{H} = \varnothing$, then $\mathsf{mtr}(\mathcal{H}) = \{\varnothing\}$.

Figure 5.1 depicts a hypergraph together with its minimal transversals.

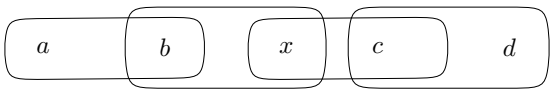


Figure 5.1 – A hypergraph $\mathcal{H} = \{\{a, b\}, \{b, x\}, \{x, c\}, \{c, d\}\}$ having 4 minimal transversals. We have $\mathsf{mtr}(\mathcal{H}) = \{\{a, x, d\}, \{a, x, c\}, \{b, x, d\}, \{b, c\}\}$.

It is worth noticing that since the sets $\mathsf{tr}(\mathcal{H})$ and $\mathsf{mtr}(\mathcal{H})$ are sets of subsets of $2^{V(\mathcal{H})}$, they may be seen as hypergraphs on $V(\mathcal{H})$. Thus, we will sometimes use the notations $\mathsf{tr}(\mathcal{H})(S)$ (resp. $\mathsf{mtr}(\mathcal{H})(S)$) which refer to the transversals (resp. minimal transversals) T of $\mathcal{H}$ such that $S \cap T \neq \varnothing$.

Blocked transversals

Our algorithm uses a dynamic programming approach by finding a relation between the number of minimal transversals of a β-acyclic hypergraph $\mathcal{H}$ with the number of minimal transversals of some specific subhypergraphs of $\mathcal{H}$. However, it is not possible to directly relate these quantities together. To illustrate this fact, let $\mathcal{H}$ be the hypergraph depicted in Figure 5.1. The minimal transversals of $\mathcal{H}$ containing x are $\{a, x, c\}, \{a, x, d\}$ and $\{b, x, d\}$. Observe that removing x from these sets directly yields a minimal transversal of $\mathcal{H} \setminus \mathcal{H}(x) = \{\{a, b\}, \{c, d\}\}$. However, adding x to a minimal transversal of $\mathcal{H} \setminus \mathcal{H}(x)$ does not give necessarily a minimal transversal of $\mathcal{H}$. For example, we have $\{b, c\} \in \mathsf{mtr}(\mathcal{H} \setminus \mathcal{H}(x))$, but $\{b, x, c\} \notin \mathsf{mtr}(\mathcal{H})$.

In fact, we can show in general that $T \cup x \in \mathsf{mtr}(\mathcal{H})$ if and only if T is a minimal transversal of $\mathcal{H} \setminus \mathcal{H}(x)$ and T is not a transversal of $\mathcal{H}$. Consequently, the number of minimal transversals of $\mathcal{H}$ containing x is

$$\#\mathsf{mtr}(\mathcal{H} \setminus \mathcal{H}(x)) - \#(\mathsf{tr}(\mathcal{H}) \cap \mathsf{mtr}(\mathcal{H} \setminus \mathcal{H}(x))). \tag{5.1}$$

We can infer from this fact a recursive formula to compute $\#\mathsf{mtr}(\mathcal{H})$. In the general case, using this formula as it is will lead to the computation of an exponential number of terms. In this subsection, we will make this relation more precise by introducing the notion of *blocked transversal*. In the next subsection, we will show how we can use this relation to evaluate the number of minimal transversals of a β-acyclic hypergraph with only a polynomial number of intermediate values.

Given a hypergraph $\mathcal{H}$, a sub-hypergraph $\mathcal{H}' \subseteq \mathcal{H}$ and $B \subseteq V(\mathcal{H})$, we define the *$B$-blocked transversals* of $\mathcal{H}'$ to be the transversals T of $\mathcal{H}'$ such that each vertex x of T has a private in $\mathcal{H}' \setminus \mathcal{H}'(B)$. In particular, if y is a vertex of B, then y cannot be in a B-blocked transversal of $\mathcal{H}'$. Observe also that if $y \in V(\mathcal{H}) \setminus V(\mathcal{H}')$, then y cannot be in a B-blocked transversal of $\mathcal{H}'$.

Fact 5.5. *Given a sub-hypergraph $\mathcal{H}'$ of a hypergraph $\mathcal{H}$ and $B \subseteq V(\mathcal{H})$, T is a B-blocked transversal of $\mathcal{H}'$ if and only if T belongs to $\mathsf{tr}(\mathcal{H}') \cap \mathsf{mtr}(\mathcal{H}' \setminus \mathcal{H}'(B))$.*

Proof. If T is a transversal of $\mathcal{H}'$ such that each vertex x of T has a private in $\mathcal{H}' \setminus \mathcal{H}'(B)$, then $T \subseteq V(\mathcal{H}' \setminus \mathcal{H}'(B))$ and is a transversal of $\mathcal{H}' \setminus \mathcal{H}'(B)$. Thus by Fact 5.4 T is a minimal transversal of $\mathcal{H}' \setminus \mathcal{H}'(B)$. Conversely, if T is a minimal transversal of $\mathcal{H}' \setminus \mathcal{H}'(B)$, then by Fact 5.4 each vertex x has a private in $\mathcal{H}' \setminus \mathcal{H}'(B)$. If in addition it belongs to the set of transversals of $\mathcal{H}'$, then it is a B-blocked transversal of $\mathcal{H}'$ by definition. □

As an example, let $\mathcal{H}$ be the hypergraph depicted in Figure 5.1, $\mathcal{H}' = \mathcal{H}$ and $B = \{x\}$. The only B-blocked transversal of $\mathcal{H}$ is $\{b, c\}$. While $\{b, d\}$ is a minimal transversal of $\mathcal{H} \setminus \mathcal{H}(x)$, it is not a B-blocked transversal as the hyperedge $\{x, c\}$ does not intersect $\{b, d\}$.

We call the set $\mathcal{H}'(B)$ the *blocked hyperedges*. Intuitively, $\mathcal{H}'(B)$ is the set of hyperedges that cannot be used as privates in a transversal of $\mathcal{H}'$. We denote by $\mathsf{btr}_B(\mathcal{H}')$ the set of B-blocked transversals of $\mathcal{H}'$. By Fact 5.5, we have

$$\mathsf{btr}_B(\mathcal{H}') := \mathsf{tr}(\mathcal{H}') \cap \mathsf{mtr}(\mathcal{H}' \setminus \mathcal{H}'(B)).$$

Observe that by definition, $\mathsf{mtr}(\mathcal{H}) = \mathsf{btr}_\varnothing(\mathcal{H})$. Moreover, if $\mathcal{H}(B) = \mathcal{H} \neq \varnothing$, then $\mathsf{btr}_{\mathcal{H}}(\mathcal{H}) = \varnothing$ as $\mathsf{mtr}(\varnothing) = \{\varnothing\}$ and $\varnothing \notin \mathsf{tr}(\mathcal{H})$. When $B = \{x\}$, we denote $\mathsf{btr}_B(\mathcal{H})$ by $\mathsf{btr}_x(\mathcal{H})$.

Given $S \subseteq V(\mathcal{H})$, we denote by $\mathsf{tr}(\mathcal{H}, S) := \{T \in \mathsf{tr}(\mathcal{H}) : T \subseteq S\}$. We extend this notation to mtr and btr as well. The following summarizes observations about blocked transversals.

Fact 5.6. *Let $\mathcal{H}'$ be a sub-hypergraph of a hypergraph $\mathcal{H}$ and $B, S \subseteq V(\mathcal{H})$. Then,*

1. $\mathsf{btr}_B(\mathcal{H}') = \mathsf{btr}_{B\cap V(\mathcal{H}')}(\mathcal{H}')$.

2. $\mathsf{btr}_B(\mathcal{H}', S) = \mathsf{btr}_B(\mathcal{H}', S \setminus B)$.

3. *If* $x \notin V(\mathcal{H}')$, *then* $\mathsf{btr}_B(\mathcal{H}', S) = \mathsf{btr}_B(\mathcal{H}', S \setminus \{x\})$.

Let us briefly explain how B-blocked transversals will be used in computing $\#\mathsf{mtr}(\mathcal{H})$. One checks easily that $\#\mathsf{mtr}(\mathcal{H}) = \#\mathsf{mtr}(\mathcal{H}, V(\mathcal{H}) \setminus \{x\}) + \#\mathsf{mtr}(\mathcal{H})(x)$. By Equation 5.1, $\#\mathsf{mtr}(\mathcal{H})(x) = \#\mathsf{mtr}(\mathcal{H} \setminus \mathcal{H}(x)) - \#\mathsf{btr}_x(\mathcal{H})$. Therefore, we have

$$\#\mathsf{mtr}(\mathcal{H}) = \#\mathsf{btr}_\varnothing(\mathcal{H}) = \#\mathsf{btr}_\varnothing(\mathcal{H}, V(\mathcal{H}) \setminus \{x\}) + \#\mathsf{btr}_\varnothing(\mathcal{H} \setminus \mathcal{H}(x)) - \#\mathsf{btr}_x(\mathcal{H}).$$

We will show in the next subsection that we can define, from the β-elimination ordering of a β-acyclic hypergraph $\mathcal{H}$, sub-hypergraphs $\mathcal{H}_1, \ldots, \mathcal{H}_q \subseteq \mathcal{H}$ and vertices $x_1, \ldots, x_q$ such that, for all $1 \leq i \leq q$, $\#\mathsf{btr}_{x_i}(\mathcal{H}_i)$ and $\#\mathsf{btr}_\varnothing(\mathcal{H}_i)$ can be computed in polynomial time if $\#\mathsf{btr}_{x_j}(\mathcal{H}_j)$ and $\#\mathsf{btr}_\varnothing(\mathcal{H}_j)$ are known for all $j < i$. As a consequence, one can compute $\#\mathsf{mtr}(\mathcal{H})$ by classical dynamic programming for any β-acyclic hypergraph.

The end of this subsection is dedicated to the proof of several crucial lemmas concerning recursive formulas for computing the number of blocked transversals, and that will be useful in our algorithm. We start by describing the blocked transversals of a hypergraph having more than one connected component.

Lemma 5.7. *Let* $\mathcal{H}$ *be a hypergraph,* $S, B \subseteq V(\mathcal{H})$ *and* $C_1, \ldots, C_k$ *the connected components of* $\mathcal{H}[S]$. *For each* $i \in [1, k]$, *let* $\mathcal{H}_i = \{e \in \mathcal{H} : e \cap S \in C_i\}$. *We have:*

$$\mathsf{btr}_B(\mathcal{H}, S) = \bigotimes_{i=1}^{k} \mathsf{btr}_B(\mathcal{H}_i, S).$$

Proof. Assume that $\varnothing \in \mathcal{H}[S]$, it means that there exists a hyperedge $e \in \mathcal{H}$ such that $e \cap S = \varnothing$. In this case, $\mathsf{btr}_B(\mathcal{H}, S) = \varnothing$. Moreover, there exists $i \in [1, k]$ such that $C_i = \{\varnothing\}$ and $e \in \mathcal{H}_i$. Thus, $\mathsf{btr}_B(\mathcal{H}_i, S) = \varnothing$ and the equality holds in this case.

Assume from now that $\varnothing \notin \mathcal{H}[S]$. Let $T \in \mathsf{btr}_B(\mathcal{H}, S)$. We show that for all $i \leq k$, $T_i = T \cap V(\mathcal{H}_i) \in \mathsf{btr}_B(\mathcal{H}_i, S)$. Let $e \in \mathcal{H}_i$. Since $e \in \mathcal{H}$, we have $e \cap T \neq \varnothing$. As $e \in \mathcal{H}_i$, we have $e \subseteq V(\mathcal{H}_i)$. Thus $e \cap T_i \neq \varnothing$, that is, $T_i \in \mathsf{tr}(\mathcal{H}_i, S)$. Moreover, let $y \in T_i$. By definition of T, there exists $e \in \mathcal{H} \setminus \mathcal{H}(B)$ such that e is private for y w.r.t. T. Observe that we have $T_i \subseteq S \cap V(\mathcal{H}_i) = V(C_i)$ since $T \subseteq S$. Thus, $y \in V(C_i)$ and $e \cap S \in C_i$, because C_i is a connected component. We can conclude that $e \in \mathcal{H}_i$ and e is private to y w.r.t. T_i and $\mathcal{H}_i \setminus \mathcal{H}(B) = \mathcal{H}_i \setminus \mathcal{H}_i(B)$. Thus T_i is a minimal transversal of $\mathcal{H}_i \setminus \mathcal{H}_i(B)$. That is $T_i \in \mathsf{btr}_B(\mathcal{H}_i, S)$.

Now let $T_1 \in \mathsf{btr}_B(\mathcal{H}_1, S), \ldots, T_k \in \mathsf{btr}_B(\mathcal{H}_k, S)$. We show that $T = \bigcup_{i=1}^k T_i \in \mathsf{btr}_B(\mathcal{H}, S)$. Let $e \in \mathcal{H}$. As $\mathcal{H} = \bigcup_{i=1}^k \mathcal{H}_i$, there exists i such that $e \in \mathcal{H}_i$. Thus $e \cap T_i \neq \varnothing$ and thus $e \cap T \neq \varnothing$, that is, $T \in \mathsf{tr}(\mathcal{H})$. It remains to show that $T \in \mathsf{mtr}(\mathcal{H} \setminus \mathcal{H}(B))$. Let $y \in T$. By definition of T, there exists i such that $y \in T_i$. Thus, there exists $e \in \mathcal{H}_i \setminus \mathcal{H}_i(B)$ that is private for y w.r.t. T_i. Moreover, since $\mathcal{H}_i(B) = \mathcal{H}_i \cap \mathcal{H}(B)$, we know that $e \notin \mathcal{H}(B)$. As $C_1, \ldots, C_k$ are the connected component of $\mathcal{H}[S]$, we have that, for every $j \neq i$, $V(C_i) \cap V(C_j) = \varnothing$. Moreover, for all $\ell \leq k$, we have $S \cap V(\mathcal{H}_\ell) = V(C_\ell)$. As $e \subseteq V(\mathcal{H}_i)$ and $T_j \subseteq S \cap V(\mathcal{H}_j)$, we have for every $j \neq i$:

$$e \cap T_j \subseteq V(\mathcal{H}_i) \cap S \cap V(\mathcal{H}_j) = V(C_i) \cap V(C_j) = \varnothing.$$

Thus, $e \cap T = e \cap T_i = \{y\}$. In other words, e is private for y w.r.t. T and $\mathcal{H} \setminus \mathcal{H}(B)$. That is $T \in \mathsf{btr}_B(\mathcal{H})$. □

We recall that $\mathsf{btr}_B(\mathcal{H}, S)(x)$ is the set of B-blocked transversals T of $\mathcal{H}$ such that $T \subseteq S$ and $x \in T$. The following lemma shows that for any B-blocked transversal $T \subseteq S$ of $\mathcal{H}$ containing x, we have $T \setminus x$ is a B-blocked transversal of $\mathcal{H} \setminus \mathcal{H}(x)$.

Lemma 5.8. *Let $\mathcal{H}$ be a hypergraph, $S, B \subseteq V(\mathcal{H})$ and $x \in S$. We have*

$$\mathsf{btr}_B(\mathcal{H}, S)(x) \subseteq \{\{x\}\} \bigotimes \mathsf{btr}_B(\mathcal{H} \setminus \mathcal{H}(x), S \setminus \{x\}).$$

Proof. Let $\mathcal{H}_1 := \mathcal{H} \setminus \mathcal{H}(x)$. Let $T \in \mathsf{btr}_B(\mathcal{H}, S)(x)$. By definition, $x \in T$ and $T \subseteq S$, thus we only have to show that $T' = T \setminus \{x\} \in \mathsf{btr}_B(\mathcal{H}_1)$. Let $e \in \mathcal{H} \setminus \mathcal{H}(x)$. Since T is a transversal of $\mathcal{H}$, there exists $y \in e \cap T$. Moreover, by definition, $x \notin e$, thus $y \in T'$, that is, T' is a transversal of $\mathcal{H} \setminus \mathcal{H}(x)$. It remains to show that T' is a minimal transversal of $\mathcal{H}_1 \setminus \mathcal{H}_1(B) = \mathcal{H} \setminus (\mathcal{H}(B) \cup \mathcal{H}(x))$. Let $y \in T'$. Since T is a minimal transversal of $\mathcal{H} \setminus \mathcal{H}(B)$, there exists $e \in \mathcal{H} \setminus \mathcal{H}(B)$ such that e is private for y w.r.t. T. Since $x \in T$, we have $x \notin e$, otherwise e would not be private for y w.r.t. T. Thus $e \in \mathcal{H} \setminus (\mathcal{H}(B) \cup \mathcal{H}(x))$, that is, e is private to y w.r.t. T' in $\mathcal{H}_1 \setminus \mathcal{H}_1(B)$. In other words, T' is a minimal transversal of $\mathcal{H}_1 \setminus \mathcal{H}_1(B)$ which concludes the proof. □

To complete the previous lemma, we show that for each B-blocked transversal $T \subseteq S$ of $\mathcal{H} \setminus \mathcal{H}(x)$, we have $T \cup \{x\}$ is a B-blocked transversal of $\mathcal{H}$ if and only if T is not a $(B \cup \{x\})$-blocked transversal of $\mathcal{H} \setminus (\mathcal{H}(B) \cap \mathcal{H}(x))$.

Lemma 5.9. *Let $\mathcal{H}$ be a hypergraph, $S, B \subseteq V(\mathcal{H})$ and $x \in S$. We have*

$$\big(\{\{x\}\} \bigotimes \mathsf{btr}_B(\mathcal{H}_1, S \setminus \{x\})\big) \setminus \mathsf{btr}_B(\mathcal{H}, S)(x) = \{\{x\}\} \bigotimes \mathsf{btr}_{B \cup \{x\}}(\mathcal{H}_2, S \setminus \{x\})$$

where $\mathcal{H}_1 := \mathcal{H} \setminus \mathcal{H}(x)$ and $\mathcal{H}_2 := \mathcal{H} \setminus (\mathcal{H}(B) \cap \mathcal{H}(x))$.

Proof. We prove the lemma by proving first the left-to-right inclusion (Claim 5.9.1) and then the right-to-left inclusion (Claim 5.9.2). But first, notice that $\mathcal{H}_1 \setminus \mathcal{H}_1(B) = \mathcal{H}_2 \setminus \mathcal{H}_2(B \cup \{x\}) = \mathcal{H} \setminus (\mathcal{H}(B) \cup \mathcal{H}(x))$ since $\mathcal{H}(B) \cup \mathcal{H}(x) = \mathcal{H}(B \cup \{x\})$.

Claim 5.9.1. *For every $T \in \big(\{\{x\}\} \bigotimes \mathsf{btr}_B(\mathcal{H}_1, S \setminus \{x\})\big) \setminus \mathsf{btr}_B(\mathcal{H}, S)(x)$, we have $T \setminus \{x\} \in \mathsf{btr}_{B \cup \{x\}}(\mathcal{H}_2, S \setminus \{x\})$.*

Proof. We start by proving that $T' = T \setminus \{x\}$ is in $\mathsf{tr}(\mathcal{H}_2)$. Assume towards a contradiction that $T' \notin \mathsf{tr}(\mathcal{H}_2)$, i.e., there exists $e \in \mathcal{H}_2$ such that $e \cap T' = \varnothing$. We prove that it implies $T \in \mathsf{btr}_B(\mathcal{H}, S)(x)$. First, observe that $T \in \mathsf{tr}(\mathcal{H})$, since $T' \in \mathsf{tr}(\mathcal{H}_1) = \mathsf{tr}(\mathcal{H} \setminus \mathcal{H}(x))$ and $T = T' \cup \{x\}$. Thus, we have $e \cap T = \{x\}$ and $e \in \mathcal{H}(x)$. As $e \in \mathcal{H}_2 = \mathcal{H} \setminus (\mathcal{H}(B) \cap \mathcal{H}(x))$, we have $e \in \mathcal{H} \setminus \mathcal{H}(B)$ and then e is a private hyperedge for x w.r.t. T and $\mathcal{H} \setminus \mathcal{H}(B)$. Furthermore, every vertex in T' has a private hyperedge w.r.t. T' and $\mathcal{H}_1 \setminus \mathcal{H}_1(B) = \mathcal{H} \setminus (\mathcal{H}(B) \cup \mathcal{H}(x))$ since $T' \in \mathsf{mtr}(\mathcal{H}_1 \setminus \mathcal{H}_1(B))$. Thus, every vertex in T' has a private hyperedge w.r.t. T and $\mathcal{H} \setminus \mathcal{H}(B)$. As $T \in \mathsf{tr}(\mathcal{H})$, we can conclude that $T \in \mathsf{mtr}(\mathcal{H} \setminus \mathcal{H}(B))$. Finally, we have $T \subseteq S$ by assumption. Therefore $T \in \mathsf{btr}_B(\mathcal{H}, S)(x)$ which is a contradiction. Thus, $T' \in \mathsf{tr}(\mathcal{H}_2)$.

We now prove that $T' \in \mathsf{mtr}(\mathcal{H}_2 \setminus \mathcal{H}_2(B \cup \{x\}))$, that is, we prove the minimality of T' in $\mathcal{H}_2 \setminus \mathcal{H}_2(B \cup \{x\})$. Let $y \in T'$. Since $T \in \mathsf{btr}_B(\mathcal{H}_1)$, there exists $f \in \mathcal{H}_1 \setminus \mathcal{H}_1(B)$ such that $f \cap T = \{y\}$. Since $\mathcal{H}_1 \setminus \mathcal{H}_1(B) = \mathcal{H}_2 \setminus \mathcal{H}_2(B \cup \{x\})$, every $y \in T'$ have a private hyperedge in $\mathcal{H}_2 \setminus \mathcal{H}_2(B \cup \{x\})$, that is $T' \in \mathsf{mtr}(\mathcal{H}_2 \setminus \mathcal{H}_2(B \cup \{x\}))$. As $T' \subseteq S \setminus \{x\}$, we can conclude that $T' \in \mathsf{btr}_{B \cup \{x\}}(\mathcal{H}_2, S \setminus \{x\})$. □

Claim 5.9.2. *For every $T \in \{x\} \bigotimes \mathsf{btr}_{B \cup \{x\}}(\mathcal{H}_2, S \setminus \{x\})$, we have $T \in \{x\} \bigotimes \mathsf{btr}_B(\mathcal{H}_1, S \setminus \{x\}) \setminus \mathsf{btr}_B(\mathcal{H}, S)(x)$.*

Proof. We start by proving that $T' = T \setminus \{x\}$ is in $\mathsf{btr}_B(\mathcal{H}_1, S \setminus \{x\})$. First, we show that T' is a transversal of $\mathcal{H}_1$. Let $e \in \mathcal{H}_1$. By definition of $\mathcal{H}_1$, $x \notin e$, thus $e \in \mathcal{H}_2$ as well. Therefore $e \cap T' \neq \varnothing$. We now prove that T' is minimal in $\mathcal{H}_1 \setminus \mathcal{H}_1(B)$. As $T' \in \mathsf{mtr}(\mathcal{H}_2 \setminus \mathcal{H}_2(B \cup \{x\}))$, every vertex in T' has a private hyperedge in $\mathcal{H}_2 \setminus \mathcal{H}_2(B \cup \{x\})$. Moreover, recall that $\mathcal{H}_2 \setminus \mathcal{H}_2(B \cup \{x\}) = \mathcal{H}_1 \setminus \mathcal{H}_1(B)$. Thus, T' is minimal in $\mathcal{H}_1 \setminus \mathcal{H}_1(B)$. As $T' \subseteq S \setminus \{x\}$, we have $T' \in \mathsf{btr}_B(\mathcal{H}_1, S \setminus \{x\})$.

We finish the proof by showing that $T \notin \mathsf{btr}_B(\mathcal{H}, S)(x)$. In order to prove it, we show that there are no private hyperedges for x w.r.t. T and $\mathcal{H} \setminus \mathcal{H}(B)$. Indeed, since $T' \in \mathsf{tr}(\mathcal{H}_2)$, every hyperedge in $\mathcal{H}_2$ contains a vertex in T'. By definition of $\mathcal{H}_2$, we have $\mathcal{H} \setminus \mathcal{H}(B) \subseteq \mathcal{H}_2$, thus for every hyperedge e in $\mathcal{H} \setminus \mathcal{H}(B)$, we have $e \cap T \neq \{x\}$, i.e., e is not a private for x. □

By Claim 5.9.1 and Claim 5.9.2 we can conclude the lemma. □

Finally, we characterize the number of B-blocked transversals of a hypergraph that do not contain a given vertex. We use the symbol $\uplus$ for the disjoint union of sets.

Lemma 5.10. *Let $\mathcal{H}$ be a hypergraph, $S, B \subseteq V(\mathcal{H})$ and $x \in S$. We have*

$$\mathsf{btr}_B(\mathcal{H}, S) = \mathsf{btr}_B(\mathcal{H}, S)(x) \uplus \mathsf{btr}_B(\mathcal{H}, S \setminus \{x\}).$$

Proof. Let $T \subseteq S$ be a B-blocked transversal of $\mathcal{H}$. Thus, either we have $x \in T$ and then $T \in \mathsf{btr}_B(\mathcal{H}, S)(x)$, or $x \notin T$ and then $T \subseteq S \setminus \{x\}$, i.e., $T \in \mathsf{btr}_B(\mathcal{H}, S \setminus \{x\})$. Since, the two cases are exclusive, we can conclude that $\mathsf{btr}_B(\mathcal{H}, S)$ is the disjoint union of $\mathsf{btr}_B(\mathcal{H}, S)(x)$ and of $\mathsf{btr}_B(\mathcal{H}, S \setminus \{x\})$. □

A direct consequence of Lemma 5.8, Lemma 5.9 and Lemma 5.10 is the following equality characterizing the number of B-blocked transversals of $\mathcal{H}$. This will be a crucial step in our dynamic programming scheme.

Theorem 5.11. *Let $\mathcal{H}$ be a hypergraph, $S, B \subseteq V(\mathcal{H})$ and $x \in S$. We have*

$$\#\mathsf{btr}_B(\mathcal{H}, S) = \#\mathsf{btr}_B(\mathcal{H}, S \setminus \{x\}) + \#\mathsf{btr}_B(\mathcal{H}_1, S \setminus \{x\}) - \#\mathsf{btr}_{B \cup \{x\}}(\mathcal{H}_2, S \setminus \{x\})$$

where $\mathcal{H}_1 := \mathcal{H} \setminus \mathcal{H}(x)$ and $\mathcal{H}_2 := \mathcal{H} \setminus (\mathcal{H}(B) \cap \mathcal{H}(x))$.

Proof. By Lemma 5.10, $\#\mathsf{btr}_B(\mathcal{H}, S) = \#\mathsf{btr}_B(\mathcal{H}, S \setminus \{x\}) + \#\mathsf{btr}_B(\mathcal{H}, S)(x)$. By Lemma 5.8 and Lemma 5.9, $\{\{x\}\} \otimes \mathsf{btr}_B(\mathcal{H}_1, S \setminus \{x\}) = \mathsf{btr}_B(\mathcal{H}, S)(x) \uplus \{\{x\}\} \otimes \mathsf{btr}_{B \cup \{x\}}(\mathcal{H}_2, S \setminus \{x\})$. Hence, $\#\mathsf{btr}_B(\mathcal{H}, S)(x) = \#\mathsf{btr}_B(\mathcal{H}_1, S \setminus \{x\}) - \#\mathsf{btr}_{B \cup \{x\}}(\mathcal{H}_2, S \setminus \{x\})$. Therefore, the claimed equality holds. □

5.2.2 Counting the minimal transversals of β-acyclic hypergraphs

In this subsection, we fix $\mathcal{H}$ a β-acyclic hypergraph, $\leq$ a β-elimination ordering of its vertices and we let $\leq_{\mathcal{H}}$ the induced lexicographic ordering on the hyperedges, i.e., $e \leq_{\mathcal{H}} f$ if $\mathsf{min}(e \triangle f) \in e$. We denote by $\mathcal{H}_e^x$ the sub-hypergraph of $\mathcal{H}$ formed by the hyperedges $f \in \mathcal{H}$ such that there exists a walk from f to e going only through hyperedges smaller than e and vertices smaller than x. For an example, take the β-elimination ordering a, b, x, c, d of the hypergraph in Figure 5.1 and the induced ordering $\{a, b\}, \{b, x\}, \{x, c\}, \{c, d\}$ on $\mathcal{H}$. For $e = \{x, c\}$, the hypergraph $\mathcal{H}_e^x$ is composed of the hyperedges $\{b, x\}$ and $\{x, c\}$.

For a vertex x of $V(\mathcal{H})$, we write $[\leq x]$, $[< x]$ and $[\geq x]$ for, respectively, $\{y \in V(\mathcal{H}) : y \leq x\}$, $\{y \in V(\mathcal{H}) : y \leq x \wedge y \neq x\}$ and $\{y \in V(\mathcal{H}) : x \leq y\}$. Moreover, we write $\mathcal{H}[\leq x]$, $\mathcal{H}[< x]$ and $\mathcal{H}[\geq x]$ instead of, respectively, $\mathcal{H}\big[[\leq x]\big]$, $\mathcal{H}\big[[< x]\big]$ and $\mathcal{H}\big[[\geq x]\big]$.

Decomposition of β-acyclic hypergraphs

The following two lemmas have been proven in [21, Section III-A].

Lemma 5.12 (Theorem 3 in [21]). *For every hyperedge $e \in \mathcal{H}$, and $x \in V(\mathcal{H})$, we have $V(\mathcal{H}_e^x) \cap [\geq x] \subseteq e$.*

Lemma 5.13 (Lemma 2 in [21]). *Let e and f be two hyperedges of $\mathcal{H}$ such that $e \leq_{\mathcal{H}} f$, and let x and y be vertices of $\mathcal{H}$ such that $x \leq y$. If $V(\mathcal{H}_e^x) \cap V(\mathcal{H}_f^y) \cap [\leq x] \neq \varnothing$, then $\mathcal{H}_e^x \subseteq \mathcal{H}_f^y$.*

We prove a lemma on the decomposition of $\mathcal{H}_e^x$ graphs that will be used with Lemma 5.7 to propagate the dynamic programming algorithm.

Lemma 5.14. *Let $x \in V(\mathcal{H})$, $e \in \mathcal{H}$ and $S \subseteq [\geq x]$. Let*

$$\mathcal{H}' := \begin{cases} \mathcal{H}_e^x & \text{if } S = \varnothing, \\ \mathcal{H}_e^x \setminus \left(\bigcap_{w \in S} \mathcal{H}_e^x(w)\right) & \text{otherwise.} \end{cases}$$

For every connected component C of $\mathcal{H}'[< x]$ different from $\{\varnothing\}$, there exists $y < x$ and $f \leq_{\mathcal{H}} e$ such that $C = \mathcal{H}_f^y[\leq y]$ and $\mathcal{H}_f^y = \{g \in \mathcal{H}' \,:\, g \cap [< x] \in C\}$.

Proof. Let $y = \mathsf{max}(V(C))$ and $f = \mathsf{max}\{g \in \mathcal{H}' \,:\, g \cap [< x] \in C\}$. We show that $\mathcal{H}_f^y = \{g \in \mathcal{H}' \,:\, g \cap [< x] \in C\}$.

First, we observe that $\mathcal{H}_f^y \subseteq \mathcal{H}'$. If $S = \varnothing$, it follows from Lemma 5.13 because, in this case, $\mathcal{H}' = \mathcal{H}_e^x$. Suppose that $S \neq \varnothing$, by definition of $\mathcal{H}'$, we have $S \nsubseteq f$. Moreover, by Lemma 5.12, we have that $V(\mathcal{H}_f^y) \cap [\geq y] \subseteq f$. As $S \subseteq [\geq x]$ and $x > y$, we have $S \subseteq [\geq y]$. Thus $S \nsubseteq V(\mathcal{H}_f^y)$ and for all $g \in \mathcal{H}_f^y$, we have $S \nsubseteq g$ since $g \subseteq V(\mathcal{H}_f^y)$. We can conclude that $\mathcal{H}_f^y \subseteq \mathcal{H}'$.

Now, we prove that every $g \in \mathcal{H}_f^y$, we have $g \cap [< x] \in C$. Let $g \in \mathcal{H}_f^y$. By definition of $\mathcal{H}_f^y$ and because $\mathcal{H}_f^y \subseteq \mathcal{H}'$, there exists a path P from f to g going only through vertices smaller than y and hyperedges smaller than f in $\mathcal{H}'$. As $y < x$, we can conclude that $f \cap [< x]$ is connected to $g \cap [< x]$ in $\mathcal{H}'$, i.e., $g \cap [< x] \in C$. In other words, we have $\mathcal{H}_f^y \subseteq \{g \in \mathcal{H}' \,:\, g \cap [< x] \in C\}$.

It remains to prove the other inclusion. Let $g \in \mathcal{H}'$ with $g \cap [< x] \in C$. Since C is a connected component of $\mathcal{H}'[< x]$, there exists a path P from $f \cap [< x]$ to $g \cap [< x]$. By the maximality of y and f, P goes only through vertices smaller than y and hyperedges smaller than f. We can construct from P a path P' from f to g in $\mathcal{H}'$ going through vertices smaller than y and hyperedges smaller than f. As $\mathcal{H}' \subseteq \mathcal{H}$, we can conclude that $g \in \mathcal{H}_f^y$ and thus, $\mathcal{H}_f^y = \{g \in \mathcal{H}' \,:\, g \cap [< x] \in C\}$. Finally, observe that $C = \mathcal{H}_f^y[< x] = \mathcal{H}_f^y[\leq y]$ since $y = \mathsf{max}(V(C))$. □

The algorithm

In this subsection, we describe the dynamic programming algorithm we use to count the number of minimal transversals of a β-acyclic hypergraph. We denote by x_1 the smallest element of $\leq$.

Our goal is to compute $\#\mathsf{btr}_{\varnothing}(\mathcal{H}_e^x, [\leq x])$ and $\#\mathsf{btr}_w(\mathcal{H}_e^x, [\leq x])$ for every $e \in \mathcal{H}$, $x \in V(\mathcal{H})$ and $w \in V(\mathcal{H})$ such that $x < w$. Observe that it is enough for computing the number of minimal transversals of $\mathcal{H}$ as $\#\mathsf{mtr}(\mathcal{H}) = \#\mathsf{btr}_{\varnothing}(\mathcal{H}_{e_m}^{x_n}, [\leq x_n])$ where e_m is the maximal hyperedge for $\leq_{\mathcal{H}}$ and x_n is the maximal vertex for $\leq$. Indeed, we have $\mathcal{H}_{e_m}^{x_n} = \mathcal{H}$ and $[\leq x_n] = V(\mathcal{H})$, thus $\mathsf{btr}_{\varnothing}(\mathcal{H}_{e_m}^{x_n}, [\leq x_n]) = \mathsf{btr}_{\varnothing}(\mathcal{H}) = \mathsf{mtr}(\mathcal{H})$.

The propagation of the dynamic programming works as follows: we use Theorem 5.11 to reduce the computation of $\#\mathsf{btr}_B(\mathcal{H}_e^x, [\leq x])$ to the computation of $\#\mathsf{btr}$ for several hypergraphs that do not contain x. We then use Lemma 5.14 to show that these hypergraphs can be decomposed into disjoint hypergraphs of the form $\mathcal{H}_f^y$ for $f \leq_{\mathcal{H}} e$ and $y < x$ which allows us to compute $\#\mathsf{btr}_B(\mathcal{H}_e^x, [\leq x])$ from precomputed values of the form $\#\mathsf{btr}_{B'}(\mathcal{H}_f^y, [\leq y])$, where $B' \in \{B, \{x\}\}$.

Before continuing, let us give a high-level description of the algorithm. For each $1 \leq i \leq n$ and each $1 \leq j \leq m$, let $tab[i,j,0]$ be $\#\mathsf{btr}_{\varnothing}(\mathcal{H}^{x_i}_{e_j}, [\leq x_i])$, and for each $\ell > i$, let $tab[i,j,\ell]$ be $\#\mathsf{btr}_{x_\ell}(\mathcal{H}^{x_i}_{e_j}, [\leq x_i])$. Because the number of minimal transversals of $\mathcal{H}$ is $\#\mathsf{btr}_{\varnothing}(\mathcal{H}^{x_n}_{e_m}, [\leq x_n])$, it is enough to show how to compute $tab[n,m,0]$ in polynomial time. The following is a high level description of the algorithm which computes $tab[i,j,\ell]$, for all $1 \leq i \leq n$, $1 \leq j \leq m$ and $\ell \in \{0, i+1, \ldots, n\}$.

Algorithm CountMinTransversals($\mathcal{H}$)
$\mathcal{H}$: a β-acyclic hypergraph
1. Let $x_1, \ldots, x_n$ a β-elimination ordering of $\mathcal{H}$.
2. Let $e_1, \ldots, x_m$ the induced lexicographic ordering on $\mathcal{H}$.
3. Precompute $\mathcal{H}^{x_i}_{e_j}, [\leq x_i]$ for every $i \leq n, j \leq m$.
4. **for** $1 \leq i \leq n$ and $i < \ell \leq n$ **do**
5. **for** $1 \leq j \leq m$ **do**
6. Compute $tab[i,j,0]$ from the recursive formula of $\#\mathsf{btr}_{\varnothing}(\mathcal{H}^{x_i}_{e_j}, [\leq x_i])$.
7. Compute $tab[i,j,\ell]$ from the recursive formula of $\#\mathsf{btr}_{x_\ell}(\mathcal{H}^{x_i}_{e_j}, [\leq x_i])$.
8. **end for**
9. **end for**
10. **return** $tab[n,m,0]$

In order to ease the presentation, the computation of $\#\mathsf{btr}_{\varnothing}(\mathcal{H}^x_e, [\leq x])$ and the computation of $\#\mathsf{btr}_w(\mathcal{H}^x_e, [\leq x])$ are separated, even though many of the arguments are similar.

Base cases. We observe that for every $e \in \mathcal{H}$, $\mathcal{H}^{x_1}_e[\leq x_1]$ is either equal to $\{x_1\}$ or $\{\varnothing\}$. Thus, for every $e \in \mathcal{H}$ and $w \in V(\mathcal{H})$ such that $w > x_1$, we can compute $\#\mathsf{btr}_{\varnothing}(\mathcal{H}^x_e, [\leq x_1])$ and $\#\mathsf{btr}_w(\mathcal{H}^x_e, [\leq x_1])$ in time $O(1)$.

Computing $\#\mathsf{btr}_{\varnothing}(\mathcal{H}^x_e, [\leq x])$ by dynamic programming. We start by explaining how we can compute $\#\mathsf{btr}_{\varnothing}(\mathcal{H}^x_e, [\leq x])$ in polynomial time if the values $\#\mathsf{btr}_{\varnothing}(\mathcal{H}^y_f, [\leq y])$ and $\#\mathsf{btr}_w(\mathcal{H}^y_f, [\leq y])$ have been precomputed for $f \leq_{\mathcal{H}} e$ and $y < x$, $y < w$.

We start by applying Theorem 5.11.

$$\begin{aligned}\#\mathsf{btr}_{\varnothing}(\mathcal{H}^x_e, [\leq x]) = {} & \#\mathsf{btr}_{\varnothing}(\mathcal{H}^x_e, [< x]) \\ & + \#\mathsf{btr}_{\varnothing}(\mathcal{H}^x_e \setminus \mathcal{H}^x_e(x), [< x]) \\ & - \#\mathsf{btr}_x(\mathcal{H}^x_e, [< x]).\end{aligned}$$

Now, let $C_1, \ldots, C_k$ be the connected components of $\mathcal{H}^x_e[< x]$. If there exists i such that $C_i = \{\varnothing\}$, then $\#\mathsf{btr}_{\varnothing}(\mathcal{H}^x_e, [< x]) = \#\mathsf{btr}_x(\mathcal{H}^x_e, [< x]) = 0$. Otherwise, by applying Lemma 5.14 with $S = \varnothing$, there exists, for each $1 \leq i \leq k$, $y_i < x$ and $f_i \leq_{\mathcal{H}} e$ such that $\mathcal{H}^{y_i}_{f_i} = \{g \in \mathcal{H}^f_e : g \cap [< x] \in C_i\}$ and $C_i = \mathcal{H}^{y_i}_{f_i}[\leq y_i]$. By Lemma 5.7,

$$\#\mathsf{btr}_{\varnothing}(\mathcal{H}^x_e, [< x]) = \prod_{i=1}^{k} \#\mathsf{btr}_{\varnothing}(\mathcal{H}^{y_i}_{f_i}, [\leq y_i]),$$

$$\#\mathsf{btr}_x(\mathcal{H}^x_e, [< x]) = \prod_{i=1}^{k} \#\mathsf{btr}_x(\mathcal{H}^{y_i}_{f_i}, [\leq y_i]).$$

We now show how to decompose $\#\mathsf{btr}_{\varnothing}(\mathcal{H}^x_e \setminus \mathcal{H}^x_e(x), [< x])$ into a product of precomputed values. Let $D_1, \ldots, D_l$ be the connected components of $\mathcal{H}^x_e \setminus \mathcal{H}^x_e(x)[< x]$. If there exists i such

that $D_i = \{\varnothing\}$, then $\#\mathsf{btr}_x(\mathcal{H}_e^x \setminus \mathcal{H}_e^x(x), [< x]) = 0$. Otherwise, if we apply Lemma 5.14 with $S = \{x\}$, then there exists, for each $1 \leq i \leq l$, $y_i < x$ and $f_i \leq_{\mathcal{H}} e$ such that $\mathcal{H}_{y_i}^{f_i} = \{g \in \mathcal{H}_e^x \setminus \mathcal{H}_e^x(x) \,:\, g \cap [< x] \;\in D_i\}$ and $D_i = \mathcal{H}_{y_i}^{f_i}[\leq y_i]$. We can thus conclude by Lemma 5.7 that

$$\#\mathsf{btr}_\varnothing(\mathcal{H}_e^x \setminus \mathcal{H}_e^x(x), [< x]) = \prod_{i=1}^{l} \#\mathsf{btr}_\varnothing(\mathcal{H}_{f_i}^{y_i}, [\leq y_i]).$$

Therefore, if $\#\mathsf{btr}_\varnothing(\mathcal{H}_f^y, [\leq y])$ and $\#\mathsf{btr}_x(\mathcal{H}_f^y, [\leq y])$ have already been computed for every $f <_{\mathcal{H}} e$ and $y \leq x$, we can compute $\#\mathsf{btr}_\varnothing(\mathcal{H}_e^x, [\leq x])$ with at most $3 \times |\mathcal{H}_e^x|$ additional multiplications and 3 additions.

Computing $\#\mathsf{btr}_w(\mathcal{H}_e^x, [\leq x])$ by dynamic programming. Let $x \leq w$. By Theorem 5.11, we have:

$$\begin{aligned}\#\mathsf{btr}_w(\mathcal{H}_e^x, [\leq x]) = {} & \#\mathsf{btr}_w(\mathcal{H}_e^x, [< x]) \\ & + \#\mathsf{btr}_w(\mathcal{H}_e^x \setminus \mathcal{H}_e^x(x), [< x]) \\ & - \#\mathsf{btr}_{\{x,w\}}(\mathcal{H}_e^x \setminus (\mathcal{H}_e^x(x) \cap \mathcal{H}_e^x(w)), [< x]).\end{aligned}$$

We start by explaining how to compute $\#\mathsf{btr}_w(\mathcal{H}_e^x, [< x])$. Let $C_1, \ldots, C_k$ be the connected components of $\mathcal{H}_e^x[< x]$. If there exists i such that $C_i = \{\varnothing\}$, then $\#\mathsf{btr}_w(\mathcal{H}_e^x, [< x]) = 0$. Otherwise, by Lemma 5.14 with $S = \{w\}$, there exists, for each $1 \leq i \leq k$, $f_i \leq_{\mathcal{H}} e$ and $y_i < x$ such that $\mathcal{H}_{f_i}^{y_i} = \{g \in \mathcal{H}_e^x \,:\, g \cap [< x] \in C_i\}$ and $C_i = \mathcal{H}_{f_i}^{y_i}[\leq y_i]$. By Lemma 5.7, we can conclude that

$$\#\mathsf{btr}_w(\mathcal{H}_e^x, [< x]) = \prod_{i=1}^{k} \#\mathsf{btr}_w(\mathcal{H}_{f_i}^{y_i}, [\leq y_i]).$$

We now explain how to compute $\#\mathsf{btr}_w(\mathcal{H}_e^x \setminus \mathcal{H}_e^x(x), [< x])$. Let $D_1, \ldots, D_l$ be the connected components of $(\mathcal{H}_e^x \setminus \mathcal{H}_e^x(x))[< x]$. If there exists i such that $D_i = \{\varnothing\}$, then $\#\mathsf{btr}_w(\mathcal{H}_e^x \setminus \mathcal{H}_e^x(x), [< x]) = 0$. Otherwise, by applying Lemma 5.14 with $S = \{x\}$, it follows that for every $1 \leq i \leq l$, there exists $y_i < x$ and $f_i \leq_{\mathcal{H}} e$ such that $\mathcal{H}_{f_i}^{y_i} = \{g \in \mathcal{H}_e^x \setminus \mathcal{H}_e^x(x) \,:\, g \cap \;[< x] \in D_i\}$ and $D_i = \mathcal{H}_{f_i}^{y_i}[\leq y_i]$. Thus, from Lemma 5.7,

$$\#\mathsf{btr}_w(\mathcal{H}_e^x \setminus \mathcal{H}_e^x(x), [< x]) = \prod_{i=1}^{l} \#\mathsf{btr}_w(\mathcal{H}_{f_i}^{y_i}, [\leq y_i]).$$

Finally, we explain how to decompose $\#\mathsf{btr}_{\{x,w\}}(\mathcal{H}_e^x \setminus (\mathcal{H}_e^x(x) \cap \mathcal{H}_e^x(w)), [< x])$ into a product of pre-computed values. To ease the notation, we denote $\mathcal{H}_e^x \setminus (\mathcal{H}_e^x(x) \cap \mathcal{H}_e^x(w))$ by $\mathcal{H}'$. Let $K_1, \ldots, K_r$ be the connected components of $\mathcal{H}'[< x]$. If there exists i such that $K_i = \{\varnothing\}$, then $\#\mathsf{btr}_{\{x,w\}}(\mathcal{H}', [< x]) = 0$. Otherwise, by Lemma 5.14 applied with $S = \{x, w\}$, we have that for every i, there exists $y_i < x$ and $f_i \leq_{\mathcal{H}} e$ such that $\mathcal{H}_{f_i}^{y_i} = \{g \in \mathcal{H}' \,:\, g \cap [< x] \;\in K_i\}$ and $K_i = \mathcal{H}_{f_i}^{y_i}[\leq y_i]$. By Lemma 5.7, we have:

$$\#\mathsf{btr}_{\{x,w\}}(\mathcal{H}_e^x \setminus (\mathcal{H}_e^x(x) \cap \mathcal{H}_e^x(w)), [< x]) = \prod_{i=1}^{r} \#\mathsf{btr}_{\{x,w\} \cap V(\mathcal{H}_{f_i}^{y_i})}(\mathcal{H}_{f_i}^{y_i}, [\leq y_i]).$$

Claim 5.14.1. *For every $i \leq p$, $\{x, w\} \cap V(\mathcal{H}_{f_i}^{y_i}) \neq \{x, w\}$.*

Proof. Assume towards a contradiction that $\{x, w\} \cap V(\mathcal{H}_{f_i}^{y_i}) = \{x, w\}$. Recall that $\mathcal{H}_{f_i}^{y_i}[\leq y_i]$. By Lemma 5.12, $\{x, w\} \subseteq V(\mathcal{H}_{f_i}^{y_i})$ implies $\{x, w\} \subseteq f$. Thus, we have $f \in \mathcal{H}_e^x(x) \cap \mathcal{H}_e^x(w)$. This is a contradiction, since $f \in \mathcal{H}_{f_i}^{y_i} \subseteq \mathcal{H}_e^x \setminus (\mathcal{H}_e^x(w) \cap \mathcal{H}_e^x(x))$. □

Thus, $\{x, w\} \cap V(\mathcal{H}^{y_i}_{f_i})$ equals either $\{x\}$, or $\{w\}$ or $\varnothing$ by Claim 5.14.1. That is, we can compute $\#\mathsf{btr}_{\{x,w\}}(\mathcal{H}^x_e \setminus (\mathcal{H}^x_e(x) \cap \mathcal{H}^x_e(w)), [< x])$ from precomputed terms.

We can conclude that, if $\#\mathsf{btr}_{\varnothing}(\mathcal{H}^y_f, [\leq y])$ and $\#\mathsf{btr}_w(\mathcal{H}^y_f, [\leq y])$ have already been computed for every $f <_{\mathcal{H}} e$ and $y \leq w$, we can compute $\#\mathsf{btr}_w(\mathcal{H}^x_e, [\leq x])$ with at most $3 \times |\mathcal{H}^x_e|$ additional multiplications and 3 additions.

It is easy to see that a straightforward greedy algorithm can be used to compute a β-elimination ordering in polynomial time (see [122] for a better algorithm due to Paige and Tarjan). Moreover, the dynamic programming algorithm describes above computes at most $O(n^2|\mathcal{H}|)$ terms and each of them can be computed from the others with a polynomial number of arithmetic operations. Finally, all these terms can be bounded by 2^n since they are all the cardinals of some collection of subsets of the vertices. Thus these arithmetic operations can be done in polynomial time in the size of the input. It follows.

Theorem 5.15. *Let $\mathcal{H}$ be a β-acyclic hypergraph. One can compute in polynomial time the number of minimal transversals of $\mathcal{H}$.*

5.2.3 Applications to the counting of minimal dominating sets

For a vertex x of a graph G, let $N[x]$ be the set $\{x\} \cup N(x)$. The *closed neighborhood hypergraph* of G, denoted by $\mathcal{N}[G]$, is the hypergraph $\{N[x] : x \in V(G)\}$. Observe that any (minimal) *dominating set* of G is a (minimal) transversal of $\mathcal{N}[G]$ and vice-versa.

In [93] the authors reduce the Hypergraph Dualisation problem into the enumeration of minimal dominating sets, showing that the two problems are equivalent in the area of enumeration problems (a fact already established in the case of optimization). The reduction indeed shows that the counting versions are equivalent (under Turing reductions), and such a reduction is of big interest because it allows to study counting and enumeration problems associated with the Hypergraph Dualisation in the perspectives of graph theory, where tools had been developed to tackle combinatorial problems.

Despite the broad application of counting the minimal dominating sets in (hyper)graphs, the problem was not investigated until recently, except in [30] where it is proved that the models of any monadic second-order formula can be counted in polynomial time in graphs of bounded clique-width. Indeed, as far as we know the counting of minimal dominating sets is only considered in [94, 73, 96]. This problem is known to be polynomial on interval graphs and permutation graphs [94]. However, the systematic study of its computational complexity in graph classes is only considered in [96], where the authors proved the #P-completeness in several graph classes and asked whether the following dichotomy conjecture is true. A k-sun is a graph obtained from a cycle of length $2k$ ($k \geq 3$) by adding edges to make the even-indexed vertices pairwise adjacent. Figure 5.2 represents a 3-sun, a 4-sun and a 5-sun.

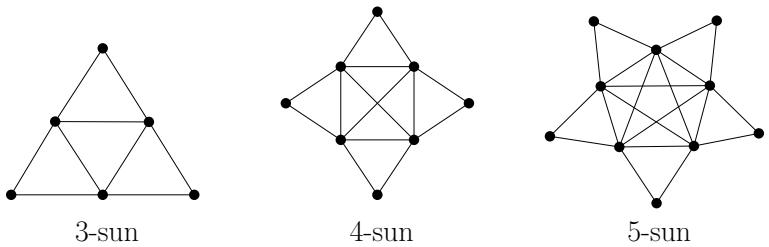


Figure 5.2 – Representations of k-suns for $k \in \{3, 4, 5\}$.

A graph is *chordal* if it does not contain cycles of length at least 4 as induced subgraphs.

Conjecture 5.16. *Let $\mathcal{C}$ be a class of chordal graphs. If $\mathcal{C}$ does not contain a k-sun as an induced subgraph, for $k \geq 4$, then one can count in polynomial time the number of minimal dominating sets of any graph in $\mathcal{C}$. Otherwise, the problem is #P-complete.*

This conjecture is motivated by the recursive structure of the chordal graphs excluding the k-sun, for $k \leq 4$.

We make a first step towards a proof of the first statement of the conjecture and provide a polynomial time algorithm for computing the number of minimal dominating sets in *strongly chordal graphs*, which are exactly chordal graphs without k-suns, for $k \geq 3$.

Corollary 5.17. *Let G be a strongly chordal graph. One can count in polynomial time the number of minimal dominating sets of G.*

Proof. Let G be a strongly chordal graph. It is well-known that the hypergraph $\mathcal{N}[G]$ is β-acyclic [14]. By Theorem 5.15, one can count in polynomial time the minimal transversals of $\mathcal{N}[G]$, which are exactly the minimal dominating sets of G. □

Observe that this result is not implied by Theorem 5.1 as the mim-width of strongly chordal graphs is unbounded [108].

5.2.4 Extension

We have proposed a polynomial time algorithm for counting the minimal transversals of β-acyclic hypergraphs, supporting Conjecture 5.16. It seems that the technique can be adapted to consider the counting of minimal (inclusion-wise) *d-transversals* of β-acyclic hypergraphs. Given $d \in \mathbb{N}^+$, a d-transversal of a hypergraph $\mathcal{H}$ is a set of vertices $X \subseteq V(\mathcal{H})$ such that for every edge $e \in E(\mathcal{H})$, we have $|X \cap e| \geq d$.

Besides resolving Conjecture 5.16, one might be interested in extending our result to the counting of minimal models of a CNF formula. Let us introduce some definitions on CNF formulas. Given a CNF formula F, for each clause C, we can associate a hyperedge E_C that contains all the variables occurring in C. The hypergraph associated with a CNF formula F is $\bigcup_{C \in F}\{E_C\}$. A CNF formula is said *β-acyclic* if its associated hypergraph is β-acyclic. A model τ of a CNF formula F is *minimal* if and only if for every variable x interpreted as true by τ, there exists a clause C such that the only literal of C interpreted as true by τ is x.

Capelli [21] proved that we can count the models of a β-acyclic formula in polynomial time. We have proved that the minimal models of a monotone β-acyclic formula can be counted in polynomial time. Indeed, there is a one-to-one correspondence between the minimal models of a monotone CNF formula and the minimal transversals of its associated hypergraph. These two results lead naturally to the following question.

Open Question 5.18. *Can we count in polynomial time the minimal models of a non-monotone β-acyclic CNF formula?*

Let us explain, briefly, why we cannot answer the above question with our techniques. Recall that, in order to compute $\#\mathsf{btr}_w(\mathcal{H}_e^x, [\leq x])$, we need to compute $\#\mathsf{btr}_{\{x,w\}}(\mathcal{H}_e^x \setminus (\mathcal{H}_e^x(x) \cap \mathcal{H}_e^x(w)), [< x])$. If we apply our techniques to count the minimal models of a β-acyclic CNF formula, then we have to face a similar situation. More precisely, for some β-acyclic CNF formula F, we have to compute the number N of assignments that are the minimal models of the two following CNF formulas (restricted to the variables in $[< x]$):

- the formula F' obtained from F by removing the clauses containing a positive occurrence of x or w,

- $\widehat{F}$ the formula obtained from F by removing the clauses where both x and w occurs positively.

This is needed to compute what we might call *w-blocked models* of F restricted to the variables in $[\leq x]$ (the analog of $\#\mathsf{btr}_w(\mathcal{H}_e^x, [\leq x])$). Similarly to the minimal transversal case, we can express N as a product of terms corresponding to the numbers of $\{x, w\}$-blocked models of some subformulas of F. Moreover, each of these subformulas can be described with one variable and one clause as we do for the hypergraphs $\mathcal{H}_f^y[\leq y]$ (see [20]). But, unlike the hypergraph $\mathcal{H}_e^x \setminus (\mathcal{H}_e^x(x) \cap \mathcal{H}_e^x(w))$, the hypergraph $\widehat{H}$ associated with $\widehat{F}$ may admit hyperedges containing both x and w. This is due to the fact that some clauses in F may contain $\{\neg x, w\}$, $\{x, \neg w\}$, or $\{\neg x, \neg w\}$. We can not delete these clauses since we are looking for the number of minimal models of F' that do not give a model of F with x and w assigned to true. Consequently, we cannot prove an analog of Claim 5.14.1 for CNF formulas. That is, to compute N, we have to compute the number of $\{x, w\}$-blocked models of some subformulas of F where both x and w may occur. Hence, adapting our algorithm to non-monotone CNF formula leads to the computation of the number of B-blocked models, for an exponential number of sets B.

It would be interesting to extend our results on β-acyclic hypergraphs via a width measure on hypergraphs. For doing so, we can use *β-hypertree-width*, a parameter introduced in [77] which is a generalization of tree-width on hypergraphs (both notions collapse when restricted to graphs). It is known that a hypergraph has β-hypertree-width 1 if and only if it is β-acyclic. Besides this, we do not know much about β-hypertree-width. In fact, the only characterization we have of β-hypertree-width is obscure and cannot be used to design algorithms. For instance, we do not even know whether SAT is solvable in polynomial time on CNF formulas whose hypergraphs have β-hypertree-width 2. Moreover, the complexity of determining the β-hypertree-width of a hypergraph is open [78]. A better understanding of this parameter is necessary in order to obtain algorithmic applications. On the other hand, Ordyniak, Paulusma, and Szeider [113] proved that SAT is $\mathsf{W}[1]$-hard parameterized by the β-hypertree-width. Consequently, counting the (minimal) models of CNF formulas is also $\mathsf{W}[1]$-hard parameterized by the β-hypertree-width of the associated hypergraphs. Whether there exist XP algorithms for these problems is open.

In [20], Capelli introduced a parameter called *cover-width* which may be an interesting alternative to β-hypertree-width. The definition of cover-width is based on a generalization of the elimination order from which we can define tree-width. Capelli [20] proved that:

- computing the cover-width of a hypergraph is NP-hard,
- a hypergraph has cover-width 1 if and only if it is β-acyclic,
- the β-hypertree-width of a hypergraph H is at most $3 \cdot k + 1$ with k the cover-width of H.

Contrary to β-hypertree-width, the definition of cover-width is suited to design algorithms. Indeed, Capelli [20] showed that, given a CNF formula F with a hypergraph H and an elimination ordering on $V(H)$ of cover-width k, we can count the models of F by doing $O(n^k \cdot m^k)$ arithmetic operations with n and m the number of variables and clauses of F (we do not know whether these arithmetic operations can be performed in polynomial time). The approach used by Capelli to obtain this algorithm is quite similar to the one he used on β-acyclic hypergraphs [20, 21]. Thus, cover-width is a good candidate to generalize our result on the counting of minimal transversals of β-acyclic hypergraphs. However, there is still much to understand on this width measure. For example, can we bound the cover-width of any hypergraph in function of its β-hypertree-width? What is the parameterized complexity of computing the cover-width of a hypergraph?

Chapter 6

Conclusion

In this thesis, we have presented many open questions about width measures. The hardest ones are certainly those concerning their computations (see Section 2.5). One of the most interesting concerns the algorithmic applications of rank-width and whether we can obtain $2^{o(\mathsf{rw}(G)^2)} \cdot n^{O(1)}$ time algorithms for NP-hard problems (see Question 2.59). In this chapter, we would like to present two general open questions that are transversal to the different chapters of this thesis.

In Chapter 2, we presented Courcelle's theorem and its variants for clique-width. These theorems are incredibly useful to know whether a problem is FPT when parameterized by a width measure such as tree-width, clique-width, rank-width, etc. But these meta-theorems do not give any hint about how fast we can solve a problem. To obtain efficient algorithms, we have to lose generality, but it does not mean that we have to look at each width measure and each problem separately. This leads naturally to the following question.

Open Question 6.1. *For each width measure* f*, can we characterize a huge number of graph problems which admit an efficient* FPT *(or* XP*) algorithm parameterized by the* f*-width of a given decomposition? For example, can we characterize a huge number of graph problems which admit a* $2^{O(\mathsf{mw}(\mathcal{L}))} \cdot n^{O(1)}$ *time algorithm with* $\mathcal{L}$ *a given rooted layout?*

This question has already been answered positively by Pilipczuk for tree-width in [123] where he introduced a variant of modal logic which captures a vast majority of problems known to be solvable in $2^{O(k)} \cdot n^{O(1)}$ with k the tree-width of the input graph. The framework designed in [16] for (σ, ρ)-Dominating Set problems and extended in Section 4.2 to the acyclic and connected variants of this family of problems shows that we can answer positively this latter question without looking at each width measure separately. In Subsection 2.6.2, we asked whether we can extend this approach to problems that are W[1]-hard parameterized by clique-width. Let us be more ambitious. Can we design a framework from which we can obtain the best known algorithms for many graph problems and many width measures? Notice that we can naturally divide this question into two parts.

- The first part consists into designing a language $\mathscr{L}$ to express graph problems like the family of (σ, ρ)-Dominating Set problems.
- The second part consists in creating a framework that, given the expression of some problem in $\mathscr{L}$, provides an efficient algorithm for each width measure. The algorithms in [18, 123] and in Section 4.2 are three examples of such framework.

To attack these questions, one could start by extending the results obtained in [18] and in Section 4.2. For example, it would be interesting to study other kinds of connectivity and acyclicity

constraints such as the strange acyclicity constraint of the Subset Feedback Vertex Set problem whose definition is the following.

Subset Feedback Vertex Set
Input: A graph G and $S \subseteq V(G)$. **Output:** A set $X \subseteq V(G)$ of minimum size such that $G[V(G) \setminus X]$ does not admit a cycle that contains a vertex in S.

Besides the fact that this problem is FPT parameterized by tree-width and clique-width (it is easily expressible in MSO_1), we do not know much on its parameterized complexity w.r.t. width measures. Studying the parameterized complexity of these kinds of problems could help to understand what makes a problem efficiently solvable with respect to the different width measures.

Nevertheless, extending our knowledge on the known width measures is not sufficient to understand entirely what makes an NP-hard problem tractable. Indeed, some NP-hard problems are tractable on graph classes where even mim-width is unbounded. For instance, many NP-hard problems are known to be tractable on chordal graphs. This is the case for Independent Set, Graph Coloring, the unweighted variant of Feedback Vertex Set, and the problem of counting the number of independent sets [55, 75, 112]. If we want to explain why these problems are tractable on chordal graphs, we need a width measure with a stronger modeling power than mim-width. Recently, Kang, Kwon, Strømme, and Telle [91] introduced such a width measure that they called *sim-width*. This new width measure corresponds to the f-width of a graph where $\mathsf{f}(A)$ is the size of a maximum induced matching between A and $\overline{A}$ in G. Kang et al. proved that the sim-width of chordal graphs is 1, however, they were not able to obtain algorithmic results directly from this new width measure. This leads to the following question.

Open Question 6.2. *Can we explain the tractability results of* NP*-hard problems on chordal graphs by the boundedness of their sim-width?*

In the same spirit, one might try to explain the tractability results we have on strongly chordal graphs. Indeed, some NP-hard (or #P-hard) problems on chordal graphs are known to be tractable on strongly chordal graphs. This is the case of Dominating Set [12, 51] and the problem of counting the number of (minimal) dominating sets (Corollary 5.3 and [96]). Unlike chordal graphs, we do not have a width measure on graphs susceptible to explain the tractability of these problems. However, in Subsection 5.2.4, we have presented two width measures on hypergraphs which could explain the tractability results concerning β-acyclic hypergraphs and thus the structured neighborhood of strongly chordal graphs.

Bibliography

[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015.

[2] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[3] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

[4] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018.

[5] Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoret. Comput. Sci.*, 511:54–65, 2013.

[6] Benjamin Bergougnoux and Mamadou Moustapha Kanté. More applications of the *d*-neighbor equivalence: acyclic and connectivity constraints. *CoRR*, abs/1805.11275, 2018.

[7] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 226–234, 1993.

[8] Hans L. Bodlaender. Treewidth: Characterizations, applications, and computations. In *Graph-Theoretic Concepts in Computer Science, 32nd International Workshop, WG 2006, Bergen, Norway, June 22-24, 2006, Revised Papers*, pages 1–14, 2006.

[9] Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inform. and Comput.*, 243:86–111, 2015.

[10] Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors. *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*. Springer, 2012.

[11] Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A c^k n 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016.

[12] Kellogg S. Booth and J. Howard Johnson. Dominating sets in chordal graphs. *SIAM J. Comput.*, 11(1):191–199, 1982.

[13] Richard B. Borie, R. Gary Parker, and Craig A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7(5&6):555–581, 1992.

[14] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: a survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1999.

[15] Johann Brault-Baron, Florent Capelli, and Stefan Mengel. Understanding model counting for beta-acyclic cnf-formulas. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 143–156, 2015.

[16] Binh-Minh Bui-Xuan, Ondřej Suchý, Jan Arne Telle, and Martin Vatshelle. Feedback vertex set on graphs of low clique-width. *European J. Combin.*, 34(3):666–679, 2013.

[17] Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Boolean-width of graphs. In *IWPEC*, pages 61–74, 2009.

[18] Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theoret. Comput. Sci.*, 511:66–76, 2013.

[19] Kathie Cameron. Induced matchings. *Discrete Applied Mathematics*, 24(1-3):97–102, 1989.

[20] Florent Capelli. *Structural restrictions of CNF formulas: application to model counting and knowledge compilation*. PhD thesis, Université Paris Diderot, 2016.

[21] Florent Capelli. Understanding the complexity of #sat using knowledge compilation. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–10, 2017.

[22] L. Sunil Chandran, Davis Issac, and Andreas Karrenbauer. On the parameterized complexity of biclique cover and partition. In *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, pages 11:1–11:13, 2016.

[23] Shiri Chechik, Daniel H. Larkin, Liam Roditty, Grant Schoenebeck, Robert Endre Tarjan, and Virginia Vassilevska Williams. Better approximation algorithms for the graph diameter. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1041–1052, 2014.

[24] James Cheetham, Frank K. H. A. Dehne, Andrew Rau-Chaplin, Ulrike Stege, and Peter J. Taillon. Solving large FPT problems on coarse-grained parallel machines. *J. Comput. Syst. Sci.*, 67(4):691–706, 2003.

[25] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158, 1971.

[26] Derek G. Corneil, Michel Habib, Jean-Marc Lanlignel, Bruce Reed, and Udi Rotics. Polynomial-time recognition of clique-width ≤ 3 graphs. *Discrete Appl. Math.*, 160(6):834–865, 2012.

[27] Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005.

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

[29] Bruno Courcelle. The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues. *ITA*, 26:257–286, 1992.

[30] Bruno Courcelle. The monadic second-order logic of graphs XVI : Canonical graph decompositions. *Logical Methods in Computer Science*, 2(2), 2006.

[31] Bruno Courcelle and Joost Engelfriet. *Graph structure and monadic second-order logic*, volume 138 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 2012. A language-theoretic approach, With a foreword by Maurice Nivat.

[32] Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *J. Comput. Syst. Sci.*, 46(2):218–270, 1993.

[33] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Applied Mathematics*, 108(1-2):23–52, 2001.

[34] Bruno Courcelle and Mohamed Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1&2):49–82, 1993.

[35] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

[36] Radu Curticapean. *The simple, little and slow things count: on parameterized counting complexity.* PhD thesis, Saarland University, 2015.

[37] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015.

[38] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time (extended abstract). In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science—FOCS 2011*, pages 150–159. IEEE Computer Soc., Los Alamitos, CA, 2011.

[39] Guillaume Damiand, Michel Habib, and Christophe Paul. A simple paradigm for graph recognition: application to cographs and distance hereditary graphs. *Theor. Comput. Sci.*, 263(1-2):99–111, 2001.

[40] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics.* Springer, 2012.

[41] Rodney G. Downey and Michael R. Fellows. Fixed-parameter intractability. In *Proceedings of the Seventh Annual Structure in Complexity Theory Conference, Boston, Massachusetts, USA, June 22-25, 1992*, pages 36–49, 1992.

[42] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer, 1999.

[43] Rodney G. Downey and Michael R. Fellows. *Fundamentals of parameterized complexity.* Texts in Computer Science. Springer, London, 2013.

[44] Arnaud Durand and Miki Hermann. On the counting complexity of propositional circumscription. *Inform. Process. Lett.*, 106(4):164–170, 2008.

[45] Thomas Eiter and Georg Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM J. Comput.*, 24(6):1278–1304, 1995.

[46] Thomas Eiter and Georg Gottlob. Hypergraph transversal computation and related problems in logic and AI. In *European Workshop on Logics in Artificial Intelligence*, pages 549–564. Springer, 2002.

[47] Thomas Eiter, Kazuhisa Makino, and Georg Gottlob. Computational aspects of monotone dualization: a brief survey. *Discrete Appl. Math.*, 156(11):2035–2049, 2008.

[48] Khaled M. Elbassioni, Rajiv Raman, Saurabh Ray, and René Sitters. On the approximability of the maximum feasible subsystem problem with 0/1-coefficients. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 1210–1219, 2009.

[49] Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In *Graph-theoretic concepts in computer science (Boltenhagen, 2001)*, volume 2204 of *Lecture Notes in Comput. Sci.*, pages 117–128. Springer, Berlin, 2001.

[50] Stefan Fafianie, Hans L. Bodlaender, and Jesper Nederlof. Speeding up dynamic programming with representative sets: An experimental evaluation of algorithms for steiner tree on tree decompositions. *Algorithmica*, 71(3):636–660, 2015.

[51] Martin Farber. Domination, independent domination, and duality in strongly chordal graphs. *Discrete Applied Mathematics*, 7(2):115–130, 1984.

[52] Michael R. Fellows. Parameterized complexity: The main ideas and some research frontiers. In *Algorithms and Computation, 12th International Symposium, ISAAC 2001, Christchurch, New Zealand, December 19-21, 2001, Proceedings*, pages 291–307, 2001.

[53] Michael R. Fellows. Parameterized complexity: The main ideas and connections to practical computing. *Electr. Notes Theor. Comput. Sci.*, 61:1–19, 2002.

[54] Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width minimization is np-hard. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 354–362, 2006.

[55] Paola Festa, Panos M. Pardalos, and Mauricio G. C. Resende. Feedback set problems. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization, Second Edition*, pages 1005–1016. Springer, 2009.

[56] Herbert Fleischner. *Eulerian graphs and related topics*, volume 1. Elsevier, 1990.

[57] Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM J. Comput.*, 33(4):892–922, 2004.

[58] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010.

[59] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014.

[60] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: hamiltonian cycle and the odd case of graph coloring. *ACM Trans. Algorithms*, 15(1):9:1–9:27, 2019.

[61] Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the tractability of optimization problems on h-graphs. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 30:1–30:14, 2018.

[62] Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016.

[63] Lance Fortnow. The status of the P versus NP problem. *Commun. ACM*, 52(9):78–86, 2009.

[64] Veronique Froidure. *Rangs des relations binaires et semigroupes de relations non ambigus.* PhD thesis, Université Pierre-et-Marie-Curie, 1995.

[65] Robert Ganian and Petr Hliněný. On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. *Discrete Appl. Math.*, 158(7):851–867, 2010.

[66] Robert Ganian, Petr Hliněný, and Jan Obdržálek. Clique-width: when hard does not mean impossible. In *28th International Symposium on Theoretical Aspects of Computer Science*, volume 9 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages 404–415. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2011.

[67] Robert Ganian, Petr Hliněný, and Jan Obdržálek. A unified approach to polynomial algorithms on graphs of bounded (bi-)rank-width. *European J. Combin.*, 34(3):680–701, 2013.

[68] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[69] M. R. Garey, David S. Johnson, G. L. Miller, and Christos H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM J. Matrix Analysis Applications*, 1(2):216–227, 1980.

[70] James F. Geelen, A. M. H. Gerards, and Geoff Whittle. Branch-width and well-quasi-ordering in matroids and graphs. *J. Comb. Theory, Ser. B*, 84(2):270–290, 2002.

[71] Benny Godlin, Tomer Kotek, and Johann A. Makowsky. Evaluations of graph polynomials. In *Graph-Theoretic Concepts in Computer Science, 34th International Workshop, WG 2008, Durham, UK, June 30 - July 2, 2008. Revised Papers*, pages 183–194, 2008.

[72] Leslie Ann Goldberg, Rob Gysel, and John Lapinskas. Approximately counting locally-optimal structures. *J. Comput. Syst. Sci.*, 82(6):1144–1160, 2016.

[73] Petr A. Golovach, Pinar Heggernes, Mamadou Moustapha Kanté, Dieter Kratsch, Sigve H. Sæther, and Yngve Villanger. Output-polynomial enumeration on graphs of bounded (local) linear mim-width. *Algorithmica*, pages 1–28, 2017.

[74] Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Cliquewidth III: the odd case of graph coloring parameterized by cliquewidth. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 262–273, 2018.

[75] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier, 2004.

[76] Martin Charles Golumbic and Udi Rotics. On the clique-width of some perfect graph classes. *Int. J. Found. Comput. Sci.*, 11(3):423–443, 2000.

[77] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

[78] Georg Gottlob and Reinhard Pichler. Hypergraphs in model checking: Acyclicity and hypertree-width versus clique-width. *SIAM J. Comput.*, 33(2):351–378, 2004.

[79] Georg Gottlob and Stefan Szeider. Fixed-parameter algorithms for artificial intelligence, constraint satisfaction and database problems. *Comput. J.*, 51(3):303–325, 2008.

[80] Petr Hlinený. A parametrized algorithm for matroid branch-width. *SIAM J. Comput.*, 35(2):259–277, 2005.

[81] Petr Hliněný and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008.

[82] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[83] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 653–663, 1998.

[84] Lars Jaffke, O-joung Kwon, and Jan Arne Telle. A note on the complexity of feedback vertex set parameterized by mim-width. *CoRR*, abs/1711.05157, 2017.

[85] Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Polynomial-time algorithms for the longest induced path and induced disjoint paths problems on graphs of bounded mim-width. In *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, pages 21:1–21:13, 2017.

[86] Lars Jaffke, O-joung Kwon, and Jan Arne Telle. A unified polynomial-time algorithm for feedback vertex set on graphs of bounded mim-width. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 42:1–42:14, 2018.

[87] Matti Järvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. The international SAT solver competitions. *AI Magazine*, 33(1), 2012.

[88] Vít Jelínek. The rank-width of the square grid. *Discrete Applied Mathematics*, 158(7):841–850, 2010.

[89] Jisu Jeong, Eun Jung Kim, and Sang-il Oum. Finding branch-decompositions of matroids, hypergraphs, and more. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 80:1–80:14, 2018.

[90] Tao Jiang and Bala Ravikumar. Minimal NFA problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, 1993.

[91] Dong Yeap Kang, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. A width parameter useful for chordal and co-comparability graphs. *Theor. Comput. Sci.*, 704:1–17, 2017.

[92] Mamadou Moustapha Kanté. The rank-width of directed graphs. *CoRR*, abs/0709.1433, 2007.

[93] Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the enumeration of minimal dominating sets and related notions. *SIAM J. Discrete Math.*, 28(4):1916–1929, 2014.

[94] Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, Lhouari Nourine, and Takeaki Uno. On the enumeration and counting of minimal dominating sets in interval and permutation graphs. In *Algorithms and computation*, volume 8283 of *Lecture Notes in Comput. Sci.*, pages 339–349. Springer, Heidelberg, 2013.

[95] Mamadou Moustapha Kanté and Michaël Rao. The rank-width of edge-coloured graphs. *Theory Comput. Syst.*, 52(4):599–644, 2013.

[96] Mamadou Moustapha Kanté and Takeaki Uno. Counting minimal dominating sets. In *Theory and Applications of Models of Computation - 14th Annual Conference, TAMC 2017, Bern, Switzerland, April 20-22, 2017, Proceedings*, pages 333–347, 2017.

[97] Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, pages 85–103, 1972.

[98] Ki Hang Kim. *Boolean matrix theory and applications*, volume 70. Dekker, 1982.

[99] Daniel Kobler and Udi Rotics. Polynomial algorithms for partitioning problems on graphs with fixed clique-width (extended abstract). In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA.*, pages 468–476, 2001.

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

[101] Anton Kotzig. Moves without forbidden transitions in a graph. *Matematický časopis*, 18(1):76–80, 1968.

[102] Michael A. Langston. Practical FPT implementations and applications (invited talk). In *Parameterized and Exact Computation, First International Workshop, IWPEC 2004, Bergen, Norway, September 14-17, 2004, Proceedings*, pages 291–292, 2004.

[103] Michael A. Langston, Andy D. Perkins, Arnold M. Saxton, Jon A. Scharff, and Brynn H. Voy. Innovative computational methods for transcriptomic data analysis: A case study in the use of FPT for practical algorithm design and implementation. *Comput. J.*, 51(1):26–38, 2008.

[104] Leonid Anatolevich Levin. Universal sequential search problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973.

[105] Johann A. Makowsky. Colored tutte polynomials and kaufman brackets for graphs of bounded tree width. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA.*, pages 487–495, 2001.

[106] Johann A. Makowsky, Udi Rotics, Ilya Averbouch, and Benny Godlin. Computing graph polynomials on graphs of bounded clique-width. In *Graph-Theoretic Concepts in Computer Science, 32nd International Workshop, WG 2006, Bergen, Norway, June 22-24, 2006, Revised Papers*, pages 191–204, 2006.

[107] Rajesh Matai, Surya Singh, and Murari Lal Mittal. Traveling salesman problem: an overview of applications, formulations, and solution approaches. In *Traveling Salesman Problem, Theory and Applications*. InTech, 2010.

[108] Stefan Mengel. Lower bounds on the mim-width of some graph classes. *Discrete Applied Mathematics*, 2017.

[109] Pedro Montealegre and Ioan Todinca. On distance-d independent set and other problems in graphs with "few" minimal separators. In *Graph-Theoretic Concepts in Computer Science - 42nd International Workshop, WG 2016, Istanbul, Turkey, June 22-24, 2016, Revised Selected Papers*, pages 183–194, 2016.

[110] Hannes Moser and Somnath Sikdar. The parameterized complexity of the induced matching problem. *Discrete Applied Mathematics*, 157(4):715–727, 2009.

[111] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[112] Yoshio Okamoto, Takeaki Uno, and Ryuhei Uehara. Counting the number of independent sets in chordal graphs. *J. Discrete Algorithms*, 6(2):229–242, 2008.

[113] Sebastian Ordyniak, Daniël Paulusma, and Stefan Szeider. Satisfiability of acyclic and almost acyclic CNF formulas. *Theor. Comput. Sci.*, 481:85–99, 2013.

[114] James Orlin. Contentment in graph theory: Covering graphs with cliques. *Indagationes Mathematicae (Proceedings)*, 80(5):406 – 424, 1977.

[115] Sang-Il Oum. *Graphs of Bounded Rank Width*. PhD thesis, Princeton University, 2005.

[116] Sang-il Oum. Rank-width and vertex-minors. *J. Combin. Theory Ser. B*, 95(1):79–100, 2005.

[117] Sang-il Oum. Rank-width is less than or equal to branch-width. *Journal of Graph Theory*, 57(3):239–244, 2008.

[118] Sang-Il Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5(1):Art. 10, 20, 2009.

[119] Sang-il Oum. Rank-width: Algorithmic and structural results. *Discrete Applied Mathematics*, 231:15–24, 2017.

[120] Sang-il Oum, Sigve Hortemo Sæther, and Martin Vatshelle. Faster algorithms for vertex partitioning problems parameterized by clique-width. *Theoret. Comput. Sci.*, 535:16–24, 2014.

[121] Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

[122] Robert Paige and Robert E Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[123] Michal Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, pages 520–531, 2011.

[124] J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.

[125] Michaël Rao. *Décompositions de Graphes et Algorithmes Efficaces*. PhD thesis, Université Paul Verlaine, Metz, 2006.

[126] Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

[127] Neil Robertson and Paul D. Seymour. Graph minors. x. obstructions to tree-decomposition. *J. Comb. Theory, Ser. B*, 52(2):153–190, 1991.

[128] Sigve Hortemo Sæther and Martin Vatshelle. Hardness of computing width parameters based on branch decompositions over the vertex set. *Electronic Notes in Discrete Mathematics*, 49:301–308, 2015.

[129] Jan Arne Telle and Andrzej Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Discrete Math.*, 10(4):529–550, 1997.

[130] L. G. Valiant. The complexity of computing the permanent. *Theoret. Comput. Sci.*, 8(2):189–201, 1979.

[131] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

[132] Martin Vatshelle. *New width parameters of graphs*. PhD thesis, University of Bergen, Bergen, Norway, 2012.

[133] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898, 2012.

[134] Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, pages 17–29, 2015.

[135] Chain-Chin Yen and Richard C. T. Lee. The weighted perfect domination problem and its variants. *Discrete Applied Mathematics*, 66(2):147–160, 1996.