Contributions to approximation and parameterization in combinatorial optimization

Nikolaos Melissinos

PSL UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Université Paris Dauphine

Contributions to Approximation and Parameterization in Combinatorial Optimization

Soutenue par
Nikolaos MELISSINOS
Le 02/12/2022

École doctorale n°543
SDOSE

Spécialité
Informatique

Composition du jury :

Cristina BAZGAN Professeur, Université Paris Dauphine	*Président du jury*
Bruno ESCOFFIER Professeur, Sorbonne Université	*Rapporteur*
Bernard RIES Professeur, University of Fribourg	*Rapporteur*
Marthe BONAMY Chargé de recherche, Université de Bordeaux	*Examinateur*
Ararat HARUTYUNYAN Maître de conférences, Université Paris Dauphine	*Examinateur*
Aris PAGOURTZIS Professeur, National Technical University of Athens	*Examinateur*
Laurent GOURVES Directeur de recherche CNRS, Université Paris Dauphine	*Directeur de thèse*

Dauphine | PSL UNIVERSITÉ PARIS

Acknowledgment

First of all, I would like to thank my supervisors Laurent Gourvès and Ararat Harutyunyan who did their best to guide me and provide me with the best possible environment to develop both as a scientist and as a person. Their doors were always open and they were more than willing to help me with any request. Thank you both for being by my side in this very beautiful journey called Ph.D.

It is also an honor to have Prof. Bruno Escoffier from Sorbonne University and Prof. Bernard Ries from University of Fribourg as reviewers of my thesis. I want to thank you for agreeing review my work. Your comments on my thesis were not only pleasant but also extremely helpful in order to improve my work.

I would also like to thank Prof. Cristina Bazgan from Dauphine University, Dr. Marthe Bonamy from University of Bordeaux and Prof. Aris Pagourtzis from National Technical University of Athens how accept to be committee members of my thesis. Thank you for the time you spend to read my thesis and to attend in me defense. I also appriciate all the comments and thoughts you shared with me about my work. Special thanks to Professor Aris Pagourtzis; I truly believe that I would not be here without your guidance in my previous studies.

I also want to thank all the people I have worked with over the years. First of all, I am really grateful to have worked with Professor Michael Lampis. Thanks to him I got to know and love parameterized complexity. All the permanent members of LAMSADE as well as all the PhD students in the lab. Thank you for providing me with a really delightful environment both in and out of the university. My dear friends Foivos Fioravantes, Theofilos Triommatis with whom we share the same passion for computer science. And finally, all the members of my former lab, CORLAB, at the National Technical University of Athens. Thank you all for the pleasant collaboration we had.

I could not forget my family and friends in Greece who have put up with me for so many years. My mother, Kostantina, my sister Panagiota, and my friends Christina, Alexia, Antreas and Achilios. Finally, to the love of my life, Georgia, thank you for all the support you have provide me all these years.

Abstract

In Computer Science an optimization problem is a problem where we try to find the best solution from all feasible solutions. This kind of problems may arise from other sciences, like mathematics, physics and economics, or even from industry. While we are dealing with such a problem our goal is to present efficient algorithms that are computing optimal solutions for any given instance. Unfortunately, in many cases we have to deal with NP-hard problems for which it is a common belief that no such algorithm exists. Due to that, several approaches and techniques have been developed to deal with NP-hardness.

In this manuscript we are interested in two specific approaches. First, we consider approximate solutions. The concept of polynomial time approximation exists from decades. Here, instead of trying to find the "best" solution we are searching for a "good" one that can be computed in polynomial time. There are problems for which we can develop polynomial-time approximation schemes that return solutions as close to the optimal as we want, others that can only admit constant-factor approximation algorithms and others do not even admit constant-factor approximation algorithms.

In the second approach, we consider restricted instances for the problem. In such a approach we make additional assumptions for the instance of the given problem. This idea comes form the fact that sometimes we are not interested in solving all the instances of a problem but just a subclass and instances in this subclass have properties that we can exploit in our algorithms. In this thesis we mostly apply this approach on graph problems. Some assumptions are related to the structure of the given input (for graph problems this could for example mean that the graph is regular or bipartite) while other assumptions deal with the size of specific parameter (like the degree or the treewidth of the given graph). In the second case, we say that we are interested in the parameterized complexity of the problem. When we consider a parameter, our goal is to find exact algorithms whose running time is in the form $f(k)n^{O(1)}$, where k is the size of the parameter and n the size of the instance. When the approach is successful, we say that we have an fpt-algorithm.

In the first chapter we investigate problems that are similar to the well-known Subset Sum problem. In particular, we study the Subset Sums Ratio problem (SSR) and some variants of it. In SSR we have as input a set A of positive integers and we want to find two disjoint subsets that have ratio as close to one as possible. We present a new FPTAS algorithm for the SSR and we develop a generic framework that yields FPTAS's for a family of variations of SSR under some conditions.

In the rest of this manuscript we study graph problems. Some of these problems are variants of well-known graph problems like the Feedback Vertex Set problem and the Vertex Coloring problem. For these problems we present approximation algorithms, inapproximability results and parameterized algorithms. Finally, using standard assumptions like the Exponential Time Hypothesis, we present lower bounds on the running time of our parameterizad algorithms that matches the running time of our algorithms.

Résumé

En informatique, un problème d'optimisation est un problème où l'on essaie de trouver la meilleure solution parmi toutes les solutions réalisables. Ce type de problème peut provenir d'autres sciences, comme les mathématiques, la physique, l'économie, ou même de l'industrie. Lorsque nous traitons un tel problème, notre objectif est de proposer des algorithmes efficaces qui calculent des solutions optimales pour toute instance donnée. Malheureusement, dans de nombreux cas, nous devons traiter des problèmes NP-difficiles pour lesquels il est communément admis qu'aucun algorithme de la sorte n'existe. Pour cette raison, plusieurs approches et techniques ont été développées pour traiter la NP-difficulté.

Dans ce manuscrit, nous nous intéressons à deux approches spécifiques. Premièrement, nous considérons les solutions approchées. Le concept d'approximation en temps polynomial existe depuis des décennies. Ici, au lieu d'essayer de trouver la "meilleure" solution, nous cherchons une "bonne" solution qui peut être calculée en temps polynomial. En particulier, nous voulons développer un algorithme d'approximation en temps polynomial qui retourne des solutions aussi proches de l'optimum que possible. Dans la deuxième approche, nous considérons des instances restreintes du problème. Dans une telle approche, nous faisons des hypothèses supplémentaires sur l'instance du problème donné. Cette idée vient du fait que parfois nous ne sommes pas intéressés à résoudre toutes les instances d'un problème mais seulement une sous-classe et les instances de cette sous-classe ont des propriétés que nous pouvons exploiter dans nos algorithmes. Dans cette thèse, nous appliquons principalement cette approche aux problèmes de graphes. Certaines hypothèses sont liées à la structure de l'entrée donnée (pour les problèmes de graphes, cela peut signifier par exemple que le graphe est régulier ou biparti) tandis que d'autres hypothèses concernent la taille d'un paramètre spécifique (comme le degré ou la largeur d'arbre du graphe donné). Dans le second cas, nous disons que nous sommes intéressés par la complexité paramétrée du problème. Lorsque nous considérons un paramètre, notre objectif est de trouver des algorithmes exacts dont le temps d'exécution est de la forme $f(k)n^{O(1)}$, où k est la taille du paramètre et n la taille de l'instance. Lorsque cette approche est possible, nous disons que nous avons un algorithme *fpt*.

Dans le premier chapitre, nous étudions des problèmes similaires au problème bien connu de la somme de sous-ensembles (Subset Sum). En particulier, nous étudions le problème Subset Sum Ratio (SSR) et certaines de ses variantes. Dans SSR, nous avons en entrée un ensemble A d'entiers positifs et nous voulons trouver deux sous-ensembles disjoints qui ont un rapport aussi proche de un que possible. Nous présentons un nouvel algorithme FPTAS pour SSR et nous développons un cadre générique qui donne des FPTAS pour une famille de variantes du SSR sous certaines conditions.

Dans la suite de ce manuscrit, nous étudions les problèmes de graphes. Certains de ces problèmes sont des variantes de problèmes de graphes bien connus comme le problème Feedback Vertex Set et le problème Vertex Coloring. Pour ces problèmes, nous présentons des algorithmes d'approximation, des résultats d'inapproximabilité et des al-

gorithmes paramétrés. Enfin, en utilisant des hypothèses standard comme l'hypothèse du temps exponentiel (ETH), nous présentons des bornes inférieures sur le temps d'exécution de nos algorithmes paramétrés qui correspondent au temps d'exécution de nos algorithmes.

Résumé étendu

Élements de complexité, approximation, graphes et paramétrisation

Dans cette thèse, nous considérons des problèmes combinatoires appartenant à la classe NP. De manière informelle, NP est la classe contenant tous les problèmes de décision pour lesquels on peut vérifier, en temps polynomial, si une réponse donnée est la bonne réponse. Une sous-classe de NP est la classe P qui contient tous les problèmes de décision qui peuvent être résolus en temps polynomial. Il est facile de voir que P $\subseteq$ NP, cependant la question de savoir si P = NP ou non est encore ouverte.

De nombreux problèmes intéressants appartiennent à la catégorie des problèmes NP-complets. Pour définir les problèmes NP-complets, nous devons d'abord définir les "réductions polynomiales".

Définition (Réduction en temps polynomial)**.** *Une réduction en temps polynomial d'un problème A à un problème B est un algorithme en temps polynomial qui transforme toute instance I_A de A en une instance I_B de B telle que I_B est une instance positive de B si et seulement si I_A est une instance positive de A.*

Nous disons qu'un problème A est NP-complet s'il appartient à NP et que tout autre problème B dans NP est réductible en temps polynomial à A. Si nous ne pouvons vérifier que la deuxième condition, nous disons que le problème est NP-difficile.

Le premier problème dont on a prouvé qu'il était NP-complet est le problème bien connu de la Satisfiabilité (connu sous le nom de SAT) [61].

Définition (SAT)**.** *Soit ϕ une formule logique sur un ensemble de n variables $\{x_1, \ldots, x_n\}$. L'objectif est de décider s'il existe une affectation de valeurs de vérité aux variables qui satisfait la formule.*

L'une des variantes les plus importantes de SAT, qui est également NP-complète, est 3-SAT. La différence entre SAT et 3-SAT est que dans cette dernière la formule ϕ est sous forme normale conjonctive (formule CNF) et que chaque clause contient trois littéraux.

Lorsque nous considérons un problème d'optimisation, dans de nombreux cas, nous pouvons être satisfaits si nous disposons d'un algorithme en temps polynomial qui renvoie des solutions réalisables proches de l'optimum. Pour quantifier la distance entre une

solution réalisable et une solution optimale, nous utilisons le rapport entre le poids de la solution réalisable et le poids de la solution optimale. Si, pour un algorithme donné, nous pouvons prouver que ce rapport est borné pour toute instance donnée, alors nous disons que cet algorithme est un algorithme d'approximation. De plus, si pour un algorithme ce rapport r satisfait $1 \geq r \geq a$ (respectivement $1 \leq r \leq a$) pour toutes les instances d'un problème de maximisation (respectivement d'un problème de minimisation), alors cet algorithme est un algorithme d'approximation de rapport a.

Comme tous les algorithmes n'ont pas le même rapport et la même durée d'exécution, il a été décidé de les classer en fonction de ces aspects. Les catégories les plus importantes sont les suivantes :

- Algorithme d'approximation à facteur constant : Tout algorithme à facteur constant qui s'exécute en temps polynomial.
- Schéma d'approximation en temps polynomial (PTAS) : Il s'agit d'un algorithme d'approximation, pour un problème de maximisation (respectivement, pour un problème de minimisation), ayant comme information supplémentaire en entrée un nombre réel $\varepsilon > 0$, ayant un rapport $1 - \varepsilon$ (respectivement, ayant un rapport $1+\varepsilon$) et son temps d'exécution est un polynôme de n (où n est la taille de l'entrée).
- Schéma d'approximation en temps entièrement polynomial (FPTAS) : Ces algorithmes sont des algorithmes PTAS dont le temps d'exécution est un polynôme de n et $1/\varepsilon$.

Par conséquent, les problèmes d'optimisation peuvent être catégorisés en fonction du meilleur rapport d'approximation qu'ils peuvent admettre. Nous présentons ci-dessous quelques-unes de ces classes :

- FPTAS: Classe de problèmes qui admettent des FPTAS.
- PTAS: Classe de problèmes qui admettent des PTAS.
- APX: Classe de problèmes admettant des algorithmes d'approximation à facteur constant.
- $f(n)$-APX: Classe de problèmes qui admettent des algorithmes d'approximation $O(f(n))$ s'exécutant en temps polynomial.

Cependant, dans plusieurs cas, nous ne pouvons pas être sûrs que nos algorithmes atteignent le meilleur rapport d'approximation pour un problème. C'est pourquoi nous cherchons des limites inférieures. Une façon de calculer les limites du rapport d'approximation pour un problème donné est de présenter des réductions génératrices d'écarts. L'idée derrière les réductions d'écart est la suivante :

Soit A un problème de minimisation et B un problème de décision. De plus, pour une instance $y \in I_A$, nous désignons par $opt(y)$ le poids d'une solution optimale de y. Supposons maintenant que pour deux nombres k et c, nous puissions donner une correspondance de l'ensemble des instances I_B de B à l'ensemble des instances I_A de A telle que :

- toutes les instances positives $x \in I_B$ sont mises en correspondance avec un $y \in I_A$ où $opt(y) \leq k$ et
- toutes les instances négatives $x \in I_B$ sont mises en correspondance avec une $y \in I_A$ où $opt(y) > ck$.

Alors, si B est un problème NP-difficile, on ne peut pas décider en temps polynomial si, pour une instance y de A, on a $opt(y) \leq k$ ou $opt(y) > ck$. Par conséquent, le problème A n'admet aucun algorithme d'approximation de r pport c qui s'exécute en temps polynomial.

Lorsqu'un problème est NP-difficile, on peut essayer de considérer des contraintes supplémentaires afin de trouver des moyens de le traiter. Certaines contraintes peuvent être liées à la taille d'un paramètre d'un problème particulier; la taille de la solution optimale en est un exemple. Une question intéressante qui se pose lors de l'examen des paramètres est la suivante : "Si la taille de ce paramètre est petite, pouvons-nous résoudre le problème plus rapidement?". Lorsque nous considérons de telles questions, nous disons que nous considérons des problèmes paramétrés (c'est-à-dire des problèmes avec un paramètre particulier k) et nous étudions la complexité paramétrée d'un problème sous ce paramètre.

Étant donné un problème paramétré avec un paramètre k, nous voulons décider s'il existe un moyen de calculer une solution optimale en temps $f(|k|)poly(n)$ où $f(|k|)$ est une fonction calculable et $poly(n)$ est un polynôme de la taille de l'entrée n. Un algorithme ayant ces propriétés est appelé algorithme fpt. En utilisant cette idée, nous définissons la classe FPT comme suit :

Définition (Classe FPT)**.** *La classe des problèmes paramétrés qui admettent des algorithmes fpt.*

Malheureusement, tous les problèmes paramétrés n'appartiennent pas à la classe FPT. Ci-dessous, nous présentons la hiérarchie W telle que proposée par Downey et Fellows [74] :

$$\mathrm{P} \subseteq \mathrm{FPT} = \mathrm{W}[0] \subseteq \mathrm{W}[1] \cdots \subseteq \mathrm{W}[\mathrm{t}] \cdots \subseteq \mathrm{XP}.$$

Ils ont conjecturé que les inclusions sont strictes; cette conjecture reste ouverte. Pour une définition formelle des classes paramétrées ci-dessus et de la hiérarchie W, le lecteur peut se référer à [75].

Comme pour la complexité classique, l'outil que nous utilisons pour décider de la complexité paramétrée d'un problème est une réduction. Comme la complexité d'un problème paramétré dépend de la taille du paramètre, nous avons besoin d'une réduction qui prend en compte le paramètre.

Définition (fpt-réduction)**.** *Soit (A, k) et (B, k') deux problèmes paramétrés. Nous disons que nous avons une réduction fpt de la forme (A, k) à (B, k') s'il existe un algorithme qui transforme toute instance (x, k) de (A, k) en une instance (y, k') de (B, k') et*

- *(x, k) est une instance* oui *si et seulement si (y, k') est une instance* oui*,*

- *il existe une fonction calculable g telle que $|k'| \leq g(|k|)$ et*
- *l'algorithme s'exécute en temps $f(|k|)poly(n)$, pour une fonction calculable f.*

De même que pour les réductions polynomiales et les problèmes NP-complets, ici, un problème paramétré (A, k) est W[i]-complet s'il appartient à W[i] et que tout autre problème paramétré dans W[i] est fpt-réductible à (A, k).

Les algorithmes paramétrés (même les algorithmes FPT) ont des temps d'exécution vraiment mauvais par rapport au paramètre. Pour cette raison, nous essayons toujours de trouver des algorithmes qui ont une dépendance minimale vis-à-vis de la taille du paramètre. Pour vérifier qu'un algorithme ne peut pas être amélioré, nous essayons de trouver des limites inférieures de calcul. Certaines de ces limites sont basées sur l'hypothèse P≠NP, mais dans de nombreux cas, nous utilisons d'autres hypothèses plus fortes. Le principal outil sur lequel nous nous appuierons est l'hypothèse de temps exponentiel (ETH) de Impagliazzo, Paturi, et Zane [120], qui énonce ce qui suit :

Conjecture. *Hypothèse du temps exponentiel: il existe un $\epsilon > 0$, tel que 3-SAT sur des instances avec n variables et m clauses ne peut être résolu en temps $2^{\epsilon(n+m)}$.*

Plusieurs des problèmes que nous considérons dans cette thèse sont des problèmes de graphes. Un graphe G est constitué de deux ensembles, un ensemble V de sommets et un ensemble E d'arêtes. L'ensemble E indique, dans un certain sens, si deux sommets sont liés ou non. Si deux sommets u et v dans V sont liés, alors une arête $e = uv$ appartient à E, sinon $uv \notin E$.

Comme nous l'avons mentionné précédemment, dans de nombreux cas, nous supposons que nous avons des contraintes supplémentaires sur nos instances du problème. Dans les problèmes de graphes, ces contraintes sont généralement liées aux propriétés structurelles du graphe d'entrée.
Nous présentons ci-dessous quelques-unes des classes les plus courantes de graphes non orientés.

- Graphes complets : Graphes qui contiennent toutes les arêtes possibles. Un graphe complet à n sommets est noté K_n.
- Arbres : graphes qui ne contiennent pas de cycles.
- Graphes bipartis : graphes qui ne contiennent pas de cycles de longueur impaire.
- Graphes bipartis complets : graphes bipartis qui contiennent toutes les arêtes possibles. Un graphe biparti complet avec $|L| = n$ et $|R| = m$ est noté $K_{n,m}$.
- Graphes réguliers : Graphes où tous les sommets ont le même degré. Si en plus on connaît le degré d des sommets alors on appelle le graphe d-régulier.
- Graphes subcubiques : Graphes de degré maximum $\Delta \leq 3$.
- Graphes planaires : Un graphe est dit planaire s'il existe une manière de le dessiner dans un plan à deux dimensions telle qu'aucune paire d'arêtes ne se croise.

L'un des problèmes de graphes les plus étudiés, NP-difficile, est le problème de la Maximum Independent Set.

Définition (Maximum Independent Set (Max IS)). *Étant donné un graphe $G = (V, E)$, un ensemble $S \subseteq V$ est appelé ensemble indépendant si pour tout $u, v \in S$ on a $uv \notin E$. Dans Max IS nous recherchons un ensemble indépendant de G, de taille maximale.*

La taille d'un ensemble indépendant maximal d'un graphe G est désignée par $\alpha(G)$. Dans [183], il a été prouvé que, pour tout $\varepsilon > 0$, il n'existe aucun algorithme en temps polynomial qui approche $\alpha(G)$, pour un graphe G, selon un rapport de $n^{1-\varepsilon}$, à moins que P = NP. Une version plus faible de ce résultat a été présentée pour la première fois dans [106] sous la condition NP $\neq$ ZPP au lieu de P $\neq$ NP. La version paramétrée de Max IS, lorsque nous considérons la taille de la solution comme paramètre, est W[1]-difficile et elle ne peut pas être résolue en $|V|^{o(k)}$ temps, sous la condition ETH [65].

Il existe plusieurs problèmes, tels que Max IS, qui recherchent des sous-ensembles de sommets ou d'arêtes qui satisfont une certaine propriété et ont la plus grande (ou la plus petite) taille possible. Certaines propriétés bien connues sont présentées ci-dessous. Supposons que l'on nous donne un graphe $G = (V, E)$, nous définissons alors ce qui suit:

- Vertex Cover de G : Un ensemble $S \subseteq V$ est appelé *vertex cover* de G si pour chaque arête $e = uv \in E$ on a que $u \in S$ ou $v \in S$.
- Dominating Set de G : Un ensemble $S \subseteq V$ est appelé *dominating set* de G si pour tout sommet $u \in V$ on a soit $u \in S$, soit il existe $v \in S$ tel que $uv \in E$.
- Matching de G : Un ensemble $S \subseteq E$ est appelé *matching* de G si pour tout sommet $u \in V$ il existe au plus une arête $e \in S$ qui est incidente à u.

En utilisant les propriétés précédentes, nous pouvons définir les problèmes suivants :

Définition (Minimum Vertex Cover (Min VC)). *Soit $G = (V, E)$ un graphe. Le problème Minimum Vertex Cover demande de trouver un vertex cover de G de plus petite cardinalité.*

Min VC est considéré comme le complément de Max IS. En effet, pour un graphe donné $G = (V, E)$, le complément $V \setminus S$, de tout vertex cover S de G, est un ensemble indépendant. Comme Max IS, Min VC est également NP-difficile; cependant, pour Min VC, il existe un algorithme d'approximation à rapport 2. De plus, sa version paramétrée, lorsqu'elle est paramétrée par la taille de la solution, appartient à FPT.

Définition (Minimum Dominating Set (Min DS)). *Soit $G = (V, E)$ un graphe. Le problème Min DS demande de trouver un dominating set de G de plus petite cardinalité.*

Min DS est NP-difficile et sa version paramétrée, lorsqu'elle est paramétrée par la taille de la solution, est W[2]-complète. La taille d'un ensemble dominant minimal, d'un graphe G donné, est appelée nombre de domination et est notée par $\gamma(G)$.

Définition (Maximum Matching problem)**.** *Soit $G = (V, E)$ un graphe. Le problème Maximum Matching demande de trouver un matching de G de taille maximale.*

Le problème du Maximum Matching appartient à P.

L'un des paramètres structurels les plus connus d'un graphe est la largeur d'arbre (treewidth). Pour définir la treewidth d'un graphe, nous devons d'abord définir ce que nous appelons la décomposition en arbre d'un graphe.

Définition (Décomposition en arbre)**.** *Étant donné un graphe $G = (V, E)$, nous appelons décomposition arborescente de G un arbre $T = (V', E')$ ainsi qu'un ensemble $\mathcal{B} = \{B_t \subseteq V \mid t \in V'\}$ tel que:*

- *Pour chaque sommet $t \in V'$ a été attribué un ensemble $B_t \subseteq V$. Nous appelons cet ensemble sac de t.*
- *$\bigcup_{t \in V'} B_t = V$.*
- *Soit $t, w \in V'$ et $u \in B_t \cap B_w$. Pour tout $x \in V'$, où x appartient au chemin (unique) de t à w dans T, on a que $u \in B_x$.*
- *Soit $uv \in E$. Il existe un $t \in V'$ tel que $u, v \in B_t$.*

La taille du plus grand sac de T moins 1 est appelée largeur de la décomposition en arbre.

Définition (Treewidth)**.** *Étant donné un graphe G, nous recherchons le nombre minimum $tw \in \mathbb{N}$ tel que, il existe une décomposition en arbre de G avec une largeur de tw et il n'existe aucune décomposition en arbre de G avec une largeur inférieure à tw.*

La treewidth d'un graphe est un paramètre important. Afin de tirer parti de la treewidth tw d'un graphe $G = (V, E)$, nous devons calculer une décomposition d'arbre $(T, \mathcal{B})$, où $T = (V', E')$ est un arbre binaire enraciné, qui satisfait les propriétés suivantes:

- la largeur de la décomposition de l'arbre est de tw,
- chaque nœud de T appartient à exactement une des catégories suivantes:
  - Nœud feuille : une feuille, v, de T telle que $|B_v| = 1$.
  - Nœud *introduire* v: un nœud, v, de T avec un fils, c, tel que $B_c \subset B_v$ et $|B_v \setminus B_c| = 1$.
  - Nœud *oublier* : un nœud, v, de T avec un fils, c, tel que $B_v \subset B_c$ et $|B_c \setminus B_v| = 1$.
  - Nœud *joindre* : un nœud, v, de T avec deux fils, c et c', tel que $B_v = B_c = B_{c'}$.

Cette décomposition en arbre dite *nice* a été introduite dans [33]. Il existe un algorithme qui transforme une décomposition en arbre donnée en une décomposition en arbre *nice* avec $O(n\,tw)$ nœuds en $O(n^2 tw)$.

En général, chaque sac d'une décomposition en arbre est un ensemble séparant du graphe original. Cette propriété nous permet de tirer avantage de la structure de la décomposition *nice* en arbre et nous permet d'utiliser la programmation dynamique à ses nœuds. Pour plus de détails sur les décompositions en arbres, voir [75].

Subset-Sums Ratio et variations

Dans le Chapitre 3, nous étudions le problème Subset-Sums Ratio. La définition formelle du problème est la suivante:

Définition (Subset-Sums Ratio (SSR)). *Étant donné un ensemble $A = \{a_1, \ldots, a_n\}$ de n entiers positifs, trouver deux ensembles non vides et disjoints $S_1, S_2 \subseteq \{1, \ldots, n\}$ qui minimisent le rapport:*

$$\frac{\max\{\sum_{i \in S_1} a_i, \sum_{j \in S_2} a_j\}}{\min\{\sum_{i \in S_1} a_i, \sum_{j \in S_2} a_j\}}.$$

Ce problème est la version optimisation du problème Equal Subset Sum (ESS) qui, étant donné un ensemble d'entrée, demande deux sous-ensembles disjoints de somme égale. ESS trouve des applications dans de nombreux domaines différents, allant de la biologie computationnelle [60, 57], du choix social computationnel [144], à la cryptographie [179], pour n'en citer que quelques-uns. De plus, elle est liée à des concepts théoriques importants tels que la complexité des problèmes de recherche de la classe TFNP [166].

Ces deux problèmes sont fortement liés au problème bien connu de la somme des sous-ensembles. Les calculs de la somme des sous-ensembles sont d'une importance capitale en informatique, car ils apparaissent soit comme des problèmes en soi, soit comme des sous-problèmes dans un grand nombre de méthodes théoriques et pratiques permettant de résoudre d'importants problèmes de calcul. Comme la plupart des problèmes de somme de sous-ensembles sont NP-difficiles, un effort a été fait au fil des ans pour trouver des moyens systématiques de dériver des schémas d'approximation pour ces problèmes. En particulier, le FPTAS le plus connu pour le problème de la somme de sous-ensembles apparaît dans [129]. Parmi les contributions importantes dans cette direction, citons les travaux de Gens et Levner [95], Horowitz et Sahni [114, 115], Ibarra et Kim [119], Sahni [174], Woeginger [180], et Woeginger et Pruhs [169].

Dans la première moitié de ce chapitre, nous présentons un nouveau schéma d'approximation pour le problème Subset-Sums Ratio. Notre algorithme utilise des algorithmes exacts et d'approximation pour Subset Sum, ainsi, toute amélioration par rapport à ces derniers s'applique à notre schéma proposé. De plus, selon la relation entre n et ε, notre algorithme améliore le meilleur schéma d'approximation existant de [151].

En particulier, dans la section 3.3, nous introduisons un FPTAS pour une version du problème appelée Subset-Sums Ratio contraint. Ensuite, nous expliquons comment utiliser l'algorithme présenté dans la section précédente, afin d'obtenir un schéma d'approximation pour le problème Subset-Sums Ratio. La complexité de ce FPTAS est analysée en détail dans la Section 3.5.

Plusieurs variantes d'ESS ont été prouvées NP-complètes par Cieliebak *et al.* dans [58, 59]. Cependant, la version d'optimisation de ces problèmes n'a pas été étudiée. Dans la deuxième moitié de ce chapitre, nous considérons une famille de variations de SSR qui inclut tous ces problèmes. La définition formelle de cette famille est assez technique et présentée ci-dessous.

Famille de problèmes de type Subset-Sums ratio (F-SSR). *Un problème $\mathcal{P}$ dans F-SSR est un problème d'optimisation combinatoire $(\mathcal{I}, k, \mathcal{F})$ où:*

- *$\mathcal{I}$ est un ensemble d'instances dont chacune est une paire (E, w) où $E = \{e_1, \ldots, e_n\}$ est un ensemble d'éléments de base et $w : E \mapsto \mathbb{R}^+$ est une fonction de poids qui fait correspondre chaque élément e_i à un nombre positif a_i;*
- *k définit le nombre de sous-ensembles de $\{1, \ldots, n\}$ que l'on recherche;*
- *$\mathcal{F}$ donne l'ensemble des solutions réalisables comme suit: pour toute entrée (E, w), $\mathcal{F}(k, E)$ est une collection de k-tuples de sous-ensembles non vides et disjoints de $\{1, \ldots, n\}$, et étant donné $(k, E, (S_1, \ldots, S_k))$, nous pouvons vérifier en temps polynomial si $(S_1, \ldots, S_k)$ est dans $\mathcal{F}(k, E)$.*

Pour une instance (E, w) de $\mathcal{P}$ nous définissons le poids d'une solution réalisable $(S_1, \ldots, S_k)$ comme étant le rapport:

$$\frac{\max\{\sum_{i \in S_1} a_i, \ldots, \sum_{j \in S_k} a_j\}}{\min\{\sum_{i \in S_1} a_i, \ldots, \sum_{j \in S_2} a_j\}}$$

où $a_i = w(e_i)$ pour tous les $e_i \in E$.

Enfin, le but de $\mathcal{P}$ est de trouver, pour une instance (E, w), une solution réalisable $(S_1^*, \ldots, S_k^*)$ avec un poids minimum.

Dans la section 3.7, nous présentons certaines conditions qui, si elles sont satisfaites par un problème $\mathcal{P}$ dans cette famille, alors $\mathcal{P}$ admet un FPTAS. La plus importante de ces conditions est l'existence d'un algorithme pseudopolynomial pour une version restreinte de $\mathcal{P}$. L'idée derrière notre cadre est de définir un paramètre d'échelle δ que nous utiliserons pour réduire les valeurs d'entrée, ce qui permet aux algorithmes pseudopolynomiaux de s'exécuter en temps polynomial. Nous devons noter que des algorithmes d'approximation plus rapides peuvent exister pour ces problèmes. Cependant, notre objectif n'était pas de trouver les algorithmes les plus rapides possibles mais de garantir des FPTAS pour le plus grand nombre de problèmes possible.

À titre d'exemples particuliers, nous présentons deux problèmes qui appartiennent à F-SSR: le 2-Set SSR et le Factor-r SSR.

Two-Set Subset-Sums Ratio problem (2-Set SSR). *Soit $A = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ un ensemble de paires de nombres positifs. Nous cherchons deux ensembles non vides et disjoints $S_1, S_2 \subseteq \{1, ..., n\}$ qui minimisent*

$$\frac{\max\{\sum_{i \in S_1} a_i, \sum_{j \in S_2} b_j\}}{\min\{\sum_{i \in S_1} a_i, \sum_{j \in S_2} b_j\}} .$$

Factor-r Subset-Sums Ratio problem (Factor-r SSR). *Étant donné un ensemble $A = \{a_1, \ldots, a_n\}$ de n nombres positifs et un nombre $r \geq 1$, trouver deux ensembles non vides et disjoints $S_1, S_2 \subseteq \{1, \ldots, n\}$ qui minimisent le rapport*

$$\frac{\max\{r \cdot \sum_{i \in S_1} a_i, \sum_{j \in S_2} a_j\}}{\min\{r \cdot \sum_{i \in S_1} a_i, \sum_{j \in S_2} a_j\}} .$$

Les preuves qu'ils appartiennent effectivement à *F-SSR* seront présentées dans la Section 3.8.1 et la Section 3.8.2 respectivement. Nous devons noter que la version décision de Factor-r SSR a été étudiée dans [59]. Pour ces deux problèmes, nous introduisons les FPTAS. Plus précisément, nous montrons que le FPTAS que nous présentons pour le 2-Set SSR peut être utilisé comme FPTAS pour le Factor-r SSR.

Max-Min and Min-Max Problems

Dans les chapitres 4 et 5, nous considérons les versions Max-Min de Edge Cover et Feedback Vertex Set. Les versions Max-Min et Min-Max de nombreux problèmes d'optimisation célèbres ont récemment suscité un grand intérêt dans la littérature. Bien que la motivation initiale pour l'étude de ces problèmes était le désir d'analyser la pire performance possible d'une heuristique naïve, ces problèmes se sont progressivement révélés posséder une riche structure combinatoire qui les rend intéressants en soi.

De nombreux autres problèmes d'optimisation classiques ont été étudiés dans le cadre Max-Min ou Min-Max, tels que : Max Min Separator [100], Max Min Cut [79], Min Max Knapsack (également connu sous le nom de Lazy Bureaucrat Problem) [12, 90, 97], et une variante pondérée de Max Min Edge Cover [130]. Certains problèmes dans ce domaine se présentent naturellement sous d'autres formes et ont été largement étudiés, comme Min Max Matching (également connu sous le nom de Edge Dominating Set [121]), Grundy Coloring, qui peut être vu comme une version Max Min de Coloring [3, 24], et Max Min VC dans les hypergraphes, qui est connu sous le nom de Upper Transversal [155, 109, 110, 111].

Upper r-Tolerant Edge Cover

Dans le chapitre 4 nous définissons et étudions les problèmes de *tolerant edge cover*. Une *edge cover* d'un graphe $G = (V, E)$ sans sommets isolés est un sous-ensemble d'arêtes $S \subseteq E$ qui couvre tous les sommets de G, c'est-à-dire que chaque sommet de G est une extrémité d'au moins une arête dans S. Le *nombre de edge cover* d'un graphe $G = (V, E)$, noté $ec(G)$, est la taille minimale d'une couverture d'arêtes de G et il peut être calculé en temps polynomial (voir le chapitre 19 dans [175]). Une couverture d'arêtes $S \subseteq E$ est dite *minimale* (par rapport à l'inclusion) si aucun sous-ensemble propre de S n'est une edge cover. Le edge cover minimal est également connu dans la littérature sous le nom d'ensemble enclaveless [176] ou d'ensemble nonblocker [71].

Alors qu'une couverture par arêtes minimale peut être calculée efficacement, trouver la plus grande couverture par arêtes minimale est un problème NP-difficile [149]. En particulier, il est montré que le problème est équivalent à la recherche d'un ensemble dominant minimal. Le problème d'optimisation associé est appelé Upper Edge Cover (et noté Upper EC) [13] et la valeur optimale correspondante, pour un graphe $G = (V, E)$, sera notée $\mathsf{uec}(\mathsf{G})$.

Ici, nous nous intéressons aux solutions edge cover minimales tolérant les défaillances d'au plus $r - 1$ arêtes.

Définition (r-Tolerant Edge Cover)**.** *Soit $r \geq 1$ un entier et $G = (V, E)$ un graphe de degré minimum r. Nous disons qu'un ensemble $S \subseteq E$ est une couverture par arêtes r-tolérante si :*

- *tous les sommets $v \in V$ sont incidents à au moins k arêtes de S,*
- *S est minimal avec cette propriété.*

De manière équivalente, nous cherchons un sous-ensemble d'arêtes S de G tel que le sous-graphe (V, S) est le plus grand sous-graphe qui a un degré minimum r et il est minimal avec cette propriété. Notons que le cas $r = 1$ correspond à la notion standard de edge cover minimale.

Définition (Upper r-Tolerant Edge Cover problem (Upper r-EC))**.** *Étant donné un entier $r \geq 1$ et un graphe $G = (V, E)$ avec un degré minimum r, nous cherchons une r-tolerant edge cover S de G avec une taille maximum. Nous désignons par $\mathsf{uec_r}$ la taille de S.*

Pour $r = 1$, le Upper r-Tolerant Edge Cover est le même problème que le Upper Edge Cover (ou simplement le Upper EC). Un problème fortement lié au problème Upper EC est le problème Minimum Dominating Set (Min DS). Un ensemble dominant dans un graphe est un sous-ensemble S de sommets tel que tout sommet qui n'est pas dans S a au moins un voisin dans S. La taille du plus petit ensemble dominant de G est notée $\gamma(G)$. Pour tout graphe G l'égalité $\mathsf{uec(G)} = \mathsf{n} - \gamma(\mathsf{G})$ est vraie [149].

Ainsi, en utilisant les résultats de complexité connus pour Min DS, nous déduisons que Upper EC est NP-difficile dans les graphes planaires de degré maximum 3 [94], les graphes cordaux [39], les graphes bipartis, les split graphes [30] et les k-arbres avec k arbitraire [62]. Par contre, Upper EC est polynomial dans les k-arbres avec k fixe, les graphes bipartis convexes [70] et les graphes fortement cordaux [81]. Concernant l'approximabilité, une preuve de APX-difficulté avec une limite d'inapproximabilité explicite et un algorithme combinatoire d'approximation de 0,6 sont proposés dans [165]. De meilleurs algorithmes avec un rapport d'approximation de 0,71 et 0,803 sont donnés respectivement dans [53] et [15]. Pour tout $\varepsilon > 0$, Upper Edge Cover est difficile à approcher à un facteur de $\frac{259}{260} + \varepsilon$ à moins que P=NP [165].

Des notions connexes d'ensembles dominants sont introduites dans la littérature sous le nom de r-tuple domination [18, 85, 91, 101, 107, 133], et r-domination [51, 85]. Un ensemble $S \subseteq V$ est appelé un r ensemble dominant de $G = (V, E)$ si pour tout sommet $v \in V$, $|N_G(v) \cap S| \geq r$. La cardinalité minimale d'un ensemble dominant r de G est appelée nombre de domination r et généralement notée $\gamma_r(G)$. De plus, un ensemble $S \subseteq V$ est appelé un r-tuple dominating set de $G = (V, E)$ si pour tout sommet $v \in V$, $|N_G[v] \cap S| \geq r$. La cardinalité minimale d'un ensemble dominant de r de G est appelée nombre de domination de r et est généralement notée $\gamma_{\times r}(G)$.

Dans la Section 4.3 nous prouvons plusieurs propriétés concernant $\mathsf{uec_r(G)}$. En particulier, nous montrons :

Property. *Soit $r \geq 1$, pour tous les graphes $G = (V, E)$ de degré minimum au moins r, l'inégalité suivante est vraie :*

$$2\mathsf{ec_r}(\mathsf{G}) \geq \mathsf{uec_r}(\mathsf{G})$$

La propriété ci-dessus montre que la taille de toute couverture par arêtes r-tolérante de G est une approximation à rapport $\frac{1}{2}$ de $\mathsf{uec_r}(\mathsf{G})$. Nous présentons ensuite deux propriétés qui relient la valeur $\mathsf{uec_r}(\mathsf{G})$ aux valeurs $\mathsf{uec}(\mathsf{G})$ et $\gamma_r(G)$ respectivement.

Property. *Soit $r \geq 1$. Pour tous les graphes $G = (V, E)$ de degré minimal au moins égal à r, les inégalités suivantes sont vraies :*

$$\mathsf{uec_r}(\mathsf{G}) \leq r \cdot \mathsf{uec}(\mathsf{G})$$
$$\mathsf{uec_r}(\mathsf{G}) \geq r(n - \gamma_r(G))$$

Dans les sections 4.4 et 4.5, nous présentons des résultats de NP-difficulté pour Upper r-EC et Double Upper EC, et des résultats de difficulté d'approximation pour Upper EC et Double Upper EC. Plus précisément, nous montrons que Double Upper EC est NP-difficile dans les graphes bipartis cubiques et les split graphes et Upper r-EC est NP-difficile dans les graphes $r+1$ réguliers. De plus, Upper $r+1$-EC est NP-difficile dans les graphes de degré maximum $\Delta+1$ si Upper r-EC est NP-difficile dans les graphes de degré maximum Δ, et ceci est valable même pour les graphes bipartis.

Comme nous avons déjà montré qu'il existe une approximation facile de rapport $\frac{1}{2}$, nous cherchons des limites inférieures sur le rapport d'approximation en temps polynomial. Nous présentons deux réductions de la difficulté d'approximation qui nous donnent (plusieurs) rapports d'inapproximabilité pour Upper Edge Cover et Upper 2-Tolerant Edge Cover dans les graphes avec un degré maximal borné.

Max-Min FVS

Dans le chapitre 5, nous considérons une variation Max-Min de Feedback Vertex Set. Typiquement, le Feedback Vertex Set est étudié avec un objectif de minimisation tel qu'il a été défini précédemment. Ici, nous nous intéressons à un objectif qui est, en un sens, l'inverse : nous cherchons un feedback vertex set S qui est aussi grand que possible, tout en étant minimal. Nous appelons ce problème Max Min FVS. Notre objectif dans ce chapitre est de montrer que Max Min FVS présente un comportement de complexité intéressant en ce qui concerne son approximabilité.

La raison pour laquelle nous nous concentrons sur le problème Max Min FVS est le contraste entre deux de ses cousins plus étudiés : les problèmes Max Min Vertex Cover (Max Min VC) et Upper Dominating Set (Upper DS), où l'objectif est de trouver la plus grande couverture minimale de sommets ou l'ensemble dominant le plus grand respectivement. À première vue, on pourrait s'attendre à ce que le problème Max Min VC soit le plus facile des deux : les deux problèmes peuvent être vus comme une tentative de trouver le plus grand ensemble minimal de sommets d'un hypergraphe, mais dans le cas de Max Min VC l'hypergraphe a une structure très restreinte, alors que dans le cas de Upper DS l'hypergraphe est essentiellement arbitraire. Cette intuition s'avère

correcte : alors que Upper DS n'admet aucune approximation de rapport $n^{1-\epsilon}$ [22], Max Min VC admet une approximation de rapport $\sqrt{n}$ (mais aucune approximation de rapport $n^{1/2-\epsilon}$) [40].

Ce contexte nous amène à la question naturelle de l'approximabilité de Max Min FVS. À un niveau intuitif, on peut être tenté de penser que ce problème devrait être plus difficile que celui de Max Min VC, puisque couvrir les cycles est plus complexe que couvrir les arêtes, mais plus facile que celui de Upper DS, puisque couvrir les cycles nous offre toujours plus de structure qu'un hypergraphe arbitraire. Cependant, à notre connaissance, aucun algorithme d'approximation de rapport $n^{1-\epsilon}$ n'est actuellement connu pour Max Min FVS (le problème pourrait donc être aussi difficile que Upper DS), et la meilleure borne de difficulté d'approximation connue est $n^{1/2-\epsilon}$ [154] (le problème pourrait donc être aussi facile que Max Min VC).

Notre principale contribution dans ce chapitre est de répondre pleinement à la question posée précédemment, en confirmant et en quantifiant précisément l'intuition selon laquelle Max Min FVS est un problème qui se situe "entre" Max Min VC et Upper DS : Nous donnons un algorithme d'approximation en temps polynomial avec un rapport $O(n^{2/3})$ et une réduction de la difficulté de l'approximation qui montre que (à moins que P = NP) aucun algorithme en temps polynomial ne peut obtenir un rapport de $n^{2/3-\epsilon}$, pour tout $\epsilon > 0$. Ceci règle complètement l'approximabilité du problème en temps polynomial. Au passage, nous prouvons également que Max Min FVS admet un noyau cubique lorsqu'il est paramétré par la taille de la solution, nous donnons un algorithme d'approximation avec un rapport $O(\Delta)$, nous montrons qu'aucun algorithme ne peut atteindre le rapport $\Delta^{1-\epsilon}$, pour tout $\epsilon > 0$, et enfin nous améliorons la meilleure preuve de NP-complétude connue pour Max Min FVS de $\Delta \geq 9$ [154] à $\Delta \geq 6$, où Δ est le degré maximum du graphe d'entrée.

Un aspect intéressant de nos résultats est qu'ils ont une interprétation issue de la combinatoire extrémale qui reflète bien la situation de Max Min VC. Rappelons qu'un corollaire de l'approximation de rapport $\sqrt{n}$ pour Max Min VC [40] est que tout graphe sans sommets isolés possède une couverture de sommets minimale de taille au moins égale à $\sqrt{n}$, et ceci est optimal (voir Remarque 5.3.2). Par conséquent, l'algorithme a seulement besoin de prétraiter trivialement le graphe (en supprimant les sommets isolés) et ensuite de trouver cet ensemble, dont l'existence est garantie. Nos algorithmes peuvent être vus sous un angle similaire: nous prouvons que si l'on applique deux règles de prétraitement presque triviales à un graphe (suppression des feuilles et contraction des arêtes entre sommets de degré deux), l'existence d'un fvs minimal de taille au moins $n^{1/3}$ (et $\Omega(n/\Delta)$) est toujours garantie, et ces quantités sont optimales (Corollaire 5.9 et Remarque 5.3.2). Ainsi, le rapport d'approximation de $n^{2/3}$ est automatiquement garanti pour tout graphe où nous appliquons exhaustivement ces règles très simples et où nos algorithmes n'ont à travailler que pour construire l'ensemble promis. Cela rend quelque peu remarquable le fait que le rapport de $n^{2/3}$ s'avère être le meilleur possible.

Ayant réglé l'approximabilité de Max Min FVS en temps polynomial, nous considérons la question du temps à investir si l'on souhaite garantir un rapport d'approximation de r (qui peut dépendre de n) où $r < n^{2/3}$. Ce type de compromis temps-approximation

a été largement étudié par Bonnet et al. [38], qui ont montré que Max Min VC admet une r-approximation en temps $2^{O(n/r^2)}$ et que celle-ci est optimale sous l'hypothèse ETH randomisée.

Pour Max Min FVS nous ne pouvons pas espérer obtenir un compromis avec des performances exponentielles en n/r^2, car cela implique une $\sqrt{n}$ approximation en temps polynomial. Il semble donc plus naturel de viser un temps d'exécution exponentiel en $n/r^{3/2}$. En effet, en généralisant notre algorithme d'approximation en temps polynomial, nous montrons que nous pouvons obtenir une r approximation en temps $n^{O(n/r^{3/2})}$. Bien que cet algorithme réutilise certains ingrédients de notre approximation en temps polynomial, il est significativement plus élaboré, car il n'est plus suffisant de comparer la taille de notre solution à n. Nous complétons notre résultat par une borne inférieure montrant que notre algorithme est essentiellement le meilleur possible sous l'hypothèse ETH randomisée pour tout r (pas seulement en temps polynomial), ou plus précisément que l'exposant du temps d'exécution de notre algorithme ne peut être amélioré que par des facteurs $n^{o(1)}$.

Digraph Coloring

Dans Digraph Coloring, on nous donne un graphe orienté (digraphe) D et on nous demande de calculer le plus petit k tel que les sommets de D peuvent être partitionnés en k ensembles *acycliques*. En d'autres termes, l'objectif de ce problème est de colorer les sommets avec le nombre minimal de couleurs de sorte qu'aucun cycle dirigé ne soit monochrome. Cette notion est appelée le *nombre dichromatique* et a été introduite par V. Neumann-Lara [164]. Plus récemment, la coloration des digraphes a fait l'objet d'une grande attention, notamment parce qu'il s'avère que de nombreux résultats concernant le nombre chromatique des graphes non orientés s'appliquent tout naturellement au nombre dichromatique des digraphes [4, 10, 26, 35, 54, 99, 102, 104, 105, 112, 141, 153, 157, 177]. Nous notons que Digraph Coloring généralise Coloring (si nous remplaçons simplement toutes les arêtes d'un graphe par des paires d'arcs anti-parallèles) et est donc NP-complet.

Dans le Chapitre 6, nous nous intéressons à la complexité de Digraph Coloring du point de vue de la complexité structurelle paramétrée. Notre principale motivation pour cette étude est que Coloring (non orienté) est un problème d'importance centrale dans ce domaine dont la complexité est bien comprise, et il est naturel d'espérer que certains des résultats de tractabilité connus puissent s'appliquer aux digraphes – en particulier parce que, comme nous l'avons mentionné, le Digraph Coloring semble se comporter comme une contrepartie très proche du Coloring à bien des égards. En particulier, pour les graphes non orientés, la complexité de Coloring pour les graphes "presque-acycliques" est très précisément connue : pour tout $k \geq 3$, il existe un algorithme de complexité $O^*(k^{tw})$, où tw est la largeur d'arbre du graphe d'entrée, et il est optimal (sous SETH) même si nous remplaçons la largeur d'arbre par des paramètres beaucoup plus restrictifs [123, 147]. Pouvons-nous obtenir le même degré de précision pour Digraph Coloring ?

Les paramétrages structurels de Digraph Coloring ont été étudiés dans [177], qui a montré que le problème est FPT par largeur modulaire en généralisant les algorithmes de [92, 136], et [99] qui a montré que le problème est dans XP par largeur de clique (notez que les résultats de difficulté pour Coloring excluent un algorithme fpt dans ce cas [86, 87, 137]). Nos résultats sur la difficulté du problème pour les DFVS et FAS bornés s'appuient sur les travaux de [153]. Le fait que le problème soit difficile pour le DFVS borné implique qu'il est également difficile pour la plupart des versions dirigées de la largeur d'arbre, y compris la largeur de DAG, la largeur de Kelly et la largeur de chemin dirigée [31, 93, 118, 125, 138]. En effet, la difficulté pour FAS implique également la difficulté pour la largeur d'élimination bornée, une restriction plus récemment introduite de la largeur d'arbre dirigée [84]. Pour la largeur d'arbre non dirigée, un problème avec un comportement similaire est DFVS : (undirected) FVS est soluble en $O^*(3^{tw})$ [68] mais DFVS ne peut pas être résolu en temps $tw^{o(tw)}n^{O(1)}$, et ceci est optimal sous l'hypothèse ETH [36]. Pour d'autres problèmes naturels dont la complexité par largeur d'arbre est $tw^{\Theta(tw)}$, voir [20, 28, 37].

En ce qui concerne le degré maximal, il n'est pas difficile de voir que k-Digraph Coloring est NP-difficile pour les graphes de degré maximal $2k+2$, car k-Coloring est NP-difficile pour les graphes de degré maximal $k+1$, pour tous les $k \geq 3$. A l'inverse, en utilisant une généralisation du théorème de Brooks dû à Mohar [158] on peut voir que k-Coloring dans les digraphes de degré maximum $2k$ est dans P. Cela laisse comme seul cas ouvert les digraphes de degré $2k+1$, ce qui dans un sens reflète nos résultats pour les digraphes de DFVS k et de degré $4k-2$. Nous notons que la NP-difficulté de 2-Digraph Coloring pour les graphes de degré borné est connue même pour les graphes de grande circonférence (girth), mais la borne sur le degré suit la limite imposée sur la circonférence [82].

La question principale qui nous motive est la suivante : Est-ce que Digraph Coloring devient aussi soluble pour les instances "presque-acycliques" ? Nous abordons cette question sous deux angles.

Tout d'abord, dans la section 6.3, nous considérons la notion d'acyclicité au sens du digraphe et étudions les cas où le digraphe d'entrée est proche d'être un DAG. La mesure la plus naturelle de ce type est probablement le *directed feedback vertex set* (DFVS).

Formellement, cet ensemble de sommets est défini comme suit :

Définition. *Étant donné un graphe dirigé $D = (V, E)$, un ensemble $S \subseteq V$ est appelé* directed feedback vertex set *de D si le graphe $D[V \setminus S]$ est un graphe acyclique dirigé.*

Le problème est paraNP-difficile pour ce paramètre, car pour tous les $k \geq 2$ fixés, on sait déjà que le problème de coloration des graphes est NP-difficile, pour des instances de DFVS de taille au plus $k+4$ [153]. Notre première contribution est de renforcer ce résultat en montrant ce qui suit :

Théorème. *Pour tout $k \geq 2$, il est NP-difficile de décider si un digraphe $D = (V, E)$ est k-colorable même lorsque la taille de son directed feedback vertex set est k. De plus, ce problème ne peut pas être résolu en temps $2^{o(n)}$ à moins que l'ETH soit fausse.*

Pour ce faire, nous présentons une réduction à partir d'une version restreinte du problème 3-SAT. Ce résultat comble le vide laissé par la réduction de [153] et fournit une dichotomie complète, puisque le problème est trivialement FPT par k lorsque le DFVS a une taille strictement inférieure à k (la seule partie non triviale du problème dans ce cas est de trouver le DFVS [52]). À la fin de cette section, nous considérons le problème 2-Digraph Coloring sur des graphes orientés. Nous prouvons qu'il est NP-difficile de décider si un graphe orienté est 2-colorable même dans les cas où la taille du DFVS est 3. Ceci est optimal car il existe un argument facile montrant que tous les graphes orientés avec DFVS k sont k-colorables.

Dans la section 6.4, nous cherchons à savoir si en considérant une notion plus restreinte de la quasi-acyclicité, ou en imposant d'autres restrictions, telles que la limitation du degré maximum du graphe, pourrait conduire à un algorithme fpt. Malheureusement, nous montrons qu'aucune de ces solutions ne suffit à rendre le problème soluble. En particulier, nous considérons d'abord comme paramètre la taille du *feedback arc set* (FAS) d'un digraphe.

Définition. *Étant donné un graphe dirigé $D = (V, E)$, un ensemble $S \subseteq E$ est appelé* feedback arc set *de D si le graphe $D' = (V, E \setminus S]$ est un graphe acyclique dirigé.*

Nous montrons que pour tout $k \geq 2$, k-Digraph Coloring est NP-difficile pour les digraphes de FAS de taille k^2 (le feedback arc set est bien sûr un paramètre plus restrictif que le feedback vertex set). De manière intéressante, cela nous conduit également à une dichotomie complète, cette fois pour le paramètre FAS : nous montrons que k-coloring devient FPT (par k) sur les graphes de FAS de taille au plus $k^2 - 1$, par un argument qui réduit ce problème à la coloration d'un sous-digraphe avec au plus $O(k^2)$ sommets, et donc le seuil de complexité correct pour ce paramètre est k^2. Deuxièmement, nous montrons que la coloration d'un digraphe avec DFVS de taille k reste NP-difficile même si le degré maximum est au plus $4k-1$. Ceci renforce la réduction de [153], qui a montré que le problème est NP-difficile pour une *dégénérescence* bornée (plutôt que pour un degré). En presque complétant le tableau, nous montrons que k-colorer un digraphe avec DFVS k et un degré maximum d'au plus $4k - 3$ est FPT par k, laissant ouvert seulement le cas où le DFVS est exactement k et le degré maximum exactement $4k - 2$.

Dans la section 6.5, en raison des résultats négatifs pour DFVS et FAS, nous avons décidé de considérer comme paramètre la treewidth du graphe sous-jacent.

Il s'avère que, finalement, cela suffit à conduire à un algorithme fpt, obtenu avec des techniques de programmation dynamique standard. En particulier, nous montrons ce qui suit.

Théorème. *Il existe un algorithme qui, étant donné un digraphe D sur n sommets et une décomposition en arbre de son graphe sous-jacent de largeur tw, décide si D est k-colorable en temps $k^{tw}(tw!)n^{O(1)}$.*

Cependant, notre algorithme a un temps d'exécution quelque peu décevant, car il est nettement inférieur à la complexité de $k^{tw}n^{O(1)}$ qui est connue pour être optimale pour les Coloring non orientés, en particulier pour les petites valeurs de k. Cela soulève la

question de savoir si le facteur supplémentaire ($tw!$) peut être supprimé. Notre principale contribution dans cette partie est de montrer que cela est probablement impossible, même pour un cas plus restreint. Plus précisément, nous montrons que:

Théorème. *S'il existe un algorithme qui décide si un digraphe donné sur n sommets et treedepth (non orientée)* td *est 2-colorable en temps* $\mathsf{td}^{o(\mathsf{td})}\mathsf{n}^{O(1)}$*, alors l'ETH est fausse.*

Enfin, dans la section 6.6, nous considérons les tournois. Il est déjà connu que 2-Digraph Coloring est NP-difficile pour les tournois [54]. L'algorithme exhaustif pour vérifier si un tournoi est 2-colorable nécessite une durée $O^*(2^n)$ car il existe 2^n 2-coloriages possibles pour un graphe. Nous améliorons ce temps d'exécution en proposant un algorithme qui répond à la même question en $O^*(\sqrt[3]{6}^n)$. Cette amélioration vient du fait qu'étant donné un tournoi $T = (V, E)$ tout cycle contient un triangle. Enfin, nous expliquons comment l'algorithme précédent peut être étendu pour k couleurs. Cela nous donne le corollaire suivant :

Corollary. *Soit $T = (V, E)$ un tournoi. Nous pouvons décider si le nombre bichromatique de T est k en temps $O^*(\sqrt[3]{k^3 - k}^n)$.*

Locally Irregular Induced Subgraphs

Un graphe G est dit *localement irrégulier*, si toutes les paires de sommets adjacents de G ont des degrés différents. Dans ce chapitre, nous introduisons et étudions le problème de trouver le plus grand sous-graphe induit localement irrégulier d'un graphe donné. Ce problème est équivalent à l'identification du nombre minimum de sommets qui doivent être supprimés de G, pour que ce qui reste soit un graphe localement irrégulier.

La notion de graphe localement irrégulier a été introduite pour la première fois dans [21]. L'aspect le plus intéressant des graphes localement irréguliers vient de leur connexion avec la conjecture dite 1-2-3, proposée dans [126]. Formellement, la conjecture 1-2-3 stipule que pour presque tous les graphes, nous devrions être capables de placer des poids de $\{1, 2, 3\}$ sur les arêtes de ce graphe, de sorte que la coloration, qui attribue à chaque sommet une couleur égale à la somme des poids sur ses arêtes adjacentes, est une coloration de sommet correcte du graphe. Un lien évident est que cette conjecture est valable pour les graphes localement irréguliers. En effet, placer un poids égal à 1 sur toutes les arêtes d'un graphe localement irrégulier, suffit à produire un vertex-coloring correct, puisque chaque sommet reçoit une couleur égale à son degré.

Le problème que nous présentons appartient à une famille de problèmes plus générale et bien étudiée, qui consiste à identifier le plus grand sous-graphe induit d'un graphe donné qui vérifie une propriété spécifique Π. C'est-à-dire, étant donné un graphe $G = (V, E)$ et un entier k, existe-t-il un ensemble $V' \subseteq V$ tel que $|V'| \leq k$ et $G[V \setminus V']$ a la propriété spécifiée Π ? Ce problème généralisé est en effet classique en théorie des graphes, et il est connu sous le nom Induced Subgraph with Property Π (ISPΠ en abrégé) dans [94]. Dans notre cas, la propriété Π est "le sous-graphe induit est localement irrégulier".

Dans [140], les auteurs ont montré que ISPΠ est un problème difficile pour toute propriété Π qui est *héréditaire*, i.e., tous les sous-graphes induits de G vérifient Π si G lui-même vérifie cette propriété. Cependant, la question reste intéressante (on pourrait dire qu'elle devient en fait plus intéressante) même si la propriété Π n'est pas héréditaire. Récemment, les auteurs de [25] ont étudié le problème pour Π étant "tous les sommets du sous-graphe induit ont un degré impair", qui n'est clairement pas une propriété héréditaire. Néanmoins, ils ont montré que c'est un problème NP-difficile, et ils ont donné un algorithme fpt qui résout le problème quand il est paramétré par la largeur de rang. De même, les auteurs de [5, 14, 159] ont étudié le problème ISPΠ, où Π est la propriété assez naturelle "le sous-graphe induit est d-régulier", où d est un entier donné en entrée (rappelons qu'un graphe est dit *d-régulier* si tous ses sommets ont le même degré d). En particulier, il est montré dans [14] que trouver un plus grand sous-graphe induit (connecté) qui est d-régulier, est NP-difficile à approximer, même en se limitant aux graphes bipartis ou planaires. Les auteurs de [14] fournissent également un algorithme en temps linéaire pour résoudre ce problème pour les graphes avec une largeur d'arbre bornée. En revanche, les auteurs de [5] adoptent une approche plus pratique, puisqu'ils se concentrent sur la résolution du problème pour les valeurs particulières de $d = 1$ et $d = 2$, en utilisant des limites issues de la programmation quadratique, de la relaxation Lagrangienne et de la programmation en nombres entiers.

Il est tout à fait clair que, dans un certain sens, la propriété qui nous intéresse se situe à l'opposé de celle étudiée dans [5, 14, 159]. Cependant, les deux propriétés, "le sous-graphe induit est régulier" et "le sous-graphe induit est localement irrégulier" ne sont pas héréditaires. Cela signifie que nous n'obtenons pas un résultat de NP-difficulté directement de [140]. De plus, le problème ISPΠ admet toujours un algorithme fpt, lorsqu'il est paramétré par la taille de la solution, si Π est une propriété héréditaire (prouvé dans [46, 131]), mais pour une propriété non héréditaire, ce n'est pas toujours vrai. En effet, dans [159], les auteurs ont prouvé que lorsque l'on considère Π comme "le sous-graphe induit est régulier", le problème ISPΠ est W[1]-difficile lorsqu'il est paramétré par la taille de la solution.

Comme nous l'avons mentionné précédemment, la conjecture 1-2-3 semble avoir des liens très intéressants avec les graphes localement irréguliers. Il y a eu quelques étapes vers la preuve de cette conjecture, qui impliquent la décomposition des arêtes d'un graphe en un nombre constant de sous-graphes localement irréguliers, i.e., étant donné G, trouver une coloration des arêtes de G en utilisant un nombre constant de couleurs, de sorte que chaque couleur induise un sous-graphe localement irrégulier de G. C'est la principale motivation derrière [21], et elle semble rester suffisamment intéressante pour attirer plus d'attention [27, 143, 171].

Notons que la classe des graphes localement irréguliers peut être vue comme un antonyme de celle des *réguliers*, i.e., graphes tels que tous leurs sommets ont le même degré. Il est important de préciser ici qu'il existe plusieurs notions alternatives de ce type. Ceci est principalement dû au fait très connu qu'il n'existe pas de graphes *irréguliers* non triviaux, i.e., graphes qui ne contiennent pas deux sommets (pas nécessairement adjacents) avec le même degré (voir [49]). Ainsi, la littérature regorge de définitions

légèrement différentes de l'irrégularité (voir par exemple [6, 49, 50, 89, 170]). Une façon de traiter la non-existence de graphes irréguliers, est de définir une notion d'irrégularité *locale*. Intuitivement, au lieu d'exiger que tous les sommets d'un graphe aient des degrés différents, nous considérons maintenant chaque sommet v séparément, et demandons que les sommets "autour" de v vérifient certaines propriétés d'irrégularité. Par exemple, les auteurs de [7] étudient des graphes G tels que pour chaque sommet v de G, deux voisins de v n'ont pas le même degré. Pour un aperçu d'autres notions intéressantes d'irrégularité (locale ou autre), nous renvoyons le lecteur à [8].

Nous commençons dans la Section 7.2 par fournir les notations, définitions et lemmes de base qui seront utilisés tout au long de ce chapitre. Plus important encore, nous définissons l'*irregulateur* minimal de G comme suit :

Définition. *Soit $G = (V, E)$ un graphe. Un ensemble $S \subseteq V$, de taille minimale, tel que $G[V \setminus S]$ est appelé* irregulateur *minimal de G et noté $ir^*(G)$. De plus, par $\mathrm{I}(G)$ nous désignons la taille $|S|$ d'un irregulateur minimum S de G.*

Dans la Section 7.3, nous traitons de la complexité du problème introduit. En particulier, nous montrons que le problème appartient à P si le graphe d'entrée est un chemin, un cycle, un arbre, un graphe biparti complet ou complet. Nous prouvons ensuite que trouver le plus grand sous-graphe localement irrégulier induit d'un graphe G donné est NP-difficile, même dans certains cas très restreints.

Théorème. *Soit G un graphe et $k \in \mathbb{N}$. Décider si $\mathrm{I}(G) \leq k$ est NP-complet, même lorsque G est un graphe biparti planaire avec un degré maximum $\Delta \leq 3$, ou un graphe biparti cubique.*

Comme le problème que nous introduisons semble être difficile à calculer même pour des familles de graphes assez restreintes, nous procédons à l'étude de son approximabilité. Malheureusement, nous prouvons dans la Section 7.4 que pour tout graphe biparti G d'ordre n et $k \geq 1$, il ne peut exister d'algorithme en temps polynomial qui trouve une approximation de $\mathrm{I}(G)$ dans le rapport $O(n^{1-\frac{1}{k}})$, à moins que P=NP. Néanmoins, nous parvenons à donner un algorithme d'approximation (trivial) de rapport d pour les graphes bipartis d-réguliers en retournant (toujours) une des bipartitions du graphe. En particulier, nous montrons ce qui suit.

Théorème. *Pour tout graphe biparti d-régulier $G = (L, R, E)$ d'ordre n, on a que $\mathrm{I}(G) \geq n/2d$.*

Nous décidons alors de nous intéresser à sa complexité paramétrée. Dans la section 7.5, nous présentons deux algorithmes qui calculent $\mathrm{I}(G)$, chacun considérant des paramètres différents. Le premier considère la taille de la solution k et le degré maximal Δ de G :

Théorème. *Pour un graphe donné $G = (V, E)$ avec $|V| = n$ et un degré maximal Δ, et pour $k \in \mathbb{N}$, il existe un algorithme qui décide si $\mathrm{I}(G) \leq k$ en temps $(2\Delta)^k n^{O(1)}$.*

L'idée de cet algorithme est basée sur l'observation suivante :

Lemma. *Soit $G = (V, E)$ un graphe tel que G n'est pas localement irrégulier, et S est un $ir^*(G)$. De plus, soit $G_v = (V', E')$ le graphe $G[V \setminus \{v\}]$ pour un sommet $v \in S$. Alors $\mathrm{I}(G_v) = \mathrm{I}(G) - 1$.*

La seconde considère la largeur d'arbre tw et Δ de G :

Théorème. *Pour un graphe donné $G = (V, E)$ et une belle décomposition arborescente de G, il existe un algorithme qui retourne $\mathrm{I}(G)$ en temps $\Delta^{3tw} n^{O(1)}$, où tw est la treewidth de la décomposition donnée et Δ est le degré maximum de G.*

Malheureusement, ces algorithmes ne peuvent être considérés comme étant FPT que si Δ fait partie du paramètre. Dans la section 7.5.3, nous présentons deux réductions qui prouvent que le problème est W[2]-difficile quand il est paramétré uniquement par la taille de la solution et W[1]-difficile quand il est paramétré uniquement par la largeur de l'arbre. La première est une réduction à partir de l'ensemble dominant qui est W[2]-complet lorsqu'il est paramétré par la taille de sa solution, tandis que la seconde est une réduction à partir du problème List Coloring qui est W[1]-difficile lorsqu'il est paramétré par la treewidth du graphe donné. Ces réductions montrent également que nous ne pouvons même pas avoir un algorithme qui calcule $\mathrm{I}(G)$ en temps $f(k)n^{o(k)}$ ou $O^*(f(tw)n^{o(tw)})$, à moins que l'ETH n'échoue.

Par conséquent, nos algorithmes sont essentiellement optimaux.

Crossword Puzzle

Les mots croisés sont des jeux à un joueur dont le but est de remplir une grille (traditionnellement bidimensionnelle) avec des mots. Depuis leur première apparition il y a plus de 100 ans, les mots croisés sont rapidement devenus populaires. Aujourd'hui, on peut les trouver dans de nombreux journaux et magazines du monde entier, comme le *New York Times* aux États-Unis ou le *Le Figaro* en France. En plus de leur intérêt récréatif évident, les mots croisés sont des outils appréciés en éducation [64] et en médecine. En particulier, la participation à des jeux de mots croisés semble retarder l'apparition d'un déclin accéléré de la mémoire [168]. Ils sont également utiles pour développer et tester des techniques de calcul; voir par exemple [173]. En fait, la conception et la réalisation d'un puzzle constituent toutes deux un défi. Dans ce chapitre, nous nous intéressons à la tâche de résolution d'un type spécifique de mots croisés.

Il existe différents types de mots croisés. Dans les plus célèbres, certains indices sont donnés en même temps que l'endroit où les réponses doivent se trouver. Une solution contient des mots qui doivent être cohérents avec les indices donnés, et les paires de mots qui se croisent sont contraintes de s'accorder sur la lettre qu'elles partagent. Les mots croisés de type "Fill-in" ne sont pas accompagnés d'indices. Étant donné une liste de mots et une grille dans laquelle certains emplacements sont identifiés, l'objectif est de remplir tous les emplacements avec les mots donnés. La liste de mots est généralement succincte et fournie de manière explicite.

Dans une variante de mots croisés "Fill-in" actuellement proposée dans un magazine télévisé français [148], il faut trouver jusqu'à 14 mots et les placer dans une grille (la

grille est la même pour chaque instance, voir la Figure 1 pour une illustration). Les mots ne sont pas explicitement listés mais ils doivent être *valide* (par exemple, appartenir à la langue française). Dans une instance du jeu, certaines lettres spécifiées ont un poids positif; les autres lettres ont un poids nul. L'objectif est de trouver une solution dont le poids - défini comme la somme totale des lettres écrites dans la grille - est au moins égal à un seuil donné.

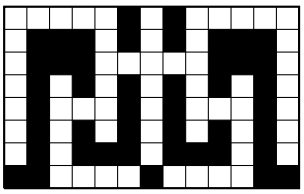


Figure 1: Placez les mots valides dans cette grille. Dans une instance possible, les lettres S, U, I, V, R, E et T ont respectivement un poids de 7, 5, 4, 2, 6, 1 et 3. Toute autre lettre a un poids nul. Essayez d'obtenir au moins 330 points.

Dans ce chapitre, nous nous proposons de faire une étude théorique de cette grille de mots croisés "Fill-in" (la grille n'est pas limitée à celle de la Figure 1). Nous nous intéressons principalement à deux problèmes : La grille peut-elle être entièrement complétée ? Comment le poids d'une solution peut-il être maximisé ? Dans la suite, ces problèmes sont appelés respectivement Crossword Puzzle Decision et Crossword Puzzle Optimization (en abrégé : CP-Dec et CP-Opt).

CP-Dec n'est pas nouveau; voir GP14 dans [94]. La preuve de NP-complétude est attribuée à une communication personnelle avec Lewis et Papadimitriou. Par la suite, une preuve alternative de NP-complétude est apparue dans [77] (voir également [139]). D'autres articles sur les mots croisés existent et il s'agit pour la plupart de techniques validées empiriquement provenant de l'intelligence artificielle et de l'apprentissage automatique; voir par exemple [96, 150, 145, 9, 173, 172] et les références qui s'y trouvent.

Tout d'abord, nous devons donner quelques détails sur l'entrée du problème. On nous donne un dictionnaire $\mathcal{D} = \{d_1, \dots d_m\}$ dont les mots sont construits sur un alphabet $\mathcal{L} = \{l_1, \dots l_\ell\}$, et une grille bidimensionnelle composée d'emplacements horizontaux et verticaux. Un emplacement est composé de cellules consécutives. Les emplacements horizontaux ne se croisent pas; il en va de même pour les emplacements verticaux. Cependant, les emplacements horizontaux peuvent croiser les emplacements verticaux.

Dans une solution réalisable, chaque emplacement S reçoit soit un mot de $\mathcal{D}$ de longueur $|S|$, soit rien (on dit parfois qu'une case ne recevant rien reçoit un *mot vide*). Chaque case reçoit au maximum une lettre, et les mots attribués à deux emplacements qui se croisent doivent s'accorder sur la lettre placée dans la case partagée. Tous les emplacements horizontaux remplis reçoivent des mots écrits de gauche à droite (en travers), tandis que tous les emplacements verticaux reçoivent des mots écrits de haut en bas.

Il existe une fonction de poids $w : \mathcal{L} \to \mathbb{N}$. Le poids d'une solution est la somme totale des poids des lettres placées dans la grille. Observez que le poids d'une solution est plus petit que la somme totale des poids de ses mots car, dans le premier cas, les

lettres des cellules partagées ne sont comptées qu'une seule fois.

Les deux principaux problèmes étudiés dans cet article sont les suivants. Étant donné une grille, un dictionnaire $\mathcal{D}$ sur un alphabet $\mathcal{L}$, et une fonction de poids $w : \mathcal{L} \to \mathbb{N}$, l'objectif de Crossword Puzzle Optimization (en bref, CP-Opt) est de trouver une solution réalisable de poids maximal. Étant donné une grille et un dictionnaire $\mathcal{D}$ sur un alphabet $\mathcal{L}$, la question posée par Crossword Puzzle Decision (CP-Dec en bref) est de savoir si la grille peut être complètement remplie ou non ?

Deux cas seront considérés : si chaque mot est utilisé au plus une fois, ou si les mots peuvent être utilisés plusieurs fois. Nous supposerons parfois que certaines cellules sont pré-remplies avec certains éléments de $\mathcal{L}$. Dans ce cas, une solution est réalisable si elle est cohérente avec les cellules pré-remplies. Nous proposons ci-dessous un premier résultat lorsque toutes les cellules partagées sont pré-remplies.

Proposition. *CP-Dec et CP-Opt peuvent être résolus en temps polynomial si toutes les cellules partagées de la grille sont pré-remplies, que la réutilisation des mots soit autorisée ou non.*

Ce résultat est utilisé à plusieurs reprises dans le reste du chapitre.

On peut associer à chaque grille un graphe biparti, appelé ci-après le *graphe de la grille*: chaque emplacement est un sommet et deux sommets partagent une arête si les emplacements correspondants se chevauchent. La grille (et donc, le graphe de grille) n'est pas nécessairement connectée.

Remarquons enfin que, en utilisant une représentation appropriée, on peut supposer que la taille de l'entrée est de $n + m$, où n est le nombre d'emplacements dans la grille et m est le nombre de mots dans le dictionnaire.

Notre objectif dans ce chapitre est d'identifier les paramètres structurels pertinents qui rendent le remplissage des mots croisés difficiles d'un point de vue computationnel. Nous commençons par examiner la structure de la grille donnée. Il est naturel de penser que, si la structure de la grille est arborescente, alors le problème devrait devenir plus facile, puisque la grande majorité des problèmes sont solubles sur des graphes de petite largeur d'arbre. Nous ne confirmons que partiellement cette intuition : en prenant en compte la structure du graphe de la grille, nous montrons dans la Section 8.3 que CP-Opt, lorsque nous autorisons la réutilisation des mots, peut être résolu en temps polynomial sur des instances de largeur d'arbre constante. En particulier :

Théorème. *Si nous autorisons la réutilisation des mots, alors CP-Opt peut être résolu en temps* $(m+1)^{tw}(n+m)^{O(1)}$ *sur des instances où tw est la treewidth du graphe de la grille.*

Cependant, notre algorithme n'est pas soluble à paramètre fixe et, comme nous le montrons, cela ne peut être évité, même si l'on considère le cas beaucoup plus restreint où le problème est paramétré par le nombre d'emplacements horizontaux, qui limite trivialement la largeur d'arbre du graphe de grille. De manière plus dévastatrice, nous montrons que si nous imposons également la règle naturelle selon laquelle les mots ne peuvent pas être réutilisés, le problème devient déjà NP-difficile lorsque le graphe de

grille est une collection d'arêtes disjointes pour les alphabets de taille 3, ou une union d'étoiles pour un alphabet binaire. Par conséquent, une structure arborescente ne semble pas être d'une grande aide pour rendre les mots croisés solubles.

Nous considérons ensuite CP-Opt paramétré par le nombre total d'emplacements n. On peut dire qu'il s'agit d'une paramétrisation très naturelle du problème, car dans les mots croisés de la vie réelle, on peut s'attendre à ce que la taille de la grille soit significativement plus petite que la taille du dictionnaire. Nous montrons que dans ce cas, le problème devient soluble à paramètre fixe :

Théorème. *Il existe un algorithme qui résout CP-Dec et CP-Opt en temps* $O^*((\ell+1)^{n^2/4})$*, où* n *est le nombre total de slots et* ℓ *la taille de l'alphabet, que la réutilisation des mots soit autorisée ou non.*

Cependant, le temps d'exécution de notre algorithme est exponentiel en n^2. Notre résultat principal est de montrer que cette dépendance décevante est probablement la meilleure possible : même pour un alphabet binaire, un algorithme résolvant CP-Dec en temps $2^{o(n^2)}$ contredirait l'ETH randomisée. Notez que tous nos résultats positifs jusqu'à ce point fonctionnent pour le cas plus général CP-Opt, tandis que nos résultats de difficulté s'appliquent à CP-Dec.

Ensuite, dans la section 8.5 nous considérons l'approximabilité de CP-Opt. Ici, il est facile d'obtenir une approximation de rapport $\frac{1}{2}$ en ne considérant que les emplacements horizontaux ou verticaux. Nous sommes seulement capables d'améliorer légèrement cette approximation en montrant que :

Théorème. *CP-Opt est* $(\frac{1}{2}+\frac{1}{2(\varepsilon n+1)})$*-approximable en temps polynomial, pour tout* $\varepsilon \in (0,1]$.

Notre principal résultat dans cette direction est de montrer que c'est essentiellement le meilleur possible : obtenir un algorithme avec le rapport $\frac{1}{2}+\epsilon$ contredirait la conjecture des jeux uniques (UGC).

Avant de conclure, nous explorons dans la Section 8.6 les cas où CP-Dec peut être résolu en temps polynomial. Nous proposons des réductions de CP-Dec à certains problèmes bien connus qui appartiennent à P.

Contents

Chapter 1

Introduction

Combinatorial problems are discrete problems defined over a finite collection of objects and the goal is to return a set, or an ordering, or an assignment of these objects that satisfies certain conditions. They arise from every day problems, puzzles, domains like logic and graph theory, and even from industry. Furthermore, the study of these problems is an important task as they are abstraction of many problems that we encounter in the real world, often since a long time (routing, scheduling, monitoring a territory with a limited number of resources, etc).

Several combinatorial problems are defined in graphs. Graphs are mathematical structures that consist of two sets; one set contains objects called vertices and the second contains pairs of vertices, called edges. One of the most classic applications of graphs is the representation of maps; the vertices represent areas while an edge exists between two vertices if the corresponding areas are connected directly. Some of the most classic combinatorial problems on graphs are the Traveling Salesman Problem, where we want to decide whether for a given graph there is a cycle that contains each vertex exactly once, the k coloring problem, which asks us to decide whether we can color the vertices of a graph with k colors and each two vertices that share an edge have different colors, and the shortest path problem, where we have to find a path with the smallest length that connects two given vertices of a graph.

Note that the latter problem requests to find the shortest path, a question that cannot be answered with a simple yes or no like the first two problems. Based on this difference, combinatorial problems are divided into two categories: decision problems and optimization problems. In decision problems we ask if there is a solution (and sometimes return one), while in optimization problems the solutions have weights and we want to find a solution with the minimum or maximum weight among all feasible solutions.

When a computer scientist considers a combinatorial problem his goal is to find an efficient way (algorithm) to solve the given problem. However, this sentence is far from been precise as both the "find an efficient way" and "solve the given problem" may have many different interpretations. First, it is important to understand the meaning behind this sentence. We say that we have a way of solving a problem if we have a set of steps,

called an algorithm, such that, given any instance of the problem (the input), following these steps we can compute the desired output, called the solution. Efficient algorithms minimize the resources we use. These resources can be energy, memory, time, etc., and can be viewed as different dimensions of an algorithm. It is often possible to reduce all these dimensions to time. For example, it is not hard to imagine that an algorithm that runs for "a long time" also consumes "a lot of energy". In other words, time is one of the most important resources to minimize. Time efficient algorithms have been studied extensively through the years. When we study how fast we can solve a problem, from a theoretical point of view, we say that we are studying its time complexity. Since the size of the instances of a problem can vary greatly, it is inevitable that the time complexity of a problem will be related to the size of its instance. Consequently, the theoretical running time of an algorithm (i.e., the number of required steps) that solves a problem is a function of the size of the problem instance. Additionally, we always need to specify if we give the worst, average or best case running time. In this thesis we are interested for the worst case analysis of an algorithm. When considering graph problems, we take the number of vertices plus the number of edges of the input graph as the input size. Moreover, in the case of simple graphs (i.e., graphs without loops or multiple edges) we can reduce ourselves to the number of vertices since the number of edges is at most quadratic to them. For example, imagine that we are given a graph and our goal is to check whether starting from a particular vertex we can reach all other vertices. This can be checked using a well-known algorithm called Breadth-first search (BFS). The running time of this algorithm in the worst case is linear in the number of vertices and edges (or quadratic in the number vertices for simplicity).

In the early developments of combinatorial optimization, researchers stated that polynomial time resolution is acceptable (i.e., efficient) in theory. This is however questionable for the practical resolution of a problem. Unfortunately, a lot of interesting combinatorial problems are conjectured to be unsolvable in polynomial time, but there is no evidence that no such algorithm exists. Many of these problems belong to the set of NP-complete problems. Moreover, there is a framework that helps us to show that a problem is at least as hard as these problems (NP-hard problems) or that it belongs to the set of NP-complete problems. This framework gives us the certainty that if we find a polynomial-time algorithm for an NP-complete problem, then all NP-complete problems admit polynomial-time algorithms. In other words, finding a polynomial-time algorithm for an NP-hard problem would be a major breakthrough since we would be able to solve (efficiently in theory) a bunch of interesting problems.

Since no one has managed to give polynomial-time algorithms for NP-complete problems, most computer scientists believe that such algorithms can not be constructed. Consequently, many different approaches have been developed to deal with these problems. To understand these approaches we need to return to our original goal and ask ourselves what kind of trade-offs we can make.

Our initial goal was to find an efficient algorithm that solves any instance of a given problem. The first compromise is related to the efficiency of the algorithm. Since we believe that there is no polynomial-time algorithm that can solve NP-complete prob-

lems, we can look for algorithms of worse time complexity. This direction includes the development of exponential and pseudo-polynomial time algorithms.

The second compromise relates to the instances of the problem. As we believe that we cannot solve all instances, we can try to deal with instances that have certain properties. As an example, imagine a graph problem. Instead of trying to solve the given problem on any graph, we can try to develop polynomial-time algorithms that solve the problem on specific classes of graphs such as trees or cycles. In other cases, we can consider parameters of the problem and cases where these parameters are bounded by some constant. This idea leads to parameterized complexity (and parameterized algorithms) where the main goal is to find algorithms where all the computational difficulty is encapsulated in the parameter and not in the size of the instance. In particular, such algorithms are called fixed-parameter tractable (fpt-)algorithms and their running time is in the form of $f(k)poly(n)$, where $f(k)$ is a function over the size k of the parameter and $poly(n)$ is a polynomial of the size n of the instance.

Finally, we can allow the return of solutions that are not optimal. Note that this approach can only be considered when studying optimization problems. As we have already said, the goal in optimization problems is to find a feasible solution with minimum or maximum weight, which we call the optimal solution. However, in some cases we may be satisfied with solutions that are "close enough" to optimal. Algorithms that are proven to return such solutions are called approximation algorithms. There are classes of problems that are defined based on the approximation algorithms they admit. Two of the most common approximation classes are the APX and the FPTAS. When, in polynomial time over n, we can guarantee for a constant c a solution that is at most c times bigger than an optimal solution then we have a constant-factor approximation algorithm. If a problem admits a constant-factor approximation algorithm then it belongs to APX. On the other hand, when the error margin ε is given in the input and we can return, in polynomial time over n and $\frac{1}{\varepsilon}$, a solution $1+\varepsilon$ close to the optimal then we have a Fully Polynomial Approximation Scheme (FPTAS). Problems that admit FPTASs belong to the class FPTAS.

As we mentioned before, there is a huge number of different combinatorial problems, and these problems often differ in nature. This variety deepens even further when we consider all the aforementioned approaches that can be used to deal with these problems. That is why the scientific community has to offer a constantly renewed/enriched toolbox for tackling combinatorial problems and this reason has provided enough motivation to study as many combinatorial problems and different approaches as possible. This thesis contributes to this agenda.

Our contribution: This manuscript consists of 8 additional chapters. In Chapter 2 we provide all the necessary terminology and definitions we use. In chapters 3 to 7 we study variants of classical combinatorial problems and each chapter is devoted to one problem. For these problems we first consider their time complexity in general and restricted cases. Then, for the hard cases of the problems, we try to develop parameterized and approximate algorithms or prove that no such algorithms exist. Finally, there is a conclusion chapter where we suggest other directions, techniques and problems that can

be studied in future work.

More precisely, in Chapter 3 we investigate problems that are similar to the well known Subset-Sum problem. In particular, we study the Subset-Sums Ratio problem (SSR) and some variants of it. In SSR we have as input a set A of positive integers and we want to find two disjoint subsets that have ratio as close to one as possible. We present a new FPTAS for SSR that identifies cases that can be easily approached, while using Subset Sum computations to deal with the hard cases. The second half of this chapter is focused on variations of SSR. In particular, we define two families of SSR problems that are able to capture additional restrictions and we present conditions which, if they are met by a problem in these families, guarantee the existence of an FPTAS for the problem. Then we are considering two specific variants of SSR, the 2-*Set SSR* problem and the *Factor-r SSR* that belong to these families. For 2-*Set SSR* we use our framework to present an FPTAS based on this algorithm and we show that this FPTAS can be also used to approximate *Factor-r Subset-Sums Ratio*.

In the rest of this manuscript we mostly study combinatorial problems on graphs. Particularly, we consider five problems; the Upper r-Tolerant Edge Cover, the MAX-MIN Feedback Vertex Set, the k-Digraph Coloring, the Maximum Locally Irregular Induced Subgraph and the problem of filling Crossword Puzzles.

In Chapter 4 we study the Upper r-Tolerant Edge Cover. This problem asks for a subset of edges of the graph, of maximum size, that covers all the vertices at least r times and it is minimal (i.e., no proper subset of it has the same property). We first give some properties of the feasible solutions of the problem and their values. Most interestingly, we show that any feasible solution is a $\frac{1}{2}$-approximation. Then we present several NP-hardness results for restricted graph classes. In particular, the problem is intractable for $r+1$-regular bipartite graphs while, for $r=2$, it is intractable even for split graphs and cubic planar graphs. Furthermore, we show that the Upper $(r+1)$-Tolerant Edge Cover is NP-hard in graphs of maximum degree $\Delta+1$ if the Upper r-Tolerant Edge Cover is NP-hard in graphs of maximum degree Δ. Then we consider the approximability of the problem. As we showed that we can easily have a $\frac{1}{2}$-approximation then we consecrate to find inapproximability ratio. We present two hardness of approximation reductions that give us several inapproximability ratios for Upper Edge Cover and Upper 2-Tolerant Edge Cover in graphs with bounded maximum degree.

Then, in Chapter 5, we study the approximability of the MAX-MIN Feedback Vertex Set problem. A subset of the vertices of a graph is a feedback vertex set if the graph we get after deleting this set is acyclic. In this problem we are searching for the largest feedback vertex set that is minimal. We present a polynomial time $n^{2/3}$-approximation algorithm and a hardness of approximation reduction which shows that this ratio is best possible we can achieve in polynomial time under standard assumptions. Then we present a time-approximation trade off algorithm that returns a r-approximation solution in time $n^{O(n/r^{3/2})}$ and we complement this with a matching lower bound (under standard assumptions). Along the way we show that the problem admits a cubic kernel when parameterized by the solution size, we prove that the

problem remains NP-hard for graphs of maximum degree $\Delta = 6$ and we give a $O(\Delta)$-approximation.

In Chapter 6 we consider a well known variant of the chromatic number in directed graphs, called dichromatic number. The dichromatic number is strongly connected with acyclic directed graphs as it is the minimum number of colors needed in order to color all the vertices of the graph and each color induces a directed acyclic graph. This leads to the natural question of calculating the dichromatic number in graphs that are almost acyclic. Therefore we decide to study the parameterized complexity of the problem for parameters like directed feedback vertex set and the feedback arc set. Interestingly enough, we show that for directed feedback vertex set we have a dichotomy as any directed graph with feedback vertex set k can be colored with $k + 1$ colors but it is NP-hard to decide if it can be colored with k colors. Then, we show that for digraphs with feedback arc set of size k^2 it is NP-hard to decide if they can be colored with k colors. This leads to another dichotomy since the problem belongs to FPT for graphs with feedback arc set of size $k^2 - 1$.

The problem we consider in Chapter 7 belongs to a family of problems where the goal is to identify the largest induced subgraph of a given graph that verifies a certain property Π. In particular, we study the case when the property Π is "the induced subgraph is locally irregular". A graph is called locally irregular when any two adjacent vertices have different degrees. We show that this problems is NP-hard even in very restricted cases like subcubic planar bipartite graphs, or cubic bipartite graphs. Furthermore, we show that the problem is NP-hard to approximate within a ratio $O(n^{1-1/k})$, for any constant $k \geq 1$, even for bipartite graphs. Then, we consider the parameterized complexity of the problem. We present two parameterized algorithms with running time $(2\Delta)^k n^{O(1)}$, where k is the number of deleted vertices, and $(\Delta)^{3tw} n^{O(1)}$, where tw is the treewidth of the given graph. Finally we provide evidence that our algorithms are essentially optimal under standard assumptions.

In Chapter 8 we study the problem of filling crossword puzzles. An instance of the problem consists of a grid, an alphabet and a dictionary over this alphabet. The goal of the problem is to fill the grid with words from the dictionary while the reuse of a word may or may not allowed. In particular, we create a graph that represents the grid of the crossword puzzle and we consider the complexity of filling the grid based on the properties of the graph and the size of the given alphabet. If we do not allow word reuse, the problem is NP-hard even in very restricted cases such as, the grid graph is a union of stars and the alphabet has only two letters, and, the grid graph is a matching and the alphabet has only three letters. If we allow word reuse, the problem becomes slightly more tractable as we obtain an $(m+1)^{tw}$ algorithm in this case, where tw is the treewidth of the grid graph. Then, we take as parameter the number of slots of the grid n and we present an $(k + 1)^{n^2}$ algorithm where k is the size of the alphabet. We finally consider the optimization version of the problem. Here, we provide a simple $\frac{1}{2}$-approximation algorithm and evidence that it is unlikely to obtain a better approximation ratio in polynomial time under standard assumptions.

Chapter 2

Basic Notation and Terminology

2.1 Computational Complexity

One of the most important topics in Computer Science is the design of computer algorithms. Algorithms are finite procedures that are well-defined and aim to return the desired solution to a particular problem. One of the things we are interested in this manuscript is the study of the "computational complexity" of a problem and in particular its "time complexity". The time complexity (or simply complexity) of an algorithm is defined as its running time, while the time complexity of a problem is defined as the running time of the fastest algorithm for that problem.

To give a proper definition for the running time of an algorithm, we must first explain what we call the instance of a problem and the size of the instance.

An instance of a problem consists of all the data required to solve the problem, while the size of the instance (or the size of the input) refers to the size of that data. Since the actual execution time of an algorithm is related to several parameters, such as processing power, memory size, and input size, it does not make sense, from a theoretical point of view, to compare algorithms based on this criteria. Therefore, the most logical way is to relate the running time to the number of steps using a function of the input size. Saying that an algorithm has complexity $f(n)$ means that for an input of size n the algorithm terminates after $f(n)$ steps. Note that, because these functions are related to running time, it makes sense to consider only non-negative and non-decreasing functions.

Even so, this notation does not specify everything. There are algorithms that do not always stop after exactly the same number of steps or we can not calculate the exact number of steps that are required. For that reason we are going to use the following asymptotic notations.

- $O(f(n))$: We say that a function $g(n) \in O(f(n))$ if there exist two positive constants c and n_0 such that,

$$g(n) \leq cf(n) \text{ for all } n \geq n_0$$

- $\Omega(f(n))$: We say that a function $g(n) \in \Omega(f(n))$ if there exist two positive constants c and n_0 such that,

$$g(n) \geq cf(n) \text{ for all } n \geq n_0$$

- $o(f(n))$: We say that a function $g(n) \in o(f(n))$ if

$$\lim_{n \to +\infty} \frac{g(n)}{f(n)} = 0$$

In this document, whenever we have a multi variable function inside an asymptotic notation all variables except one are considered as constants. As a particular example take the statement $g(n, m) \in O(f(n, m))$. This may mean that for any given $m \geq 0$, $g(n, m) \in O(f(n, m))$, or that for any given $n \geq 0$, $g(n, m) \in O(f(n, m))$. Which of n, m is considered fixed and which variable will be clear from the context.

Class NP and Polynomial Reductions

As mentioned, there are several problems for which we have not been able to find polynomial time algorithms or show that it is impossible to find such algorithms. In this thesis we consider problems belonging to the NP class. Informally, NP is the class containing all decision problems for which we can check, in polynomial time, whether a given answer is the correct answer. A subclass of NP is the class P which contains all decision problems that can be solved in polynomial time. It is easy to see that P $\subseteq$ NP, however the question whether P = NP or not is still open.

Many interesting problems belong to the category of NP-complete problems. To define NP-complete problems we first need to define "polynomial reductions".

Definition 2.1 (Polynomial-time reduction)**.** *A polynomial-time reduction from a problem A to a problem B is a polynomial-time algorithm that transforms any instance I_A of A to an instance I_B of B such that I_B is a positive instance of B if and only if I_A is a positive instance of A.*

We say that a problem A is NP-complete if it belongs to NP and any other problem B in NP is polynomial-time reducible to A. In case we can only verify the second condition then we say that the problem is NP-hard.

The first problem proved to be NP-complete is the well-known Satisfiability problem (known as SAT) [61].

Definition 2.2 (SAT)**.** *Let ϕ be a logical formula, in conjunctive normal form (CNF formula), over a set of variables $\{x_1, \ldots, x_n\}$. The goal is to decide if there exists a truth assignment over the variables that satisfies the formula.*

One of the most important variants of SAT, which is also NP-complete, is 3-SAT. The difference between SAT and 3-SAT is that in the latter the each clause contains three literals.

Note that if we have an NP-complete problem A, then we can show that a problem B is NP-hard by presenting a polynomial-time reduction from A to B. Informally, this means that problem B is at least as hard as A since any polynomial-time algorithm for B can be used to solve A. Moreover, the existence of any polynomial-time algorithm that solves an NP-complete problem can lead to solving all NP-complete problems in polynomial time, and to the conclusion that P=NP. Conversely, if we prove that a particular NP-complete problem does not admit a polynomial-time algorithm, then we are led to the conclusion that P≠NP and that no NP-complete (or NP-hard) problem admits a polynomial-time algorithm.

In this thesis we focus on combinatorial optimization problems. Compared to decision problems, the standard definition for optimization problems is a bit more complicated.

Definition 2.3 (Optimization problems)**.** *An optimization problem is a quadruple* (I, f, w, g) *such that:*

- I *is the set of the instances of the problem.*
- *For a given instance* $x \in I$, $f(x)$ *is the set of all feasible solutions of* x.
- *For a given* $x \in I$ *and* $y \in f(x)$, $w(x, y)$ *is the weight of the solution* y *for the instance* x.
- g *is either the min or the max function.*

The objective of an optimization problem is, for a given case $x \in I$, *to find a feasible solution* $y_{best} \in f(x)$ *such that* $w(x, y_{best}) = g\{w(x, y) \mid y \in f(x)\}$.

The class of optimization problems we are mainly interested in is NPO. This class is the analogue of NP for optimization problems and is defined as follows:

Definition 2.4. *An optimization problem* $A = (I, f, w, g)$ *belongs to NPO if:*

- *there exists a polynomial* q *such that for any* $x \in I$ *and* $y \in f(x)$ *we have* $|y| \leq q(|x|)$.
- *we can decide in polynomial time if* $x \in I$ *or* $y \in f(x)$ *for a given* x.
- $w(x, y)$ *is polynomial time computable.*

Starting from any optimization problem we can define a decision version of this problem based on the function g. If $g = min$ then the decision version asks if there is a feasible solution with weight less than or equal to a given value k, while if $g = max$ we want to check for a feasible solution with weight greater than or equal to a k. When the decision version of an optimization problem is NP-complete then the corresponding optimization problem is at least as hard as NP-complete problems. Therefore, we should not expect the optimization version of an NP-complete problem to be solvable in polynomial time (unless P=NP), as otherwise we can answer the decision problem by solving the optimization. We say that these optimization problems are NP-hard.

Since for NP-hard optimization problems we do not expect to find polynomial-time algorithms, we need to find different ways of dealing with them. There are three classical approaches:

1. we can develop algorithms that do not run in polynomial time,
2. we can try to find, in polynomial time, solutions that are "close enough" to the optimal one and
3. we can try to solve, in polynomial time, instances with additional restrictions.

In this manuscript we mainly try to find good approximation algorithms (i.e., algorithms that run in polynomial time and return a solution as close to optimal as possible) or try to develop algorithms that solve exactly instances with specific constraints. In the second case we also consider parameterized algorithms, which can be seen as a combination of the first and third approaches. More details about these approaches are presented in the sections 2.2, 2.3 and 2.4.

2.2 Approximation

In the previous section we explained that when dealing with an NP-hard optimization problem it is unlikely to find a polynomial-time algorithm that returns an optimal solution. However, in many cases we can be satisfied if we have a polynomial-time algorithm that returns feasible solutions that are close to optimal. When the weight function w gives all feasible solutions a positive weight, then we can quantify the distance between a feasible and an optimal solution using the ratio of the weight of the feasible solution to the weight of the optimal solution. If for a given algorithm we can prove that this ratio is bounded for any given instance, then we say that this algorithm is an approximation algorithm. In particular, for minimization problems this ratio r is always greater than or equal to 1; furthermore, if for a given algorithm we have $1 \leq r \leq a$ for all cases then this algorithm is an a-approximation algorithm . Similarly, for maximization problems this ratio r is always less than or equal to 1. Moreover, if for a given approximation algorithm we have $1 \geq r \geq a$ for all instances then this algorithm is an a-approximation algorithm.

Since not all algorithms have the same ratio and running time, we decide to categorize them as using these aspects. The most important categories are the following:

- Constant factor approximation algorithm: Any algorithm with constant ratio that runs in polynomial time.
- Polynomial-time approximation scheme (PTAS): This is an approximation algorithm, for a maximization problem (respectively for a minimization problem), has as additional information in the input a real number $\varepsilon > 0$, has ratio $1 - \varepsilon$ (resp. $1 + \varepsilon$) and its running time is a polynomial n (where n is the size of the input).
- Fully polynomial-time approximation scheme (FPTAS): These algorithms are PTAS algorithms that the running time is a polynomial in n and in $1/\varepsilon$.

In the following sections we will see that, under certain assumptions (such as P $\neq$ NP), there are problems that do not admit algorithms with a good approximation ratio.

Therefore, optimization problems can be categorized based on the best approximation ratio they can admit. Below we present some of these classes:

- FPTAS: Class of problems that admit FPTAS's.
- PTAS: Class of problems that admit PTAS's.
- APX: Class of problems that admit constant factor approximation algorithms.
- $f(n)$-APX: Class of problems that admit $O(f(n))$-approximation algorithms that run in polynomial time.

As we have already mentioned, there are problems that cannot be approximated under a particular ratio in polynomial time. One way of computing the bounds on the approximation ratio for a given problem is by using gap reductions. The idea behind gap-generating reductions is as follows:

Let A be a minimization problem and B a decision problem. Moreover, for an instance $y \in I_A$, we denote as $opt(y)$ the weight of an optimal solution of y. Now, suppose that, for two numbers k and c, we can give a mapping from the set of instances I_B of B to the set of instances I_A of A such that:

- all the yes-instances $x \in I_B$ are mapped to a $y \in I_A$ where $opt(y) \leq k$ and
- all the no-instances $x \in I_B$ are mapped to a $y \in I_A$ where $opt(y) > ck$.

Then, if B is an NP-hard problem, we cannot decide in polynomial time if, for an instance y of A, we have $opt(y) \leq k$ or $opt(y) > ck$. Therefore, the problem A does not admit any c-approximation algorithm that runs in polynomial time.

Note that, the previous conclusion follows from the fact that, any polynomial-time c-approximation algorithm for A could be used to separate the instances of A with optimal solution with weight less than k from those with optimal solution with weight greater than ck. Therefore, we could answer whether an instance of B is yes or no in polynomial time.

In some cases, we use additional assumptions in order to find bounds on the approximation ratio that can be achieved in polynomial time. One such hypothesis is the Unique Games Conjecture. For more information on the Unique Games Conjecture and the Unique Label Cover problem see Section 2.4.

2.3 Restricted Instances and Parameterization

When a problem is NP-hard, we can try to consider additional constraints in order to find ways to deal with it. In the next section we present several constraints for graph problems. Some constraints may be related to the size of a parameter of a particular problem. As a parameter we can consider any function of the instance of the problem. In many cases the parameter is related to the structural properties of the problem instance; for example, when dealing with graph problems, the maximum degree of the given graph

graph can be considered as a parameter. An interesting question that arises when considering parameters is the following: "If the size of this parameter is small, can we solve the problem faster?". When we consider such questions we say that we are considering parameterized problems (i.e., problems that have a particular parameter k as part of the input) and we are investigating the parameterized complexity of a problem under that parameter.

Given a parameterized problem with a parameter k, we want to decide whether there is a way to compute an optimal solution in time $f(|k|)poly(n)$ where $f(|k|)$ is a computable function and $poly(n)$ is a polynomial of the size of the input n. An algorithm with these properties is called fpt-algorithm. Using this idea we define the class FPT as follows:

Definition 2.5 (Class FPT)**.** *The class of parameterized problems that admit fpt-algorithms.*

Unfortunately, not all of the parameterized problems belong to FPT. Below, we present the W-hierarchy as proposed by Downey and Fellows:

$$\mathrm{P} \subseteq \mathrm{FPT} = \mathrm{W}[0] \subseteq \mathrm{W}[1] \cdots \subseteq \mathrm{W}[\mathrm{t}] \cdots \subseteq \mathrm{XP}$$

They conjectured that the containments are proper; this conjecture remains open. For a formal definition of the above parameterized classes and the W-hierarchy you can see [75].

Similarly to classical complexity, the tool we use to decide the parameterized complexity of a problem is a reduction. Because the complexity of a parameterized problem depend on the size of the parameter, we need a reduction that takes the parameter into account.

Definition 2.6 (fpt-reduction)**.** *Let (A, k) and (B, k') be two parameterized problems. We say that we have an fpt reduction form (A, k) to (B, k') if there exists an algorithm that transforms any instance (x, k) of (A, k) to an instance (y, k') of (B, k') and*

- *(x, k) is a yes instance if and only if (y, k') is a yes instance,*
- *there exists a computable function g such that $|k'| \leq g(|k|)$ and*
- *the algorithm runs in time $f(|k|)poly(n)$, for a computable function f.*

Similarly to polynomial reductions and NP-complete problems, here, a parameterized problem (A, k) is W[i]-complete if it belongs to W[i] and any other parameterized problem in W[i] is fpt-reducible to (A, k). One of the most fundamental problems on parameterized complexity is the Circuit Satisfiability.

Definition 2.7. *A Boolean circuit is a directed acyclic graph whose vertices belong to one of the following categories:*

- *input vertices: vertices of indegree* 0

- *negation vertices: vertices of indegree* 1
- *add vertices: vertices of in degree at least* 2
- *or vertices: vertices of in degree at least* 2

Finally, there is a unique vertex of out degree 0 *(that belongs to one of the previous categories) that is called output vertex.*

Given a circuit C, we say that a vertex is small if its indegree is upper bounded by a constant; otherwise it is large. We define the depth of a circuit as the maximum number of vertices in a path from an input vertex to the output vertex. The weft of a circuit is the maximum number of large vertices in any path from an input vertex to the output vertex. Finally, a truth assignment over the input vertices has weight k is we have assigned "true" to exactly k of the input vertices.

The parameterized version of the Circuit Satisfiability problem we are interested in is the following:

Definition 2.8 (Weft-t Circuit Satisfiability (W_tCS))**.** *Let C be a Boolean circuit of weft at most t and bounded depth, and k a positive integer. Is there a satisfying assignment over the input vertices of weight k?*

The reason why the Circuit Satisfiability problem is fundamental to parameterized complexity is because the problem (W_tCS,k'), where k' is the size of the assignment, is complete for the class W[t]. Therefore, using fpt reductions and Boolean circuits, we can show that a parameterized problem (A, k) belongs to class W[t] by presenting an fpt reduction form (A, k) to (W_tCS,k') where k' is the size of the assignment.

Parameterized algorithms (even FPT algorithms) can have really bad running times relative to the parameter. For this reason we always try to find algorithms that have minimal dependence on the parameter size. To verify that an algorithm cannot be improved we try to find computational lower bounds. Some of these bounds are based on the assumption P≠NP, however, in many cases we use other stronger assumptions. The main tool we will rely on is the exponential time hypothesis (ETH) of Impagliazzo, Paturi, and Zane [120], which states the following:

Conjecture 1. *Exponential Time Hypothesis: there exists an $\epsilon > 0$, such that 3-SAT on instances with n variables and m clauses cannot be solved in time $2^{\epsilon(n+m)}$.*

Note that it is common to use the slightly weaker formulation of ETH stating that 3-SAT cannot be solved in $2^{o(n+m)}$. We will also rely on the randomized version of ETH, which has the same statement as Conjecture 1 but for randomized algorithms with an expected running time of $2^{\epsilon(n+m)}$.

2.4 Graphs

Several of the problems we consider in this thesis are graph problems. A graph G consists of two sets, a set $V(G)$ of vertices and a set $E(G)$ of edges. The set $E(G)$ shows, in

some sense, whether two vertices are related or not. If two vertices u and v in $V(G)$ are related then an edge $e = uv \in E(G)$ otherwise $uv \notin E(G)$. Whenever the graph G is clear from the context, we will omit it from the notation and just write V and E. Below we give some standard notation for graphs.

We have two main classes of graphs: directed and undirected. Usually, instead of the term directed graphs, we use the term digraphs. The difference between the two classes is that in directed graphs, for two vertices u and v, edges uv and vu are considered different edges, while in undirected graphs they are not. Furthermore, in directed graphs the edges are directed and are called arcs. An arc uv is called an outgoing arc of u and an incoming arc of v.

In this thesis we consider graphs with no self-loops (i.e. there is no vertex $u \in V(G)$ such that $uu \in E(G)$) and (usually) we do not have the same edge multiple time; such graphs are called simple. In digraphs, a pair of arcs uv and vu is called a digon. Digraphs without self-loops, digons or, the same edge multiple times, are called oriented graphs.

Given an undirected graph $G = (V, E)$ we define the neighborhood of a vertex $u \in V$ to be the set of vertices $N_G(u) = \{v \in V \mid uv \in E\}$ and the degree $d_G(u) = |N_G(u)|$. The closed neighborhood of $u \in V$ is defined as $N_G[u] = \{u\} \cup N_G(u)$. For a given set of vertices $X \subseteq V$, we define the neighborhood of X as $N_G(X) = \bigcup_{v \in X} N_G(v)$ and the closed neighborhood of X as $N_G[X] = \bigcup_{v \in X} N_G[v]$. In a directed graph, for a vertex $u \in V$, we can define the out-neighborhood and the in-neighborhood as $N_G^+(u) = \{v \in V \mid uv \in E\}$ and $N_G^-(u) = \{v \in V \mid vu \in E\}$ respectively. In addition, we have the out-degree and the in-degree defined as $d_G^+(u) = |N^+(u)|$ and $d_G^-(u) = |N^-(u)|$. In both directed and undirected graphs, the maximum degree is denoted by $\Delta(G)$ and the minimum degree by $\delta(G)$. Once again, whenever the graph G is clear from the context, we will omit form the subscript.

Given a graph G and a $uv \in E(G)$ the graph G/uv is the graph obtained by contracting the edge uv, that is, replacing u, v by a new vertex connected to $N_G(u) \cup N_G(v)$. In this manuscript we will only apply this operation when $N_G(u) \cap N_G(v) = \emptyset$, so the result will always be a simple graph. Given a set of edges $S \subseteq E(G)$ we denote by $V(S)$ the set of vertices of $V(G)$ that meet edges of S.

2.4.1 Graph Classes

In many cases, we assume that we have additional constraints on our problem instances. In graph problems these constraints are usually related to the structural properties of the input graph. Based on such properties, several classes of graphs have been defined.

In order to simplify the definitions of some classes we first need to explain what we call induced subgraph and independent set.

Definition 2.9 (Subgraph)**.** *A graph $G' = (V', E')$ is a subgraph of $G = (V, E)$ if $V' \subseteq V$, $E' \subseteq E$ and for all $uv \in E'$ we have $u, v \in V'$.*

Definition 2.10 (Induced Subgraph)**.** *Let $G = (V, E)$ be a graph, $V' \subseteq V$ and $E' = \{uv \mid u, v \in V' \text{ and } uv \in E\}$. The graph (V', E') is called induced subgraph of G it is denoted by $G[V']$.*

Definition 2.11 (Independent Set)**.** *Let $G = (V, E)$ be a graph and $S \subseteq V$ a non empty set of vertices. If the induced subgraph $G[S]$ has no edges then we say that S is an independent set of G.*

Below we present the most common classes of undirected graphs.

- Complete graphs: Graphs that contain all possible edges. A complete graph with n vertices denoted by K_n.
- Trees: graphs that do not contain cycles.
- Bipartite graphs : graphs that do not contain odd length cycles. Alternatively, a graph $G = (V, E)$ is called bipartite if there exists a partition L, R of V such that L and R are independent sets of G.
- Complete bipartite graphs: bipartite graphs that contain all possible edges (i.e. for any $u \in L$ and $v \in R$ we have $uv \in E$). A complete bipartite graph with $|L| = n$ and $|R| = m$ is denoted by $K_{n,m}$.
- Split graphs: A graph $G = (V, E)$ is called split graph if there exists a partition L, R of V such that $G[L]$ is a complete graph and R is an independent set of G.
- Regular graphs: Graphs where all the vertices have the same degree. If in addition we know the degree d of the vertices then we call the graph d-regular.
- Cubic graphs: 3-regular graphs
- Subcubic graphs: Graphs of maximum degree $\Delta \leq 3$.
- Planar graphs: A graph is called planar if there exists a way to draw it on a 2-dimentional plan such that no edges cross each other.

2.4.2 Graph Problems

In this section we present some of the most important graph problems.

Definition 2.12 (Maximum Independent Set (Max-IS))**.** *Given a graph $G = (V, E)$ find an independent set S of G of maximum size (i.e. for any other independent set S' of G we have $|S| \geq |S'|$).*

Max-IS is one of the most studied, NP-hard, graph problems. The size of a maximal independent set of a graph G is denoted by $\alpha(G)$. In [183], it was proved that, for any $\varepsilon > 0$, there is no polynomial-time algorithm that approximates $\alpha(G)$, for a graph G, within a ratio of $n^{1-\varepsilon}$, unless P = NP. A weaker version of this result was first presented in [106] under the condition NP $\neq$ ZPP instead of P $\neq$ NP. The parameterized version of Max-IS, when we consider the solution size as the parameter, is W[1]-hard and it cannot be solved in $|V|^{o(k)}$ time, under the ETH [65].

Before we continue with the rest of the problems, let us introduce some useful notation. A set S with a property Π is called maximum if any other set S' with the same property has size $|S'| \leq |S|$. Using this notation we can say that the goal of Max-IS is to find a maximum independent set. Similarly, a set S with a property Π is called minimum if any other set S' with the same property has size $|S'| \geq |S|$.

Two other, similar, notions are these of *maximal* and *minimal* sets. A set S with a property Π is called maximal (resp. minimal) if any proper superset $S' \supset S$ (any proper subset $S' \subset S$) does not satisfy the property Π.

There are several problems, such as Max-IS, which search for subsets of vertices or edges that satisfy a certain property and have the largest (or smallest) possible size. Some well-known properties are shown below. Suppose we are given a graph $G = (V, E)$ then we define the following:

- Vertex Cover of G: A set $S \subseteq V$ is called *vertex cover* of G if for each edge $e = uv \in E$ we have that $u \in S$ or $v \in S$.
- Feedback Vertex Set of G: A set $S \subseteq V$ is called *feedback vertex set* of G if the graph $G[V \setminus S]$ does not contain any cycle.
- Dominating Set of G: A set $S \subseteq V$ is called *dominating set* of G if for any vertex $u \in V$ we have $u \in S$ or there exists $v \in S$ such that $uv \in E$.
- Edge Cover of G: A set $S \subseteq E$ is called *edge cover* of G if for any vertex $u \in V$ there exists a edge $e = uv \in E$, for some $v \in V$, where $e \in E$.
- Matching of G: A set $S \subseteq E$ is called *matching* of G if for any vertex $u \in V$ there exists at most one edge $e \in S$ which is incident to u.

Note that, in contrast with the first three properties, which are properties of vertex sets, the edge cover and matching are a properties that can be satisfied by sets of edges.

Using the previous properties, we can define the following problems:

Definition 2.13 (Minimum Vertex Cover (Min-VC))**.** *Let $G = (V, E)$ be a graph. The Minimum Vertex Cover problem asks to find a minimum vertex cover of G.*

Min-VC is considered as complement of the Max-IS. The reason is that, for a given graph $G = (V, E)$, the compliment $V \setminus S$, of any vertex cover S of G, is an independent set. Like Max-IS, Min-VC is also NP-hard; however, for Min-VC, there exists a 2-approximation algorithm. Furthermore, its parameterized version, when parameterized by the size of the solution, belongs to FPT.

Definition 2.14 (Minimum Feedback Vertex Set (Min-FVS))**.** *Let $G = (V, E)$ be a graph. The Minimum Feedback Vertex Set problem asks to find a minimum feedback vertex set of G.*

Min-FVS is NP-hard in the general case but it can be computed in polynomial time in graphs with maximum degree 3. It is APX-complete and it belongs to FPT when parameterized by its size.

Definition 2.15 (Minimum Dominating Set (Min-DS)). *Let $G = (V, E)$ be a graph. The Minimum Dominating Set problem asks to find a minimum dominating set of G.*

Min-DS is NP-hard and its parameterized version, when parameterized by the size of the solution, it is W[2]-complete. The size of a minimum dominating set, of a given graph G, is called domination number and denoted by $\gamma(G)$.

Definition 2.16 (Minimum Edge Cover (Min-EC)). *Let $G = (V, E)$ be a graph. The Minimum Edge Cover problem asks to find a minimum edge cover of G.*

Definition 2.17 (Maximum Matching problem). *Let $G = (V, E)$ be a graph. The Minimum Edge Cover problem asks to find a minimum edge cover of G.*

The Min-EC and Maximum Matching problems are problems that belong to P.

One, other, really important graph problem is the Coloring problem. First we need to explain what is a proper coloring of a graph. Let $G = (V, E)$ be a graph, k a positive integer and $C : V \to \{1, \ldots, k\}$ a function. We say that C is a proper coloring of G if for each edge $uv \in E$, $C(u) \neq C(v)$. Additionally, if such a function exists we say that G is k-colorable.

Definition 2.18 (k-Coloring). *Let $G = (V, E)$ be a graph. The k-Coloring problem asks if G is k-colorable or not.*

Note that this is a decision version of the Coloring problem. The optimization version asks to find the minimum k such that the given graph is k colorable. The k-Coloring problem is NP-complete even when $k = 3$. Using this, it is easy to see that the parameterized version of Coloring, when parameterized by the number of colors k, does not admit an fpt-algorithm. Indeed, if the problem belongs to FPT then there must be an algorithm that runs in $f(k)poly(n)$, where $n = |V|$, and this gives us a polynomial-time algorithm for 3-Coloring. However, such an algorithm cannot exist unless P = NP.

The Unique Label Cover problem is defined as follows:

Definition 2.19 (Unique Label Cover). *Let $G = (V, E)$ be a graph with some arbitrary total ordering $\prec$ of V, an integer R, and for each $uv \in E$ with $u \prec v$ a 1-to-1 constraint π_{uv} which can be seen as a permutation on $[R]$. The vertices of G are considered as variables of a constraint satisfaction problem, which take values in $[R]$. Each constraint π_{uv} defines for each value of u a unique value that must be given to v in order to satisfy the constraint. The goal is to find an assignment to the variables that satisfies as many constraints as possible.*

The above problem is important because it is related to the Unique Game Conjecture (UGC). The UGC states that for all $\epsilon > 0$, there exists R, such that distinguishing instances of Unique Label Cover for which it is possible to satisfy a $(1 - \epsilon)$-fraction of the constraints from instances where no assignment satisfies more than an ϵ-fraction of the constraints is NP-hard. A slightly different version of this conjecture was defined

by Khot and Regev and is called the Strong Unique Games Conjecture (SUGC). Despite the name, Khot and Regev showed that this version is implied by the standard UGC. The precise formulation is the following:

Theorem 2.1. *[Theorem 3.2 of [132] (SUGC)] If the Unique Games Conjecture is true, then for all $\epsilon > 0$ it is NP-hard to distinguish between the following two cases of instances of* Unique Label Cover *$G = (V, E)$:*

- *(Yes case): There exists a set $V' \subseteq V$ with $|V'| \geq (1 - \epsilon)|V|$ and an assignment for V' such that all constraints with both endpoints in V' are satisfied.*
- *(No case): For any assignment to V, for any set $V' \subseteq V$ with $|V'| \geq \epsilon|V|$, there exists a constraint with both endpoints in V' that is violated by the assignment.*

Both UGC and SUGC are important tools in hardness of approximation.

Treewidth and Tree decomposition To define the treewidth of a graph we need first to define what we call tree decomposition of a graph.

Definition 2.20 (Tree decomposition)**.** *Given a graph $G = (V, E)$, we call tree decomposition of G a tree $T = (V', E')$ together with a set $\mathcal{B} = \{B_t \subseteq V \mid t \in V'\}$ such that:*

- *For each vertex $t \in V'$ has been assigned a set $B_t \subseteq V$. We call this set bag of t.*
- $\bigcup_{t \in V'} B_t = V$
- *Let $t, w \in V'$ and $u \in B_t \cap B_w$. For any $x \in V'$, where x belongs to the (unique) path for t to w in T, we have that $u \in B_x$.*
- *Let $uv \in E$. There exists a $t \in V'$ such that $u, v \in B_t$.*

The size of biggest bag of T minus 1 called width of the tree decomposition .

Assume that we are given a graph $G = (V, E)$ and a tree decomposition $(T, \mathcal{B})$, where $T = (V', E')$ is a tree, of G. In order to avoid confusion, the vertices of the tree decomposition, $t \in V'$, will be called nodes.

Definition 2.21 (Treewidth)**.** *Given a graph G we are searching of the minimum number $tw \in \mathbb{N}$ such that, there exists a tree decomposition of G with width tw.*

2.4.3 Structural parameters of a graph

When we considered structural parameters of a graph we want to take advantage of the properties of the parameter in order to construct an algorithm. Two other important parameters are

- the size of minimum vertex cover and

- the size of minimum feedback vertex set.

Note that both the MIN-VC and MIN-FVS are FPT when parameterized by the size of their solutions. This is important because we can compute a minimum vertex cover or a minimum feedback vertex set before starting dealing with a problem.

The treewidth of a graph is an other really important parameter. In order to take advantage of the treewidth, tw, of a graph $G = (V, E)$, we need to compute a tree decomposition $(T, \mathcal{B})$, where $T = (V', E')$ is a rooted binary tree, that satisfies the following properties:

- the width of the tree decomposition is tw,
- each node of T belongs to exactly one of the following categories:
  - Leaf Node: a leaf, v, of T such that $|B_v| = 1$.
  - Introduce Node v: a node, v, of T with one child, c, such that $B_c \subset B_v$ and $|B_v \setminus B_c| = 1$.
  - Forget node: a node, v, of T with one child, c, such that $B_v \subset B_c$ and $|B_c \setminus B_v| = 1$.
  - Join Node: a node, v, of T with two children, c and c', such that $B_v = B_c = B_{c'}$.

This tree decomposition called nice and it was introduced in [33]. There exists an algorithm that transforms a given tree decomposition to a nice tree decomposition with $O(n\,tw)$ nodes in $O(n^2 tw)$ time. In case we are not given a tree decomposition, then we can use one of the various algorithms that compute one. In particular, if we are looking for a tree decomposition of width at most k, we can use the algorithm presented in [32]. This algorithm returns a tree decomposition of width at most k or reports that no such tree decomposition exists, in time $2^{O(tw^3)}O(n)$. Alternatively, there are approximation algorithms that can be used; an algorithm that returns a tree decomposition of width at most $2k+1$, or reports that the width is greater than k, and runs on $2^{O(tw)}O(n)$ was presented in [135].

The property that makes a nice tree decomposition of a graph so interesting is that each of its bags is a cut set of the original graph. This property allows us to take advantage of the structure of the nice tree decomposition and let us use dynamic programming to its nodes. For more details on tree decompositions see [75].

Chapter 3

Subset-Sums Ratio and Variants

3.1 Introduction

Subset sum computations are of key importance in computing, as they appear either as standalone tasks or as subproblems in a vast amount of theoretical and practical methods coping with important computational challenges. As most of subset sum problems are NP-hard, an effort was made over the years to come up with systematic ways of deriving approximation schemes for such problems. Important contributions in this direction include works by Gens and Levner [95], Horowitz and Sahni [114, 115], Ibarra and Kim [119], Sahni [174], Woeginger [180], and Woeginger and Pruhs [169]. In particular, the best known FPTAS for the Subset Sum problem appears in [129]. Moreover, the aforementioned problems have been recently considered under the perspective of fine-grained complexity [45, 161].

A problem, closely related to Subset Sum, is the Equal Subset Sum problem (ESS), which, given an input set, asks for two disjoint subsets of equal sum. It finds applications in multiple different fields, ranging from computational biology [60, 57] and computational social choice [144], to cryptography [179], to name a few. In addition, it is related to important theoretical concepts such as the complexity of search problems in the class TFNP [166].

In this section we are considering the optimization version of ESS problem, called Subset-Sums Ratio problem (SSR). SSR was introduced and shown NP-hard by Woeginger and Yu [181]. The formal definition of the problem is as follows:

Definition 3.1 (Subset-Sums Ratio (SSR))**.** *Given a set $A = \{a_1, \ldots, a_n\}$ of n positive integers, find two nonempty and disjoint sets $S_1, S_2 \subseteq \{1, \ldots, n\}$ that minimize the ratio:*

$$\frac{\max\{\sum_{i\in S_1} a_i, \sum_{j\in S_2} a_j\}}{\min\{\sum_{i\in S_1} a_i, \sum_{j\in S_2} a_j\}}.$$

Additionally, we are going to define and study a family of variations of this optimization problem.

Related Work As we mentioned, ESS has been proven NP-Complete by Woeginger and Yu [181] (see also the full version of [160] for an alternative proof) and several variations have been proven NP-Complete by Cieliebak *et al.* in [58, 59]. A 1.324-approximation algorithm has been proposed for SSR in [181] and several FPTASs appeared in [23, 163, 151], the fastest so far being the one in [151] of complexity $O(n^4/\varepsilon)$.

As far as exact algorithms are concerned, recent progress has shown that ESS can be solved probabilistically in[1] $O^*(1.7088^n)$ time [160], faster than a standard "meet-in-the-middle" approach yielding an $O^*(3^{n/2}) \leq O^*(1.7321^n)$ time algorithm.

These problems are tightly connected to Subset Sum, which has seen impressive advances recently, due to Koiliaris and Xu [134] who gave a deterministic $\tilde{O}(\sqrt{nt})$ algorithm, where n is the number of input elements and t is the target, and by Bringmann [43] who gave a $\tilde{O}(n+t)$ randomized algorithm, which is essentially optimal under SETH [1]. See also [11] for an extension of these algorithms to a more general setting. Jin and Wu subsequently proposed a simpler randomized algorithm [124] achieving the same bounds as [43], which however seems to only solve the decision version of the problem. Recently, Bringmann and Nakos [44] have presented an $O(|\mathcal{S}_t(Z)|^{4/3}\text{poly}(\log t))$ algorithm, where $\mathcal{S}_t(Z)$ is the set of all subset sums of the input set Z that are smaller than t, based on top-k convolution.

Regarding approximation schemes for Subset Sum, recently Bringmann and Nakos [45] presented the first improvement in over 20 years, since the scheme of [129] had remained the state of the art. Making use of modern techniques, they additionally provide lower bounds based on the popular *min-plus convolution* conjecture [67]. Furthermore, they present a new FPTAS for the closely related Partition problem, by observing that their techniques can be used to approximate a slightly more *relaxed* version of the Subset Sum problem, firstly studied in [160].

Our Contribution In Section 3.4 we present a new approximation scheme for this problem, highlighting its close relationship with the classical Subset Sum problem. The previously existing algorithms depends on the classic scaling techniques which transforms pseudo-polynomial algorithms to approximation algorithms. Our proposed algorithm is the first that does not, heavily, depend to such techniques and associates SSR with Subset Sum. Furthermore, improvements on the running time of exact or approximate algorithms for Subset Sum can also improve the running time of our approximation algorithm. Finally, we achieve a running time

$$O\left(\min\left\{\frac{n^{2.3}}{\varepsilon^{2.6}}\cdot\log(n/\varepsilon^2), \frac{n^2}{\varepsilon^3}\cdot\log(n/\varepsilon^2), \frac{n^2}{\varepsilon^2}\left(\log(n/\varepsilon^2)+\frac{1}{\varepsilon^2}\cdot\log(1/\varepsilon)\right)\right\}\right)$$

which improves over the $O(\frac{n^4}{\varepsilon})$ which is the current state of the art [151].

In Section 3.6 we define two families of variations of SSR problems that are able to capture additional restrictions. Then, in Section 3.7, we present some conditions that guarantee that a problem in these families admits an FPTAS. Finally, Section 3.8 we use

[1]Standard O^* notation is used to hide polynomial and $\tilde{O}$ to hide polylogarithmic factors.

our framework to present FPTASs for two variations of SSR, namely 2-Set Subset-Sums Ratio (2-Set SSR) and Factor-r Subset-Sums Ratio (Factor-r SSR); to the best of our knowledge, no approximation algorithm has been known so far for these problems.

3.2 Preliminaries

In what follows, we assume that we are given a set of positive integers $A = \{a_1, \ldots, a_n\}$ such that $a_j \leq a_i$ for all $1 \leq j \leq i \leq n$. Let $[n] = \{1, \ldots, n\}$ denote the set of integers in the interval $[1, n]$. For a set of indices $S \subseteq [n]$ we denote $\max(S, A) = \max\{a_i \mid i \in S\}$. Furthermore, we use the function $\mathcal{R}(S_1, S_2, A)$ introduced in [151]; this function was defined as:

Ratio of two subsets. *Given a set $A = \{a_1, \ldots, a_n\}$ of n positive numbers and two sets $S_1, S_2 \subseteq \{1, \ldots, n\}$ we define $\mathcal{R}(S_1, S_2, A)$ as follows:*

$$\mathcal{R}(S_1, S_2, A) = \begin{cases} 0 & \text{if } S_1 = \emptyset \text{ and } S_2 \neq \emptyset \\ \frac{\sum_{i \in S_1} a_i}{\sum_{i \in S_2} a_i} & \text{if } S_2 \neq \emptyset, \\ +\infty & \text{otherwise.} \end{cases}$$

We also define and use $\mathcal{MR}(S_1, \ldots, S_k, A)$ which is a generalization of $\mathcal{R}(S_1, S_2, A)$ to $k > 2$ sets:

Max ratio of k subsets. *Given a set $A = \{a_1, \ldots, a_n\}$ of n positive numbers and k sets $S_1, \ldots, S_k \subseteq \{1, \ldots, n\}$ we define:*

$$\mathcal{MR}(S_1, \ldots, S_k, A) = \max\{\mathcal{R}(S_i, S_j, A) \mid i \neq j \text{ and } i, j \in \{1, \ldots, k\}\}$$

Note that we are using these expressions only if for all indices $j \in S_i$, $i \in [k]$, we know that $a_j \in A$. In order to keep our expressions as simple as possible we use the above functions throughout the whole chapter.

If, in addition to the set A, we are given a set value $\varepsilon \in (0, 1)$, define the following *partition* of the elements of A:

- The set of its *large* elements as $L(A, \varepsilon) = \{a_i \in A \mid a_i \geq \varepsilon \cdot a_n\}$. Note that $a_n \in L(A, \varepsilon)$, for any $\varepsilon \in (0, 1)$.
- The set of its *small* elements as $M(A, \varepsilon) = \{a_i \in A \mid a_i < \varepsilon \cdot a_n\}$.

In the following, since the values of the associated parameters will be clear from the context, they will be omitted and we will refer to these sets simply as L and M. Furthermore, we denote the set of indices of elements in $L(A, \varepsilon)$ as $I_L = \{i \mid a_i \in L(A, \varepsilon)\}$ and the set of indices of elements in $M(A, \varepsilon)$ as $I_M = \{i \mid a_i \in M(A, \varepsilon)\}$.

The following definitions will be useful for the sections 2.3 to 2.5 of this chapter.

Definition 3.2 (Closest set and pair)**.** *Given a set $A = \{a_1, \ldots, a_n\}$ and a set $S \subseteq [n]$, we define as* closest set *to S a set S_{opt} such that $S_{opt} \subseteq [n] \setminus S$ and $\Sigma_{i \in S}\ a_i \geq \Sigma_{i \in S_{opt}}\ a_i \geq \Sigma_{i \in S'}\ a_i$ for all $S' \subseteq [n] \setminus S$. The pair (S, S_{opt}) is called* closest pair*.*

Definition 3.3 (ε-close set and pair)**.** *Given a set $A = \{a_1, \ldots, a_n\}$ and a set $S \subseteq [n]$, we define as ε-*close set *to S a set S_ε such that $S_\varepsilon \subseteq [n] \setminus S$ and $\Sigma_{i \in S}\ a_i \geq \Sigma_{i \in S_\varepsilon}\ a_i \geq (1 - \varepsilon) \cdot \Sigma_{i \in S'}\ a_i$ for all $S' \subseteq [n] \setminus S$. The pair (S, S_ε) is called ε-*close pair*.*

Remark. *Note that S_{opt} is also an ε-close set of S for any $\varepsilon \in (0, 1)$.*

Definition 3.4 (Subset Sum)**.** *Given a set $A = \{a_1, \ldots, a_n\}$ and target t, compute a set $S \subseteq [n]$, such that $\Sigma_{i \in S}\ a_i = \max\{\Sigma_{i \in S'}\ a_i \mid S' \subseteq [n], \Sigma_{i \in S'}\ a_i \leq t\}$.*

Definition 3.5 (Approximate Subset Sum)**.** *Given a set $A = \{a_1, \ldots, a_n\}$, target t and a number $\varepsilon \in (0, 1)$, compute a set $S \subseteq [n]$, such that $(1 - \varepsilon) \cdot OPT \leq \Sigma_{i \in S}\ a_i \leq OPT$, where $OPT = \max\{\Sigma_{i \in S'}\ a_i \mid S' \subseteq [n], \Sigma_{i \in S'}\ a_i \leq t\}$.*

For a given set $A = \{a_1, \ldots, a_n\}$ and a set of indices $S \subseteq [n]$ one can compute a closest set S_{opt} or an ε-close set S_ε by solving Subset Sum or its approximate version as follows.

1. **Closest set (S_{opt}) computation**
   Compute the subset sums of set $A \setminus \{a_i \mid i \in S\}$ with target $\Sigma_{i \in S}\ a_i$ and return the set of indices of the solution set. This can be achieved by a standard meet in the middle [114] algorithm.

2. **ε-close set (S_ε) computation**
   Run an approximate Subset Sum algorithm [129, 45] with error margin ε on set $A \setminus \{a_i \mid i \in S\}$ with target $\Sigma_{i \in S}\ a_i$.

Note that in most papers the requested output for the Subset Sum or Approximate Subset Sum is the subset of the given set A and not the set of indices. However, here we assume that the output is given as the set of indices.

3.3 Approximation scheme for Constrained SSR

In this section, we present an FPTAS for the constrained version of the Subset-Sums Ratio problem where we are only interested in approximating the ratio of solutions that involve large subset sums. By this, we mean that for at least one of the subsets of the optimal solution, the sum of its large elements must be no less than $\max(A) = a_n$ (assuming that $A = \{a_1, \ldots, a_n\}$ is the *sorted* input set); let r_{opt} denote the subset sum ratio of such an optimal solution. Our FPTAS will return a solution of ratio r, such that $1 \leq r \leq (1 + \varepsilon) \cdot r_{opt}$, for a given $\varepsilon \in (0, 1)$; however, we allow that the sets of the returned solution do not necessarily satisfy the aforementioned constraint (i.e. the sum of their large elements may be less than a_n).

3.3.1 Outline of the algorithm

We now present a rough outline of the algorithm, along with its respective pseudocode:

- At first, we search for approximate solutions involving exclusively large elements from $L(A, \varepsilon)$.
- To this end, we produce the subset sums formed by these large elements. If their number exceeds n/ε^2, then we can easily find an approximate solution.
- Otherwise, for each of the produced subsets, we find its corresponding ε'-close set, for some appropriate ε' defined later.
- Then, it suffices to consider only these pairs of subsets when searching for an approximate solution.
- In the case that the optimal solution involves small elements, we can approximate it by adding elements of $M(A, \varepsilon)$ in a greedy way.

Algorithm 1 ConstrainedSSR(A, ε, T)

Input: Sorted set $A = \{a_1, \ldots, a_n\}$, a number $\varepsilon \in (0, 1)$ and table of partial sums T.
Output: $(1 + \varepsilon)$-approximation of the optimal solution respecting the constraint.

```
Partition A to M = {a_i ∈ A | a_i < ε · a_n} and L = {a_i ∈ A | a_i ≥ ε · a_n}.
Split interval [0, n · a_n] to n/ε^2 bins of size ε^2 · a_n.
while filling the bins with the subset sums of L do
    if two subset sums correspond to the same bin then
        return an approximation solution based on these.      ▷ O(n/ε^2) complexity.
    end if
end while
2^|L| ≤ n/ε^2  ⟺  |L| ≤ log(n/ε^2).
for each subset in a bin do                                   ▷ O(n/ε^2) subsets.
    Find its ε'-close set.                                    ▷ Complexity analysis in Section 3.5.
    Add small elements accordingly.      ▷ O(log n) complexity, see Subsection 3.3.3.
end for
```

3.3.2 Regarding only large elements

We firstly search for an $(1 + \varepsilon)$-approximate solution with $\varepsilon \in (0, 1)$, without involving any of the elements that are smaller than $\varepsilon \cdot a_n$. Let $M = \{a_i \in A \mid a_i < \varepsilon \cdot a_n\}$ be the set of small elements and $L = A \setminus M = \{a_i \in A \mid a_i \geq \varepsilon \cdot a_n\}$ be the set of large elements.

After partitioning the input set, we split the interval $[0, n \cdot a_n]$ into smaller intervals, called bins, of size $l = \varepsilon^2 \cdot a_n$ each, as depicted in figure 3.1.

Thus, there are a total of $B = n/\varepsilon^2$ bins. Notice that each possible subset of the input set will belong to a respective bin constructed this way, depending on its sum.

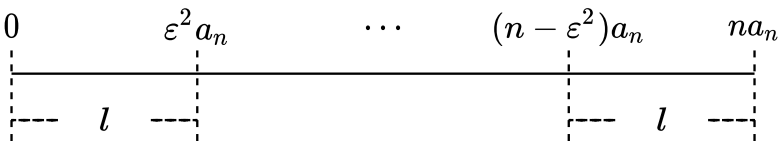


Figure 3.1: Split of the interval $[0, n \cdot a_n]$ to bins of size l.

Additionally, if two sets correspond to the same bin, then the difference of their subset sums will be at most l.

The next step of our algorithm is to generate all the possible subset sums, occurring from the set of large elements L. The complexity of this procedure is $O\left(2^{|L|}\right)$, where $|L|$ is the cardinality of set L. Notice however, that it is possible to bound the number of the produced subset sums by the number of bins B, since if two sums belong to the same bin they constitute a solution, as shown in Lemma 3.1, in which case the algorithm terminates in time $O(n/\varepsilon^2)$.

Lemma 3.1. *If two subsets correspond to the same bin, we can find an* $(1+\varepsilon)$*-approximation solution.*

Proof. Suppose there exist two sets $L_1, L_2 \subseteq I_L$ such that the sums $\sum_{i\in L_1} a_i$ and $\sum_{i\in L_1} a_i$ correspond to the same bin and $\sum_{i\in L_1} a_i \leq \sum_{i\in L_2} a_i$. Notice that there is no guarantee regarding the disjointness of said subsets, thus consider $L_1' = L_1 \setminus L_2$ and $L_2' = L_2 \setminus L_1$, for which it is obvious that $\sum_{i\in L_1} a_i \leq \sum_{i\in L_2} a_i$.

Additionally, assume that $L_1' \neq \emptyset$. Then it holds that

$$\sum_{i\in L_2'} a_i - \sum_{i\in L_1'} a_i = \sum_{i\in L_2} a_i - \sum_{i\in L_1} a_i \leq l$$

Therefore, the sets L_1' and L_2' constitute an $(1+\varepsilon)$-approximation solution, since

$$\begin{aligned}\mathcal{R}(L_2', L_1', A) = \mathcal{R}(L_2', L_1', L) &\leq \frac{l + \sum_{i\in L_1'} a_i}{\sum_{i\in L_1'} a_i} \\ &= 1 + \frac{l}{\sum_{i\in L_1'} a_i} \leq 1 + \frac{\varepsilon^2 \cdot a_n}{\varepsilon \cdot a_n} = 1 + \varepsilon\end{aligned}$$

where the last inequality is due to the fact that $L_1' \subseteq I_L$ and all elements in L are grater than or equal to $\varepsilon \cdot a_n$, thus $\sum_{i\in L_1'} a_i \geq \varepsilon \cdot a_n$.

It remains to show that $L_1' \neq \emptyset$. Assume that $L_1' = \emptyset$. This implies that $L_1 \subseteq L_2$ and since we consider each subset of L only once and the input is a set and not a multiset, it holds that $L_1 \subset L_2 \implies L_2' \neq \emptyset$. Since the sums $\sum_{i\in L_1} a_i$ and $\sum_{i\in L_2} a_i$ correspond to the same bin, it holds that

$$\sum_{i\in L_2} a_i - \sum_{i\in L_1} a_i \leq l \implies \sum_{i\in L_2'} a_i - \sum_{i\in L_1'} a_i \leq l \implies \sum_{i\in L_2'} a_i \leq l$$

Since L'_2 is a non empty subset of I_L, which contain indices of elements that are greater than or equal to $\varepsilon \cdot a_n$, we have $\sum_{i \in L'_2} a_i \geq \varepsilon \cdot a_n > \varepsilon^2 \cdot a_n = l$, since $\varepsilon < 1$; contradiction. □

Consider an ε' such that $1/(1-\varepsilon') \leq 1+\varepsilon$ for all $\varepsilon \in (0,1)$, for instance $\varepsilon' = \varepsilon/2$ (the exact value of ε' will be computed in Section 3.5). If every produced subset sum of the previous step belongs to a distinct bin, then, we compute their respective ε'-close sets, as described in Section 3.2. We can approximate an optimal solution that involves exclusively large elements using these pairs.

Before we prove the previous statement, observe that, if the optimal solution involves sets $L_1, L_2 \subseteq I_L$ composed only of indices of large elements, where $\sum_{i \in L_1} a_i \leq \sum_{i \in L_2} a_i$, then $\sum_{i \in L_1} a_i = \sum_{i \in L_{2,opt}} a_i$, where $L_{2,opt}$ is a closest set of L_2, with respect to the set $I_L \setminus L_2$.

Lemma 3.2. *If the optimal ratio r_{opt} involves sets consisting of only large elements, then there exists an ε'-close pair with ratio $r \leq (1+\varepsilon) \cdot r_{opt}$.*

Proof. Assume that the sets $S_1^*, S_2^* \subseteq I_L$ form the optimal solution (S_2^*, S_1^*) and $\mathcal{R}(S_2^*, S_1^*, L) = r_{opt} \geq 1$ is the optimal ratio. Then, as mentioned, it holds that $\sum_{i \in S_1^*} a_i = \sum_{i \in S_{2,opt}^*} a_i$. For each set of large elements, there exists an ε'-close set and a corresponding ε'-close pair; let $(S_2^*, S_{2,\varepsilon'}^*)$ be this pair for set S_2^*. Then,

$$\sum_{i \in S_2^*} a_i \geq \sum_{i \in S_1^*} a_i = \sum_{i \in S_{2,opt}^*} a_i \geq \sum_{i \in S_{2,\varepsilon'}^*} a_i \geq (1-\varepsilon') \cdot \sum_{i \in S_1^*} a_i$$

Thus, it holds that

$$1 \leq \mathcal{R}(S_2^*, S_{2,\varepsilon'}^*, L) \leq \frac{1}{(1-\varepsilon')} \cdot \mathcal{R}(S_2^*, S_1^*, L) \leq (1+\varepsilon) \cdot r_{opt}$$

□

Therefore, we have proved that in the case where the optimal solution consists of sets comprised of only large elements, it is possible to find an $(1+\varepsilon)$-approximation solution. This is achieved by computing an ε'-close set for each subset $S \subseteq I_L$ belonging in some bin, using the algorithms described in the preliminaries, with respect to set $L \setminus \{a_i \mid i \in S\}$ and target $\sum_{i \in S} a_i$. The total cost of these algorithms will be thoroughly analyzed in Section 3.5 and depends on the algorithm used.

It is important to note that by utilizing an (exact or approximation) algorithm for Subset Sum, we establish a connection between the complexities of Subset Sum and approximating Subset-Sums Ratio in a way that any future improvement in the first carries over to the second.

3.3.3 General FPTAS for Constrained SSR

Whereas we previously considered optimal solutions involving exclusively large elements, here we will search for approximations for those optimal solutions that use all the elements of the input set, hence include small elements, and satisfy our constraint. We will prove that in order to approximate those optimal solutions, it suffices to consider only the ε'-close pairs corresponding to each distinct bin and add small elements to them. In other words, instead of considering any two random disjoint subsets consisting of large elements[2] and subsequently adding to these the small elements, we can instead consider only the pairs computed in the previous step, the number of which is bounded by the number of bins $B = n/\varepsilon^2$. Moreover, we will prove that it suffices to add the small elements to our solution in a greedy way.

Since the algorithm has not detected a solution so far, due to Lemma 3.1 every computed subset sum of set L belongs to a different bin. Thus, their total number is bounded by the number of bins B, i.e.

$$2^{|L|} \leq \left(\frac{n}{\varepsilon^2}\right) \iff |L| \leq \log\left(\frac{n}{\varepsilon^2}\right).$$

We proceed by involving small elements in order to reduce the difference between the sums of ε'-close pairs, thus reducing their ratio.

Lemma 3.3. *Given the ε'-close pairs, one can find an $(1+\varepsilon)$-approximation solution for the constrained version of* Subset-Sums Ratio*, in the case that the optimal solution involves small elements.*

sketch. Due to page limitations, we only give a short sketch of the proof here; the full proof is deferred to the appendix (paper).

Let $S_1^* = L_1^* \cup M_1^*$ and $S_2^* = L_2^* \cup M_2^*$ be disjoint subsets of $[n]$ that form an optimal solution, where $\sum_{i \in S_1^*} a_i \leq \sum_{i \in S_2^*} a_i$, $L_1^*, L_2^* \subseteq I_L$ and $M_1^*, M_2^* \subseteq I_M$. Recall that we have assume that $a_i \leq a_j$ for all $1 \leq i \leq j \leq n$. Therefor $I_M = [|M|]$.

W.l.o.g., assume that $\sum_{i \in L_1^*} a_i < \sum_{i \in L_2^*} a_i$ (respectively $\sum_{i \in L_2^*} a_i < \sum_{i \in L_1^*} a_i$). We show that is suffices to find an appropriate k, such that $[k] \subseteq I_M$, and add $[k]$ to $L_{2,\varepsilon'}^*$ (respectively $L_{1,\varepsilon'}^*$) in order to approximate the optimal solution $r_{opt} = \mathcal{R}(S_2^*, S_1^*, L)$.

Therefore, by adding in a greedy way small elements to an ε'-close set of the set with the largest sum among L_1^* and L_2^*, we can successfully approximate the optimal solution. □

Adding small elements efficiently.

Here, we will describe a method to efficiently add small elements to our sets. As a reminder, up to this point the algorithm has detected an ε'-close pair (L_2, L_1), such that $L_1, L_2 \subseteq I_L$ with $\sum_{i \in L_1} a_i < \sum_{i \in L_2} a_i$. Thus, we search for some k such that $\sum_{i \in L_1 \cup M_k} a_i \leq \sum i \in L_2 a_i + \varepsilon \cdot a_n$, where $M_k = [k]$. Notice that if $\sum_{i \in I_M} a_i \geq \sum_{i \in L_2} a_i -$

[2]Note that the number of these random pairs is $3^{|L|}$.

$\sum_{i \in L_1} a_i$, there always exists such a set M_k, since by definition, each element of set M is smaller than $\varepsilon \cdot a_n$. In order to determine k, we make use of an array of partial sums $T[k] = \sum_{i \in [k]} a_i$, where $k \leq |M|$. Notice that T is sorted; therefore each time we need to compute a subset with the desired property, this can be done in $O(\log |M|) = O(\log n)$ time.

3.4 FPTAS for SSR

The algorithm presented in the previous section constitutes an approximation scheme for Subset-Sums Ratio, in the case where at least one of the solution subsets has sum of its large elements greater than, or equal to the max element of the input set. Thus, in order to solve the Subset-Sums Ratio problem, it suffices to run the previous algorithm n times, where n depicts the cardinality of the input set A, while each time removing the max element of A.

In particular, suppose that the optimal solution involves disjoint sets S_1^* and S_2^*, where $a_k = \max\{a_i \mid i \in S_1^* \cup S_2^*\}$. There exists an iteration for which the algorithm considers as input the set $A_k = \{a_1, \ldots, a_k\}$. In this iteration, the element a_k is the largest element and the algorithm searches for a solution where the sum of the large elements of one of the two subsets is at least a_k. The optimal solution has this property so the ratio of the approximate solution that the algorithm of the previous section returns is at most $(1 + \varepsilon)$ times the optimal.

Consequently, n repetitions of the algorithm suffice to construct an FPTAS for Subset-Sums Ratio.

Notice that if at some repetition, the sets returned due to the algorithm of Section 3.3 have ratio at most $1 + \varepsilon$, then this ratio successfully approximates the optimal ratio $r_{opt} \geq 1$, since $1 + \varepsilon \leq (1 + \varepsilon) \cdot r_{opt}$, therefore they constitute an approximation solution.

Algorithm 2 SSR(A, ε)

Input: Sorted set $A = \{a_1, \ldots, a_n\}$ and a number $\varepsilon \in (0, 1)$.
Output: $(1 + \varepsilon)$-approximation of the optimal solution for Subset-Sums Ratio.

1: Create array T such that $T[k] = \sum_{i=1}^{k} a_i$. ▷ $\Theta(n)$ time.
2: **for** $i = n, \ldots, 1$ **do**
3: ConstrainedSSR$(\{a_1, \ldots, a_i\}, \varepsilon, T)$
4: **end for**

3.5 Complexity

The total complexity of the final algorithm is determined by three distinct operations, over the n iterations of the algorithm:

1. The cost to compute all the possible subset sums occurring from large elements. It suffices to consider the case where this is bounded by the number of bins $B = n/\varepsilon^2$, due to Lemma 3.1.

2. The cost to find the ε'-close pair for each subset in a distinct bin. The cost of this operation will be analyzed in the following subsection.

3. The cost to include small elements to the ε'-close pairs. There are B ε'-close pairs, and each requires $O(\log n)$ time, thus the total time required is $O\Big(\frac{n}{\varepsilon^2} \cdot \log n\Big)$.

3.5.1 Complexity to find the ε'-close pairs

Using exact Subset Sum computations.

The first algorithm we mentioned is a standard meet in the middle algorithm. Here we will analyze its complexity.

Let subset $L' \subseteq L$ such that $|L'| = k$. The meet in the middle algorithm on the set $L \setminus L'$ costs time

$$O\left(\frac{|L \setminus L'|}{2} \cdot 2^{\frac{|L \setminus L'|}{2}}\right).$$

Notice that the number of subsets of L of cardinality k is $\binom{|L|}{k}$ and that $|L| \leq \log(n/\varepsilon^2)$. Additionally,

$$\begin{aligned}\sum_{k=0}^{|L|} \binom{|L|}{k} \cdot 2^{\frac{|L|-k}{2}} \cdot \frac{|L|-k}{2} &= 2^{|L|/2} \cdot \sum_{k=0}^{|L|} \binom{|L|}{k} \cdot 2^{-k/2} \cdot \frac{|L|-k}{2} \\ &\leq 2^{|L|/2} \cdot \frac{|L|}{2} \cdot \sum_{k=0}^{|L|} \binom{|L|}{k} \cdot 2^{-k/2}\end{aligned}$$

Furthermore, let $c = (1 + 2^{-1/2})$, where $\log c = 0.7715... < 0.8$. Due to Binomial Theorem, it holds that

$$\sum_{k=0}^{|L|} \binom{|L|}{k} \cdot 2^{-k/2} = (1 + 2^{-1/2})^{|L|} = c^{|L|} \leq c^{\log(n/\varepsilon^2)} = (n/\varepsilon^2)^{\log c}$$

Consequently, the complexity to find a closest set for every subset in a bin is

$$\begin{aligned}O\Big(2^{|L|/2} \cdot \frac{|L|}{2} \cdot (n/\varepsilon^2)^{\log c}\Big) &= O\Big((n/\varepsilon^2)^{1/2} \cdot \log(n/\varepsilon^2) \cdot (n/\varepsilon^2)^{\log c}\Big) \\ &= O\left(\frac{n^{1.3}}{\varepsilon^{2.6}} \cdot \log(n/\varepsilon^2)\right)\end{aligned}$$

Using approximate Subset Sum computations.

Here we will analyze the complexity in the case we run an approximate Subset Sum algorithm in order to compute the ε'-close pairs.

For subset $L_i \subseteq L$ of sum $\sum(L_i)$, we run an approximate Subset Sum algorithm ([129, 45]), with error margin ε' such that

$$\frac{1}{1-\varepsilon'} \leq 1 + \varepsilon \iff \varepsilon' \leq \frac{\varepsilon}{1+\varepsilon}$$

By choosing the maximum such ε', we have that

$$\varepsilon' = \frac{\varepsilon}{1+\varepsilon} \implies \frac{1}{\varepsilon'} = \frac{1+\varepsilon}{\varepsilon} = \frac{1}{\varepsilon} + 1 \implies \frac{1}{\varepsilon'} = O\left(\frac{1}{\varepsilon}\right)$$

Thus, if we use for instance the approximation algorithm[3] presented at [129], the complexity of finding all the ε'-close sets (one for every subset in a bin, for a total of a maximum of $B = n/\varepsilon^2$ subsets) is

$$O\left(\frac{n}{\varepsilon^2} \cdot \min\left\{\frac{|L|}{\varepsilon'}, |L| + \frac{1}{(\varepsilon')^2} \cdot \log(1/\varepsilon')\right\}\right) =$$
$$O\left(\frac{n}{\varepsilon^2} \cdot \min\left\{\frac{|L|}{\varepsilon}, |L| + \frac{1}{\varepsilon^2} \cdot \log(1/\varepsilon)\right\}\right) =$$
$$O\left(\frac{n}{\varepsilon^2} \cdot \min\left\{\frac{\log(n/\varepsilon^2)}{\varepsilon}, \log(n/\varepsilon^2) + \frac{1}{\varepsilon^2} \cdot \log(1/\varepsilon)\right\}\right)$$

3.5.2 Total complexity

The total complexity of the algorithm occurs from the n distinct iterations required and depends on the algorithm chosen to find the ε'-close pairs, since both of the presented algorithms dominate the time of the rest of the operations. Thus, by choosing the fastest one (depending on the relationship between n and ε), the final complexity is

$$O\left(\min\left\{\frac{n^{2.3}}{\varepsilon^{2.6}} \cdot \log(n/\varepsilon^2), \frac{n^2}{\varepsilon^3} \cdot \log(n/\varepsilon^2), \frac{n^2}{\varepsilon^2}\left(\log(n/\varepsilon^2) + \frac{1}{\varepsilon^2} \cdot \log(1/\varepsilon)\right)\right\}\right)$$

3.6 Families of Variations of SSR

In this section we define two families of variations of the SSR problem, we present two problems that belong in these families and finally we prove two lemmas that give us some properties of the feasible solutions of the problems in these families.

The first family of variations of SSR we define is the *Family of Subset-Sums Ratio* problems (F-SSR). We want this family to contain as many problems as possible. In general we may not have just a set of numbers as input, but also a graph with weights on the edges or the vertices, or any other structure. For such reasons we will use the following notation.

Family of Subset-Sums Ratio problems (F-SSR). *A problem $\mathcal{P}$ in F-SSR is a combinatorial optimization problem $(\mathcal{I}, k, \mathcal{F})$ where:*

- *$\mathcal{I}$ is a set of instances each of which is a pair (E, w) where $E = \{e_1, \ldots, e_n\}$ is a set of ground elements and $w : E \mapsto \mathbb{R}^+$ is a weight function which maps every element e_i to a positive number a_i;*

[3] Of complexity $O\big(\min\big\{\frac{n}{\varepsilon}, n + \frac{1}{\varepsilon^2} \cdot \log(1/\varepsilon)\big\}\big)$ for n elements and error margin ε.

- *k defines the number of subsets of $\{1, \ldots, n\}$ we are searching for;*
- *$\mathcal{F}$ gives the set of feasible solutions as follows: for any input (E, w), $\mathcal{F}(k, E)$ is a collection of k-tuples of nonempty and disjoint subsets of $\{1, \ldots, n\}$, and given $(k, E, (S_1, \ldots, S_k))$ we can check in polynomial time whether $(S_1, \ldots, S_k) \in \mathcal{F}(k, E)$.*

The goal of $\mathcal{P}$ is to find for an instance (E, w) a feasible solution $(S_1^*, \ldots, S_k^*)$ such that

$$\mathcal{MR}(S_1^*, \ldots, S_k^*, A) = \min\{\mathcal{MR}(S_1, \ldots, S_k, A) \mid (S_1, \ldots, S_k) \in \mathcal{F}(k, E)\}$$

where $A = \{a_i = w(e_i) \mid e_i \in E\}$

Remark. *Note that under this definition of F-SSR the function w of an instance (E, w) does not play any role in deciding whether a k-tuple $(S_1, \ldots, S_k)$ is feasible or infeasible solution; in other words, the element weights do not affect feasibility, only their indices do. Consequently, for a specific problem $\mathcal{P} = (\mathcal{I}, k, \mathcal{F}) \in F\text{-}SSR$ and two different instances (E, w) and (E, w') in $\mathcal{I}$ with the same ground elements E, the feasible solutions of the two instances are the same.*

We will now introduce a family of problems that is similar to *F-SSR*, with a major difference which is an additional condition. In this family we know (it is given as input), the minimum among the maximum values of the solution sets. This is rather technical and its role will become clear in the following text.

Family of Semi-Restricted Subset-Sums Ratio Problems (*Semi-Restricted F-SSR*). *For every problem $\mathcal{P} = (\mathcal{I}, k, \mathcal{F})$ in F-SSR, we define an associated optimization problem $\mathcal{P}' = (\mathcal{I}', k', \mathcal{F}')$ as follows:*

- *the set of instances of $\mathcal{P}'$ is:*

$$\mathcal{I}' = \{(E, w, m) \mid (E, w) \in \mathcal{I} \text{ and } m \in \{1, \ldots |E|\}\}$$

- $k' = k$
- *the collection of feasible solutions of instance $(E, w, m) \in \mathcal{I}'$ is given by:*

$$\mathcal{F}'(k, E, w, m) = \{(S_1, ..., S_k) \in \mathcal{F}(k, E) \mid \min_{j \in \{1,...,k\}} \{\max_{i \in S_j} w(e_i)\} = w(e_m)\}$$

and the goal of $\mathcal{P}'$ is to find a feasible solution $(S_1^*, \ldots, S_k^*)$ for an instance (E, w, m) such that

$$\mathcal{MR}(S_1^*, \ldots, S_k^*, A) = \min\{\mathcal{MR}(S_1, \ldots, S_k, A) \mid (S_1, \ldots, S_k) \in \mathcal{F}'(k, E, w, m)\}$$

where $A = \{a_i = w(e_i) \mid e_i \in E\}$. We define the family of problems Semi-Restricted F-SSR as the class of problems $\{\mathcal{P}' \mid \mathcal{P} \in F\text{-}SSR\}$.

Remark. *(i) We note that if a problem belongs to Semi-Restricted F-SSR it cannot belong to F-SSR because of the additional condition for a solution $(S_1, ..., S_k)$ to be feasible, namely $\min_{j\in\{1,...k\}}\{\max_{i\in S_j} w(e_i)\} = w(e_m)$, which depends on the weight function w and not only on the set of elements E as is the case for problems in F-SSR.*

(ii) It is obvious that if there exists an efficient algorithm that can decide if a solution is feasible for a problem $\mathcal{P}$ in F-SRR then we can construct another efficient algorithm that takes into account the additional conditions and decides if a solution is feasible for the semi-restricted version $\mathcal{P}'$.

Let us note that *F-SSR* contains many problems from several different areas in computer science and could prove useful to get an FPTAS for them if we could develop a pseudopolynomial algorithm with a particular property (as will be explained in the next section) for their semi-restricted versions. This family includes subset-sums ratio problems with matroid restrictions, graph restrictions, cardinality restrictions (including partition problems). Certain scheduling and knapsack problems also belong to this family.

As particular examples, we present some problems that belong in *F-SSR*. First we present the following job scheduling problem.

Minimum Ratio Scheduling (MinRS). *Let $j_1, \ldots, j_n$ be a list of jobs and $m_1, \ldots, m_k$ a list of machines. Each job has a processing time t_i, $i \in [n]$. The goal is to partition the jobs into k sets, $S_1, \ldots, S_k$ that minimizes the ratio $\mathcal{MR}(S_1, \ldots, S_k, T)$, where $T = \{t_1, \ldots, t_n\}$.*

Note that Min-RS is a generalization of the optimization version of Partition.

For the next problem, first, assume that we are given a graph $G = (V, E)$ on n vertices.

***G*-Independence Subset-Sum Ratio (*G*-IND SSR).** *Let $w : V \to \mathbb{R}^+$ be a function. We seek two nonempty and disjoint sets $S_1, S_2 \subseteq \{1, ..., n\}$ such that, the sets $V_1 = \{v_i \in V, i \in S_1\}$ and $V_2 = \{v_i \in V, i \in S_2\}$ are independent sets of G, and minimize the ratio $\mathcal{MR}(S_1, S_2, W)$, where $W = \{w_i = w(v_i) \mid v_i \in V\}$.*

Note that when the graph G is fixed, and not part of the input, then the G-IND SSR belongs in *F-SSR*. In the case where we include the graph in the input, then the problem is an optimization version of ESS with Exclusions [59]; this problem does not belong to *F-SSR*.

Remark. *If we wanted to include such problems in F-SSR, then we can define it in a slightly different way. In particular, instead of having the problems defined by the triplet $(\mathcal{I}, k, \mathcal{F})$ we can remove $\mathcal{F}$ and include a structure, $\mathcal{S}$, in the input (as in [169]). This structure is used to determine the feasible and infeasible solutions. For simplicity, we retain the definition of F-SSR as presented in the definition 3.6, however, all the theorems we present in the following sections can be modified to apply to even the most general definition of the family.*

In [98] there were introduced digraph constraints for the subset sum problem which can easily be modeled via our framework. In general, when the graph is fixed, we could also ask for S_1 and S_2 to satisfy a certain property related to other graph parameter(s), for example we may demand that the solution consists of dominating sets or matchings. In a similar manner, we may demand that the solution consists of, say, independent sets of a given matroid.

Bellow we present two problems that are spacial cases of G-IND SSR.

Two-Set Subset-Sums Ratio problem (2-Set SSR). *Let $A = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ be a set of pairs of positive numbers. We seek two nonempty and disjoint sets $S_1, S_2 \subseteq \{1, ..., n\}$ that minimize*

$$\frac{\max\{\sum_{i\in S_1} a_i, \sum_{j\in S_2} b_j\}}{\min\{\sum_{i\in S_1} a_i, \sum_{j\in S_2} b_j\}} .$$

Note that 2-Set SSR can be formulated as instance of G-IND SSR where $G = K_{n,n}$.

Factor-r Subset-Sums Ratio problem (Factor-r SSR). *Given a set $A = \{a_1, \ldots, a_n\}$ of n positive numbers and a number $r \geq 1$, find two nonempty and disjoint sets S_1, S_2 $\subseteq \{1, \ldots, n\}$ that minimize the ratio*

$$\frac{\max\{r \cdot \sum_{i\in S_1} a_i, \sum_{j\in S_2} a_j\}}{\min\{r \cdot \sum_{i\in S_1} a_i, \sum_{j\in S_2} a_j\}} .$$

The proofs that 2-Set SSR and Factor-r SSR actually do belong in F-SSR will be presented in Section 3.8.1 and Section 3.8.2 respectively. We must note that the decision version of Factor-r SSR was studied in [59]. For these two problems, we will introduce FPTAS's.

Before we continue to the next section we will present two lemmas which reveal properties under which solutions are feasible for both problems in F-SSR and *Semi-Restricted F-SSR*.

Lemma 3.4. *Let $\mathcal{P} = (\mathcal{I}, k, \mathcal{F})$ a problem in F-SSR and $\mathcal{P}' = (\mathcal{I}', k', \mathcal{F}')$ the semi restricted version of $\mathcal{P}$. If $(E, w) \in \mathcal{I}$ and $(E, w', m) \in \mathcal{I}'$ are the instances of $\mathcal{P}$ and $\mathcal{P}'$ respectively then any feasible solution $(S_1, \ldots, S_k)$ of the instance (E, w', m) of $\mathcal{P}'$ is also a feasible solution of the instance (E, w) of $\mathcal{P}$.*

Proof. The feasible solutions $(S_1, \ldots, S_k)$ of instance (E, w', m) of $\mathcal{P}'$ are those in $\mathcal{F}'(k, E, w', m)$. By the definition of $\mathcal{P}'$ in *Semi-Restricted F-SSR* we have $\mathcal{F}'(k, E, w', m) \subseteq \mathcal{F}(k, E)$ thus the lemma holds. □

Lemma 3.5. *Let $\mathcal{P} = (\mathcal{I}, k, \mathcal{F})$ a problem in F-SSR and $\mathcal{P}' = (\mathcal{I}', k', \mathcal{F}')$ the semi-restricted version of $\mathcal{P}$. If E is a set of elements and w, w' two weight functions such that:*

$$\forall i, j \in \{1, \ldots n\}, \quad w(e_i) < w(e_j) \Leftrightarrow w'(e_i) \leq w'(e_j)$$

then any feasible solution $(S_1, \ldots, S_k)$ *for the instance* (E, w) *of* $\mathcal{P}$ *is a feasible solution for the instance* (E, w', m) *of* $\mathcal{P}'$ *if*

$$w(e_m) = \min_{j \in \{1,\ldots,k\}} \{\max\{w(e_i) \mid i \in S_j\}\}$$

Proof. Let $(S_1, \ldots, S_k)$ be a feasible solution for the problem $\mathcal{P}$ with instance (E, w). This means that $(S_1, \ldots, S_k) \in \mathcal{F}(k, E)$. Assuming that:

$$w(e_m) = \min_{j \in \{1,\ldots,k\}} \{\max\{w(e_i) \mid i \in S_j\}\}$$

is easy to see that $w(e_m) \leq \max\{w(e_i) \mid i \in S_j \text{ and } j \in \{1, \ldots k\}\}$ which by the lemma assumptions gives $w'(e_m) \leq \max\{w'(e_i) \mid i \in S_j \text{ and } j \in \{1, \ldots k\}\}$. This means that $(S_1, \ldots, S_k)$ is a feasible solution for the instance (k, E, w', m) of $\mathcal{P}'$ that meets both conditions

$$(S_1, \ldots, S_k) \in \mathcal{F}(k, E)$$

and

$$w'(e_m) = \min_{j \in \{1,\ldots,k\}} \{\max\{w'(e_i) \mid i \in S_j\}\}.$$

□

3.7 A Framework Yielding FPTAS for Problems in F-SSR

In the following theorem we define a scale parameter δ which we will use later to reduce the input values, thus making the pseudopolynomial algorithms run in polynomial time. This parameter in turn, depends on the error ε and on two other parameters, namely w and m, to be chosen as explained in Theorem 3.6. By means of these parameters, we are able to present properties which, if satisfied by the output sets, guarantee that the solution is $(1 + \varepsilon)$-approximate. Note that the values of these parameters do not necessarily have to be unique; several values within certain ranges would do as long as the properties of the theorem below are satisfied.

Theorem 3.6. *Let* $A = \{a_1, ..., a_n\}$ *be a set of positive numbers,* $\varepsilon \in (0, 1)$*, two sets* $S_{1Opt}, S_{2Opt} \subseteq \{1, ..., n\}$ *and any numbers* w, m, δ *that satisfy:*

- $0 < w \leq \min\left(\sum_{i \in S_{1Opt}} a_i, \sum_{i \in S_{2Opt}} a_i\right)$
- $n \geq m \geq \max(|S_{1Opt}|, |S_{2Opt}|)$,
- $\delta = (\varepsilon w)/(3m)$

If $S_1, S_2 \subseteq \{1, ..., n\}$ *are two non-empty sets such that:*

- $w \leq \min\left(\sum_{i \in S_1} a_i, \sum_{i \in S_2} a_i\right)$
- $n \geq m \geq \max(|S_1|, |S_2|)$,

- $1 \leq \mathcal{MR}(S_1, S_2, A') \leq \mathcal{MR}(S_{1Opt}, S_{2Opt}, A')$ *with* $A' = \{\lfloor \frac{a_1}{\delta} \rfloor, ..., \lfloor \frac{a_n}{\delta} \rfloor\}$

Then the following inequality holds

$$1 \leq \mathcal{MR}(S_1, S_2, A) \leq (1 + \varepsilon) \cdot \mathcal{MR}(S_{1Opt}, S_{2Opt}, A) \ .$$

The following three lemmas will be used to prove Theorem 3.6. We will start with the following one, which relates A with A'.

Lemma 3.7. *Let a_i, a'_i and δ be defined as in Theorem 3.6, then for any $S \in \{S_{1Opt}, S_{2Opt}, S_1, S_2\}$ the following holds:*

$$\sum_{i \in S} a_i - m\delta \leq \sum_{i \in S} a'_i \delta \leq \sum_{i \in S} a_i \tag{3.1}$$

$$m\delta \leq \frac{\varepsilon}{3} \sum_{i \in S} a_i \tag{3.2}$$

Proof. To prove Eq. 3.1, notice that for all $i \in \{1, \ldots, n\}$ we define $a'_i = \lfloor \frac{a_i}{\delta} \rfloor$. This gives us

$$\frac{a_i}{\delta} - 1 \leq a'_i \leq \frac{a_i}{\delta} \Rightarrow a_i - \delta \leq \delta a'_i \leq a_i \ .$$

In addition for any $S \in \{S_{1Opt}, S_{2Opt}, S_1, S_2\}$ we have $|S| \leq m$, which means that

$$\sum_{i \in S} a_i - m\delta \leq \sum_{i \in S} a_i - |S|\delta \leq \sum_{i \in S} a'_i \delta \leq \sum_{i \in S} a_i \ .$$

As for Eq. 3.2 we have to take into account the theorem's assumptions. Specifically we know that

$$m \leq \sum_{i \in S} a_i \text{ for all } S \in \{S_{1Opt}, S_{2Opt}, S_1, S_2\}$$

which gives

$$m\delta = \frac{\varepsilon w}{3} \leq \frac{\varepsilon}{3} \sum_{i \in S} a_i$$

□

Lemma 3.8. *For sets S_1 and S_2 it holds*

$$\mathcal{MR}(S_1, S_2, A) \leq \mathcal{MR}(S_1, S_2, A') + \frac{\varepsilon}{3}$$

Proof. Without loss of generality we will assume that

$$\mathcal{MR}(S_1, S_2, A) = \mathcal{R}(S_1, S_2, A)$$

$$\begin{aligned}
\mathcal{R}(S_1, S_2, A) = \frac{\sum_{i\in S_1} a_i}{\sum_{j\in S_2} a_j} &\leq \frac{\sum_{i\in S_1} a'_i\delta + \delta m}{\sum_{j\in S_2} a_j} && \text{[by Eq. 3.1]}\\
&\leq \frac{\sum_{i\in S_1} a'_i\delta}{\sum_{j\in S_2} a'_j} + \frac{\delta m}{\sum_{j\in S_2} a_j} && \text{[by Eq. 3.1]}\\
&\leq \mathcal{MR}(S_1, S_2, A') + \frac{\varepsilon}{3} && \text{[by Eq. 3.2]}
\end{aligned}$$

□

Lemma 3.9. *For every $\varepsilon \in (0, 1)$ we have that*

$$\mathcal{MR}(S_{1Opt}, S_{2Opt}, A') \leq (1 + \varepsilon/2) \cdot \mathcal{MR}(S_{1Opt}, S_{2Opt}, A)$$

Proof. Without loss of generality we will assume that

$$\mathcal{MR}(S_{1Opt}, S_{2Opt}, A') = \mathcal{R}(S_{1Opt}, S_{2Opt}, A')$$

From Eq. 3.1 we have that

$$\begin{aligned}
\mathcal{MR}(S_{1Opt}, S_{2Opt}, A') = \frac{\sum_{i\in S_{1Opt}} a'_i}{\sum_{i\in S_{2Opt}} a'_i} &\leq \frac{\sum_{i\in S_{1Opt}} a_i}{\sum_{i\in S_{2Opt}} a_i - m\delta}\\
&= \frac{\sum_{i\in S_{1Opt}} a_i}{\sum_{i\in S_{2Opt}} a_i - m\delta} \cdot \frac{\sum_{i\in S_{2Opt}} a_i}{\sum_{i\in S_{2Opt}} a_i}\\
&= \frac{\sum_{i\in S_{2Opt}} a_i}{\sum_{i\in S_{2Opt}} a_i - m\delta} \cdot \frac{\sum_{i\in S_{1Opt}} a_i}{\sum_{i\in S_{2Opt}} a_i}\\
&= \left(1 + \frac{m\delta}{\sum_{i\in S_{2Opt}} a_i - m\delta}\right) \cdot \mathcal{R}(S_{1Opt}, S_{2Opt}, A)
\end{aligned}$$

by Eq. 3.2 it follows that

$$\begin{aligned}
\mathcal{MR}(S_{1Opt}, S_{2Opt}, A') &\leq \left(1 + \frac{1}{\frac{3}{\varepsilon} - 1}\right) \cdot \mathcal{R}(S_{1Opt}, S_{2Opt}, A)\\
&= \left(1 + \frac{\varepsilon}{3 - \varepsilon}\right) \cdot \mathcal{R}(S_{1Opt}, S_{2Opt}, A)\\
&\leq \left(1 + \frac{\varepsilon}{2}\right) \cdot \mathcal{R}(S_{1Opt}, S_{2Opt}, A) \qquad \text{[because } \varepsilon \in (0, 1)\text{]}\\
&\leq \left(1 + \frac{\varepsilon}{2}\right) \cdot \mathcal{MR}(S_{1Opt}, S_{2Opt}, A).
\end{aligned}$$

This concludes the proof.

□

Now we are ready to prove Theorem 3.6.

Proof. The theorem follows from a sequence of inequalities:

$$\begin{aligned}
\mathcal{MR}(S_1, S_2, A) &\leq \mathcal{MR}(S_1, S_2, A') + \frac{\varepsilon}{3} && \text{[by Lemma 3.8]} \\
&\leq \mathcal{MR}(S_{1Opt}, S_{2Opt}, A') + \frac{\varepsilon}{3} \\
&\leq (1 + \frac{\varepsilon}{2}) \cdot \mathcal{MR}(S_{1Opt}, S_{2Opt}, A) + \frac{\varepsilon}{3} && \text{[by Lemma 3.9]} \\
&\leq (1 + \varepsilon) \cdot \mathcal{MR}(S_{1Opt}, S_{2Opt}, A).
\end{aligned}$$

□

The next theorem presents the conditions that should be met to construct an FPTAS algorithm for a problem that belongs to *F-SSR*. Keep in mind that this framework provides a generic method to obtain an efficient approximation scheme for a problem: if one manages to prove that a problem belongs to *F-SSR* and at the same time proposes a pseudopolynomial time algorithm for its Semi-Restricted version then one obtains an FPTAS for the problem.

Theorem 3.10. *Let $\mathcal{P} = (\mathcal{I}, \mathcal{F}, \mathcal{M}, \mathcal{G})$ be a problem in F-SSR and $\mathcal{P}' = (\mathcal{I}', \mathcal{F}', \mathcal{M}, \mathcal{G})$ its corresponding problem in Semi - Restricted F-SSR. If for problem $\mathcal{P}'$ there exists an algorithm that solves exactly each instance $(A^{(m)}, m) = (\{a_1, \dots a_n\}, m) \in \mathcal{I}'$, in which all a_i values are integers, and runs in $O(n^{c_1} a_m^{c_2}))$ time, then $\mathcal{P}$ admits an FPTAS that runs in $O(n^{c_1+c_2+1}/\varepsilon^{c_2}))$ time.*

Proof. Recall that for every problem in *F-SSR*, two different instances A and A' with the same number of elements have exactly the same feasible solutions (see Remark 3.6). We need to prove that the output of Algorithm 3 is a $(1+\varepsilon)$-approximation of the optimum solution of $\mathcal{P}$ with input $A = \{a_1, \dots a_n\}$. Let $\varepsilon \in (0, 1)$, $S_1, ..., S_k$ be the sets of the optimum solution of $\mathcal{P}$ and $S_1^{(m)}, ..., S_k^{(m)}$ the solution of $\mathcal{P}'$ with input $(A^{(m)}, m) \in \mathcal{I}'$. By Lemma 3.4, we have that a feasible solution of $\mathcal{P}'$, with input $(A^{(m)}, m)$ is a feasible solution of $\mathcal{P}$ with input A. So the optimal solution $S_1^{(m)}, \dots, S_k^{(m)}$ of the $\mathcal{P}'$ with input $(A^{(m)}, m) \in \mathcal{I}'$ is a feasible solution of $\mathcal{P}$ with input A. We will also denote with $a_{n_0} \in A$ the minimum element among the maximum of the sets $S_1, \dots, S_k$ of the optimal solution, i.e.

$$a_{n_0} = \min_{j \in \{1,\dots,k\}} \left(\max_{i \in S_j} a_i \right)$$

This means that for the output of the Algorithm 3, $S_1^*, \dots, S_k^*$ we have

$$\mathcal{MR}(S_1^*, ..., S_k^*, A) \leq \mathcal{MR}(S_1^{(n_0)}, ..., S_k^{(n_0)}, A)$$

so it is sufficient to prove that

$$\mathcal{MR}(S_1^{(n_0)}, ..., S_k^{(n_0)}, A) \leq (1 + \varepsilon) \cdot \mathcal{MR}(S_1, ..., S_k, A)$$

By Lemma 3.5, the optimal solution $S_1, ..., S_k$ is a feasible solution for problem $\mathcal{P}'$ on input $(A^{(n_0)}, n_0)$ so we have

$$\mathcal{MR}(S_1^{(n_0)}, ..., S_k^{(n_0)}, A^{(n_0)}) \leq \mathcal{MR}(S_1, ..., S_k, A^{(n_0)}) \tag{3.3}$$

Without loss of generality let $\mathcal{MR}(S_1^{(n_0)}, S_2^{(n_0)}, A) = \mathcal{MR}(S_1^{(n_0)}, \ldots, S_k^{(n_0)}, A)$ and $\mathcal{MR}(S_1, S_2, A^{(n_0)}) = \mathcal{MR}(S_1, \ldots, S_k, A^{(n_0)})$. Then due to the definition of the $\mathcal{MR}$ function

$$\begin{aligned}
\mathcal{MR}(S_1^{(n_0)}, S_2^{(n_0)}, A^{(n_0)}) &\leq \mathcal{MR}(S_1^{(n_0)}, \ldots, S_k^{(n_0)}, A^{(n_0)}) \\
&\leq \mathcal{MR}(S_1, \ldots, S_k, A^{(n_0)}) && \text{[by Eq. 3.3]} \\
&= \mathcal{MR}(S_1, S_2, A^{(n_0)})
\end{aligned}$$

From the above equation and the definition of $A^{(n_0)}$ it is easy to see that the pairs of sets $(S_1^{(n_0)}, S_2^{(n_0)})$ and (S_1, S_2) satisfy the requirements of Theorem 3.6 which gives us that

$$\begin{aligned}
\mathcal{MR}(S_1^{(n_0)}, \ldots, S_k^{(n_0)}, A) &= \mathcal{MR}(S_1^{(n_0)}, S_2^{(n_0)}, A) && \text{[by assumption]} \\
&\leq (1+\varepsilon) \cdot \mathcal{MR}(S_1, S_2, A) && \text{[by Theorem 3.6]} \\
&\leq (1+\varepsilon) \cdot \mathcal{MR}(S_1, \ldots, S_k, A)\ .
\end{aligned}$$

Regarding the running time we observe that the algorithm begins with a for loop that repeats n times. In each iteration the algorithm computes $A^{(m)}$ in time $O(n)$, executes algorithm $\mathcal{SOL}(A^{(m)}, m)$ in time $O(n^{c_1} a_m'^{c_2})$ and finally it has to evaluate $\mathcal{MR}(S_1', \ldots, S_k', A)$ and $\mathcal{MR}(S_1^*, \ldots, S_k^*, A)$. This evaluation takes time $O(k^2) = O(n^2)$ due to $k \leq n$ (because $S_1, \ldots, S_k$ are disjoint). So one iteration takes time $O(\max(n^{c_1} a_m'^{c_2}, n^2))$, and Algorithm 3 takes, in total, $O(\max(n^{c_1+1} a_m'^{c_2}, n^3))$.

We will prove that the value a_m' which we use in each iteration is polynomially bounded by n and $1/\varepsilon$. Indeed, $a_m' = \lfloor a_m/\delta \rfloor = \lfloor 3n a_m/\varepsilon a_m \rfloor \leq 3n/\varepsilon$. Hence the running time is $O(poly(n, 1/\varepsilon))$ proving that Algorithm 3 is an FPTAS for problem $\mathcal{P}$. Finally, by replacing a_m' by $O(n/\varepsilon)$ the running time of the statement follows.

□

We will now present an algorithm that approximates $\mathcal{P}$ using the algorithm for $\mathcal{P}'$. We will denote the algorithm that returns the exact solution for $\mathcal{P}'$ by $\mathcal{SOL}(A, m)$.

Algorithm 3 FPTAS for the problem $\mathcal{P}$ [$\mathcal{SOL}_{apx,\mathcal{P}}(A)$ function]

Input: A set $A = \{a_1, \ldots, a_n\}, a_i \in \mathbb{R}^+$.
Output: Sets with max ratio $(1+\varepsilon)$ to the optimal max ratio for the problem $\mathcal{P}$.

```
(S*_1, ..., S*_k) ← {∅, ..., ∅}
for m ← 1 to n do
    δ ← εa_m / 3n
    A^(m) ← ∅
    for i ← 1 to n do
        a'_i ← ⌊a_i / δ⌋
        A^(m) ← A^(m) ∪ {a'_i}
    end for
    (S'_1, ..., S'_k) ← SOL(A^(m), m)
    if MR(S'_1, ..., S'_k, A) ≤ MR(S*_1, ..., S*_k, A) then
        (S*_1, ..., S*_k) ← (S'_1, ..., S'_k)
    end if
end for
return (S*_1, ..., S*_k)
```

In the next section we will give some examples of how this framework works by using Theorem 3.10 to find an FPTAS algorithm for certain problems.

3.8 2-Set SSR and Factor-r SSR

In this section we consider two variants of SSR, the 2-Set SSR and the Factor-r SSR. In particular, we use the previous presented framework in order to give an FPTAS for the first problem and then we show that the same FPTAS can be used in order to approximate the second. To the best of our knowledge, there are no previous approximation schemes for these problem. We must note that faster approximation algorithms may exist for these problems but it is not the goal of this section to find the fastest possible algorithms.

3.8.1 FPTAS for 2-Set SSR

We begin this section by showing that 2-Set SSR belongs to *F-SSR*. To do so, we match this problem with a problem $(\mathcal{I}, \mathcal{F}, \mathcal{M}, \mathcal{G})$ in *F-SSR*. If we let the set of instances $\mathcal{I}$ contain sets of positive numbers $A = \{a_1, \ldots, a_{2n}\} = \{a_1, \ldots, a_n, b_1, \ldots, b_n\}$, and the set of feasible solutions $\mathcal{F}$ contain all the pairs of sets (S_1, S_2) such that $S_1 \subseteq \{1, ..., n\}$, $S_2 \subseteq \{n+1, ..., 2n\}$, $\nexists\ (i, j)$ such that $i \in S_1, j \in S_2$ with $i \equiv j \pmod{n}$, the objective $\mathcal{M} = \mathcal{MR}(S_1, S_2, A))$ and the goal function $\mathcal{G} = \min$, then the 2-Set SSR problem coincides with $(\mathcal{I}, \mathcal{F}, \mathcal{M}, \mathcal{G})$ which is a problem in *F-SSR*.

Now we shall present a pseudopolynomial time algorithm that finds an optimal solution for *Semi-Restricted* 2-Set SSR. Our algorithm employs two separate algorithms for two different cases.

Algorithm 4 *Semi-Restricted* 2-SET SSR solution [$\mathcal{SOL}(A, m)$ function]

Input: a set $A = \{a_1, \ldots, a_{2n}\}$, $a_i \in \mathbb{Z}^+$, and an integer m, $1 \leq m \leq 2n$.
Output: the sets of an optimal solution for *Semi-Restricted* 2-SET SSR.

1: $S'_1 \leftarrow \emptyset$, $S'_2 \leftarrow \emptyset$, $S_{min} \leftarrow \emptyset$, $S_{max} \leftarrow \emptyset$
2: **if** $m \leq n$ **then**
3: $\quad (p, p') \leftarrow (0, n)$
4: **else if** $n < m \leq 2n$ **then**
5: $\quad (p, p') \leftarrow (n, 0)$
6: **end if**
7: $S_{min} \leftarrow \{i \mid i \in \{1, \ldots, n\}$ and $a_{i+p} \leq a_m\} \setminus \{m - p\}$
8: $S_{max} \leftarrow \{i \mid i \in \{1, \ldots, n\}$ and $a_{i+p'} \geq a_m\} \setminus \{m - p + p'\}$
9: **if** $S_{max} \neq \emptyset$ **then**
10: $\quad (S_1, S_2) \leftarrow \mathcal{SOL}_{Case1}(A, m, S_{min}, S_{max})$
11: $\quad (S'_1, S'_2) \leftarrow \mathcal{SOL}_{Case2}(A, m, S_{min}, S_{max})$
12: $\quad$ **if** $\mathcal{MR}(S_1, S_2, A) < \mathcal{MR}(S'_1, S'_2, A)$ **then**
13: $\quad\quad (S'_1, S'_2) \leftarrow (S_1, S_2)$
14: $\quad$ **end if**
15: **end if**
16: **return** S'_1, S'_2

We continue with the presentation of the algorithms $\mathcal{SOL}_{Case1}(A, m, S_{min}, S_{max})$ and $\mathcal{SOL}_{Case2}(A, m, S_{min}, S_{max})$. Let us first define a function that will simplify their presentation.

Larger Total Sum Tuple selection (LTST). *Given two tuples* $\vec{v_1} = (S_1, S_2, x)$ *and* $\vec{v_2} = (S'_1, S'_2, x')$ *we define the function* $\mathsf{LTST}(\vec{v_1}, \vec{v_2})$ *as follows:*

$$\mathsf{LTST}(\vec{v_1}, \vec{v_2}) = \begin{cases} \vec{v_2} & \text{if } \vec{v_1} = (\emptyset, \emptyset, 0) \text{ or } x' > x, \\ \vec{v_1} & \text{otherwise .} \end{cases}$$

We use this function to compare the sum of the sets $S_1 \cup S_2$ and $S'_1 \cup S'_2$ i.e.

$$x = \sum_{i \in S_1 \cup S_2} a_i \qquad \text{and} \qquad x' = \sum_{i \in S'_1 \cup S'_2} a_i$$

The next algorithm deals with Case 1. In Case 1 the solution contains an element of weight greater than the sum of weights of all elements of the other set. In this case the set with the largest total weight contains only one element and the other set contains all the allowed elements.

Algorithm 5 Case 1 solution [$\mathcal{SOL}_{Case1}(A, m, S_{min}, S_{max})$ function]

Input: a set $A = \{a_1, \ldots, a_{2n}\}$, $a_i \in \mathbb{Z}^+$ and an integer m, $1 \leq m \leq 2n$ and $S_{min}, S_{max} \subseteq \{1, ..., n\}$.

Output: Case 1 optimal solution for *Semi-Restricted* 2-Set SSR.

$S'_1 \leftarrow \emptyset$, $S'_2 \leftarrow \emptyset$
if $m \leq n$ **then**
    $p \leftarrow 0$, $p' \leftarrow n$
else
    $p \leftarrow n$, $p' \leftarrow 0$
end if
$Q \leftarrow a_m + \sum_{i \in S_{min}} a_{i+p}$
for all $i \in S_{max}$ **and** $a_{i+p'} > Q$ **do**
    $a \leftarrow 0$
    if $i \in S_{min}$ **then**
        $a \leftarrow a_{i+p}$
    end if
    if $a_{i+p'}/(Q-a) < \mathcal{MR}(S'_1, S'_2, A)$ **then**
        $S \leftarrow \{j + p \mid j \in S_{min} \text{ or } j = m - p\} \setminus \{i + p\}$
        $(S'_1, S'_2) \leftarrow (S, \{i + p'\})$
    end if
end for
return S'_1, S'_2

In Case 2 we consider that the largest weight element does not dominate the sum of the weights of the second set. In this case we create a 3-dimensional matrix whose first dimension represents the elements we have already used, the second represents the difference of the set sums and the third is rather technical and it used to ensure that we will not overwrite tuples that have desired properties. In the cells of the table we store the two sets of indices and the total sum of their weights. Moreover, when the third coordinate has value 1, this means that these sets could be a part of a feasible solution.

Algorithm 6 Case 2 solution [$\mathcal{SOL}_{Case2}(A, m, S_{min}, S_{max})$ function]

Input: a set $A = \{a_1, \ldots, a_{2n}\}$, $a_i \in \mathbb{Z}^+$ and an integer m, $1 \leq m \leq 2n$ and $S_{min}, S_{max} \subseteq \{1, ..., n\}$.

Output: Case 2 optimal solution for *Semi-Restricted* 2-Set SSR.

$S'_1 \leftarrow \emptyset$, $S'_2 \leftarrow \emptyset$
if $m \leq n$ **then**
    $p \leftarrow 0$, $p' \leftarrow n$
else

$p \leftarrow n$, $p' \leftarrow 0$
$Q \leftarrow a_m + \sum_{i \in S_{min}} a_{i+p}$
$T[i,d,l] \leftarrow \{\emptyset, \emptyset, 0\}$, $\forall\ (i,d,l) \in \{0,\dots,n\} \times \{-2 \cdot Q, \dots, Q\} \times \{0,1\}$
$T[0, a_m, 0] \leftarrow (\{m\}, \emptyset, a_m)$
end if
for $i \leftarrow 1$ **to** n **do**
  for all $(d,l) \in \{-2 \cdot Q, \dots, Q\} \times \{0,1\}$
    $(S_1, S_2, x) \leftarrow T[i-1, d, l]$ **do**
    $T[i,d,l] \leftarrow \mathsf{LTST}(T[i,d,l], T[i-1,d,l])$
    $d' \leftarrow d + a_{i+p}$
    if $i \in S_{min}$ **then**
      $T[i,d',l] \leftarrow \mathsf{LTST}(T[i,d',l], (S_1 \cup \{i+p\}, S_2, x + a_{i+p}))$
    end if
    $d' \leftarrow d - a_{i+p'}$
    if $i \in S_{max}$ **and** $d' \geq -2 \cdot Q$ **then**
      $T[i,d',1] \leftarrow \mathsf{LTST}(T[i,d',1], (S_1, S_2 \cup \{i+p'\}, x + a_{i+p'}))$
    else if $i \notin S_{max}$ **and** $d' \geq -2 \cdot Q$ **then**
      $T[i,d',l] \leftarrow \mathsf{LTST}(T[i,d',l], (S_1, S_2 \cup \{i+p'\}, x + a_{i+p'}))$
    end if
  end for
end for
for $d \leftarrow -2 \cdot Q$ **to** Q **do**
  $(S_1, S_2, x) \leftarrow T[n,d,1]$
  if $\mathcal{MR}(S_1, S_2, A) < \mathcal{MR}(S'_1, S'_2, A)$ **then**
    $S'_1 \leftarrow S_1$, $S'_2 \leftarrow S_2$
  end if
end for
return (S'_1, S'_2)

Theorem 3.11. *The Algorithm 4 returns an optimal solution for the semi restricted version of* 2-SET *SSR.*

In order to prove this theorem we need to notice that, for all $i \in [n]$, the algorithm produce all the differences of sums from subsets of $\{a_1, \dots, a_i, a_{n+1}, \dots, a_{n+i}\} \cup \{a_m\}$ by adding the a_i or a_{n+i} to the differences of the sums can be achieved from the set $\{a_1, \dots, a_{i-1}, a_{n+1}, \dots, a_{n+i-1}\}$ accordingly. The complete proof of this theorem is rather technical and can be found in the appendix.

Theorem 3.12. *Algorithm 4 runs in time* $O(n^2 \cdot a_m)$.

Proof. Observe that in Algorithm 4 we initialize our variables and we select S_{min} and S_{max} according to m. These selections take time $O(n)$. Then we execute Algorithm 5 and Algorithm 6. Algorithm 5 runs in $O(n)$ due to the fact that the cardinality of S_{max} can not be greater than n. Furthermore in Algorithm 6 we fill a matrix with size

$n \times 3Q \times 2$ and by using a suitable data structure, we can store the sets in time (and space) $O(1)$ per cell. This implies that Algorithm 6 runs in $O(nQ)$. Last, is easy to see that $Q = a_m + \sum_{i \in S_{min}} a_{i+p}$ and $a_i \leq a_m$ for every a_i this sum which gives us that $Q \leq na_m$. So the Algorithm 4 runs in time $O(n^2 \cdot a_m)$. □

Since Algorithm 4 is a pseudopolynomial time algorithm for the *Semi-Restricted* 2-Set SSR which solves the instances with integer values and runs in time $O(n^2 \cdot a_m)$, by using Theorem 3.10 we get that 2-Set SSR admits an $FPTAS$ that runs in time $O(n^4/\varepsilon)$. Therefore, the following theorem holds.

Theorem 3.13. *For 2-Set SSR and for every $\varepsilon \in (0, 1)$ we can find an $(1+\varepsilon)$ approximation solution in time $O(n^4/\varepsilon)$.*

3.8.2 Approximation of Factor-r SSR

In this section we use the FPTAS we design for the 2-Set SSR in order to approximate another variation of SSR, the Factor-r SSR.

Before we approximate this problem we prove that it belongs to F-SSR. We identify Factor-r SSR with a problem $(\mathcal{I}, \mathcal{F}, \mathcal{M}, \mathcal{G})$ in $F{-}SSR$, by letting the set of instances $\mathcal{I}$ contain sets of positive numbers $A = \{a_1, \ldots, a_{2n}\} = \{a_1, \ldots a_n, r \cdot a_1, \ldots, r \cdot a_n\}$ with $a_i \in \mathbb{Z}^+$ for $i \in \{1, \ldots, n\}$, $r \in \mathbb{R}$, the set of feasible solutions $\mathcal{F}$ contain all pairs of sets (S_1, S_2) such that $S_1 \subseteq \{1, ..., n\}$ and $S_2 \subseteq \{n+1, ..., 2n\}$ and $\forall\ (i, j), i \in S_1 \wedge j \in S_2 \Rightarrow i + n \neq j$, the measure be $\mathcal{M} = \mathcal{MR}(S_1, S_2, A))$, and the goal function be $\mathcal{G} = \min$.

Notice that we can modify the input of Factor-r SSR in order to solve it using 2-Set SSR. Specifically, an optimal solution for Factor-r SSR with input $(\{a_1, \ldots, a_n\}, r)$ and an optimal solution of 2-Set SSR with input $A = \{(a_1, r \cdot a_1) \ldots, (a_n, r \cdot a_n)\}$ are the same. Furthermore, when considering the above inputs, there exists a bijection from feasible solutions for 2-Set SSR to the feasible solutions of Factor-r SSR. So if we find an $(1 + \varepsilon)$ approximating solution for the 2-Set SSR problem with input $A = \{(a_1, r \cdot a_1) \ldots, (a_n, r \cdot a_n)\}$ then this is an $(1 + \varepsilon)$ approximating solution for Factor-r SSR.

3.9 Conclusions

The first part of this chapter, apart from the introduction of a new FPTAS for the Subset-Sums Ratio problem, we establish a connection between Subset Sum and approximating Subset-Sums Ratio. In particular, we showed that any improvement Subset Sum will result in an improved FPTAS for Subset-Sums Ratio. In the second half, we presented a framework that can yield FPTASs for a large spectrum of problems that have weights in their input, an optimization function and some restrictions.

As a direction for future research, we consider the use of exact Subset Sum or Partition algorithms parameterized by a *concentration* parameter β, as described in [16, 17], where they solve the decision version of Subset Sum. See also [76] for a use of this

parameter under a pseudopolynomial setting. It would be interesting to investigate whether analogous arguments could be used to solve the optimization version.

Finally, since the framework we presented for problems in $F\text{-}SSR$ requires the existence of a pseudopolynomial time algorithms, it would be interesting to study the restrictions that guarantee the existence of a such algorithms for these problems.

Chapter 4

Upper Tolerant Edge Cover

4.1 Introduction

In this chapter we define and study *tolerant edge cover* problems. An *edge cover* of a graph $G = (V, E)$ without isolated vertices is a subset of edges $S \subseteq E$ which covers all vertices of G, that is, each vertex of G is an endpoint of at least one edge in S. The *edge cover number* of a graph $G = (V, E)$, denoted by $ec(G)$, is the minimum size of an edge cover of G and it can be computed in polynomial time (see Chapter 19 in [175]). An edge cover $S \subseteq E$ is called *minimal* (with respect to inclusion) if no proper subset of S is an edge cover. Minimal edge cover is also known in the literature as an *enclaveless* set [176] or as a *nonblocker* set [71]. While a minimum edge cover can be computed efficiently, finding the *largest minimal edge cover* is NP-hard [149], where it is shown that the problem is equivalent to finding a dominating set of minimum size. The associated optimization problem is called *upper edge cover* (and denoted Upper EC) [13] and the corresponding optimal value, for a graph $G = (V, E)$, will be denoted by $\mathsf{uec}(\mathsf{G})$.

Here, we are interested in minimal edge cover solutions tolerant to the failures of at most $r - 1$ edges. Formally, given an integer $r \geq 1$, an edge subset $S \subseteq E$ of $G = (V, E)$ is a *tight r-tolerant edge-cover* (r-tec for short) if the deletion of any set of at most $r - 1$ edges from S maintains an edge cover[1] and the deletion of any edge from S yields a set which is not a (tight) r-tolerant edge cover. Equivalently, we seek an edge subset S of G such that the subgraph (V, S) has minimum degree r and it is minimal with this property. For the sake of brevity we will omit the word 'tight' in the rest of the chapter. Note that the case $r = 1$ corresponds to the standard notion of minimal edge cover.

As an illustrating example consider the situation in which the mayor of a big city seeks to hire a number of guards, from a security company, who will be constantly patrolling streets between important buildings. An r-tolerant edge cover reflects the desire of the mayor to guarantee that the security is not compromised even if $r-1$ guards are attacked. Providing a maximum cover would be the goal of a selfish security company, who would

[1] It might be more intuitive to call such an edge cover $(r-1)$-tolerant, but for simplicity and due to the fact that each vertex has at least degree r in the cover, we have chosen to use the term r-tolerant.

like to propose a patrolling schedule with as many guards as possible, but in which all the proposed guards are necessary in the sense that removing any of them would leave some building not r-covered.

Related Work UPPER EC has been investigated intensively during recent years, mainly using the terminologies of *spanning star forests* and *dominating sets*. A *dominating set* in a graph is a subset S of vertices such that any vertex not in S has at least one neighbor in S. The *minimum dominating set problem* (denoted MINDS) seeks the smallest dominating set of G of value $\gamma(G)$. We have the equality $\mathsf{uec}(\mathsf{G}) = \mathsf{n} - \gamma(\mathsf{G})$ [149].

Thus, using the complexity results known for MINDS, we deduce that UPPER EDGE COVER is NP-hard in planar graphs of maximum degree 3 [94], chordal graphs [39] (even in *undirected path graphs*, the class of vertex intersection graphs of a collection of paths in a tree), bipartite graphs, split graphs [30] and k-trees with arbitrary k [62], and it is *polynomial* in k-trees with fixed k, convex bipartite graphs [70], strongly chordal graphs [81]. Concerning the approximability, an APX-hardness proof with explicit inapproximability bound and a combinatorial 0.6-approximation algorithm is proposed in [165]. Better algorithms with approximation ratio 0.71 and 0.803 are given respectively in [53] and [15]. For any $\varepsilon > 0$, UPPER EDGE COVER is hard to approximate within a factor of $\frac{259}{260} + \varepsilon$ unless P=NP [165]. The weighted version of the problem, denoted as UPPER WEIGHTED EDGE COVER, have been recently studied in [130], in which it is proved that the problem is not $O(\frac{1}{n^{1/2-\varepsilon}})$ approximable nor $O(\frac{1}{\Delta^{1-\varepsilon}})$ in edge weighted graphs of order n and maximum degree Δ.

Related notions of dominating sets are introduced in the literature under the name *r-tuple domination* [18, 85, 91, 101, 107, 133], and *r-domination* [51, 85]. A set $S \subseteq V$ is called a *r-tuple dominating set* of $G = (V, E)$ if for every vertex $v \in V$, $|N_G[v] \cap S| \geq r$. The minimum cardinality of a r-tuple dominating set of G is called *r-tuple domination number* and usually denoted by $\gamma_{\times r}(G)$. The case $r = 2$ is often called *the double domination number* [101]. Complexity and approximation results on $\gamma_{\times r}(G)$ are given in [18, 133, 142] where it is proved that for any $r \geq 2$ fixed the problem is APX-complete in graphs of maximum degree $r + 2$ [133], NP-hard in split graphs and bipartite graphs for any $r \geq 1$ [142] and it admits a PTAS for unit disk graphs [18]. Finally, *the upper r-tuple domination number*[2] of a graph G has been recently investigated in [48] where an upper bound on this quantity for regular graphs is presented, with together a characterization of extremal graphs achieving this upper bound depending on parameter r. The particular case $r = 1$, corresponding to the *upper domination number* (denoted $\mathsf{uds}(\mathsf{G})$ here but also known as $\Gamma(G)$), has been proved NP-hard in [55] and extensively studied from complexity and approximability point of view in [22].

Our Contribution In Section 4.3 we present several properties of *r-tec* solutions and of the $\mathsf{ec_r}$ and $\mathsf{uec_r}$ values in graphs. Furthermore, we give a characterization of *r-tec*

[2]The maximum cardinality of a minimal r-tuple dominating set of G.

solutions based on the $\gamma(G)$ and $\gamma_2(G)$ values of graphs, where we have equality between uec and uec_2 values.

In Section 4.4, we provide several complexity results. More specifically, for the Double Upper EC we show that it is NP-hard in cubic planar graphs and split graphs and for the general Upper r-EC we show that it is NP-hard in r-regular bipartite graphs. Furthermore, we show that the Upper $(r+1)$-EC is NP-hard in graphs with maximum degree $\Delta+1$ if the Upper r-EC is NP-hard in graphs with maximum degree Δ (and this holds even for bipartite graphs).

In Section 4.5 we present some inapproximability results starting by proving that, unless P=NP, Upper EC is not approximable within $\frac{593}{594}$ in graphs of max degree 4 and $\frac{363}{364}$ in graphs of max degree 5 (the previous known result was for graphs with maximum degree 6). Furthermore, we present the first inaproximability results for the Double Upper EC for graphs with maximum degree 6 and 9.

4.2 Preliminaries

Let $G=(V,E)$ be a graph where the minimum degree is at least $r \geq 1$, i.e., $\delta(G) \geq r$. We assume r is a constant fixed greater than one (but all results given here hold even if r depends on the graph). A r-degree edge-cover [3] is defined as a subset of edges $G'=G_S=(V,S)$, such that each vertex of G is incident to at least $r \geq 1$ distinct edges $e \in S$. As r-tolerant edge-cover (or simply r-tec) we will call an edge set $S \subseteq E$ if it is a *minimal r-degree edge-cover* i.e. if for every $e \in S$, $G'-e=(V,S\setminus\{e\})$ is not an r-degree edge-cover. Alternatively, $\delta(G')=r$, and $\delta(G'-e)=r-1$. If you seek the minimization version, all the problems are polynomial-time solvable. In particular, the case of $r=1$ corresponds to the *edge cover* in graphs. The optimization version of a generalization of r-EC known as the Min lower-upper-cover problem (MinLUCP), consists of, given a graph G where $G=(V,E)$ and two non-negative functions a,b from V such that $\forall v \in V$, $0 \leq a(v) \leq b(v) \leq d_G(v)$, of finding a subset $M \subseteq E$ such that the partial graph $G_M=(V,M)$ induced by M satisfies $a(v) \leq d_{G_M}(v) \leq b(v)$ (such a solution will be called a *lower-upper-cover*) and minimizing its total size $|M|$ among all such solutions (if any). Hence, an r-EC solution corresponds to a lower-upper-cover with $a(v)=r$ and $b(v)=d_G(v)$ for every $v \in V$. MinLUCP is known to be solvable in polynomial time even for edge-weighted graphs (Theorem 35.2 in Chapter 35 of Volume A in [175]). We are considering two associated problems, formally described as follows.

r-EC
Input: A graph $G=(V,E)$ of minimum degree r.
Solution: r-tec $S \subseteq E$ of G.
Output: Minimize $\|S\|$.

[3] A different generalization of edge cover was considered in [83], requiring that each connected component induced by the edge cover solution contains at least r edges. Clearly, if every vertex is incident to at least r edges from the cover, then each connected component induced by the edge cover solution contains at least r edges.

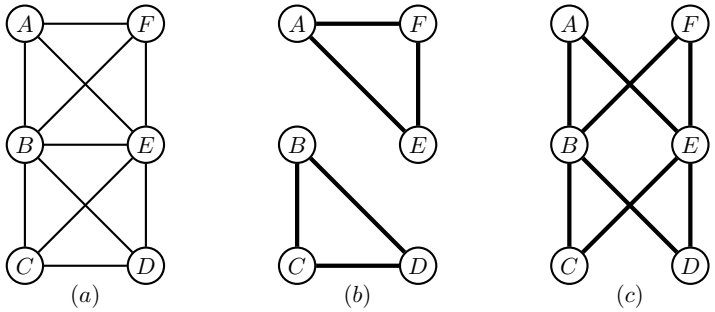


Figure 4.1: Graph $G = (V, E)$ with 6 vertices and 11 edges is shown in (a); (b) and (c) give a solution for 2-EC and DOUBLE UPPER EC of size 6 and 8 respectively.

UPPER r-EC **Input:** A graph $G = (V, E)$ of minimum degree r. **Solution:** r-tec $S \subseteq E$ of G. **Output:** Maximize $\|S\|$.

For a graph G, the optimal values of r-EC and UPPER r-EC will be denoted by $\mathsf{ec_r(G)}$ and $\mathsf{uec_r(G)}$ respectively. In particular, $\mathsf{ec_1(G)} = \mathsf{ec(G)}$ and $\mathsf{uec_1(G)} = \mathsf{uec(G)}$. As indicated above, $\mathsf{ec_r(G)}$ can be computed in polynomial-time. In Fig 4.1, we illustrate the difference between the problems r-EC and UPPER r-EC for $r = 2$. Note that throughout the chapter we will also use the term DOUBLE UPPER EC to refer to UPPER 2-EC.

An *edge dominating set* $S \subseteq E$ of a simple graph $G = (V, E)$ is a subset S of edges such that for any edge $e \in E$ of G, at least one edge of S is incident to e. The EDGE DOMINATING SET problem (EDS in short) consists in finding an edge dominating set of minimum size; the optimal value of an edge dominating set is usually denoted $\mathsf{eds(G)}$. EDS is known to be **NP**-hard in general graphs (problem [GT2] in [94]). It is well known that the problem is equivalent to solve the problem consisting of finding a maximal matching of G with minimum size. According to standard terminology, this problem is also called *lower matching* (LOWER EM in short).

4.3 Basic properties of r-tolerant solutions

The next property presents a simple characterization of feasible r-tec solution generalizing the well known result given for minimal edge covers, i.e., 1-tec, affirming that S is a 1-tec solution of G if and only if S is spanning and the subgraph (V, S) induced by S is (K_3, P_4)-free.

Property 4.1. *Let $r \geq 1$ and let $G = (V, E)$ be a graph with minimum degree $\delta \geq r$. S is an r-tec solution of G if and only if the following conditions meet on $G_S = (V, S)$:*

(1) $V = V_1(S) \cup V_2(S)$ *where* $V_1(S) = \{v \in V : \ d_{G_S}(v) = r\}$ *and* $V_2(S) = \{v \in V : \ d_{G_S}(v) > r\}$.

(2) $V_2(S)$ *is an independent set of* G_S.

Proof. Let $r \geq 1$ be a fixed integer and let $G = (V, E)$ be an instance of Upper r-EC, i.e., a graph of minimum degree at least r. Let us prove the necessary conditions: if $S \subseteq E$ is an r-tec solution, then by construction, $V = V_1(S) \cup V_2(S)$ is a partition of vertices with minimum degree r in S. Now, if $uv \in S$ with $u, v \in V_2(S)$, then $S - uv$ is also r-tec which is a contradiction of minimality.

Now, let us prove the other direction. Consider a subgraph $G' = (V, S)$ induced by edge set S satisfying (1) and (2). By (1) it is clear G_S has minimum degree at least r. If $uv \in S$, then by (2) one vertex, say $u \in V_1(S)$ because $V_2(S)$ is an independent set. Hence, the deletion of uv leaves u of degree $r - 1$ in the subgraph induced by $G_{S\setminus\{uv\}}$ and then S is an r-tec solution. □

Property 4.2. *Let $r \geq 1$, for all graphs $G = (V, E)$ of minimum degree at least r, the following inequality holds:*

$$2\mathsf{ec_r(G)} \geq \mathsf{uec_r(G)} \tag{4.1}$$

Proof. For a given graph $G = (V, E)$ with n vertices, let S^* be an optimal solution of Upper r-EC, that is $|S^*| = \mathsf{uec_r(G)}$. Let (V_1^*, V_2^*) be the associated partition related to solution S^* as indicated in Property 4.1. Using this characterization, we deduce $\mathsf{uec_r(G)} \leq \mathsf{r}|\mathsf{V_1^*}| \leq \mathsf{rn}$. On the other side, if G' denotes the subgraph induced by a minimum r-tec solution of value $\mathsf{ec_r(G)}$, we get $2\mathsf{ec_r(G)} = \sum_{\mathsf{v}\in\mathsf{V}} \mathsf{d_{G'}(v)} \geq \mathsf{rn}$. Combining these two inequalities, the results follows. □

In particular, inequality (4.1) of Property 4.2 shows that any r-tec solution is a $\frac{1}{2}$-approximation of Upper r-EC.
The next property is quite natural for induced subgraphs and indicates that the size of an optimal solution of a maximization problem does not decrease with the size of the graph. Nevertheless, this property is false in general when we deal with partial subgraphs; for instance, for the upper domination number, we get $\mathsf{uds(K_3)} = 1 < 2 = \mathsf{uds(P_3)}$. It turns out that this inequality is valid for the upper edge cover number.

Property 4.3. *Let $G = (V, E)$ be a graph such that $0 < r \leq \delta(G)$. For every partial subgraph $G' \subseteq G$ with $\delta(G') \geq r$, the following inequality holds:*

$$\mathsf{uec_r(G)} \geq \mathsf{uec_r(G')} \tag{4.2}$$

Proof. Fix an integer $r \geq 1$ and a graph $G = (V, E)$ with $\delta(G) \geq r$. Let $G' = (V', E')$ with $\delta(G') \geq r$ be a partial subgraph of G, i.e., $V' \subseteq V$ and $E' \subseteq E$. Consider an upper r-tec solution S' of G' with size $|S'| = \mathsf{uec_r(G')}$. We prove inequality (4.2) by starting from $S = S'$ and by iteratively repeating the following procedure:

1. Select a vertex $v \in V$ with $d_{G_S}(v) < r$ and $e = uv \in E \setminus S$.
2. If u is covered less or more than r times by S, then $S := S + e$.
3. If vertex u is covered exactly r times by S, consider two cases:
   (a) If every vertex $u' \in N_{G_S}(u)$ has degree $d_{G_S}(u') \leq r$, then $S := S + e$.
   (b) Otherwise there exists a vertex $u' \in N_{G_S}(u)$ with $d_{G_S}(u') > r$ then $S := S + e - u'u$.

We repeat this process until every vertex of G is covered at least r times. Obviously, this algorithm terminates because $\sum_{v \notin V_2(S)} d_{G_S}(v)$ increase at each iteration where we recall $V_2(S) = \{v \in V : d_{G_S}(v) > r\}$. Since at each step, the size of S never decreases, we only need to show that we end the process with a r-tec solution of the whole graph. In order to prove that, we will show by induction that $V_2(S)$ is an independent set of G_S for each iteration. Using Property 4.1, we get an r-tec solution because $V_1(S) = V \setminus V_2(S)$. Initially $V_2(S) = V_2(S')$ is an independent set of G_S because S' is a r-tec solution of G'. Assume it is true for iteration k and consider iteration $k+1$. During step 2, $V_2(S)$ remains unchanged in both cases $d_{G_S}(u) < r$ or $d_{G_S}(u) > r$. So by induction the result is valid. During step 3.(a), $V_2(S) := V_2(S) + u$, but $V_2(S) \cap N_{G_S}(u) = \emptyset$; so $V_2(S)$ remains an independent set of G_S. During step 3.(b), $V_2(S)$ remains unchanged if $d_{G_S}(u') > r + 1$, otherwise $V_2(S) := V_2(S) - u'$. Thus, $V_2(S)$ remains an independent set of G_S by induction. In conclusion, $\mathsf{uec_r(G)} \geq |\mathsf{S}| \geq |\mathsf{S'}| = \mathsf{uec_r(G')}$ and the property holds. □

We can prove a similar property for the size of a partial solution and the upper edge cover number. More specifically, in a graph $G = (V, E)$ a partial r-tec defined as a set $S \subseteq E$ such that the set $V_2(S) = \{v \in V \mid d_{G_S}(v) > r\}$ is an independent set of G_S.

Property 4.4. *Let $G = (V, E)$ be a graph such that $0 < r \leq \delta(G)$. If $E' \subseteq E$ is a partial r-tec of G then the following inequality holds:*

$$\mathsf{uec_r(G)} \geq |\mathsf{E'}| \tag{4.3}$$

The proof of the above property is similar to Property's 4.3 except that we will start with $S := E'$.

Property 4.5. *Let $G = (V, E)$ and $E' \subseteq E$ be a solution for Upper r-EC of G then for any $1 \leq m < r$ we can find a set $E'' \subseteq E'$ such that E'' is a m-tec of G. Especially, if $m = r - 1$ then $E' \setminus E''$ is a tec of $G[V(E' \setminus E'')]$.*

Proof. It is easy to find a m-tec of G that is a subset of E' if we start from the graph $G_{E'}$ which has minimum degree r. Any m-tec of $G_{E'}$ is a m-tec of G. Now, if $m = r - 1$ and $G' = G[V(E' \setminus E'')]$ we will prove that $E' \setminus E''$ is a tec of G'. In order to do this, we need to show that the set $V_2 = \{v \in V \mid d_{G'_{E' \setminus E''}}(v) > 1\}$ is an independent set in $G'_{E' \setminus E''}$. Let $u, v \in V_2$ be two vertices which are connected in $G'_{E' \setminus E''}$ and they have degree greater

than one. Then, because E'' is a $(r-1)$-tec of G, we know that the vertices u and v are covered $r-1$ so in E' are covered at least $r+1$ times and they are connected. This is a contradiction because E' is an a solution for Upper r-EC. So, if $m = r-1$ then $E' \setminus E''$ is a tec of $G[V(E' \setminus E'')]$.

□

Property 4.6. *Let $r \geq 1$. For all graphs $G = (V, E)$ of minimum degree at least r, the following inequality holds:*

$$\mathsf{uec_r(G)} \leq \mathsf{r} \cdot \mathsf{uec(G)} \tag{4.4}$$

Proof. Let $E_1 \subseteq E$ and $E_r \subseteq E$ be the solutions of Upper EC and Upper r-EC. By Property 4.5 we can have a set $E' \subseteq E$ such that E' is a $(r-1)$-tec and $E_r \setminus E'$ is a TEC of $G[V(E_r \setminus E')]$. Then by Property 4.3 and because E' is a $(r-1)$-tec we have that:

$$\mathsf{uec_r(G)} = |\mathsf{E'}| + |\mathsf{E_r} \setminus \mathsf{E'}| \leq \mathsf{uec_{r-1}(G)} + \mathsf{uec(G)} \tag{4.5}$$

Therefore, by induction to r we will show that $\mathsf{uec_r(G)} \leq \mathsf{r} \cdot \mathsf{uec(G)}$.

- Induction base: for $r = 2$ we have that $\mathsf{uec_2(G)} \leq \mathsf{2uec(G)}$.
- Induction hypothesis: for $0 < r' < \delta(G)$ we have $\mathsf{uec_{r'}(G)} \leq \mathsf{r'} \cdot \mathsf{uec(G)}$.
- Induction step: we will show that $\mathsf{uec_{r'+1}(G)} \leq (\mathsf{r'}+1) \cdot \mathsf{uec(G)}$.

The induction step can be proved using the inequality 4.5 and the induction hypothesis: $\mathsf{uec_{r'+1}(G)} \leq \mathsf{uec_{r'}(G)} + \mathsf{uec(G)} \leq \mathsf{r'} \cdot \mathsf{uec(G)} + \mathsf{uec(G)} = (\mathsf{r'}+1) \cdot \mathsf{uec(G)}$.

□

A well known relation between the domination number and the upper edge cover number for any graph G with n vertices is the following:

$$\mathsf{uec(G)} = \mathsf{n} - \gamma(\mathsf{G}).$$

The previous equation cannot be generalized for the r-domination number and the r upper edge cover number in any graph. However, we can prove the next relation.

Property 4.7. *Let $r \geq 1$. For all graphs $G = (V, E)$ with n vertices and minimum degree at least r. The next inequality between $\mathsf{uec_r(G)}$ and the r-domination number holds:*

$$\mathsf{uec_r(G)} \geq \mathsf{r}(\mathsf{n} - \gamma_\mathsf{r}(\mathsf{G}))$$

Proof. In order to prove this we will start from a minimum r-dominating set S and we will construct a partial r-tec of G. S dominates all the vertices in $V \setminus S$ at least r times so for each $v \in V \setminus S$ we can select $u_1, \ldots, u_r \in S$ such that $vu_i \in E$, $\forall i = 1, \ldots, r$ and we construct the set $E_v = \{vu_i \mid i = 1, \ldots, r\}$. We will prove that the set $E' = \bigcup_{v \in V \setminus S} E_v$ is a partial r-tec. Recall that a subset E' of edges is a partial r-tec if in the graph $G_{E'}$

the set $V_2 = \{v \in V \mid d_{G_{E'}} > r\}$ is an independent set. This property holds in our set because we selected the edges in order to be incident to at least one vertex of degree exactly r. So, by the Property 4.4 we have that $\mathsf{uec_r(G)} \geq |\mathsf{E'}|$. As for the size of E' we observe that for each vertex $v \in V \setminus S$ we have used exactly r edges that are incident to a vertex of S and both of those sets ($V \setminus S$ and S) are independent in $G_{E'}$. So the size of E' is equal to $r(n - |S|)$ which implies:

$$\mathsf{uec_r(G)} \geq |\mathsf{E'}| = \mathsf{r(n - \gamma_r(G))}$$

.

□

In the next lemma we are interested in graphs where the equality between uec and $\mathsf{uec_r}$ holds.

Lemma 4.8. *Let $G = (V, E)$ be a graph such that $1 < r \leq \delta(G)$, then the following are equivalent:*

1. $\gamma_r(G) = \gamma(G)$
2. $\mathsf{uec_r(G)} = \mathsf{r} \cdot \mathsf{uec(G)}$

Proof. Starting from a graph G were $\gamma_r(G) = \gamma(G)$ we will prove that $\mathsf{uec_r(G)} = \mathsf{r \cdot uec(G)}$. In this case we have that:

$$\begin{aligned} r \cdot \mathsf{uec(G)} &= r \cdot (n - \gamma(G)) = r \cdot (n - \gamma_r(G)) \\ &\leq \mathsf{uec_r} && \text{(by Prop. 4.7)} \end{aligned}$$

In addition, we have that $r \cdot \mathsf{uec(G)} \geq \mathsf{uec_r(G)}$ (by Prop. 4.6) so we proved that:

$$\gamma_r(G) = \gamma(G) \Longrightarrow \mathsf{uec_r} = \mathsf{r} \cdot \mathsf{uec}.$$

For the converse, let $G = (V, E)$ be a graph such that $\mathsf{uec_r} = \mathsf{r} \cdot \mathsf{uec}$, E_r be a maximum r-tec and E_1 be a maximum tec. First suppose that there exists a partition of the vertices $V_1' \cup V_2' = V$ such that for all $v \in V_1'$ we have $d_{G_{E_r}}(v) = r$ and both of V_1', V_2' are independent in G_{E_r}. It is easy to see that V_2' is a r-dominating set of G because V_1' is independent in G_{E_r} so each vertex of it is dominated at least r times by the vertices in V_2'. Therefore, because V_1', V_2' are independent in G_{E_r} and for all $v \in V_1'$ we have $d_{G_{E_r}}(v) = r$ we can conclude that $|E_r| = r \cdot |V_1'|$. This gives us the following:

$$\mathsf{uec_r(G)} = |\mathsf{E_r}| = \mathsf{r} \cdot |\mathsf{V_1'}| = \mathsf{r(n - |V_2'|)} \leq \mathsf{r(n - \gamma_r(G))}.$$

Because in Prop.4.7 we showed that $\mathsf{uec_r(G)} \geq \mathsf{r(n - \gamma_r(G))}$ we have that:

$$r(n - \gamma_r(G)) = \mathsf{uec_r(G) r \cdot uec} = \mathsf{r(n - \gamma(G))}$$

which implies that:

$$\mathsf{uec_r} = \mathsf{r \cdot uec} \Rightarrow \gamma_\mathsf{r}(\mathsf{G}) = \gamma(\mathsf{G}).$$

Thus, in order to complete the proof we have to show that when the values uec and $\mathsf{uec_r}$ of a graph are equal then we always have a partition as the one described in the above assumption. Assuming that there is no such partition then for any partition V_1', V_2' such that V_2' is independent, all the vertices $v \in V_1'$ have degree $d_{G_{E_r}}(v) = r$ and V_1' cannot be independent. First we will show that, for the previous partition, the following property holds:

$$\text{if } v \in V_1' \text{ then } N_{G_{E_r}}(v) \nsubseteq V_1'.$$

Starting from $V_1' := V_1(E_r)$ and $V_2' := V_2(E_r)$ (see Prop. 4.1) we have to remove vertices $u \in V_1'$ such that $N_{G_{E_r}}(v) \subseteq V_1'$ until we have the desired condition. So, we repeat the following:

1. Select a vertex $u \in V_1'$ such that $N_{G_{E_r}}(u) \subseteq V_1'$.
2. Set $V_1' := V_1' - u$ and $V_2' := V_2' + u$.

We have to show that the previous process terminates when the desired properties hold for V_1' and V_2'. It is easy to see this holds because whenever we remove a $u \in V_1$ vertex such that $N_{G_{E_r}}(u) \subseteq V_1'$ from it, we reduce, at least by one, the number of vertices with this property in V_1'. Therefore, the remaining vertices $u \in V_1'$ have $d_{G_{E_r}}(u) = r$ and the vertices in V_2' are independent in G_{E_r}.

Now, due to the assumption, we know that V_1' cannot be independent (in G_{E_r}) and for all $u \in V_1'$ we have $d_{G_{E_r}}(u) = r$. This observation combined with the fact that V_2' is independent in G_{E_r} gives $r \cdot |V_1'| > E_r$. Furthermore, because each vertex in V_1' has at least one neighbor in V_2' we can select one edge for each vertex of V_1' adjacent to a vertex in V_2'. The set which consists of this edges is a partial tec of size $|V_1'|$ so by the Prop. 4.4 we have $\mathsf{uec} \geq |\mathsf{V_1'}|$. This gives us:

$$E_r < r \cdot |V_1'| \leq r \cdot \mathsf{uec}$$

which is a contradiction because we have $\mathsf{uec_r} = \mathsf{r} \cdot \mathsf{uec}$. □

A known relation between the domination number and the r-domination number is:

$$\gamma_r(G) \geq \gamma(G) + r - 2$$

so, by the above relation and the lemma 4.10 we can conclude the following:

Corollary 4.9. *There is no graph G such that $uec_r = r \cdot uec(G)$ for any $r \geq 3$.*

Now, we will present some graphs where the equation $\mathsf{uec_2} = \mathsf{2uec}$ holds. It is easy to see that the smallest graph in which this relation holds is C_4. Conversely, we have the following lemma.

Lemma 4.10. *For any graph $G = (V, E)$ with $|V| = n \geq 4$ if $\mathsf{uec_2(G)} = \mathsf{2uec(G)}$ then $|E(G)| \leq \frac{n(n-2)}{2}$ when n is even and $|E(G)| \leq \frac{n(n-2)-1}{2}$ when n is odd.*

Proof. By the lemma 4.8 we know that the eq. $\mathsf{uec_2(G)} = \mathsf{2uec(G)}$ is equivalent to $\gamma_2(G) = \gamma(G)$. It is obvious that in order to dominate twice every vertex we need at least two vertices, therefore $\gamma_2(G) \geq 2$. So, we need to show that if n is even (resp. odd) we have to remove from a clique at least $\frac{n}{2}$ edges (resp. $\frac{n+1}{2}$ edges) in order the domination number to be at least two.

If the domination number is at least two we know that for any vertex $v \in V$ there exists another vertex $u \in V$ such that $uv \notin E$. This means that every vertex have degree at most $n-2$, so:

$$|E| = \sum_{v \in V} d(v) \leq \frac{n(n-2)}{2}.$$

This completes the case when n is even. In the case when n is odd the previous bound is not an integer so the real bound is the greatest integer which is smaller than $\frac{n(n-2)}{2}$, so $\frac{n(n-2)-1}{2}$. □

We will show that the previous upper bounds are tight. Let $G = (V, E) = K_n$ be a clique of size n. First we construct a maximum matching $M \subseteq E$ of G; if n is even then the graph $G_{\overline{M}}$ has $\gamma_2(G_{\overline{M}}) = \gamma(G_{\overline{M}})$ and the number of edges in $G_{\overline{M}}$ is equal to $\frac{n(n-1)}{2} - |M| = \frac{n(n-1)}{2} - \frac{n}{2} = \frac{n(n-2)}{2}$. In the case when n is odd any maximum matching covers $n-1$ vertices. Let $v \in V$ be the vertex that is not covered by the matching M. Then the only dominating set of $G_{\overline{M}}$ of size one is the $\{v\}$. Therefore if we remove any edge incident to v, let $uv \in E$ be that edge, then in $G_{\overline{M-uv}}$ we have $\gamma_2(G_{\overline{M-uv}}) = \gamma(G_{\overline{M-uv}})$ and the number of edges in $G_{\overline{M-uv}}$ are equal to $\frac{n(n-1)}{2} - |M| - 1 = \frac{n(n-1)}{2} - \frac{n+1}{2} = \frac{n(n-2)-1}{2}$.

4.4 Complexity results

In this section we provide several complexity results for the Double Upper EC and the Upper r-EC in some graph classes as regular graphs and split graphs.

Theorem 4.11. *Let $G = (V, E)$ be an $(r+1)$-regular graph with $r \geq 2$. Then,*

$$\mathsf{uec_r(G)} = |\mathsf{E}| - \mathsf{eds(G)}. \tag{4.6}$$

Proof. In order to prove this equation, first we will show that if $S \subseteq E$ is a r-tec of G, then $\overline{S} = E \setminus S$ is an edge dominating set of G. Let (V_1, V_2) be the associated partition related to S. By the Property 4.1 we know that V_2 is an independent set. Because our graph is $(r+1)$-regular, it is easy to see that $\forall v \in V_1$, $d_{G_S}(v) = r$ and $\forall u \in V_2$, $d_{G_S}(u) = r+1$. This observation gives us that the set $\overline{S}$ covers each vertex of V_1 (they have degree r) and that all the edges in E are incident to a vertex in V_1 (because V_2 is an independent set). So, $\overline{S}$ is an edge dominating set.

Conversely, let S be a solution of EDS. We will show that there exists a r-tec of size $|\overline{S}|$. Because EDS is equivalent to Lower EM, we consider the edge set S as a maximal

matching. Now, let $V' = V \setminus V[S]$. Observe that V' is an independent set in G and each vertex $v \in V'$ has $d_{G_{\overline{S}}}(v) = r + 1$. The first holds because if there exists an edge e between two vertices of V' then $S \cup \{e\}$ will be a matching with size greater than S, which contradicts the maximality of S. The second holds because the edges in S are not incident to vertices of V' by definition, and thus, all the edges in $\overline{S}$ are incident to at least one vertex in $V[S]$. Finally, because S is a matching we have that all the vertices in $V[S]$ have degree r in $G_{\overline{S}}$ so by the property 4.1, $\overline{S}$ is a r-tec. So the Eq. (4.6) holds. □

Corollary 4.12. *Upper r-EC is NP-hard in $(r+1)$-regular bipartite graphs.*

Proof. Using the NP-hardness proof of EDS in r-regular bipartite graphs given in [73], the results follows from Theorem 4.11. □

Corollary 4.13. *Double Upper EC is NP-hard in cubic planar graphs.*

Proof. Using the NP-hardness proof of EDS for cubic planar graphs given in [116], the results follows from Theorem 4.11. □

Theorem 4.14. *Upper $(r+1)$-EC is NP-hard in graphs of maximum degree $\Delta + 1$ if Upper r-EC is NP-hard in graphs of maximum degree Δ, and this holds even for bipartite graphs.*

Proof. Let $G = (V, E)$ be a bipartite graph of max degree Δ. For our reduction we will construct a new graph $G' = (V', E')$ by starting from $r + 1$ copies of the graph G. Let $G_i = (V_i, E_i)$, $i = 1, \ldots, r + 1$ be the copies G. Then for each vertex $v \in V$ we add a new vertex u_v in G' and we connect it with each one of the $r + 1$ copies, $v_1, v_2, \ldots, v_{r+1}$, of the vertex v. It is obvious by the construction that the new graph remains bipartite (see Fig. 4.2).

Now, we claim that the relation between $\mathsf{uec_r(G)}$ and $\mathsf{uec_{r+1}(G')}$ is the following:

$$\mathsf{uec_{r+1}(G') = (r+1)(n + uec_r(G))} \tag{4.7}$$

In order to prove this first we have to observe that all the vertices of the set $U = \{u_v \mid v \in V\} \subset V'$ have degree exactly $r + 1$ in G' (by construction) so we need all the edges that are incident to them in any $(r + 1)$-tec of the G'. Furthermore, because U is independent and has size equal to V we have that the number of the edges that are incident to U is $n(r + 1)$. Let E_U be the set of those edges.

Now we will show that from an upper r edge cover S of G we can construct an $(r+1)$-tec of G' with size $\mathsf{uec_{r+1}(G') = (r+1)(n+uec_r(G))}$. If in each G_i we select the edges that are copies of those in S (let us call S_i this sets) then we claim that $S' = \bigcup_{i=1}^{k+1} S_i \cup E_U$ is an $(r + 1)$-tec of G'. Is easy to see that this set covers each vertex in U exactly $r + 1$ times by the set E_U and each vertex $v_i \in V_i$ at least r time from the S_i and exactly once from the edge $v_i u_v \in E_U$. It remains to show that the set $V_2'(S')$ is an independent set of $G'_{S'}$ (see property 4.1). The only vertices in $V_2'(S')$ are the one that had degree greater than r in each (V_i, S_i) (all the other vertices have degree $r + 1$ in $G'_{S'}$) but this vertices must be independent because either the belong to different copies of G or, if they belong

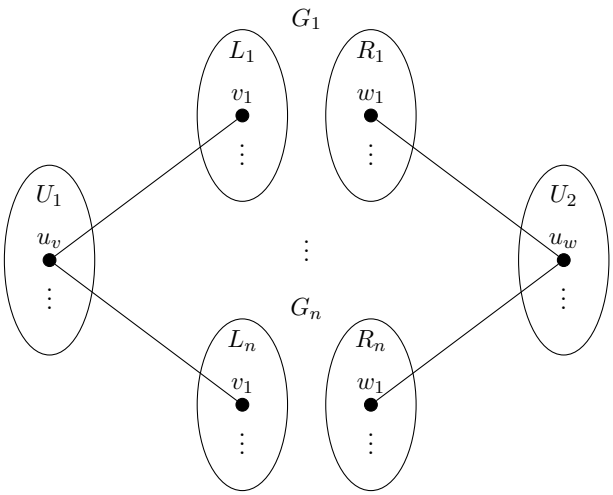


Figure 4.2: If we call R_i and L_i the right and left parts of the bipartite graph G_i for all $i = 1, \ldots, n$ then we can observe that the vertices of U ($U = U_1 \cup U_2$) been connected either only to the right or only to the left vertices. So the new graph is bipartite

to the same copy, they belong to the independent set of an upper r edge cover. So we have that:

$$\mathsf{uec_{r+1}(G')} \geq \mathsf{S'} = \mathsf{(r+1)(n + uec_r(G))}$$

Now we will prove the converse. Let S' be an upper $(r+1)$ edge cover of G' then $S' \supseteq E_U$ because otherwise we could not cover all the vertices of U $r+1$ times. We claim that the set $S' \setminus E_U$ cannot has size $|S' \setminus E_U| \geq (r+1)\mathsf{uec_r(G)}$.In order to prove this, first observe that E_U is an tec of G' because it covers every vertex at least once and the vertices with degree greater that one are all in U which is independent set. Observe that all the vertices in $\bigcup_{i=1}^{k+1}$ are incident to exactly one edge of E_U. Furthermore, we have that $S' \setminus E_U$ is a r-tec of $G'_{V'[S' \setminus E_U]}$ because all vertices in $V'[S' \setminus E_U]$ with degree greater than r are independent (otherwise we could have two adjacent vertices with degree greater that $r+1$ in $G'_{S'}$) and all the other covered exactly r times (they covered $r+1$ by S'). Therefore, because the remaining graph ($G'_{V'[S' \setminus E_U]}$) consisted only by the $r+1$ copies of G we can separate $S' \setminus E_U$ in $r+1$ r-tec, one for each copy if G. Due to that $|S' \setminus E_U| \leq (r+1)\mathsf{uec_r(G)}$ so:

$$\mathsf{uec_{r+1}(G')} = |\mathsf{S' \setminus E_U}| + |\mathsf{E_U}| \leq \mathsf{(r+1)uec_r(G) + n(r+1)}.$$

It remains to show that the new graph has greater maximum degree than than the original by one. Indeed, this holds because in the copies of G the maximum degree is $\Delta + 1$ (we added one edge to each vertex of each cope of G) and the degree of the vertices of U is exactly $r+1$ so $\Delta(G') = \max\{\Delta + 1, r+1\} = \Delta + 1$ (we can assume that $\Delta \geq r$). □

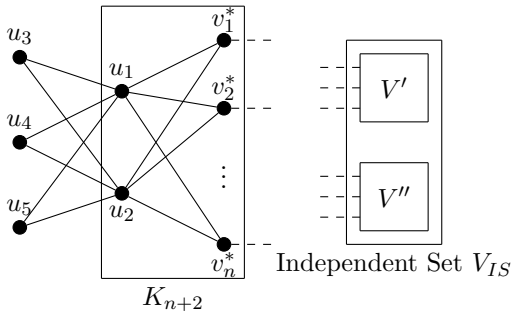


Figure 4.3: We add the all the edges between the vertices of the set $V^* \cup \{u_1, u_2\}$ in order to be a clique. The dashed edges represent the edges between the sets V^* and $V' \cup V''$ (V_{IS}). A vertex $v'_1 \in V'$ adjacent to a vertex $v^*_2 \in V^*$ if $v_1v_2 \in E$ in the starting graph and to the vertex $v_1* \in V*$; the same holds for the vertices in V''.

Remark. *Observe that the reverse direction of the proof applies to any* $(r+1)$-tec *of* G'. *So starting from an* $(r+1)$-tec *of* G' *we can construct an* r-tec *of* G *by checking the* r-tec *of the copies of* G *in* G'. *Furthermore, if we select the greater one we have the following relation:*

$$|r\text{-tec}(G)| \geq \frac{1}{r+1}|(r+1)\text{-tec}(G')| - n$$

Theorem 4.15. *Double Upper EC is NP-hard in split graphs.*

Proof. The proof is based on a reduction from the 2-tuple dominating set. Let $G = (V, E)$ be an instance of 2-tuple dominating set; using this instance we will construct a split graph G'. First, for every vertex $v \in V$ we make three copies v^*, v', v'' (e.i. three copies of V: V^*, V' and V'') and for each vertex $u \in N[v]$ we add the edges v^*u', v^*u'' (so, we contain the edges v^*v' and v^*v'' in the new graph). After that we construct a $K_{3,2}$ and in the end we add edges in order to make a complete graph with the vertices of V^* and the two vertices of the one side of $K_{3,2}$ (Fig 4.3). From now on let $V_{IS} = V' \cup V''$.

Now we will prove that for a the minimum 2-tuple dominating S set of G and a maximum 2-tec of G' holds the following:

$$uec_2(G') = 8n - 2\gamma_{\times 2}(G) + 6 \tag{4.8}$$

Let S be a minimum 2-tuple dominating set of G. Using S we will construct an 2-tec E' for G'. For any vertex $v \in V$ we will select two vertices $w_1, w_2 \in S$ that dominates it (there exist because S is a 2-tuple dominating set); then we will put in E' all the edges $w_1^*v', w_1^*v'', w_1^*v''', w_2^*v', w_2^*v''$ and w_2^*v''' of G' (by the construction we are sure that we have all the needed edges). By repeating this process for all the vertices of G we ending with a set of edges E' which covers all the vertices in V_{IS} twice. Except that, E' covers more that twice all $v* \in V^*$ such that $v \in S$ because S is minimum which means that for all $v \in S$ there exists a vertex $w \in V$ that is not dominated second time without v so we have used at least two edges adjacent to it (one for w' and one for w''). Now

for all $v \in V \setminus S$ we add to E' the edges v^*u_1 and v^*u_2 in order to cover the remaining vertices of V^*. Finally we add all the edges between u_1, u_2 and u_3, u_4, u_5. We claim that E' is a 2-tec of G' with size $8n - 2|S| + 6$. Is easy to see that we cover all the vertices at least twice by the construction of E'; so we have to prove that we cannot remove any edge from it. Let $V_1(E')$ and $V_2(E')$ to be the associated partition related to E' (see property 4.1). We have to show that $V_2(E')$ is independent set of $G'_{E'}$. Let $S^* = \{v^* \in V^* \mid v \in S\}$ then is easy to see that $V_2(E') \subseteq S^* + u_1 + u_2$ because for all the other vertices we have use exactly two edges. Then, because we have not use any edges between the vertices of the set $S^* + u_1 + u_2$ in E' we have that $V_2(E')$ is independent.

For the size of E' we have $2(3n)$ edges that cover the vertices of V_{IS} and $\{v* \mid v \in S\}$, six edges between u_1, u_2 and u_3, u_4, u_5 and $2(n - |S|)$ edges for the vertices $v \in V \setminus S$. In total, $|E'| = 8n - 2|S| + 6$. So we have that:

$$uec_2(G') \geq 8n - 2|S| + 6$$

Now we have to show the converse. Let E' be an optimal solution of Double Upper EC of G'; using E' we will construct a 2-tuple dominating set of G (not necessarily minimum). In order to prove this we will need to construct from E' an other optimal solution that has some specific properties. Before we start, observe that if need to cover any vertex $v \in V*$ we can always use the edges u_1v and u_2v because u_1 and u_2 are covered at least three times in any solution. First we will prove that:

- For any optimal solution of Double Upper EC E' there exists an equivalent solution E'' such that $|E'| = |E''|$ and for any vertex $v \in V_{IS}$ we have $d_{G'_{E''}}(v) = 2$.

This can be proved easily because if there exists a vertex $v \in V_{IS}$ with degree three or more in our solution then we will remove an edge which is incident to v from E' and we will add an other edge in order to keep the size of our solution the same. Let $e = uv$ be the edge that we want to remove. We know that $u \in V^*$ (by the construction of G') and it is covered only once by $E' - e$ (otherwise E' wouldn't be minimal). That means that we have to cover the vertex u once more. For that, we can use one of the edges u_1u or u_2u because we know that there are not both in our solution (like we said, u covered once by $E' - e$) and the degree of both of u_1, u_2 is greater than two in any solution. Let u_1u be the new edge, then $E'' = (E' - e + u_1u$ is an optimal solution. If we repeat the process we will have a solution without vertices $v \in V_{IS}$ with degree three or more. Now we will prove that:

- For any optimal solution E' of Double Upper EC there exists an equivalent solution E'' such that $|E'| = |E''|$ and there is no edge $uv \in E''$ with both $u, v \in V^*$.

Let us say that there exists $uv \in E'$ such that $u, v \in V^*$. That means that at least one of u, v has degree two in our solution. If both of them are degree two then, like before, we can remove uv from our solution and add one of u_1u, u_2u in order to cover u and one of u_1v, u_2v in order to cover v. This cannot be happened because the new solution will be greater than the starting and we have assumed that we start from an optimal. So, we

have only one of u, v to be degree two in our solution, let v be that vertex. If we remove the edge uv from E' we have to cover only v and we can do it by adding one of u_1v, u_2v and we keep the same size of solution. By repeating the process we can construct the wanted solution. The third property that is needed for our solution is the following:

- For any optimal solution E' of Double Upper EC there exists an equivalent solution E'' such that $|E'| = |E''|$ and $\forall v \in V^*$ either $N_{G'_{E''}}(v) = \{u_1, u_2\}$ or $N_{G'_{E''}}(v) \subseteq V_{IS}$.

By starting from any optimal solution we are constructing a new one that keeps both the first two properties; let E' be that solution. By the second property we do not have edges between vertices of V^* in the solution so, if there exists a vertex such that neither $N_{G_{E'}}(v) = \{u_1, u_2\}$ nor $N_{G_{E'}}(v) \subseteq V_{IS}$ then for v we must have the following three properties:

1. $N_{G'_{E'}}(v) \cap V_{IS} \neq \emptyset$
2. $N_{G'_{E'}}(v) \cap \{u_1, u_2\} \neq \emptyset$
3. $(N_{G'_{E'}}(v) \cap V_{IS}) \cup (N_{G'_{E'}}(v) \cap \{u_1, u_2\}) = N_{G'_{E'}}(v)$

It is easy to realize that the degree of v must be exactly two, otherwise we could remove the edge between v and the set $\{u_1, u_2\}$ that used in our solution. Now, w.l.o.g. let $N_{G'_{E'}}(v) = \{u_1, u'\}$ for some $u' \in V'$. In order to change the neighborhood of v, we will remove the edge vu' and we will add vu_2. After this the neighborhood of v is $\{u_1, u_2\}$ but u' is covered only once (by the first property). Let $u'' \in V''$ be the copy of u' and $N_{G'_{E'}}(u'') = \{v_1, v_2\}$ (is degree two by the first property). Because we have not change the edges incident to any of v_1, v_2 we have that the degree of both is at least two in our solution. If one of them has degree greater than two, let v_1 be that vertex, then we could add the edge $u'v_1$ (by the construction of G' u' and u'' have the same neighbors) and then we could have a solution greater than the optimal, which is a contradiction. So, both v_1, v_2 have degree two in our solution and cannot be connected both with u' (degree one at the moment). Let $u'v_1$ not used in our solution, then we want to use it in order to cover u'. If v_1 is connected only to vertices in V_{IS} then again we could use $u'v_1$ and give a solution greater than the optimal (we cannot remove edges incident to vertices in V_{IS} because are all degree two), so v_1 must be connected to one of u_1, u_2 and we can remove this edge from our solution and keep the same size for the solution. So we can construct the wanted solution by repeating the process for all the needed vertices.

In order to complete this we must prove that

$$uec_2(G') \leq 8n - 2|S| + 6$$

where S is a minimum 2-tuple dominating set. For that purpose, first we will show that if E' is an optimal solution of Double Upper EC that with all the above properties and $S' = \{v \mid v^* \in V^* \text{ and } N_{G'_{E'}}(v^*) \subseteq V_{IS}\}$ then S' is a 2-tuple dominating set of G

and $uec_2(G') = 8n - 2|S^*| + 6$ where $S^* = \{v^* \in V^* \mid v \in S'\}$. S' is a 2-tuple dominating set of G because every vertex $u' \in V'$ is dominated twice by the vertices in S^* which means that every $u \in V$ has $N[u] \cap S' \geq 2$. Now, our solution contains all the six edges between $u_1, \ldots, u_5$, two edges for each vertex in $V^* \setminus S^*$ (because by the last property their neighborhood in E' is $\{u_1, u_2\}$) and two edges for each vertex in V_{IS} (which covers twice all vertices in S^*). So we have that

$$uec_2(G') = 8n - 2|S^*| + 6 = 8n - 2|S'| + 6 \leq 8n - 2|S| + 6$$

because S is a minimum 2-tuple dominating set (so $|S| \leq |S'|$). So, by the NP-hardness of 2-tuple dominating set in general graphs we have that the Double Upper EC is NP-hard in split graphs.

□

4.5 Hardness of Approximation

In the following theorems we provide some inapproximability results for the Upper EC and the Double Upper EC.

Theorem 4.16. *It is NP-hard to approximate the solution of Upper EC to within* $\frac{593}{594}$ *and* $\frac{363}{364}$ *in graphs of max degree* 4 *and* 5 *respectively.*

Proof of Theorem 4.16. In order to prove this we will use a reduction from Min VC. Starting from an r-regular graph $G = (V, E)$ we will construct a new graph G'. First we will add a P_2 to each vertex $v \in V$ (let v' and v'' be the vertices of P_2). After that for each edge $e = vu \in E$ we add a new vertex v_e adjacent to v and u. In the end we remove all the starting edges E. Let $G' = (V', E')$ be the new graph; we claim that:

$$uec(G') = 2n + m - |minVC(G)| \tag{4.9}$$

By starting from a minimum vertex cover of G we will construct an *ec* of G'. It is easy, by using m edges, to cover the vertices v_e of G' by selecting the edge $v_e v$ if $v \in minVC(G)$ and the edge $v_e u$ otherwise. This way we covered and all the vertices in $minVC$ because if any remains uncovered this way we could have a $VC \subset minVC$ which is impossible. In order to cover the ending vertices of the adding paths we must use the edge $v'v''$ (n edges in total). It remains to cover the vertices of V that are not in the $minVC$. This can be done by adding to our solution for each uncovered vertex $v \in V$ the edge $v'v$. Is easy to see that the selected edges is a miminal edge cover of size $m + n + n - |minVC|$ so we have:

$$uec(G') \geq 2n + m - |minVC|$$

In reverse, if we start from an upper edge cover S of G' we will make a vertex cover of G. First observe that all the edges $v'v''$ must be in our edge cover because this is the only way to cover the vertices v''. Second, if we have the edge $v_e v$ in our solution then the edge $v'v$ is not and reverse (otherwise our solution would not be minimal). Now we will prove that:

- For any optimal solution S' of Double Upper EC there exists an equivalent solution S' such that $|S'| = |S|$ and $\forall v_e \in V'$ there exists only one edge that covers it.

In order to prove that we will start from a vertex v_e that covered twice in our solution. Because the solution must be minimal then the vertices been adjacent to v_e, let them be v and u, must have degree one in our solution. So, we can remove one of the edges; let this be the $v_e v$, and cover the vertex v by using the edge $v'v$. The new set of edges remain a minimal edge cover and has the same size. By doing this to all vertices v_e, $e \in E$, we have the wanted property.

Now we will show that if we have a solution S that has the previous property then the set $U = \{v \in V' | v_e v \in S\}$ is a vertex cover of G. This is obvious if we realize that the edge $v_e v$ exists only if v is incident to the edge e and that in S we covered all the vertices v_e so U covers all the edges $e \in E$.

It remains to check the size of our solution. Like we have said before, we need n edges to cover the vertices v''. Because each vertex v_e covered exactly once we need m edges and in the same time these edges covers the set U; so we have to cover the vertices $V \setminus U$. These vertices does not covered by edges that are incident to vertices v_e so by our previous observation the have degree exactly one (they covered by the edges $v'v$). All the previous gives us the following:

$$uec(G') = 2n + m - |U| \leq 2n + m - |minVC|.$$

Before we continue, it is important to realize that if we start from a minimal edge cover of value $|ec|$ we can construct a vertex cover U of size $|U| = 2n + m - |ec|$ by using the same method as above (because we use only the minimality of our solution in the costruction).

So, in order to prove those inapproximability results we will start from an r-regular graph and we will do the construction we describe above. Now we have to observe that:

- $m \leq r|minVC|$
- $n = \frac{2m}{r} \leq 2|minVC|$

By starting from an approximation solution, of value $|ec|$, of Upper EC we can construct a vertex cover U. Suppose that we can approximate Upper EC in a factor grater than $(1 - a)$ in graphs with max degree $r + 1$ then:

$$\begin{aligned} |U| - |minVC| \leq & uec(G') - |ec| \\ \leq_a & \ uec(G') \\ =_a & \ (2n + m - |minVC|) \\ \leq_a & \ (3 + r)|minVC| \end{aligned}$$

By this we could have an approximating solution for Min VC within a ratio $1+(3+r)a$ for r-regular graphs. Because we know that Min VC cannot be approximated within a

factor $\frac{100}{99}$ in 3-regular graphs and within a factor $\frac{53}{52}$ in 4-regular graphs (both in [56]) we have that:

- a cannot be equal or less than $\frac{1}{594}$ in 3-regular graphs
- a cannot be equal or less than $\frac{1}{364}$ in 4-regular graphs

So, the Upper EC cannot be approximated within a factor:

- $1 - \frac{1}{594}$ for graphs with max degree $\Delta = 4$ and
- $1 - \frac{1}{364}$ for graphs with max degree $\Delta = 5$.

□

Theorem 4.17. *It is NP-hard to approximate the solution of Double Upper EC to within* $\frac{883}{884}$ *in graphs of max degree* 6.

Proof of Theorem 4.17. We will start from the same reduction as in theorem 4.16. So, from a 4-regular graph $G = (V, E)$ we are forming a graph $G' = (V', E')$ where $|V'| = 3n + m$, max degree $\Delta = 5$ and $uec(G') = 2n + m - |minVC(G)|$. After that we will do the same reduction between Upper EC and Upper 2-EC as in Theorem 4.14 so we have the following relation:

$$uec_2(G'') = 2uec(G') + 2|V'|$$

and generally by the remark 4.4 we have that from any 2-tec, of value $|ec_2|$, of G'' we can construct a tec, of value $|ec|$, of G' such that the following equation holds:

$$|ec_2| \leq 2|ec| + 2|V'|.$$

By combine the two reductions we have that:

$$uec_2(G'') = 2(2n + m - |minVC|) + 2|V'| = 10n + 4m - 2|minVC(G)|$$

Now, if we can approximate the solution of Double Upper EC in factor $1 - a$ then we can have a solution of value $|ec_2|$ such that $\frac{|ec_2|}{uec_2} \geq 1 - a$. By starting from this solution we can construct a vertex cover U for G. That gives us the following:

$$\begin{aligned} |U| - |minVC| \leq & \frac{1}{2}(uec_2(G'') - |ec_2|) \\ \leq & \frac{1}{2} a uec(G'') \\ = & \frac{1}{2} a(10n + 4m - 2|minVC|) \end{aligned}$$

Because we have start from a 4-regular graph we know that $m \leq 4minVC$ and $n \leq 2minVC$ so by the previous we have that:

$$\begin{aligned}|U| - |minVC| \leq & \frac{1}{2}a(10n + 4m - 2|minVC|) \\ \leq & 17a|minVC|\end{aligned}$$

So, we could have a $1+17a$ approximation for vertex cover in 4-regular graphs. Because we know that vertex cover is not $\frac{53}{52}$ approximable (see [56]) that gives us that a cannot be less than or equal to $\frac{1}{17} \cdot \frac{1}{52}$ so the solution of Double Upper EC is NP-hard to approximate within a factor $\frac{883}{334}$ in graphs of max degree 6.

□

Theorem 4.18. *It is NP-hard to approximate the solution of Double Upper EC to within $\frac{571}{572}$ in graphs of max degree 9.*

Proof. Again, we will make a reduction from Vertex Cover problem. Starting from a 4-regular graph $G = (V, E)$ we construct a new graph by adding a set of new vertices V_E which has one vertex v_e for each edge $e \in E$, and then adding new edges $v_e u$ if the edge e was incident to u in the original graph G. Let $G' = (V', E')$ be the new graph. It is easy to see that $|V'| = |V| + |E|$ and $\Delta(G') = 2\Delta(G) = 8$. Furthermore, we can show that from any VC of G we can construct a tec of G' of size $|\text{TEC}| = |E| + |V| - |VC|$ and conversely, from any tec of G' we can construct a VC of G of size $|VC| = |E| + |V| - |\text{TEC}|$. In order to prove the first direction we will start from a VC of G. Let S be the set of all the edges $v_e u$ where $u \in VC$. S is a partial tec of G because it covers only the vertices in $VC \cup V_E$, any vertex of V_E has degree one in G'_S and the vertices of VC are independent in G'_S. It is easy to extend S to a tec of G' by adding one edge for every vertex $v \in V \setminus VC$ that is adjacent to a vertex in VC (there exists because $v \in V$ and VC is a vertex cover of G). The extended S is a tec due to the fact that the vertices that may have greater degree than one are all in VC, which is independent in G_S and all the vertices are covered. Now we have to observe that this tec contains exactly one edge for each vertex in V_E and one for each vertex in $V \setminus VC$ so the size is exactly

$$|\text{TEC}| = |E| + |V| - |VC|.$$

Conversely, we will start from a tec of G' and we will construct a vertex cover of G of the wanted size. First we have to show that for any tec S of G', if there exists $v_e \in V_E$ such that $d_{G'_S}(v_e) = 2$ (it cannot be greater because $d_{G'}(v_e) = 2$) then there exists an other tec S' of G' that has the same size and every vertex $v_e \in V_E$ has degree $d_{G_{S'}}(v_e) = 1$. This is easy to prove by repeating the following:

$$\text{If there exists } e = uv \in E \text{ such that } d_{G_{S'}}(v_e) = 2 \text{ then } S' := S' + v_e u - v_e v.$$

This process terminates because it reduces the number of such vertices each time by one. We have to show that the last set has the same size and remains a tec. Because v_e has degree two in the starting tec this means that the vertices u and v that were adjacent

to it had degree one. In the new set we have degree two in vertex u and degree one in the two neighbors of it, v_e and v. So, the new set remains a tec and has the same size because we remove one and add one edge. Now, from S' we will construct a vertex cover of G. We claim that the set $U = \{v \in V \mid N_{G'_{S'}}(v) \cap V_E \neq \emptyset\}$ is a vertex cover of G of the desired size. Because for each edge $e \in E$ there exists a vertex $v_e \in V_E$, we have that U is a vertex cover of G (because it dominates the V_E). Because we know that in the modified tec every vertex in V_E has degree exactly one and those edges covers only U (by construction) we need to count the edges that covers the remaining vertices. Assume that in our tec the remaining vertices $(V \setminus U)$ have degree one and are independent, then the size of our tec is

$$|\text{TEC}| = |E| + |V| - |U| \tag{4.10}$$

which gives us what we needed. In order to complete the reduction between Vertex Cover and Upper Edge Cover we need to prove that the last assumption is always true. First observe that if two vertices $v, u \in V \setminus U$ are covered by the same edge in our tec then there exists an edge $uv = e \in E$ and a vertex $v_e \in V_E$. Then in our tec the vertex v_e must be covered by u or v, which is a contradiction because none of them are in U. Now suppose that there exists vertex $v \in V \setminus U$ such that $d_{G'_{S'}}(v) \geq 2$. Because $v \notin U$ we know that there is a $u \in U$ such that $uv \in S'$ and because u must be adjacent to a vertex in V_E in our tec this means that we have two vertices of degree at least two in a tec which is a contradiction.

After that we will do the same reduction between Upper EC and Upper 2-EC as in Theorem 4.14 so we have $uec_2(G'') = 2uec(G') + 2|V'|$ and generally by the remark 4.4 we have that from any 2-tec, of value $|ec_2|$, of G'' we can construct a tec, of value $|ec|$, of G' such that the following equation holds,

$$|ec_2| \leq 2|ec| + 2|V'|. \tag{4.11}$$

Furthermore we have $m = |E| \leq 4|minVC|$ and $n = |V| \leq 2|minVC|$ (because we started from a 4-regular graph) which implies that from a 2-tec, of value $|ec_2|$, of G'' we can construct a vertex cover, U, of G that has the following property.

$$|ec_2| \leq 4|E| + 4|V| - 2U \tag{4.12}$$

The previous equation is easy to prove by the equations 4.10, 4.11 and the construction of the graphs. Now we are ready to finish the proof. If we can approximate the solution of Upper 2-EC within a factor of $1 - a$ then we can have a solution of value $|\mathsf{ec_2}|$ such that $\frac{|\mathsf{ec_2}|}{\mathsf{uec_2}} \geq 1 - a$. By starting from this solution we can construct a vertex cover U for the graph G. Then we have that:

$$\begin{aligned} |U| - |minVC| &\leq \frac{1}{2}(\mathsf{uec_2}(\mathsf{G''}) - |\mathsf{ec_2}|) \leq \frac{1}{2}\mathsf{a}\ \mathsf{uec_2}(\mathsf{G''}) \\ &= \frac{1}{2}a\ (4m + 4n - 2|minVC|) \end{aligned}$$

By the relations between m, n and $|minVC|$ we have the following:

$$|U| - |minVC| \leq \frac{1}{2}a\ (4m + 4n - 2|minVC|) \leq 11a\ |minVC|$$

So, we could have a $1 + 11a$ approximation for vertex cover in 4-regular graphs. Because the vertex cover is not $\frac{53}{52}$ approximable (see [56]) that gives us that a cannot be less than or equal to $\frac{1}{11} \cdot \frac{1}{52}$ so the solution of Double Upper EC is NP-hard to approximate within a factor $\frac{571}{572}$ in graphs of max degree 9. □

4.6 Conclusions

In this chapter we studied a variant of the classic Edge Cover called Upper r-Tolerant Edge Cover. Interestingly enough, we proved that any feasible solution of the problem is $\frac{1}{2}$-approximation solution. Furthermore, we show that the problem remains NP-hard in many restricted cases like split graphs and cubic planar graphs. Finally we present some inapproximability results for graph with bounded degree.

Some questions we can consider in the future are the following:

- Can we improve the approximation ratio in general or in specific instances?
- Can we present hardness of approximation reduction that give us a more tight inapproximability ratio?
- Are there classes of graph where the problem belongs in P?

Finally, it would be interesting to investigate the parameterized complexity of the problem. In particular, since Upper Edge Cover is polynomial in many classes that have constant treewidth, it would be interesting to consider as parameter the treewidth of the given graph. If we manage to set the problem in FPT, when parameterized by treewidth, it will also give us some graph classes where the problem can be solved in polynomial time.

Chapter 5

Max-Min Feedback Vertex Set

5.1 Introduction

In this chapter we consider a variation of Feedback Vertex Set. Typically, Feedback Vertex Set is studied with a minimization objective as we have define it in the Section 2.4. Here, we are interested in an objective which is, in a sense, the inverse: we seek a feedback vertex set S which is as *large* as possible, while still being minimal. We call this problem Max Min FVS.

MaxMin and MinMax versions of many famous optimization problems have recently attracted much interest in the literature (we give references below) and Max Min FVS can be seen as a member of this framework. Although the initial motivation for studying such problems was a desire to analyze the worst possible performance of a naive heuristic, these problems have gradually been revealed to possess a rich combinatorial structure that makes them interesting in their own right. Our goal in this chapter is to show that Max Min FVS displays an interesting complexity behavior with respect to its approximability.

Our motivation for focusing on Max Min FVS is the contrast between two of its more well-studied cousins: the Max Min Vertex Cover and Upper Dominating Set problems, where the objective is to find the largest minimal vertex cover or dominating set respectively. At first glance, one would expect Max Min VC to be the easier of these two problems: both problems can be seen as trying to find the largest minimal hitting set of a hypergraph, but in the case of Max Min VC the hypergraph has a very restricted structure, while in Upper DS the hypergraph is essentially arbitrary. This intuition turns out to be correct: while Upper DS admits no $n^{1-\epsilon}$-approximation [22], Max Min VC admits a $\sqrt{n}$-approximation (but no $n^{1/2-\epsilon}$-approximation) [40].

This background leads us to the natural question of the approximability of Max Min FVS. On an intuitive level, one may be tempted to think that this problem should be harder than Max Min VC, since hitting cycles is more complex than hitting edges, but easier than Upper DS, since hitting cycles still offers us more structure than an arbitrary hypergraph. However, to the best of our knowledge, no $n^{1-\epsilon}$-approximation algorithm is currently known for Max Min FVS (so the problem could be as hard as

Upper DS), and the best hardness of approximation bound known is $n^{1/2-\epsilon}$ [154] (so the problem could be as easy as Max Min VC).

Related Work To the best of our knowledge, Max Min FVS was first considered by Mishra and Sikdar [154], who showed that the problem does not admit an $n^{1/2-\epsilon}$ approximation (unless P = NP), and that it remains APX-hard for $\Delta \geq 9$. Furthermore, these seem to be the only results related to the classic complexity of the problem. On the other hand, Upper DS and Max Min VC are well-studied problems, both in the context of approximation and in the context of parameterized complexity [2, 22, 40, 42, 55, 63, 72, 108, 122, 182, 184]. Many other classical optimization problems have recently been studied in the MaxMin or MinMax framework, such as Max Min Separator [100], Max Min Cut [79], Min Max Knapsack (also known as the Lazy Bureaucrat Problem) [12, 90, 97], and some variants of Max Min Edge Cover [130, 103]. Some problems in this area also arise naturally in other forms and have been extensively studied, such as Min Max Matching (also known as Edge Dominating Set [121]), Grundy Coloring, which can be seen as a Max Min version of Coloring [3, 24], and Max Min VC in hypergraphs, which is known as Upper Transversal[155, 109, 110, 111].

The idea of designing super-polynomial time approximation algorithms which obtain guarantees better than those possible in polynomial time has attracted much attention in the last decade [19, 41, 66, 69, 78, 88, 128]. As mentioned, the result closest to the time-approximation trade-off we give, in this chapter, is the approximation algorithm for Max Min VC given by Bonnet et al. [38]. It is important to note that such trade-offs are only generally known to be tight up to poly-logarithmic factors in the exponent of the running time. As explained in [38], current lower bound techniques can rule out improvements in the running time that shave at least n^{ϵ} from the exponent, but not improvements which shave poly-logarithmic factors, due to the state of the art in quasi-linear PCP constructions. Indeed, such improvements are sometimes possible [19] and are conceivable for Max Min VC and Max Min FVS. Lower bounds for this type of algorithm rely on the (randomized) Exponential Time Hypothesis (ETH).

Our Contribution Our main contribution in this chapter is to fully answer the question addressed in the end of Section 5.1, confirming and precisely quantifying the intuition that Max Min FVS is a problem that lies "between" Max Min VC and Upper DS: We give a polynomial-time approximation algorithm with ratio $O(n^{2/3})$ and a hardness of approximation reduction which shows that (unless P = NP) no polynomial-time algorithm can obtain a ratio of $n^{2/3-\epsilon}$, for any $\epsilon > 0$. This completely settles the approximability of the problem in polynomial time. Along the way, we also prove that Max Min FVS admits a cubic kernel when parameterized by the solution size, give an approximation algorithm with ratio $O(\Delta)$, show that no algorithm can achieve ratio $\Delta^{1-\epsilon}$, for any $\epsilon > 0$, and improve the best known NP-completeness proof for Max Min FVS from graphs of $\Delta \geq 9$ [154] to planar bipartite graphs of $\Delta \geq 6$, where Δ is the maximum degree of the input graph. Note that, this is one of the very few hardness

results for this problem. Furthermore, to the best of our knowledge, the complexity of Max Min FVS was unknown for both planar and bipartite graphs until now.

One interesting aspect of our results is that they have an interpretation from extremal combinatorics which nicely mirrors the situation for Max Min VC. Recall that a corollary of the $\sqrt{n}$-approximation for Max Min VC [40] is that any graph without isolated vertices has a minimal vertex cover of size at least $\sqrt{n}$, and this is tight (see Remark 5.3.2). Hence, the algorithm only needs to trivially preprocess the graph (deleting isolated vertices) and then find this set, which is guaranteed to exist. Our algorithms can be seen in a similar light: we prove that if one applies two almost trivial pre-processing rules to a graph (deleting leaves and contracting edges between degree-two vertices), a minimal fvs of size at least $n^{1/3}$ (and $\Omega(n/\Delta)$) is always guaranteed to exist, and this is tight (Corollary 5.9 and Remark 5.3.2). Thus, the approximation ratio of $n^{2/3}$ is automatically guaranteed for any graph where we exhaustively apply these very simple rules and our algorithms only have to work to construct the promised set. This makes it somewhat remarkable that the ratio of $n^{2/3}$ turns out to be best possible.

Having settled the approximability of Max Min FVS in polynomial time, we consider the question of how much time needs to be invested if one wishes to guarantee an approximation ratio of r (which may depend on n) where $r < n^{2/3}$. This type of time-approximation trade-off was extensively studied by Bonnet et al. [38], who showed that Max Min Vertex Cover admits an r-approximation in time $2^{O(n/r^2)}$ and this is optimal under the randomized ETH.

For Max Min FVS we cannot hope to obtain a trade-off with performance exponential in n/r^2, as this implies a polynomial-time $\sqrt{n}$-approximation. It therefore seems more natural to aim for a running time exponential in $n/r^{3/2}$. Indeed, generalizing our polynomial-time approximation algorithm, we show that we can achieve an r-approximation in time $n^{O(n/r^{3/2})}$. Although this algorithm reuses some ingredients from our polynomial-time approximation, it is significantly more involved, as it is no longer sufficient to compare the size of our solution to n. We complement our result with a lower bound showing that our algorithm is essentially best possible under the randomized ETH for any r (not just for polynomial time), or more precisely that the exponent of the running time of our algorithm can only be improved by $n^{o(1)}$ factors.

5.2 Preliminaries

It is not hard to see that an fvs S is minimal if every $u \in S$ has a *private cycle*, that is, there exists a cycle in $G[(V \setminus S) \cup \{u\}]$, which goes through u. A vertex u of a feedback vertex set S that does not have a private cycle (that is, $S \setminus \{u\}$ is also an fvs), is called *redundant*. For a given fvs S, we call the set $F := V \setminus S$ the corresponding induced forest. If S is minimal, then F is maximal.

The main problem we are interested in is Max Min FVS: given a graph $G = (V, E)$, find a minimal fvs of G of maximum size. Since this problem is NP-hard, we will be interested in approximation algorithms. An approximation algorithm with ratio $r \geq 1$ (which may depend on n, the order of the graph) is an algorithm which, given a graph

G, returns a solution of size at least $\frac{\text{mmfvs}(G)}{r}$, where mmfvs$(G)$ is the size of the largest minimal fvs of G.

We make two basic observations about our problem: deleting vertices or contracting edges can only decrease the size of the optimal solution.

Lemma 5.1. *Let $G = (V, E)$ be a graph and $u \in V$. Then,* mmfvs$(G) \geq$ mmfvs$(G - u)$. *Furthermore, given any minimal feedback vertex set S of $G - u$, it is possible to construct in polynomial time a minimal feedback vertex set of G of the same or larger size.*

Lemma 5.2. *Let $G = (V, E)$ be a graph, $u, v \in V$ with $N(u) \cap N(v) = \emptyset$ and $uv \in E$. Then* mmfvs$(G) \geq$ mmfvs(G/uv). *Furthermore, given any minimal feedback vertex set S of G/uv, it is possible to construct in polynomial time a minimal feedback vertex set of G of the same or larger size.*

The proofs of both lemmas are simple; we start from a minimal fvs S of the new graph and we construct a minimal fvs S' of the starting graph G such that $|S| \leq |S'|$. The complete proofs of these lemmas can be found in the appendix.

5.3 Polynomial Time Approximation Algorithm

In this section we present a polynomial-time algorithm which guarantees an approximation ratio of $n^{2/3}$. As we show in Theorem 5.14, this ratio is the best that can be hoped for in polynomial time. Later (Theorem 5.12) we show how to generalize the ideas presented here to obtain an algorithm that achieves a trade-off between the approximation ratio and the (sub-exponential) running time, and show that this trade-off is essentially optimal.

On a high level, our algorithm proceeds as follows: first we identify some easy cases in which applying Lemma 5.1 or Lemma 5.2 is *safe*, that is, the value of the optimal is guaranteed to stay constant, namely deleting vertices of degree at most 1, and contracting edges between vertices of degree 2. After we apply these reduction rules exhaustively, we compute a minimal fvs S in an arbitrary way. If S is large enough (larger than $n^{1/3}$), we simply return this set.

If not, we apply some counting arguments to show that a vertex $u \in S$ with high degree ($\geq n^{2/3}$) must exist. We then have two cases: either we are able to construct a large minimal fvs just by looking at the neighborhood of u in the forest (and ignoring $S \setminus \{u\}$), or u must share many neighbors with another vertex $v \in S$, in which case we construct a large minimal fvs in the common neighborhood of u, v.

Because our algorithm is constructive (and runs in polynomial time), we find it interesting to remark an interpretation from the point of view of extremal combinatorics, given in Corollary 5.9.

5.3.1 Basic Reduction Rules and Combinatorial Tools

We begin by showing two *safe* versions of Lemmas 5.1, 5.2.

Lemma 5.3. *Let G, u be as in Lemma 5.1 with $d(u) \leq 1$. Then* $\mathrm{mmfvs}(G - u) = \mathrm{mmfvs}(G)$.

Proof. We only need to show that $\mathrm{mmfvs}(G) \leq \mathrm{mmfvs}(G - u)$ (the other direction is given by Lemma 5.1). Let S be a minimal fvs of G. Then, S is an fvs of $G - u$. Furthermore, $u \notin S$, as S is minimal in G. To see that S is also minimal in $G - u$, note that any cycle of G also exists in $G - u$ (as no cycle contains u). □

Lemma 5.4. *Let G, u, v be as in Lemma 5.2 with $d(u) = d(v) = 2$. Then* $\mathrm{mmfvs}(G/uv) = \mathrm{mmfvs}(G)$.

Proof. Let $G' = G/uv$, w be the vertex that replaced u, v in G', and $V' = V(G')$.

We only need to show that $\mathrm{mmfvs}(G) \leq \mathrm{mmfvs}(G')$, as the other direction is given by Lemma 5.2. Let S be a minimal fvs of G. We consider two cases:

If $u, v \notin S$, then we claim that S is also a minimal fvs of G'. Indeed, $G'[V' \setminus S]$ is obtained from $G[V \setminus S]$ by contracting uv, so both are acyclic. Furthermore, for all $z \in S$, $G'[(V' \setminus S) \cup \{z\}]$ is obtained from $G[(V \setminus S) \cup \{z\}]$ by contracting uv, therefore both have a cycle, hence no vertex of S is redundant in G'.

If $\{u, v\} \cap S \neq \emptyset$, we claim that exactly one of u, v is in S. Indeed, if $u, v \in S$, then $G[(V \setminus S) \cup \{u\}]$ does not contain a cycle going through u, as u has degree 1 in this graph. Without loss of generality, let $u \in S$, $v \notin S$. We set $S' := (S \setminus \{u\}) \cup \{w\}$ and claim that S' is a minimal fvs of G'. It is not hard to see that S' is an fvs of G', since it corresponds to deleting $S \cup \{v\}$ from G. To see that it is minimal, for all $z \in S' \setminus \{w\}$ we observe that $G'[(V' \setminus S') \cup \{z\}]$ obtained from $G'[(V \setminus S) \cup \{z\}]$ by deleting v, which has degree 1. Therefore, this deletion strongly preserves acyclicity. Finally, to see that w is not redundant for S' we observe that $G[(V \setminus S) \cup \{u\}]$ has a cycle, and this cycle must be present in $G'[(V' \setminus S') \cup \{w\}]$, which is obtained from the former graph by contracting uv. □

Definition 5.1. *For a graph $G = (V, E)$ we say that G is reduced if it is not possible to apply Lemma 5.3 or Lemma 5.4 to G.*

We now present a counting argument which will useful in our algorithm and states, roughly, that if in a reduced graph we find a (not necessarily minimal) fvs, that fvs must have many neighbors in the corresponding forest.

Lemma 5.5. *Let $G = (V, E)$ be a reduced graph and $S \subseteq V$ a feedback vertex set of G. Let $F = V \setminus S$. Then, $|N(S) \cap F| \geq \frac{|F|}{4}$.*

Proof. Let n_1 be the number of leaves of F, n_3 the number of vertices of F with at least three neighbors in F, n_{2a} the number of vertices of F with two neighbors in F and at least one neighbor in S, and n_{2b} the number of remaining vertices of F. We have $n_1 + n_{2a} + n_{2b} + n_3 = |F|$. Furthermore, $n_3 \leq n_1$ because the average degree of any forest is less than 2.

We observe that all leaves of the tree have a neighbor in S (otherwise we would have applied Lemma 5.3). This gives $|N(S) \cap F| \geq n_1 + n_{2a}$.

Furthermore, none of the n_{2b} vertices which have degree two in the tree and no neighbors in S can be connected to each other, since then Lemma 5.4 would apply. Therefore, $n_{2b} \leq n_1 + n_{2a} + n_3$. Indeed, if $n_{2b} > n_1 + n_{2a} + n_3$, then $n_{2b} > |F|/2$ and, since these n_{2b} vertices form an independent set, we would have $|E(F)| \geq 2n_{2b} > |F|$, contradicting the assumption that F is a forest.

Putting things together we get $|F| = n_1 + n_{2a} + n_{2b} + n_3 \leq 2n_1 + 2n_{2a} + 2n_3 \leq 4n_1 + 2n_{2a} \leq 4|N(S) \cap F|$. □

We note that Lemma 5.5 immediately gives an approximation algorithm with ratio $O(\Delta)$.

Lemma 5.6. *In a reduced graph G with n vertices and maximum degree Δ, every feedback vertex set has size at least $\frac{n}{5\Delta}$.*

Proof. Let S be a feedback vertex set of G and F the corresponding forest. If $|S| < \frac{n}{5\Delta}$ then $|N(S) \cap F| < \frac{n}{5}$ so by Lemma 5.5 we have $|F| < \frac{4n}{5}$. But then $|V| = |S| + |F| < n$, which is a contradiction. □

Remark. *Lemma 5.5 is tight.*

Proof. Take two copies of a rooted binary tree with n leaves and connect their roots. The resulting tree has $2n$ leaves and $2n-2$ vertices of degree 3. Subdivide every edge of this tree. Add two vertices u, v connected to every leaf. In the resulting graph $S = \{u, v\}$ is an fvs. The corresponding forest has $8n - 5$ vertices, of which $2n$ are connected to S. The graph is reduced. □

5.3.2 Polynomial Time Approximation and Extremal Results

We begin with a final intermediate lemma that allows us to construct a large minimal fvs in any reduced graph that is a forest plus one vertex.

Lemma 5.7. *Let $G = (V, E)$ be a reduced graph and $u \in V$ such that $G - u$ is acyclic. Then it is possible to construct in polynomial time a minimal feedback vertex set S of G with $|S| \geq d(u)/2$.*

Proof. Let $F = V \setminus \{u\}$. Since the graph is reduced, all trees of $G[F]$ contain at least two neighbors of u. Indeed, since each tree T of $G[F]$ has at least two leaves, both of them must be neighbors of u (otherwise Lemma 5.3 applies).

As long as there exist $v, w \in F$ with $vw \in E$ and $\{v, w\} \not\subseteq N(u)$ we contract the edge vw. Note that we can apply Lemma 5.2 as v, w have no common neighbors (u is not a common neighbor by assumption, and they cannot have a common neighbor in the forest without forming a cycle). Furthermore, this operation does not change $d(u)$. Therefore, it will be sufficient to construct a minimal fvs in the resulting graph after applying this operation exhaustively.

Suppose now that we have applied the contraction operation described above exhaustively. We eventually arrive at a graph where u is connected to all vertices of F,

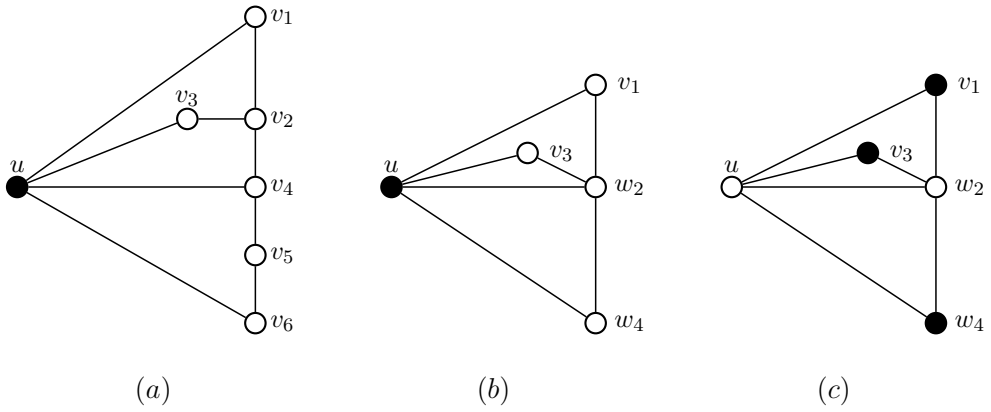


Figure 5.1: (a) vertex u is a minimal fvs of the given graph and has 4 neighbors in $G[F]$. (b) a contracted form of $G[F]$ with 4 vertices. (c) a new minimal fvs of the result graph of size 3.

as all trees of F initially contain some neighbors of u and, after repeated contractions, all non-neighbors of u are absorbed into its neighbors (more precisely, each contraction decreases $|F \setminus N(u)|$). Therefore, we arrive at a graph with $d(u) = |F|$. Furthermore, every component of F contains strictly more than one vertex.

Now, since $G[F]$ is bipartite, there is a bipartition $F = L \cup R$. Without loss of generality $|L| \leq |R|$. We return the solution $S = R$. First, S does have the promised size, as $|S| \geq |F|/2 = d(u)/2$. Second, S is an fvs, as L is an independent set, so $L \cup \{u\}$ induces a star. Finally, S is minimal, because all $v \in S$ are connected to u, and also have at least one neighbor $w \in L$, with w also connected to u. An illustration of the process is presented in Figure 5.1.

□

Theorem 5.8. *There is a polynomial time approximation algorithm for* Max Min FVS *with ratio* $O(n^{2/3})$.

Proof. We are given a graph $G = (V, E)$. We begin by applying Lemmas 5.3,5.4 exhaustively in order to obtain a reduced graph $G' = (V', E')$. Clearly, if we obtain a $|V'|^{1/3}$ approximation in G', since the reductions we applied do not change the optimal, and we can construct a solution of the same size in G, we get a $|V'|^{2/3} \leq |V|^{2/3}$ approximation ratio in G. So, in the remainder, to ease presentation, we assume G is already reduced and has n vertices.

Our algorithm begins with an arbitrary minimal fvs S. This can be constructed, for example, by starting with $S = V$ and removing vertices from S until it becomes minimal. If $|S| \geq n^{1/3}$ then we return S. Since the optimal solution cannot have size more than n, we achieve the claimed ratio.

Suppose then that $|S| < n^{1/3}$. Let F be the corresponding forest. We have $|F| > n - n^{1/3} > n/2$ for sufficiently large n. By Lemma 5.5, $|N(S) \cap F| \geq n/8$. Since $|S| < n^{1/3}$ there must exist $u \in S$ such that u has at least $\frac{n^{2/3}}{8}$ neighbors in F.

Let $w \in F \cap N(u)$. We say that w is a *good* neighbor of u if there exists $w' \in F \cap N(u)$ with $w' \neq w$ and w' is in the same tree of $G[F]$ as w. Otherwise w is a bad neighbor of u. By extension, a tree of $G[F]$ that contains a good (resp. bad) neighbor of u will be called a good (resp. bad) tree. Every vertex of $N(u) \cap F$ is either good or bad.

We have argued that $|N(u) \cap F| \geq \frac{n^{2/3}}{8}$. We distinguish two cases: either u has at least $\frac{n^{2/3}}{16}$ good neighbors in F, or it has at least that many bad neighbors in F.

In the former case, we delete from the graph the set $S \setminus \{u\}$ and apply Lemmas 5.3, 5.4 exhaustively again. We claim that the number of good neighbors of u does not decrease in this process. Indeed, two good neighbors of u cannot be contracted using Lemma 5.4, since they have a common neighbor (namely u). Furthermore, suppose w is the first good neighbor of u to be deleted using Lemma 5.3. This would mean that w currently has no other neighbor except u. However, since w is good, initially there was a $w' \in N(u)$ in the same tree of $G[F]$ as w. The vertex w' has not been deleted (since we assumed w is the first good neighbor to be deleted). Furthermore, Lemmas 5.3, 5.4 cannot disconnect two vertices which are in the same component, so we get a contradiction. We therefore have a reduced graph, where $\{u\}$ is an fvs, and $d(u) \geq \frac{n^{2/3}}{16}$. By Lemma 5.7 we obtain a minimal fvs of size at least $\frac{n^{2/3}}{32}$, which is an $O(n^{1/3})$ approximation.

In the latter case, u has at least $\frac{n^{2/3}}{16}$ bad neighbors in F. Consider a bad tree T. We claim that T must have a neighbor in $S \setminus \{u\}$, because T has at least two leaves, at most one of which is a neighbor of u (since T is bad). If the second leaf is not connected to S, it will be deleted by Lemma 5.3. Furthermore, since u is connected to one vertex in each bad tree, u is connected to at least $\frac{n^{2/3}}{16}$ bad trees.

We now find the vertex $v \in S \setminus \{u\}$ such that v is connected to the maximum number of bad trees connected to u. Since $|S| \leq n^{1/3}$, v must be connected to at least $\frac{n^{1/3}}{16}$ bad trees connected to u. We now delete from the graph the set $S \setminus \{u, v\}$ as well as all trees of $G[F]$, except the bad trees connected to u, v. Furthermore, in each bad tree T connected to both u, v let $u' \in T \cap N(u)$ and $v' \in T \cap N(v)$ such that u', v' are as close as possible in T (note that perhaps $v' = u'$). We delete all vertices of the tree T except those on the path from v' to u'. Then, we contract all internal edges of this path (note that internal vertices of the path are not connected to $\{u, v\}$ by the selection of u', v'). It is not hard to verify that, by using Lemmas 5.1, 5.2, if we are able to produce a large minimal fvs in the resulting graph, we obtain a solution for G. Furthermore, in the resulting graph, every bad tree T connected to u, v has been reduced to a single vertex connected to u, v. So the graph is now either a $K_{2,s}$, with $s \geq \frac{n^{1/3}}{16}$, or the same graph with the addition of the edge uv. In either case, it is not hard to see that starting with the fvs that contains all vertices except $\{u, v\}$, and making it minimal, we obtain a solution of size at least $s - 1$ which gives an approximation ratio of $O(n^{2/3})$. □

Corollary 5.9. *For any reduced graph G on n vertices we have* $\text{mmfvs}(G) = \Omega(n^{1/3})$.

Proof. We simply note that the algorithm of Theorem 5.8 always constructs a solution of size at least $\frac{n^{1/3}}{c}$, where c is a small constant, assuming that the original n-vertex graph G was reduced. □

Remark. *Corollary 5.9 is tight.*

Proof. Take a K_n and for every pair of vertices u, v in the clique, add $2n$ new vertices connected only to u, v. The graph has order $n + 2n\binom{n}{2} = n + n^2(n-1) = n^3 - n^2 + n \geq n^3/2$. Any minimal fvs of this graph must contain at least $n-2$ vertices of the clique. As a result its maximum size is at most $n - 2 + 2n \leq 3n$. We have $\frac{\text{mmfvs}(G)}{|V(G)|} \leq \frac{6n}{n^3} = O(\frac{1}{n^2})$ therefore $\text{mmfvs}(G) = O(|V(G)|^{1/3})$. □

Theorem 5.8 also implies the existence of a cubic kernel of Max Min FVS when parameterized by the solution size k. Recall that the reduction rules do not change the solution size. We suppose that the reduced graph has n vertices. For a small constant c, if $n \geq c^3k^3$, then we can always produce a solution of size at least $n^{1/3}/c = k$, and thus the answer is YES. Otherwise, we have a cubic kernel.

Corollary 5.10. *Max Min FVS admits a cubic kernel when parameterized by the solution size.*

Finally, we remark that a similar combinatorial point of view can be taken for the related problem of Max Min VC, giving another intuitive explanation for the difference in approximability between the two problems.

Remark. *Any graph $G = (V, E)$ without isolated vertices, has a minimal vertex cover of size at least $\sqrt{|V|}$, and this is asymptotically tight.*

Proof. We will prove the statement under the assumption that G is connected. If not, we can treat each component separately. If the components of G have sizes $n_1, \ldots, n_k$, then we rely on the fact that $\sum_{i=1}^{k} \sqrt{n_i} \geq \sqrt{\sum_{i=1}^{k} n_i}$ and that the union of the minimal vertex covers of each component is a minimal vertex cover of G.

If $G = (V, E)$ has a vertex u of degree at least $\sqrt{n}$, then we begin with the vertex cover $V \setminus \{u\}$ and remove vertices until it becomes minimal. In the end, our solution contains a superset of $N(u)$, therefore we have a minimal vertex cover of size at least $\sqrt{n}$ as promised. If, on the other hand, $\Delta(G) < \sqrt{n}$, then any vertex cover of G must have size at least $\sqrt{n}$. Indeed, a vertex cover of size at most $\sqrt{n} - 1$ can cover at most $(\sqrt{n} - 1)\sqrt{n} < n - 1$ edges, but since G is connected we have $|E(G)| \geq n - 1$. So, in this case, any minimal vertex cover has the promised size.

To see that the bound given is tight, take a K_n and attach n leaves to each of its vertices. This graph has $n^2 + n$ vertices, but any minimal vertex cover has size at most $2n$. □

5.4 Sub-exponential Time Approximation

In this section we give an approximation algorithm that generalizes our $n^{2/3}$-approximation and is able to guarantee any desired performance, at the cost of increased running time. On a high level, our initial approach again constructs an arbitrary minimal fvs S and if S is clearly large enough, returns it. However, things become more complicated from

then on, as it is no longer sufficient to consider vertices of S individually or in pairs. We therefore need several new ideas, one of which is given in the following lemma, which states that we can find a constant factor approximation in time exponential in the size of a given fvs. This will be useful as we will use the assumption that S is "small" and then cut it up into even smaller pieces to allow us to use Lemma 5.11.

Lemma 5.11. *Given a graph $G = (V, E)$ on n vertices and a feedback vertex set $S \subseteq V$ of size k, it is possible to produce a minimal fvs S' of G of size $|S'| \geq \frac{\text{mmfvs}(G)}{3}$ in time $n^{O(k)}$.*

Proof. Before we begin, let us point out that for $k = 1$, Max Min FVS can be solved optimally in time $O(n)$, using standard arguments from parameterized complexity, namely the fact that in this case G has treewidth 2, and invoking Courcelle's theorem, since the properties "S is an fvs" and "S is minimal" are MSO-expressible [65]. Unfortunately, this type of argument is not good enough for larger values of k, as the running time guaranteed by Courcelle's theorem could depend super-exponentially on k. We could try to avoid this by formulating a treewidth-based DP algorithm to obtain a better running time, but we prefer to give a simpler more direct branching algorithm, since this is good enough for Theorem 5.12.

We will assume that S is minimal (if not, we can remove vertices from it to make it minimal and this only decreases the available running time of our algorithm). As a result, we assume that $\text{mmfvs}(G) \geq 3k$, as otherwise S is already a 3-approximation.

Let S_{OPT} be a maximum minimal fvs of G, and $F_{OPT} = V \setminus S_{OPT}$. We formulate an algorithm that maintains two disjoint sets of vertices S_{SOL}, F_{SOL} which, intuitively, correspond to vertices we have decided to place in the fvs or the induced forest, respectively. We will denote $U := V \setminus (S_{SOL} \cup F_{SOL})$ the set of undecided vertices. Our algorithm will be non-deterministic, that is, it will sometimes "guess" some vertices of U that will be placed in S_{SOL} or F_{SOL}. We will bound the total number of guessing possibilities by $n^{O(k)}$, which will imply that the algorithm can be made deterministic by trying all possibilities for every guess and returning the best returned solution.

Throughout the algorithm, we will work to maintain the following invariants:

1. $S_{SOL} \cup F_{SOL}$ is an fvs of G.
2. $S_{SOL} \subseteq S_{OPT}$ and $F_{SOL} \subseteq F_{OPT}$.
3. $G[F_{SOL}]$ is acyclic and has at most $2k$ components.
4. All vertices of S_{SOL} have at least two neighbors in F_{SOL}.

To begin, we guess a set $F' \subseteq S$ such that $G[F']$ is acyclic and set $F_{SOL} = F'$ and $S_{SOL} = S \setminus F'$. Property 1 is satisfied as $F_{SOL} \cup S_{SOL} = S$. Property 2 is satisfied for the guess $F' = F_{OPT} \cap S$. If there exists $u \in S_{SOL}$ which does not satisfy Property 4, we guess one or two vertices from $N(u) \cap U$ and place them into F_{SOL} so that u has two neighbors in F_{SOL}. Since u has a private cycle in $G[F_{OPT}]$, if the vertices we guessed

are the neighbors of u in that cycle, we maintain Property 2. We continue in this way until Property 4 is satisfied. We now observe that F_{SOL} is acyclic (as $F_{SOL} \subseteq F_{OPT}$), and that since we have added at most two vertices for each vertex of S_{SOL}, it contains at most $2k$ vertices, hence at most $2k$ components, so we satisfied Property 3. So far, the total number of possible guesses is upper-bounded by $2^k n^{2k}$: 2^k for guessing F' and n^{2k} for guessing at most two neighbors for each $u \in S_{SOL}$.

We will now say that a "connector" is a path $P \subseteq F_{OPT} \setminus F_{SOL}$, such that $G[F_{OPT} \cup P]$ has strictly fewer components that $G[F_{SOL}]$. Our algorithm will now repeateadly guess if a connector exists, and if it does it will guess the first and last vertex u, v of P. Note that $u, v \in U$ and if we guess u, v correctly we can infer all of P, as $G[U]$ is acyclic, so there is at most one path from u to v in $G[U]$. We set $F_{SOL} := F_{SOL} \cup P$ and continue guessing, until we guess that no connector exists. Observe that guessing the endpoints of a connector gives n^2 possibilities, and that adding a connector to F_{SOL} decreases the number of connected components of F_{SOL}, which can happen at most $2k$ times by Property 3. So we have a total of $n^{O(k)}$ possible guesses and for the correct guess Property 2 is maintained.

We now consider every vertex of $u \in U$ that has at least two neighbors in F_{SOL} and place all such vertices in S_{SOL}. Properties 1, 3, and 4 are trivially still satisfied. Furthermore, if our guesses so far are correct, all such vertices u belong in S_{OPT}, as they either already have a private cycle in F_{OPT}, or if they have neighbors in distinct components of F_{SOL}, they would function as connectors in F_{OPT} (and we assume we have correctly guessed that no more connectors exist).

We are now in a situation where every vertex of U has at most one neighbor in F_{SOL}. We construct a new graph H by deleting from G all of S_{SOL} and replacing F_{SOL} by a single vertex f that is connected to $N(F_{SOL})$. Note that H is a simple graph (it has no parallel edges) with an fvs of size 1 (as $H - w$ is acyclic). We therefore use the aforementioned algorithm implied by Courcelle's theorem to produce a maximum minimal fvs of H which, without loss of generality, does not contain w. Let $S^* \subseteq U$ be this set. In G, we check if $S_{SOL} \cup S^*$ is an fvs. If it is we delete vertices from it (if necessary) to make it redundant and return the resulting set S^{**}, which is a minimal fvs.

To see that the resulting solution has the desired size we focus on the case where all guesses were correct and therefore Properties 1-4 were maintained throughout the execution of the algorithm. As mentioned, since the total number of possibilities considered in $n^{O(k)}$, a deterministic algorithm can simply try out all possible choices and return the best solution.

We first observe that $\text{mmfvs}(H) \geq \text{mmfvs}(G) - |S_{SOL}|$, where S_{SOL} is the set of vertices we deleted from G to obtain H. Indeed, $S_0 := S_{OPT} \setminus S_{SOL}$ is a minimal fvs of H. To see that S_0 is an fvs, suppose that H contains a cycle after deleting S_0. This cycle must necessarily go through w. Let P be the vertices of this cycle except w. We have $P \subseteq U \setminus S_{OPT}$ therefore, $P \subseteq F_{OPT}$. However, this means either that P forms a cycle with a component of F_{SOL} (which contradicts the acyclicity of F_{OPT} by Property 2), or that P is a connector, which contradicts our guess that no other connector exists.

Therefore, S_0 must be an fvs of H. To see that it is minimal we note that for all $u \in S_0$ there exists a private cycle in $G[U \cup F_{SOL} \cup \{u\}]$, and this cycle is not destroyed by contracting the vertices of F_{SOL} into w.

We now have that $|S^* \cup S_{SOL}| \geq |S_{OPT}|$, because $|S^*| \geq |S_{OPT} \setminus S_{SOL}|$. We argue that in the process of making S^* minimal to obtain S^{**} we delete at most $2k$ vertices. Indeed, every time a vertex u of S_{SOL} is removed from $S^* \cup S_{SOL}$ as redundant, since u has at least two neighbors in F_{SOL} by Property 4, the number of components of $G[F_{SOL}]$ must decrease. Similarly, if we remove a vertex $u \in S^*$ as redundant, we consider the private cycle of u in $H \setminus S^*$. All of the vertices of this cycle are present in G after we delete S^*, except w, therefore, this cycle forms a path between two distinct components of $G[F_{SOL}]$. We conclude that, since removing a vertex from our fvs decreases the number of connected components of $G[F_{SOL}]$, by Property 3 we have $|S^{**}| \geq |S_{OPT}| - 2k$. But recall that we have assumed that $k \leq \frac{S_{OPT}}{3}$ (otherwise S was already a sufficiently good approximation), so we have $|S^{**}| \geq \frac{\text{mmfvs}(G)}{3}$. □

Theorem 5.12. *There is an algorithm which, given an n-vertex graph $G = (V, E)$ and a value r, produces an r-approximation for* Max Min FVS *in G in time $n^{O(n/r^{3/2})}$.*

Proof. First, let us note that we may assume that r is $\omega(1)$, because if r is bounded by a constant, then we can solve the problem exactly in the given time. To ease presentation, we will give an algorithm with approximation ratio $O(r)$. A ratio of exactly r can be obtained by multiplying r with an appropriate (small) constant.

Our algorithm borrows several of the basic ideas from Theorem 5.8, but requires some new ingredients (including Lemma 5.11). The first step is, again, to construct a minimal fvs S in some arbitrary way, for example by setting $S = V$ and then removing vertices from S until it becomes minimal. If $|S| \geq n/r$ we are done, as we already have an r-approximation, so we simply return S. From this point, this algorithm departs from the algorithm of Theorem 5.8, because it is no longer sufficient to compare the size of the returned solution with a function of n (we need to compare it to the actual optimal in order to obtain a ratio of r), and because we need to partition S into non-trivial parts that contain more than one vertex. The algorithm proceeds as follows:

Let $k = \lceil \sqrt{r} \rceil$ and partition S into k parts of (almost) equal size $S_1, \ldots, S_k$. Our algorithm proceeds as follows: for each $i, j \in \{1, \ldots, k\}$ (not necessarily distinct) consider the graph $G_{i,j}$ obtained by deleting all vertices of $S \setminus (S_i \cup S_j)$. Compute, using Lemma 5.11 a solution for $G_{i,j}$, taking into account that $S_i \cup S_j$ is a feedback vertex set of this graph. Output the largest of the solutions found, using Lemma 5.1 to transform them into solutions of G (or output S if it is larger than all solutions).

The algorithm clearly runs in the promised time: $|S_i \cup S_j| \leq \frac{2n}{rk}$, so the algorithm of Lemma 5.11 takes time $n^{O(n/r^{3/2})}$ and is executed a polynomial number of times.

Let us now analyze the approximation ratio of the produced solution. Let S_{OPT} be an optimal solution and let $F := V \setminus S$ and $F_{OPT} = V \setminus S_{OPT}$ be the induced forests corresponding to S and to the optimal solution. We would like to argue that one of the considered subproblems contains at least a $\frac{1}{r}$ fraction of S_{OPT} and that most (though not all) of these vertices form part of a minimal fvs of that subgraph.

To be more precise, we will define the notion of "type" for each $u \in S_{OPT} \cap F$. For each such u there must exist a cycle in the graph $G[F_{OPT} \cup \{u\}]$ (if not, this would contradict the minimality of S_{OPT}). Call this cycle $c(u)$ (select one such cycle arbitrarily if several exist). The cycle $c(u)$ must intersect S, as S is an fvs. Let v be the vertex of $c(u) \cap S$ closest to u on the cycle. Let v' be the vertex of $c(u) \cap S$ that is closest to u if we traverse the cycle in the opposite direction (note that v, v' are not necessarily distinct). Suppose that $v \in S_i, v' \in S_j$ and without loss of generality $i \leq j$. We then say that $u \in S_{OPT} \cap F$ has type (i, j). In this way, we define a type for each $u \in S_{OPT} \cap F$. Note that according to our definition, all internal vertices of the path in $c(u)$ from u to v (and also from u to v') belong in $F_{OPT} \cap F$.

According to the definition of the previous paragraph, there are $k(k+1)/2 \leq r$ possible types of vertices in $S_{OPT} \cap F$. Therefore, there must be a type (i, j) such that at least $\frac{|S_{OPT} \cap F|}{r}$ vertices have this type. We now concentrate on the graph $G_{i,j}$, for the type (i, j) which satisfies this condition. Our algorithm constructed $G_{i,j}$ by deleting all of S except $S_i \cup S_j$. We would like to claim that this graph has a minimal feedback vertex set of size comparable to $\frac{|S_{OPT} \cap F|}{r}$.

For the sake of the analysis, construct a minimal feedback vertex set S^* of $G_{i,j}$ as follows: we begin with the fvs $S^* = S_{OPT} \cap (F \cup S_i \cup S_j)$ and the corresponding induced forest $F^* = F_{OPT} \cap (F \cup S_i \cup S_j)$. The set S^* is a feedback vertex set as it contains all vertices of S_{OPT} found in $G_{i,j}$ and S_{OPT} is a feasible feedback vertex set of all of G. We then make S^* minimal by arbitrarily removing redundant vertices. Call the resulting set $S^{**} \subseteq S^*$ and the corresponding induced forest $F^{**} \supseteq F^*$.

Our main claim now is that the number of vertices of $S^* \cap F$ of type (i, j) which were "lost" in the process of making S^* minimal, is upper-bounded by $|S_i \cup S_j|$. Formally, we claim that $|\{u \in (S^* \cap F) \setminus S^{**} \mid u \text{ has type } (i, j)\}| \leq |S_i \cup S_j|$. Indeed, consider such a vertex $u \in (S^* \cap F) \setminus S^{**}$ of type (i, j), let $c(u)$ be the cycle that defines its type and v, v' the vertices of $S_i \cup S_j$ which are closest to u on the cycle in either direction. All vertices of $c(u)$ in the paths from u to v and from u to v' belong to $F_{OPT} \cap F$, therefore also to F^*. If u was removed as redundant, this means that v, v' must have been in distinct connected components at the moment u was removed from the feedback vertex set (and also that v, v' are distinct). However, the addition of u to the induced forest creates a path from v to v' in the induced forest and hence decreases the number of connected components (that is, trees in the induced forest) containing vertices of $S_i \cup S_j$. The number of such connected components cannot decrease more than $|S_i \cup S_j|$ times, therefore, during the process of making S^* minimal we may have removed at most $|S_i \cup S_j|$ vertices of type (i, j) from $S^* \cap F$.

Using the above analysis and the assumption that S^* contains at least $\frac{|S_{OPT} \cap F|}{r}$ vertices of type (i, j), we conclude that $\text{mmfvs}(G_{i,j}) \geq |S^{**}| \geq \frac{|S_{OPT} \cap F|}{r} - |S_i \cup S_j|$. We now note that if $|S_{OPT} \cap S| \geq \frac{|S_{OPT}|}{r}$, then S is already an r-approximation, so it is safe to assume $|S_{OPT} \cap F| \geq \frac{(r-1)|S_{OPT}|}{r}$. Furthermore, $|S_i \cup S_j| \leq \frac{2|S|}{\sqrt{r}} \leq \frac{2|S_{OPT}|}{r\sqrt{r}}$, where again we are assuming that S is not already an r-approximation. Putting things together we get $\text{mmfvs}(G_{i,j}) \geq \frac{(r-1)|S_{OPT}|}{r^2} - \frac{2|S_{OPT}|}{r\sqrt{r}} \geq \frac{|S_{OPT}|}{2r}$, for sufficiently large r. Hence, since

the algorithm will return a solution that is at least as large as $\frac{\text{mmfvs}(G_{i,j})}{3}$, we obtain an $O(r)$-approximation. □

5.5 Hardness of Approximation and NP-hardness

In this section we establish lower bound results showing that the approximation algorithms given in Theorems 5.8 and 5.12 are essentially optimal, under standard complexity assumptions.

5.5.1 Hardness of Approximation in Polynomial Time

We begin by showing that the best approximation ratio achievable in polynomial time is indeed (essentially) $n^{2/3}$. For this, we rely on the celebrated result of Håstad on the hardness of approximating Max Independent Set, which was later derandomized by Zuckerman, cited below.

Theorem 5.13. *[106, 183] For any $\epsilon > 0$, there is no polynomial time algorithm which approximates Max Independent Set with a ratio of $n^{1-\epsilon}$, unless $P = NP$.*

Starting from this result, we present a reduction to Max Min FVS.

Theorem 5.14. *For any $\epsilon > 0$, Max Min FVS is inapproximable within a factor of $n^{2/3-\epsilon}$ unless $P = NP$.*

Proof. We give a gap-preserving reduction from Max Independent Set, which cannot be approximated within a factor of $n^{1-\epsilon}$, unless P = NP. We are given a graph $G = (V, E)$ on n vertices as an instance of Max Independent Set. Recall that $\alpha(G)$ denotes the size of the maximum independent set of G.

We transform G into an instance of Max Min FVS as follows: For every pair of $u, v \in V$, we add n vertices such that they are adjacent only to u and v. We denote by I_{uv} the set of such vertices. Then I_{uv} is an independent set. Let $G' = (V', E')$ be the constructed graph.

We now make the following two claims:

Claim 5.15. $\text{mmfvs}(G') \geq (n-1)\binom{\alpha(G)}{2}$

Proof. We construct a minimal fvs of G' as follows: let C be a minimum vertex cover of G. Then we begin with the set that contains C and the union of all I_{uv} (which is clearly an fvs) and remove vertices from it until it becomes minimal. Let S be the final minimal fvs. We observe that for all $u, v \in V \setminus C$, S contains at least $n-1$ of the vertices of I_{uv}. Since C is a minimum vertex cover of G, there are $\binom{\alpha(G)}{2}$ pairs $u, v \in V \setminus C$. □

Claim 5.16. $\text{mmfvs}(G') \leq n\binom{2\alpha(G)}{2} + n$

Proof. Let S be a minimal fvs of G' and F be the corresponding forest. It suffices to show that $|S \setminus V| \leq n\binom{2\alpha(G)}{2}$, since $|S \cap V| \leq n$. Consider now a set I_{uv}. If $u \in S$ or $v \in S$, then $I_{uv} \cap S = \emptyset$, because all vertices of I_{uv} have at most one neighbor in F, and are therefore redundant. So, I_{uv} contains (at most n) vertices of S only if $u, v \in F$. However, $|F \cap V| \leq 2\alpha(G)$, because F is bipartite, so $F \cap V$ induces two independent sets, both of which must be at most equal to the maximum independent set of G. So the number of pairs $u, v \in F \cap V$ is at most $\binom{2\alpha(G)}{2}$ and since each corresponding I_{uv} has size n, we get the promised bound. □

The two claims together imply that there exist constants c_1, c_2 such that (for sufficiently large n) we have $c_1 n(\alpha(G))^2 \leq \text{mmfvs}(G') \leq c_2 n(\alpha(G))^2$. That is, $\text{mmfvs}(G') = \Theta(n(\alpha(G))^2)$.

Suppose now that there exists a polynomial-time approximation algorithm which, given a graph G', produces a minimal fvs S with the property $\frac{\text{mmfvs}(G')}{r} \leq |S| \leq \text{mmfvs}(G')$, that is, there exists an r-approximation for Max Min FVS. Running this algorithm on the instance we constructed, we obtain that $\frac{c_1 n(\alpha(G))^2}{r} \leq |S| \leq c_2 n(\alpha(G))^2$. Therefore, $\frac{\alpha(G)}{\sqrt{rc_2/c_1}} \leq \sqrt{\frac{|S|}{c_2 n}} \leq \alpha(G)$. As a result, we obtain an $O(\sqrt{r})$ approximation for the value of $\alpha(G)$. We therefore conclude that, unless P = NP, any such algorithm must have $\sqrt{r} > n^{1-\epsilon}$, for any $\epsilon > 0$, hence, $r > n^{2-\epsilon}$, for any $\epsilon > 0$. Since the graph G' has $N = \Theta(n^3)$ vertices, we get that no approximation algorithm can achieve a ratio of $N^{2/3-\epsilon}$. □

We notice that in the construction of the previous theorem, the maximum degree of the graph is approximately equal to the approximation gap. Thus, the following corollary also holds.

Corollary 5.17. *For any positive constant ϵ, Max Min FVS is inapproximable within a factor of $\Delta^{1-\epsilon}$ unless P = NP.*

5.5.2 Hardness of Approximation in Sub-Exponential Time

In this section we extend Theorem 5.14 to the realm of sub-exponential time algorithms. We recall the following result of Chalermsook et al.

Theorem 5.18. *[47] For any $\epsilon > 0$ and any sufficiently large r, if there exists an r-approximation algorithm for Max Independent Set running in $2^{(n/r)^{1-\epsilon}}$, then the randomized ETH is false.*

We remark that Theorem 5.18, which gives an almost tight running time lower bound for Max Independent Set, has already been used as a starting point to derive a similarly tight bound for the running time of any sub-exponential time approximation for Max Min VC. Here, we modify the proof of Theorem 5.14 to obtain a similarly tight result for Max Min FVS. Nevertheless, the reduction for Max Min FVS is significantly more challenging, because the ideas used in Theorem 5.14 involve an inherent quadratic (in n) blow-up of the size of the instance. As a result, in addition to executing an

appropriately modified version of the reduction of Theorem 5.14, we are forced to add an extra "sparsification" step, and use a probabilistic analysis with Chernoff bounds to argue that this step does not destroy the inapproximability gap.

Theorem 5.19. *For any $\epsilon > 0$ and any sufficiently large r, if there exists an r-approximation algorithm for Max Min FVS running in $2^{(n/r^{3/2})^{1-\epsilon}}$, then the randomized ETH is false.*

Proof. We recall some details about the reduction used to prove Theorem 5.18. The reduction of [47] begins from a 3-SAT instance ϕ on n variables, and for any ϵ, r, constructs a graph G with $n^{1+\epsilon}r^{1+\epsilon}$ vertices which (with high probability) satisfies the following properties: if ϕ is satisfiable, then $\alpha(G) \geq n^{1+\epsilon}r$; otherwise $\alpha(G) \leq n^{1+\epsilon}r^{2\epsilon}$. Hence, any approximation algorithm with ratio $r^{1-2\epsilon}$ for Max Independent Set would be able to distinguish between the two cases (and solve the initial 3-SAT instance). If, furthermore, this algorithm runs in $2^{(|V|/r)^{1-2\epsilon}}$, we get a sub-exponential algorithm for 3-SAT.

Suppose we are given ϵ, r, and we want to prove the claimed lower bound on the running time of any algorithm that r-approximates Max Min FVS. To ease presentation, we will assume that r is the square of an integer (this can be achieved without changing the value of r by more than a small constant). We will also perform a reduction from 3-SAT to show that an algorithm that achieves this ratio too rapidly would give a sub-exponential (randomized) algorithm for 3-SAT. We begin by executing the reduction of [47], starting from a 3-SAT instance ϕ on n variables, but adjusting their parameter r appropriately so we obtain a graph G with the following properties (with high probability):

- $|V(G)| = n^{1+\epsilon}r^{1/2+\epsilon}$
- If ϕ is satisfiable, then $\alpha(G) \geq n^{1+\epsilon}r^{1/2}$
- If ϕ is not satisfiable, then $\alpha(G) \leq n^{1+\epsilon}r^{2\epsilon}$

We now construct a graph G' as follows: for each pair $u, v \in V(G)$, we introduce an independent set I_{uv} of size $\sqrt{r}$ connected to u, v. We claim that G' has the following properties (assuming G has the properties cited above):

- $|V(G')| = \Theta(n^{2+2\epsilon}r^{3/2+2\epsilon})$
- If ϕ is satisfiable, then $\text{mmfvs}(G') = \Omega(n^{2+2\epsilon}r^{3/2})$
- If ϕ is not satisfiable, then $\text{mmfvs}(G') = O(n^{2+2\epsilon}r^{1/2+4\epsilon})$

Before proceeding, let us establish the properties mentioned above. The size of $|V(G')|$ is easy to bound, as for each of the $\binom{|V(G)|}{2}$ pairs of vertices of G we have constructed an independent set of size $\sqrt{r}$. If ϕ is satisfiable, we construct a minimal fvs of G' by starting with a minimum vertex cover of G to which we add all vertices of all

I_{uv}. We then make this fvs minimal. We claim that for each I_{uv} for which $u, v \in V \setminus C$, our set will in the end contain all of I_{uv}, except maybe at most one vertex. Furthermore, if one vertex of I_{uv} is removed from the fvs as redundant, this decreases the number of components of the induced forest that contain vertices of V (as u, v are now in the same component). This cannot happen more than $|V(G)|$ times. The number of I_{uv} with $u, v \in V \setminus C$ is $\binom{\alpha(G)}{2} = \Omega(n^{2+2\epsilon}r)$. So, mmfvs$(G') = \Omega(n^{2+2\epsilon}r^{3/2} - |V(G)|)$.

For the third property, take any minimal fvs S of G' and let F be the corresponding forest. We have $|F \cap V| \leq 2\alpha(G)$, because F is bipartite. It is sufficient to bound $|S \setminus V|$ to obtain the bound (as $|S \cap V|$ is already small enough). To do this, we note that in a set I_{uv} where u, v are not both in F, we have $I_{uv} \cap S = \emptyset$, as all vertices of I_{uv} are redundant. So, the number of sets I_{uv} which contribute vertices to S is at most $\binom{|F \cap V|}{2} = O(n^{2+2\epsilon}r^{4\epsilon})$. Each such set has size $\sqrt{r}$, giving the claimed bound.

We have now constructed an instance where the gap between the values for mmfvs(G'), depending on whether ϕ is satisfiable, is almost r (in fact, it is $r^{1-4\epsilon}$, but we can make it equal to r by adjusting the parameters accordingly). The problem is that the order of the new graph depends quadratically on n. This blow-up makes it impossible to obtain a running time lower bound, as a fast approximation algorithm for Max Min FVS (say with running time $2^{n/r^2}$) would not result in a sub-exponential algorithm for 3-SAT. We therefore need to "sparsify" our instance.

We construct a graph G'' by taking G' and deleting every vertex of $V(G') \setminus V(G)$ with probability $\frac{n-1}{n}$. That is, every vertex of the independent sets I_{uv} we added survives (independently) with probability $1/n$. We now claim the following properties hold with high probability:

- $|V(G'')| = \Theta(n^{1+2\epsilon}r^{3/2+2\epsilon})$
- If ϕ is satisfiable, then mmfvs$(G'') = \Omega(n^{1+2\epsilon}r^{3/2})$
- If ϕ is not satisfiable, then mmfvs$(G'') = O(n^{1+2\epsilon}r^{1/2+4\epsilon})$

Before we proceed, let us explain why if we establish that G'' satisfies these properties, then we obtain the theorem. Indeed, suppose that for some sufficiently large r and $\epsilon > 0$, there exists an approximation algorithm for Max Min FVS with ratio $r^{1-5\epsilon}$ running in time $2^{(N/r^{3/2})^{1-10\epsilon}}$ for graphs with N vertices. The algorithm has sufficiently small ratio to distinguish between the two cases in our constructed graph G'', as the ratio between mmfvs(G'') when ϕ is satisfiable or not is $\Omega(r^{1-4\epsilon})$ (and r is sufficiently large), so we can use the approximation algorithm to solve 3-SAT. Furthermore, to compute the running time we see that $N/r^{3/2} = \Theta(n^{1+2\epsilon}r^{2\epsilon}) = O(n^{1+4\epsilon})$. Therefore, $(N/r^{3/2})^{1-10\epsilon} = o(n)$ and we get a sub-exponential time algorithm for 3-SAT. We conclude that for any sufficiently large r and any $\epsilon > 0$, no algorithm achieves ratio $r^{1-5\epsilon}$ in time $2^{(N/r^{3/2})^{1-10\epsilon}}$. By adjusting r, ϵ appropriately we get the statement of the theorem.

Let us therefore try to establish that the three claimed properties all hold with high probability. We will use the following standard Chernoff bound: suppose $X = \sum_{i=1}^{n} X_i$ is the sum of n independent random 0/1 variables X_i and that $E[X] = \sum_{i=1}^{n} E[X_i] = \mu$. Then, for all $\delta \in (0, 1)$ we have $Pr[|X - \mu| \geq \delta\mu] \leq 2e^{-\mu\delta^2/3}$

The first property is easy to establish: we define a random variable X_i for each vertex of each I_{uv} of G'. This variable takes value 1 if the corresponding vertex appears in G'' and 0 otherwise. Let X be the sum of the X_i variables, which corresponds to the number of such vertices appearing in G''. Suppose that the number of vertices in sets I_{uv} of G' is $cn^{2+2\epsilon}r^{3/2+2\epsilon}$, where c is a constant. Then, $E[X] = cn^{1+2\epsilon}r^{3/2+2\epsilon}$. Also, $Pr[|X - E[X]| \geq \frac{E[X]}{2}] \leq 2e^{-E[X]/12} = o(1)$. So with high probability, $|V(G'')|$ is of the promised magnitude.

The second property is also straightforward. This time we consider a maximum minimal fvs S of G' of size $cn^{2+2\epsilon}r^{3/2}$. Again, we define an indicator variable for each vertex of this set in sets I_{uv}. The expected number of such vertices that survive in G'' is $cn^{1+2\epsilon}r^{3/2}$. As in the previous paragraph, with high probability the actual number will be close to this bound. We now need to argue that (almost) the same set is a minimal fvs of G''. We start in G'' with (the surviving vertices of) S, which is clearly an fvs of G'', and delete vertices until the set is minimal. We claim that the size of the set will decrease by at most $|V(G)| = n^{1+\epsilon}r^{1+\epsilon}$. Indeed, if $S \cap I_{uv} \neq \emptyset$, then $u, v \notin S$. The two vertices u, v are (deterministically) included in G'' and start out in the corresponding induced forest in our solution. If a vertex of $S \cap I_{uv}$ is deleted as redundant, placing that vertex in the forest will put u, v in the same component, reducing the number of components of the forest with vertices from $|V(G)|$. This can happen at most $|V(G)|$ times. Since $|V(G)| < \frac{c}{10}(n^{1+2\epsilon}r^{3/2})$ (for n, r sufficiently large), deleting these redundant vertices will not change the order of magnitude of the solution.

Finally, in order to establish the third property we need to consider every possible minimal fvs of G'' and show that none of them end up being too large. Consider a set $F \subseteq V(G)$ that induces a forest in G. Our goal is to prove that any minimal fvs S of G'' that satisfies $V(G) \setminus S = F$ has a probability of being "too large" (that is, violating our claimed bound) much smaller than $2^{-|V(G)|}$. If we achieve this, then we can take a union bound over all sets F and conclude that with high probability no minimal fvs of G'' is too large.

Suppose then that we have fixed an acyclic set $F \subseteq V(G)$. We have $|F| \leq 2\alpha(G) \leq 2n^{1+\epsilon}r^{2\epsilon}$. Any minimal fvs with $V(G) \setminus S = F$ can only contain vertices from a set I_{uv} if $u, v \in F$. The total number of such vertices in G' is at most $O(n^{2+2\epsilon}r^{1/2+4\epsilon})$. The expected number of such vertices that survive in G'' is (for some constant c) at most $\mu = cn^{1+2\epsilon}r^{1/2+4\epsilon}$. Now, using the Chernoff bound cited above we have $Pr[|X - \mu| \geq \frac{\mu}{2}] \leq 2e^{-\mu/12}$. We claim $2e^{-\mu/12} = o(2^{-|V(G)|})$. Indeed, this follows because $|V(G)| = n^{1+\epsilon}r^{1/2+\epsilon} = o(\mu)$. As a result, the probability that a large minimal fvs exists for a fixed set $F \subseteq V(G)$ exists is low enough that taking the union bound over all possible sets F we have that with high probability no minimal fvs exists with value higher than $3\mu/2$, which establishes the third property. □

5.5.3 NP-hardness for $\Delta = 6$

Theorem 5.20. *Max Min FVS is NP-hard on planar bipartite graphs with $\Delta = 6$.*

Proof. We give a reduction from Max Min VC, which is NP-hard on planar bipartite

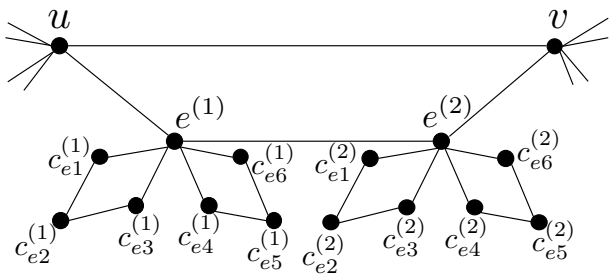


Figure 5.2: The edge gadget of $e = (u, v)$ in the constructed graph G.

graphs of maximum degree 3 [184]. Note that the NP-hardness in [184] is stated for Minimum Independent Dominating Set, but any independent dominating set is also a maximal independent set (and vice-versa) and the complement of the minimum maximal independent set of any graph is a maximum minimal vertex cover. Thus, we also obtain NP-hardness for Max Min VC on the same instances.

We are given a graph $G = (V, E)$. For each edge $e = (u, v) \in E$, we add a path of length three from u to v going through two new vertices $e^{(1)}, e^{(2)}$ (see Figure 5.2). Note that $u, e^{(1)}, e^{(2)}, v$ form a cycle of length 4. Then we add two cycles of length 4, $e^{(i)}, c^{(i)}_{e1}, c^{(i)}_{e2}, c^{(i)}_{e3}$ and $e^{(i)}, c^{(i)}_{e4}, c^{(i)}_{e5}, c^{(i)}_{e6}$ for $i \in \{1, 2\}$. Let $G' = (V', E')$ be the constructed graph. Because $\Delta(G) = 3$, we have $\Delta(G') = 6$. Moreover, since G is planar and bipartite, G' is also planar and bipartite. We will show that there is a minimal vertex cover of size at least k in G if and only if there is a minimal feedback vertex set of size at least $k + 4|E|$ in G'.

Given a minimal vertex cover S of size at least k in G, we construct the set $S' = S \cup \bigcup_{e \in E} \{c^{(1)}_{e1}, c^{(1)}_{e4}, c^{(2)}_{e1}, c^{(2)}_{e4}\}$. Then $|S'| \geq k + 4|E|$. Let us first argue that S' is an fvs of G'. For each $e = (u, v) \in E$ we have at least one of $u, v \in S$, without loss of generality let $u \in S$. Now in $G'[V' \setminus S']$ the edges $(e^{(1)}, e^{(2)})$ and $(e^{(2)}, v)$ are bridges and therefore cannot be part of any cycle. The remaining cycles going through $e^{(1)}, e^{(2)}$ are handled by $\{c^{(1)}_{e1}, c^{(1)}_{e4}, c^{(2)}_{e1}, c^{(2)}_{e4}\}$. Furthermore, since $G'[V \setminus S]$ is an independent set, it is also acyclic. To see that S' is a minimal fvs, we remark that for each $c^{(i)}_{e1}, c^{(i)}_{e4}$ contained in S' there is a private cycle in $G'[V' \setminus S']$. We also note that since S is a minimal vertex cover of G, for each $u \in S$, there exists $v \notin S$ with $e = (u, v) \in E$. This means that u has the private cycle formed by $\{u, v, e^{(1)}, e^{(2)}\}$ in $G'[V' \setminus S']$. Therefore, S' is a minimal fvs.

Conversely, suppose we are given a minimal fvs S' of G' with $|S'| \geq k + 4|E|$. We will edit S' so that is contains only vertices in $V' \setminus \bigcup_{e \in E} \{e^{(1)}, e^{(2)}\}$, without decreasing its size.

First, suppose $e^{(1)}, e^{(2)} \in S'$, for some $e \in E$. We construct a new minimal fvs $S'' = S' \setminus \{e^{(2)}\} \cup \{c^{(2)}_{e1}, c^{(2)}_{e4}\}$ which is larger that S', since by minimality we have $c^{(2)}_{ei} \notin S'$ for $i \in \{1, \ldots, 6\}$. It is not hard to see that S'' is indeed an fvs, as no cycle can go through $e^{(2)}$ in $G'[V' \setminus S'']$. The two vertices we added have a private cycle, while all vertices of $S' \cap S''$ retain their private cycles, so S'' is a minimal fvs. As a result in the remainder

we assume that S' contains at most one of $\{e^{(1)}, e^{(2)}\}$ for all $e \in E$.

Suppose now that for some $e = (u, v) \in E$, we have $S' \cap \{u, v\} \neq \emptyset$ and $S' \cap \{e^{(1)}, e^{(2)}\} \neq \emptyset$. Without loss of generality, let $e^{(1)} \in S'$. We set $S'' = S' \setminus \{e^{(1)}\} \cup \{c_{e1}^{(1)}, c_{e4}^{(1)}\}$ and claim that S'' is a larger minimal fvs than S. Indeed, no cycle goes through $e^{(1)}$ in $G'[V' \setminus S'']$, the new vertices we added to S' have private cycles, and all vertices of $S' \cap S''$ retain their private cycles in $G'[V' \setminus S'']$. Therefore, we can now assume that if for some $e = (u, v) \in E$ we have $S' \cap \{e^{(1)}, e^{(2)}\} \neq \emptyset$ then $u, v \notin S'$.

For the remaining case, suppose that for some $e = (u, v) \in E$ we have $u, v \notin S'$ and (without loss of generality) $e^{(1)} \in S'$. We construct the set $S'' = S' \setminus \{e^{(1)}\} \cup \{c_{e1}^{(1)}, c_{e4}^{(2)}, u\}$. Note that $|S''| \geq |S'|+2$. It is not hard to see that S'' is an fvs, since by adding $c_{e1}^{(1)}, c_{e4}^{(1)}, v$ to our set we have hit all cycles containing $e^{(1)}$ in G'. The problem now is that S'' is not necessarily minimal. We greedily delete vertices from S'' to obtain a minimal fvs S^*. We claim that in this process we cannot delete more than two vertices, that is $|S^* \setminus S''| \leq 2$. To see this, we first note that $c_{e1}^{(1)}, c_{e4}^{(2)}, u$ cannot be removed from S'' as they have private cycles in $G[V' \setminus S'']$. Suppose now that $w_1 \in S'' \setminus S^*$ is the first vertex we removed from S'', so $G'[(V' \setminus S'') \cup \{w_1\}]$ is acyclic. This vertex must have had a private cycle in $G'[V' \setminus S']$, which was necessarily going through u. Therefore, $G'[(V' \setminus S'') \cup \{w_1\}]$ has a path connecting two neighbors of u and this path does not exist in $G'[(V' \setminus S'')]$. With a similar reasoning, removing another vertex $w_2 \in S''$ from the fvs will create a second path between neighbors of u in the induced forest. We conclude that this cannot happen a third time, since $|N(u)| \leq 3$, and if we create three paths between neighbors of u, this will create a cycle. As a result, $|S^*| \geq |S'|$. We assume in the remainder that S' does not contain $e^{(1)}, e^{(2)}$ for any $e \in E$.

Now, given a minimal fvs S' of G' with $|S'| \geq k+4|E|$ and $S' \cap (\cup_{e \in E}\{e^{(1)}, e^{(2)}\}) = \emptyset$ we set $S = S' \cap V$ and claim that S is a minimal vertex cover of G with $|S| \geq k$. Indeed S is a vertex cover, as for each $e = (u, v) \in E$, if $u, v \notin S'$ then we would get the cycle formed by $\{u, v, e^{(1)}, e^{(2)}\}$. To see that S is minimal, suppose $N_G[u] \subseteq S'$. We claim that in that case u has no private cycle in $G'[V' \setminus S']$ (this can be seen by deleting all bridges in $G'[V' \setminus S']$, which leaves u isolated). This contradicts the minimality of S' as an fvs of G'. Finally, we argue that $|S' \setminus V| \leq 4|E|$, which gives the desired bound on $|S|$. Consider an $e = (u, v) \in E$. S' cannot contain more than one vertex among $c_{e1}^{(1)}, c_{e2}^{(1)}, c_{e3}^{(1)}$, since any of these vertices hits the cycle that goes through the others. With similar reasoning for the three other length-four cycles we conclude that S' contains at most 4 vertices for each edge $e \in E$. □

Note that the previous result not only improves upon the maximum degree of graphs where the problem is NP-hard. Moreover, it shows that the problem is NP-hard even in instances at the intersection between planar graphs and bipartite graphs, two classes for which the complexity of the problem was previously unknown.

5.6 Conclusions

We have essentially settled the approximability of Max Min FVS for polynomial and sub-exponential time, up to sub-polynomial factors in the exponent of the running time. It would be interesting to see if the running time of our sub-exponential approximation algorithm can be improved by poly-logarithmic factors in the exponent, as in [19]. In particular, improving the running time to $2^{O(n/r^{3/2})}$ seems feasible, but would likely require a version of Lemma 5.11 which uses more sophisticated techniques, such as Cut&Count [34, 65, 68]. For the parameterized complexity perspective, we gave a cubic kernel when parameterized by solution size. A natural direction of future work is the deep analysis of parameterized complexity of Max Min FVS. Finally, we showed that Max Min FVS is NP-hard even on planar bipartite graphs of maximum degree 6. This, to the best of our knowledge, is one of the very few hardness results for this problem. It would be interesting to investigate the tractability in other restricted graph classes, and in particular in graphs of graphs of maximum degree 3, where Min FVS can be solved in polynomial time [178].

Another problem of similar spirit which deserves to be studied is Max Min OCT, where an odd cycle transversal (OCT) is a set of vertices whose removal makes the graph bipartite. This problem could also potentially be "between" Max Min VC and Upper DS, but obtaining a $n^{1-\epsilon}$ approximation for it seems much more challenging than for Max Min FVS.

Chapter 6

Digraph Coloring

6.1 Introduction

In Digraph Coloring, we are given a digraph D and are asked to calculate the smallest k such that the vertices of D can be partitioned into k *acyclic* sets. In other words, the objective of this problem is to color the vertices with the minimum number of colors so that no directed cycle is monochromatic. This notion is called the *dichromatic number* and it was introduced by V. Neumann-Lara [164]. More recently, digraph coloring has received much attention, in part because it turns out that many results about the chromatic number of undirected graphs quite naturally carry over to the dichromatic number of digraphs [4, 10, 26, 35, 54, 99, 102, 104, 105, 112, 141, 153, 157, 177]. We note that Digraph Coloring generalizes Coloring (if we simply replace all edges of a graph by pairs of anti-parallel arcs) and is therefore NP-complete.

In this chapter we are interested in the computational complexity of Digraph Coloring from the point of view of structural parameterized complexity. Our main motivation for studying this is that (undirected) Coloring is a problem of central importance in this area whose complexity is well-understood, and it is natural to hope that some of the known tractability results may carry over to digraphs – especially because, as we mentioned, Digraph Coloring seems to behave as a very close counterpart to Coloring in many respects. In particular, for undirected graphs, the complexity of Coloring for "almost-acyclic" graphs is very precisely known: for all $k \geq 3$ there is a $O^*(k^{tw})$ algorithm, where tw is the input graph's treewidth, and this is optimal (under the SETH) even if we replace treewidth by much more restrictive parameters [123, 147]. Can we achieve the same amount of precision for Digraph Coloring?

Related Work Structural parameterizations of Digraph Coloring have been studied in [177], who showed that the problem is FPT by modular width generalizing the algorithms of [92, 136]; and [99] who showed that the problem is in XP by clique-width (note that hardness results for Coloring rule out an fpt-algorithm in this case [86, 87, 137]). Our results on the hardness of the problem for bounded DFVS and FAS build upon the work of [153]. The fact that the problem is hard for bounded DFVS

implies that it is also hard for most versions of directed treewidth, including DAG-width, Kelly-width, and directed pathwidth [31, 93, 118, 125, 138]. Indeed, hardness for FAS implies also hardness for bounded elimination width, a more recently introduced restriction of directed treewidth [84]. For undirected treewidth, a problem with similar behavior is DFVS: (undirected) FVS is solvable in $O^*(3^{tw})$ [68] but DFVS cannot be solved in time $tw^{o(tw)}n^{O(1)}$, and this is tight under the ETH [36]. For other natural problems whose complexity by treewidth is $tw^{\Theta(tw)}$ see [20, 28, 37]

With respect to maximum degree, it is not hard to see that k-Digraph Coloring is NP-hard for graphs of maximum degree $2k + 2$, because k-Coloring is NP-hard for graphs of maximum degree $k + 1$, for all $k \geq 3$ [1]. On the converse side, using a generalization of Brooks' theorem due to Mohar [158] one can see that k-Digraph Coloring digraphs of maximum degree $2k$ is in P. This leaves as the only open case digraphs of degree $2k + 1$, which in a sense mirrors our results for digraphs of DFVS k and degree $4k-2$. We note that the NP-hardness of 2-Digraph Coloring for bounded degree graphs is known even for graphs of large girth, but the degree bound follows the imposed bound on the girth [82].

Our Contribution The main question motivating this chapter is the following: Does Digraph Coloring also become tractable for "almost-acyclic" inputs? We attack this question from two directions.

First, in Section 6.3, we consider the notion of acyclicity in the digraph sense and study cases where the input digraph is close to being a DAG. Possibly the most natural such measure is directed feedback vertex set (DFVS), which is the minimum number of vertices whose removal destroys all directed cycles. The problem is paraNP-hard for this parameter, as for all fixed $k \geq 2$, k-Digraph Coloring is already known to be NP-hard, for inputs of DFVS at most $k+4$ [153]. Our first contribution is to tighten this result by showing that actually k-Digraph Coloring is already NP-hard for DFVS of size *exactly* k. This closes the gap left by the reduction of [153] and provides a complete dichotomy, as the problem is trivially FPT by k when the DFVS has size strictly smaller than k (the only non-trivial part of the problem in this case is to find the DFVS [52]). In the end of this section we consider 2-Digraph Coloring on oriented graphs. We prove that it is NP-hard to decide if an oriented graph is 2-colorable even in cases where the size of DFVS is 3. This is tight as there exists an easy argument showing that all oriented graphs with DFVS k are k-colorable.

In Section 6.4 we investigate if by considering a more restricted notion of near-acyclicity, or by imposing further restrictions, such as bounding the maximum degree of the graph, could lead to an fpt-algorithm. Unfortunately, we show that neither of these suffices to make the problem tractable. In particular, by refining our reduction we obtain the following: First, we show that for all $k \geq 2$, k-Digraph Coloring is NP-hard for digraphs of feedback *arc* set (FAS) k^2, that is, digraphs where there exists a set of k^2 arcs whose removal destroys all cycles (feedback arc set is of course a more restrictive

[1] Note that this argument does not prove that 2-Digraph Coloring is NP-hard for maximum degree 6, but this is not too hard to show. We give a proof in Theorem 6.1 for the sake of completeness.

parameter than feedback vertex set). Interestingly, this also leads us to a complete dichotomy, this time for the parameter FAS: we show that k-coloring becomes FPT (by k) on graphs of FAS at most k^2-1, by an argument that reduces this problem to coloring a subdigraph with at most $O(k^2)$ vertices, and hence the correct complexity threshold for this parameter is k^2. Second, we show that k-coloring a digraph with DFVS k remains NP-hard even if the maximum degree is at most $4k-1$. This further strengthens the reduction of [153], which showed that the problem is NP-hard for bounded *degeneracy* (rather than degree). Almost completing the picture, we show that k-coloring a digraph with DFVS k and maximum degree at most $4k-3$ is FPT by k, leaving open only the case where the DFVS is exactly k and the maximum degree exactly $4k-2$.

In Section 6.5, because of the negative results for DFVS and FAS, we deiced to consider as parameter the treewidth of the underlying graph. It turns out that, finally, this suffices to lead to an *fpt*-algorithm, obtained with standard DP techniques. However, our algorithm has a somewhat disappointing running time of $(tw!)k^{tw}n^{O(1)}$, which is significantly worse than the $k^{tw}n^{O(1)}$ complexity which is known to be optimal for undirected Coloring, especially for small values of k. This raises the question of whether the extra $(tw!)$ factor can be removed. Our main contribution in this part is to show that this is likely impossible, even for a more restricted case. Specifically, we show that if the ETH is true, no algorithm can solve 2-Digraph Coloring in time $td^{o(\mathsf{td})}n^{O(1)}$, where td is the input graph's treedepth, a parameter more restrictive than treewidth (and pathwidth).

Finally, in Section 6.6, we consider tournaments. It is already known that 2-Digraph Coloring is NP-hard for tournaments [54]. The exhaustive algorithm to check if a tournament is 2-colorable takes $O^*(2^n)$ time as there exists 2^n possible 2-colorings for a graph. We improve this running time by proposing an algorithm that answers the same question in $O^*(\sqrt[3]{6}^n)$.

6.2 Preliminaries

All digraphs are loopless and have no parallel arcs; two oppositely oriented arcs between the same pair of vertices, however, are allowed and are called a *digon*. Oriented graphs are digraphs which do not contain any digons. The in-degree (respectively, out-degree) of a vertex is the number of arcs coming into (respectively going out of) a vertex. The degree of a vertex is the sum of its in-degree and out-degree. For a set of arcs F, $V(F)$ denotes the set of their endpoints.

The *chromatic number* of a graph G is the minimum number of colors k needed to color the vertices of G such that each color class is an independent set. We say that a digraph $D = (V, E)$ is *k-colorable* if we can color the vertices of D with k colors such that each color class induces an acyclic subdigraph (such a coloring is called a *proper k-coloring*). The *dichromatic number*, denoted by $\vec{\chi}(D)$, is the minimum number k for which D is k-colorable. The maximum degree of a graph or digraph is denoted with Δ.

Recall that a subset of vertices $S \subset V$ of D is called a *feedback vertex set* if $D - S$ is acyclic.

Remark. *Every digraph $D = (V, E)$ with feedback vertex set of size at most $k - 1$ is k-colorable.*

The remark holds because we can use distinct colors for the vertices of the feedback vertex set and the remaining color for the rest of the graph.

A subset of arcs $A \subset E$ of D is called a *feedback arc set* if $D - A$ is acyclic. A graph G has treedepth at most k if one of the following holds: (i) G has at most k vertices (ii) G is disconnected and all its components have treedepth at most k (iii) there exists $u \in V(G)$ such that $G - u$ has treedepth at most $k - 1$. We use $tw(G), \mathsf{td}(\mathsf{G})$ to denote the treewidth and treedepth of a graph. It is known that $tw(G) \leq \mathsf{td}(\mathsf{G})$ for all graphs G.

For a set V an ordering of V is an injective function $\sigma : V \to [|V|]$. It is a well-known fact that a digraph D is acyclic if and only if there exists an ordering σ of $V(D)$ such that for all arcs uv we have $\sigma(u) < \sigma(v)$. This is called a topological ordering of D.

We conclude this section with a preliminary theorem. As we mentioned, the argument from (undirected) graph coloring that shows why k-Digraph Coloring is NP-hard for digraphs of $\Delta = 2k + 2$ does not hold for $k = 2$. Our first theorem, the proof of which is given in the appendix, shows that the previous statement holds even for $k = 2$.

Theorem 6.1. *It is NP-hard to decide if a given digraph with maximum degree 6 is 2-colorable.*

6.3 Bounded Directed Feedback Vertex Set

In this section we study the complexity of the problem parameterized by the size of the directed feedback vertex set of a digraph. Throughout we will assume that a directed feedback vertex set is given to us; if not we can use known fpt-algorithms to find the smallest such set [52].

As we are mentioned in Remark 6.2, a digraph of directed feedback vertex set of size $k - 1$ can be always colored with k colors. Our main result in this section is that k-Digraph Coloring is NP-hard for digraphs of directed feedback vertex set of size k. Observe that Remark 6.2 indicates that this result will be best possible.

Remark. *Let $D = (V, E)$ be a digraph with directed feedback vertex set F of size $|F| = k$. If F does not induce a bi-directed clique, then D is k-colorable.*

Indeed, if $u, v \in F$ are not connected by a digon we can use one color for $\{u, v\}$, $k-2$ distinct colors for the rest of F, and the remaining color for the rest of the graph. Remark 6.3 will also be useful later in designing an algorithm, but at this point it is interesting because it tells us that, since the graphs we construct in our reduction have directed feedback vertex set of size k and must in some cases have $\vec{\chi}(D) > k$, our reduction needs to construct a bi-directed clique of size k.

Before we go on to our reduction let us also mention that we will reduce from a restricted version of 3-SAT with the following properties: (i) all clauses must have

either only positive literals or only negative literals (ii) all variables appear at most 2 times positive and 1 time negative. We call this RESTRICTED-3-SAT.

Lemma 6.2. *RESTRICTED-3-SAT is NP-hard and cannot be solved in $2^{o(n+m)}$ time unless the ETH is false.*

Proof. Start with an arbitrary instance ϕ of 3-SAT with n variables and m clauses. We first make sure that every variable appears at most 3 times as follows. First use the trick of Theorem 6.1 to decrease the number of appearances of each literal to two. We now edit ϕ' as follows: for each variable x of ϕ' we replace every occurence of $\neg x$ with a fresh variable x'. We then add the clause $(\neg x \vee \neg x')$. This gives a new equivalent instance ϕ'' which also has $O(n+m)$ variables and clauses and satisfies all properties of RESTRICTED-3-SAT. □

Theorem 6.3. *For all $k \geq 2$, it is NP-hard to decide if a digraph $D = (V, E)$ is k-colorable even when the size of its directed feedback vertex set is k. Furthermore, this problem cannot be solved in time $2^{o(n)}$ unless the ETH is false.*

Proof. We give a reduction from RESTRICTED-3-SAT, which is NP-hard by Lemma 6.2. Our reduction will produce an instance of size linear in the input formula, which leads to the ETH-based lower bound. Let ϕ be the given formula with variables $x_1, \ldots, x_n$, and suppose that clauses $c_1, \ldots, c_\ell$ contain only positive literals, while clauses $c_{\ell+1}, \ldots, c_m$ contain only negative literals. We will assume without loss of generality that all variables appear in ϕ both positive and negative (otherwise ϕ can be simplified).

First we are going to create a bi-directed clique of size k. We will call this clique "palette" and the vertices of this clique "palette" vertices. We will prove the theorem for $k = 2$. To obtain the proof for larger values one can add to the palette $k-2$ new vertices which are connected to everything (including each other) with digons: this increases both the dichromatic number and the directed feedback vertex set by $k-2$. Note that this does indeed construct a "palette" clique of size k, as indicated by Remark 6.3.

We begin by constructing the two palette vertices v_1, v_2 which are connected by a digon. Then, for each clause $c_i, i \in [m]$ we do the following: if the clause has size three we construct a directed path with vertices $l_{i,1}, w_{i,1}, l_{i,2}, w_{i,2}, l_{i,3}$, where the vertices $l_{i,1}, l_{i,2}, l_{i,3}$ represent the literals of the clause; if the clause has size two we similarly construct a directed path with vertices $l_{i,1}, w_{i,1}, l_{i,2}$, where again $l_{i,1}, l_{i,2}$ represent the literals of the clause.

For each variable $x_j, j \in [n]$ we do the following: for each clause c_{i_1} where x_j appears positive and clause c_{i_2} where x_j appears negative we construct a vertex w'_{j,i_1,i_2} and add an incoming arc from the vertex that represents the literal x_j in the directed path of c_{i_1} to w'_{j,i_1,i_2}; and an outgoing arc from w'_{j,i_1,i_2} to the vertex that represents the literal $\neg x_j$ in the directed path of c_{i_2}.

Finally, to complete the construction we connect the palette vertices to the rest of the graph as follows: v_1 is connected with a digon to all existing vertices $w_{i,j}$, $i \in [m], j \in [2]$; v_2 is connected with a digon to all existing vertices w'_{j,i_1,i_2}; v_2 has an outgoing arc to the first vertex of each directed path representing a clause and an incoming arc from

the last vertex of each such path; v_1 has an outgoing arc to all vertices that represent positive literals and an incoming arc from all vertices representing negative literals. (See Fig. 6.1)

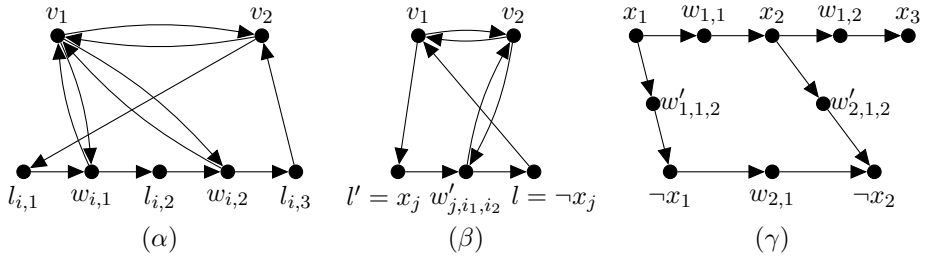


Figure 6.1: (α): The cycles created by $\{v_1, v_2\}$ and clauses with three literals. (β): The cycles created by $\{v_1, v_2\}$ and each pair $\{x, \neg x\}$. (γ): An example digraph for the formula $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2)$, without showing v_1, v_2.

Let us now prove that this reduction implies the theorem. First, we claim that in the digraph we constructed $\{v_1, v_2\}$ is a directed feedback vertex set. Indeed, suppose we remove these two vertices. Now every arc in the remaining graph either connects vertices that represent the same clause, or is incident on a vertex w'_{j,i_1,i_2}. Observe that these vertices have only one incoming and one outgoing arc and because of the ordering of the clauses $i_1 < i_2$ (since clauses that contain negative literals come later in the numbering). We conclude that every directed path must either stay inside the path representing the same clause or lead to a path that represents a later clause. Hence, the digraph is acyclic.

Let us now argue that if ϕ is satisfiable then the digraph is 2-colorable. We give color 1 to v_1 and 2 to v_2. We give color 2 to each $w_{i,j}$ and color 1 to each w'_{j,i_1,i_2}. Fix a satisfying assignment for ϕ. We give color 1 to all vertices $l_{i,j}$ that represent literals set to True by the assignment and color 2 to all remaining vertices. Let us see why this coloring is acyclic. First, consider a vertex w'_{j,i_1,i_2}. This vertex has color 1 and one incoming and one outgoing arc corresponding to opposite literals. Because the literals are opposite, one of them has color 2, hence w'_{j,i_1,i_2} cannot be in any monochromatic cycle and can be removed. Now, suppose there is a monochromatic cycle of color 1. As $\{v_1, v_2\}$ is a directed feedback vertex set, this cycle must include v_1. Since v_2 and all $w_{i,j}$ have color 2 the vertex after v_1 in the cycle must be some $l_{i,j}$ representing a positive literal which was set to True by our assignment. The only outgoing arc leaving from $l_{i,j}$ and going to a vertex of color 1 must lead it to a vertex $w'_{j',i,i'}$, which as we said cannot be part of any cycle. Hence, no monochromatic cycle of color 1 exists. Consider then a monochromatic cycle of color 2, which must begin from v_2. The next vertex on this cycle must be a $l_{i,1}$ and since we have eliminated vertices w'_{j,i_1,i_2} the cycle must continue in the directed path of clause i. But, since we started with a satisfying assignment, at least one of the literal vertices of this path has color 1, meaning the cycle cannot be monochromatic.

Finally, let us argue that if the digraph is 2-colorable, then ϕ is satisfiable. Consider

a 2-coloring which, without loss of generality, assigns 1 to v_1 and 2 to v_2. The coloring must give color 2 to all $w_{i,j}$ and color 1 to all w_{j,i_1,i_2}, because of the digons connecting these vertices to the palette. Now, we obtain an assignment for ϕ as follows: for each x_j, we find the vertex in our graph that represents the literal $\neg x_j$ (this is unique since each variable appears exactly once negatively): we assign x_j to True if and only if this vertex has color 2. Let us argue that this assignment satisfies all clauses. First, consider a clause with all negative literals. If this clause is not satisfied, then all the vertices representing its literals have color 2. Because vertices $w_{i,j}$ also all have color 2, this creates a monochromatic cycle with v_2, contradiction. Hence, all such clauses are satisfied. Second, consider a clause c_i with all positive literals. In the directed path representing c_i at least one literal vertex must have color 1, otherwise we would get a monochromatic cycle with v_2. Suppose this vertex represents the literal x_j and has an out-neighbor w'_{j,i,i_2}, which is colored 1. If the out-neighbor of w'_{j,i_1,i_2} is also colored 1, we get a monochromatic cycle with v_1. Therefore, that vertex, which represents the literal $\neg x_j$ has color 2. But then, according to our assignment x_j is True and c_i is satisfied. □

The last result of this section concerns 2-coloring of oriented graphs.

Theorem 6.4. *It is NP-hard to decide if an oriented graph $D = (V, E)$ is 2-colorable even when the size of its directed feedback vertex set is 3.*

Proof. We adapt the proof of Theorem 6.3. First let us give an intuition behind the gadget we are going to use. In the proof of Theorem 6.3 the digraph we created is not an oriented graph as it contains digons. All the digons of that digraph are connected to vertices v_1 or v_2, and therefore, we want to replace v_1 and v_2 with a gadget that contains two arcs t_1t_2 and f_1f_2 such that the vertices t_1 and t_2 have the same color as v_1 and the vertices f_1 and f_2 have the same color as v_2. Then we can replace all cycles that contained v_1 (respectively, v_2) with cycles that contain the arc t_1t_2 (respectively, f_1f_2) and the rest of the proof will remain the same.

The gadget we use in place of $\{v_1, v_2\}$ is the one in the Fig. 6.2. Furthermore, we will not use the digon between v_1 and v_2 and we replace all the other incoming arcs of v_1 from the previous construction with incoming arcs to t_1, the outgoing arcs of v_1 with outgoing arcs from t_2, the incoming arcs of v_2 with incoming arcs to f_1, the outgoing arcs of v_2 with outgoing arcs from f_2. For example, the digon $v_1w_{i,1}$ in the gadget (α) from the previous theorem becomes a triangle $t_1t_2w_{i,1}$.

Now we need to show that in any proper 2-coloring of this gadget both pairs f_1, f_2 and t_1, t_2 are monochromatic and we use different colors per pair.

First observe that there exists such a coloring (see Fig. 6.2) We will show that the vertices f_1 and t_1 cannot have the same color. Assume that they are both colored 0; then the vertex f_2 must be colored 1 because we have the cycle f_1, f_2, t_1. Because the vertex f_2 is colored 1 and there exists the cycle f_2, v_3, v_4 we know that at least one of v_3, v_4 must be colored 0. Let v_3 (respectively, v_4) be colored 0, then the coloring is not proper because there exists the cycle t_1, f_1, v_3 (resp. t_1, f_1, v_4) with all the vertices colored 0. This is a contradiction so the f_1 and t_1 cannot have the same color. Similarly we can prove that f_2 and t_1 cannot have the same color. So we must color the vertices

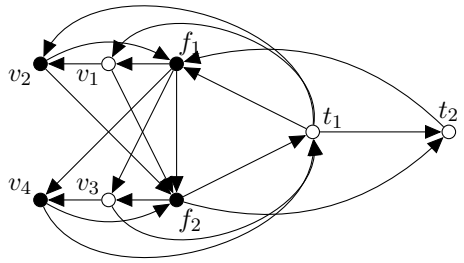
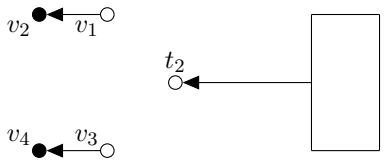


Figure 6.2: Gadget H: It is 2-colorable, and in any 2-coloring both pairs $\{f_1, f_2\}$ and $\{t_1, t_2\}$ must be monochromatic but with different colors per pair.

f_1 and f_2 with one color and t_1 with the second. Furthermore because we have the cycle f_1, f_2, t_2, the vertex t_2 must use the same color as t_1.

It remains to show that the size of a minimum directed feedback vertex set is at most 3; observe that the set $\{f_1, f_2, t_1\}$ is a directed feedback vertex set (see Fig. 6.3).



H without $\{f_1, f_2, t_1\}$ remaining digraph

Figure 6.3: For the remaining digraph, it has been proved that is acyclic in the previous theorem so $\{f_1, f_2, t_1\}$ is a directed feedback vertex set of the whole digraph.

□

This result is tight as, by Remark 6.3, we know that oriented graphs with directed feedback vertex set of size k are k-colorable.

6.4 Bounded Feedback Arc Set and Bounded Degree

In this section we first present two algorithmic results: we show that k-Digraph Coloring becomes FPT (by k) if either the input graph has directed feedback vertex set k and maximum degree at most $4k - 3$; or if it has feedback arc set at most $k^2 - 1$ (and unbounded degree). Interestingly, the latter of these results is exactly tight and the former is almost tight: in the second part we refine the reduction of the previous section to show that k-Digraph Coloring is NP-hard for digraphs which have simlutaneously

a feedback arc set of size k^2, a directed feedback vertex set of size k and maximum degree $\Delta = 4k - 1$.

6.4.1 Algorithmic Results

Our first result shows that for k-Digraph Coloring, if we are promised a directed feedback vertex set of size k (which is the smallest value for which the problem is non-trivial), then the problem remains tractable for degree up to $4k-3$. Observe that in the case of general digraphs (where we do not bound the directed feedback vertex set) the problem is already hard for maximum degree $2k + 2$ (see Introduction section), so this seems encouraging. However, we show in Theorem 6.8 that this tractability cannot be extended much further.

Theorem 6.5. *Let $D = (V, E)$ be a digraph with feedback vertex set F of size $|F| = k$ and maximum degree $\Delta \leq 4k - 3$. Then, D is k-colorable if and only if $D[N[F]]$ is k-colorable. Furthermore, a k-coloring of $D[N[F]]$ can be extended to a k-coloring of D in polynomial time.*

Proof. Let $D = (V, E)$ be such a digraph. If $D[N[F]]$ is not k-colorable, then D is not k-colorable, so we need to prove that if $D[N[F]]$ is k-colorable then D is k-colorable and we can extend this coloring to D. Assume that $D[N[F]]$ is k-colorable. By Remark 6.3 we can assume that $D[F]$ is a bi-directed clique. Let $c : N[F] \to [k]$ be the assumed k-coloring and without loss of generality say that $F = \{v_1, \ldots, v_k\}$ and $c(v_i) = i$ for all $i \in [k]$.

Before we continue let us define the following sets of vertices: we will call $V_{i,in}$ the set of vertices $v \in N[F] \setminus F$ such that $c(v) = i$ and there exists an arc $vv_i \in E$. Similarly we will call $V_{i,out}$ the set of vertices $v \in N[F] \setminus F$ where $c(v) = i$ and there exists an arc $v_iv \in E$. The sets $V_{i,in}$ and $V_{i,out}$ are disjoint in any proper coloring (otherwise we would have a monochromatic digon). Furthermore, $V_{i,in} \cup V_{i,out}$ is disjoint from $V_{j,in} \cup V_{j,out}$ for $j \neq i$ (because their vertices have different colors), so all these $2k$ sets are pairwise disjoint. We first show that if one of these $2k$ sets is empty, then we can color D.

Lemma 6.6. *If for some $i \in [k]$ one of the sets $V_{i,in}$, $V_{i,out}$ is empty then we can extend c to a k-coloring of D in polynomial time.*

Proof. We keep c unchanged and color all of $V(D) \setminus N[F]$ with color i. This is a proper k-coloring. Indeed, this cannot create a monochromatic cycle with color $j \neq i$. Furthermore, if a monochromatic cycle of color i exists, since this cycle must intersect F, we conclude that it must contain v_i. However, in the current k-coloring v_i either has in-degree or out-degree 0 in the vertices colored i, so no monochromatic cycle can go through it. □

In the remainder we assume that all sets $V_{i,in}, V_{i,out}$ are non-empty. Our strategy will be to edit the k-coloring of $D[N[F]]$ so that we retain a proper k-coloring, but one of these $2k$ sets becomes empty. We will then invoke Lemma 6.6 to complete the proof.

We now define, for each pair $i, j \in [k]$ with $i < j$ the set $E_{i,j}$ which contains all arcs with one endpoint in $\{v_i, v_j\}$ and the other in $V_{i,in} \cup V_{i,out} \cup V_{j,in} \cup V_{j,out}$ and whose endpoints have *distinct* colors. We call $E_{i,j}$ the set of *cross* arcs for the pair (i, j). We will now argue that for some pair (i, j) we must have $|E_{i,j}| \leq 3$. For the sake of contradiction, assume that $|E_{i,j}| \geq 4$ for all pairs. Then, by summing up the degrees of vertices of F we have:

$$\sum_{i \in [k]} d(v_i) \geq 2k + k(2k-2) + \sum_{i,j \in [k], i<j} |E_{i,j}| \geq 2k^2 + 4\binom{k}{2} = 4k^2 - 2k$$

In the first inequality we used the fact that each $v_i \in F$ has at least two arcs connecting it to $V_{i,in} \cup V_{i,out}$ (since these sets are non-empty); $2k-2$ arcs connecting it to other vertices of F (since F is a clique); and each cross arc of a set $E_{i,j}$ contributes one to the degree of one vertex of F. From this calculation we infer that the average degree of F is at least $4k-2$, which is a contradiction, since we assumed that the digraph has maximum degre $4k-3$.

Fix now i, j such that $|E_{i,j}| \leq 3$. We will recolor $V_{i,in} \cup V_{i,out} \cup V_{j,in} \cup V_{j,out}$ in a way that allows us to invoke Lemma 6.6. Since we do not change any other color, we will only need to prove that our recoloring does not create monochromatic cycles of colors i or j in $D[N[F]]$. We can assume that $|E_{i,j}| = 3$, since if $|E_{i,j}| < 3$ we can add an arbitrary missing cross arc and this can only make the recoloring process harder. Furthermore, without loss of generality, we assume that v_i has strictly more cross arcs of $E_{i,j}$ incident to it than v_j.

We now have to do a case analysis. First, suppose all three arcs of $E_{i,j}$ are incident on v_i. Then, there exists a set among $V_{j,in}, V_{j,out}$ that has at most one arc connecting it to v_i. We color this set i, and leave the other set colored j. We also color $V_{i,in} \cup V_{i,out}$ with j. This creates no monochromatic cycle because: (i) v_i now has at most one neighbor colored i in $V_{i,in} \cup V_{i,out} \cup V_{j,in} \cup V_{j,out}$, so no monochromatic cycle goes through v_i; (ii) v_j has either no out-neighbors or no in-neighbors colored j in $V_{i,in} \cup V_{i,out} \cup V_{j,in} \cup V_{j,out}$. With the new coloring we can invoke Lemma 6.6. In the remainder of this proof we therefore assume that two arcs of $E_{i,j}$ are incident on v_i and one is incident on v_j.

Second, suppose that one of $V_{j,in}, V_{j,out}$ has no arcs connecting it to v_i. We color this set i and leave the other set colored j. Observe that one of $V_{i,in}, V_{i,out}$ has no arc connecting it to v_j. We color that set j and leave the other set colored i. In the new coloring both v_i, v_j either have no out-neighbor or no in-neighbor with the same color in $V_{i,in} \cup V_{i,out} \cup V_{j,in} \cup V_{j,out}$, so the coloring is proper and we can invoke Claim 6.6. In the remainder we assume that v_i has one arc connecting it to each of $V_{j,in}, V_{j,out}$.

Third, suppose that both arcs of $E_{i,j}$ incident on v_i have the same direction (into or out of v_i). We then color $V_{i,in} \cup V_{i,out}$ with j and $V_{j,in} \cup V_{j,out}$ with i. In the new coloring v_j has at most one neighbor with the same color and v_i has either only in-neighbors or only out-neighbors with color i, so the coloring is acyclic and we again invoke Lemma 6.6.

Finally, we have the case where two arcs of $E_{i,j}$ are incident on v_i, they have different directions, one has its other endpoint in $V_{j,in}$ and the other in $V_{j,out}$. Observe that one of $V_{i,in}, V_{i,out}$ has no arc connecting it to v_j and suppose without loss of generality that

it is $V_{i,in}$ (the other case is symmetric). We color $V_{i,in}$ with j and leave $V_{i,out}$ with color i. One of $V_{j,in}, V_{j,out}$ has an incoming arc from v_i; we color this set i and leave the other colored j. Now, v_i only has out-neighbors with color i, while v_j has at either only in-neighbors or only out-neighbors colored j, so we are done in this final case. □

Our second result concerns a parameter more restricted than directed feedback vertex set, namely feedback arc set. Note that, in a sense, the class of graphs of bounded feedback arc set contains the class of graphs that have bounded directed feedback vertex set and bounded degree (selecting all incoming or outgoing arcs of each vertex of a directed feedback vertex set produces a feedback arc set), so the following theorem may seem more general. However, a closer look reveals that the following result is incomparable to Theorem 6.5, because graphs of directed feedback vertex set k and maximum degree $4k-3$ could have feedback arc set of size up to almost $2k^2$ (consider for example a bi-direction of the complete graph $K_{k,2k-2}$), while the following theorem is able to handle graphs of unbounded degree but feedback arc set up to (only) k^2-1. As we show in Theorem 6.8, this is tight.

Theorem 6.7. *Let D be a digraph with a feedback arc set F of size at most k^2-1. Then D is k-colorable if and only if $D[V(F)]$ is k-colorable, and such a coloring can be extended to D in polynomial time.*

Proof. It is obvious that if $D[V(F)]$ is not k-colorable then D is not k-colorable. We will prove the converse by induction. For $k=1$ it is trivial to see that if $|F|=0$ then D is acyclic so is 1-colorable. Assume then that any digraph D with a feedback arc set F of size at most $(k-1)^2-1$ is $(k-1)$-colorable, if and only if $D[V(F)]$ is $(k-1)$-colorable.

Suppose now that we have D with a feedback arc set F with $|F| \leq k^2-1$ and we find that $D[V(F)]$ is k-colorable (this can be tested in $2^{O(k^2)}$ time). Let $c : V(F) \to [k]$ be a coloring of $V(F)$. We distinguish two cases:

Case 1. There exists a color class (say V_k) such that at least $2k-1$ arcs of F are incident on V_k. Then $D - V_k$ has a feedback arc set of size at most $|F| - (2k-1) \leq k^2-1-(2k-1) \leq (k-1)^2-1$ and $V_1, \ldots, V_{k-1}$ remains a valid coloring of the remainder of $V(F)$. So by inductive hypothesis $D - V_k$ has a $(k-1)$-coloring. By using the color k on V_k we have a k-coloring for D.

Case 2. Each color class is incident on at most $2k-2$ arcs of F. We then claim that there is a way to color $V(F)$ so that all arcs of F have distinct colors on their endpoints. If we achieve this, we can trivially extend the coloring to the rest of the graph, as arcs of F become irrelevant. Now, let us call $v \in V(F)$ as *type one* if v is incident on at least k arcs of F. We will show that there is at most one *type one* vertex in each color class. Indeed, if $u, v \in V_i$ are both *type one*, then they are incident on at least $2k-1$ arcs of F (there is no digon between u and v because they share a color), which we assumed is not the case, as V_i is incident on at most $2k-2$ arcs of F. Therefore, we can use k distinct colors to color all the *type one* vertices of $V(F)$. Each remaining vertex of $V(F)$ is incident on at most $k-1$ arcs of F. We consider these vertices in some arbitrary order, and give each a color that does not already appear on the other endpoints of its

incident arcs from F. Such a color always exists, and proceeding this way we color all arcs of F with distinct colors. This completes the proof. □

6.4.2 Hardness

In this section we improve upon our previous reduction by producing a graph which has bounded degree and bounded feedback arc set. Our goal is to do this efficiently enough to (almost) match the algorithmic bounds given in the previous section.

Theorem 6.8. *For all $k \geq 2$, it is NP-hard to decide if a digraph $D = (V, E)$ is k-colorable, even if D has a directed feedback vertex set of size k, a feedback arc set of size k^2, and maximum degree $\Delta = 4k - 1$.*

Proof. Recall that in the proof of Theorem 6.3 for $k \geq 2$ we construct a graph that is made up of two parts: the palette part, which is a bi-directed clique that contains v_1, v_2 and the $k-2$ vertices we have possibly added to increase the chromatic number (call them $v_3, \ldots, v_k$); and the part that represents the formula. We perform the same reduction except that we will now edit the graph to reduce its degree and its feedback arc set. In particular, we delete the palette vertices and replace them with a gadget that we describe below.

We construct a new palette that will be a bi-directed clique of size k, whose vertices are now labeled $v^i, i \in [k]$. Let M be the number of vertices of the graph we constructed for Theorem 6.3. We construct M "rows" of $2k$ vertices each. More precisely, for each $\ell \in [M], i \in [k]$ we construct the two vertices $v^i_{\ell,in}, v^i_{\ell,out}$. In the remainder, when we refer to row ℓ, we mean the set $\{v^i_{\ell,in}, v^i_{\ell,out} \mid i \in [k]\}$. For all $i, j \in [k], i < j$ we connect the vertices of row 1 to the vertices of the clique as shown in Fig. 6.4. For all $i, j \in [k], i < j$ and $\ell \in [M-1]$ we connect the vertices of rows $\ell, \ell+1$ as shown in Fig. 6.5.

In more detail we have:

1. For each $i \in [k]$ the vertex v^i has an arc to all $v^j_{1,out}$ for $j \geq i$, an arc to $v^j_{1,in}$ for all $j \neq i$, and an arc from $v^j_{1,in}$ for all $j \leq i$.

2. For each $\ell \in [M]$, for all $i < j$ we have the following four arcs: $v^j_{\ell,out}v^i_{\ell,out}$, $v^i_{\ell,out}v^j_{\ell,in}$, $v^j_{\ell,in}v^i_{\ell,in}$, and $v^j_{\ell,out}v^i_{\ell,in}$. As a result, inside a row arcs are oriented from *out* to *in* vertices; and between vertices of the same type from larger to smaller indices i.

3. For each $\ell \in [M-1]$, for all $i \in [k]$ we have the arcs $v^i_{\ell,out}v^i_{\ell+1,out}$ and $v^i_{\ell+1,in}v^i_{\ell,in}$. As a result, the $v^i_{\ell,out}$ vertices form a directed path going out of v^i and the $v^i_{\ell,in}$ vertices form a directed path going into v^i.

4. For each $\ell \in [M-1]$, for all $i, j \in [k]$ with $i < j$ we have the arcs $v^i_{\ell,out}v^j_{\ell+1,in}$, $v^i_{\ell,out}v^j_{\ell+1,out}$, $v^i_{\ell+1,in}v^j_{\ell,in}$, $v^j_{\ell,out}v^i_{\ell+1,in}$. Again, arcs incident on an *out* and an *in* vertex are oriented towards the *in* vertex.

Let P be the gadget we have constructed so far, consisting of the clique of size k and the M rows of $2k$ vertices each. We will establish the following properties.

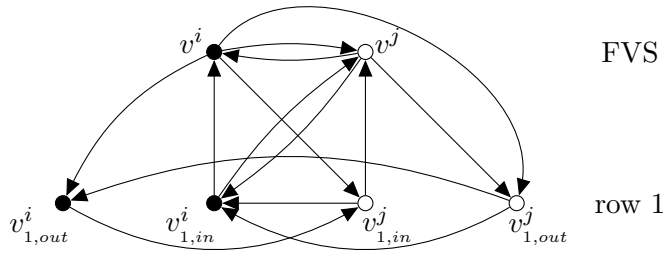


Figure 6.4: Graph showing the connections between two vertices of the clique palette (v^i, v^j, where $i < j$) and the corresponding vertices of row 1.

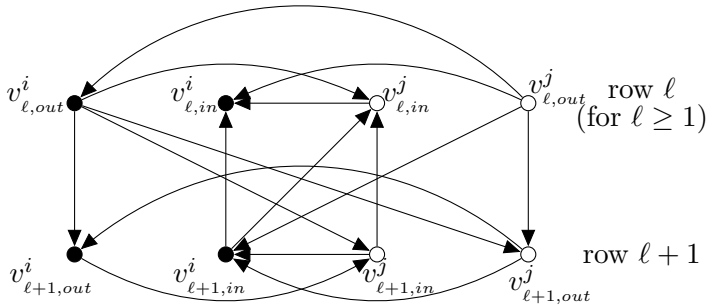


Figure 6.5: Here we present the way we are connecting the vertices of the rows i and $i+1$

1. Deleting all vertices $v^i, i \in [k]$ makes P acyclic and eliminates all directed paths from any vertex $v^i_{\ell,in}$ to any vertex $v^j_{\ell',out}$, for all $i, j \in [k], \ell, \ell' \in [M]$.
2. The maximum degree of any vertex of P is $4k-2$.
3. There is a k-coloring of P that gives all vertices of $\{v^i_{\ell,in}, v^i_{\ell,out} \mid \ell \in [M]\}$ color i, for all $i \in [k]$.
4. In any k-coloring of P, for all i, all vertices of $\{v^i_{\ell,in}, v^i_{\ell,out} \mid \ell \in [M]\}$ receive the same color as v^i.

We include the proof of these properties in the appendix. Now, let us explain why they imply the theorem. To complete the construction, we insert P in our graph in the place of the palette clique we were previously using. To each vertex of the original graph, we associate a distinct row of P (there are sufficiently many rows to do this). Now, if vertex u of the original graph, which is associated to row ℓ, had an arc from (respectively to) the vertex v_i in the palette, we add an arc from $v^i_{\ell,out}$ (respectively to $v^i_{\ell,in}$).

Let us first establish that the new graph has the properties promised in the theorem. The maximum degree of any vertex in the main (non-palette) part remains unchanged

and is $2k+2 \leq 4k-1$ while the maximum degree of any vertex of P is now at most $4k-1$, as we added at most one arc to each vertex. Deleting $\{v^i \mid i \in [k]\}$ eliminates all cycles in P, but also all cycles going through P, because such a cycle would need to use a path from a vertex $v^i_{\ell,in}$ (since these are the only vertices with incoming arcs from outside P) to a vertex $v^j_{\ell',out}$. Deleting all of P leaves the graph acyclic (recall that the palette clique was a directed feedback vertex set in our previous construction), so there is a directed feedback vertex set of size k.

For the feedback arc set we remove the arcs $\{v^j v^i \mid j > i,\ i,j \in [k]\} \cup \{v^i_{1,in} v^j \mid j > i,\ i,j \in [k]\} \cup \{v^i_{1,in} v^i \mid i \in [k]\}$. Note that these are indeed k^2 arcs. To see that this is a feedback arc set, suppose that the graph contains a directed cycle after its removal. This cycle must contain some vertex v^i, because we argued that $\{v^i \mid i \in [k]\}$ is a directed feedback vertex set. Among these vertices, select the v^i with minimum i. We now examine the arc of the cycle going into v^i. Its tail cannot be v^j for $j > i$, as we have removed such arcs, nor v_j for $j < i$, as this contradicts the minimality of i. It cannot be $v^i_{1,in}$ as we have also removed these arcs. And it cannot be $v^j_{1,in}$ for $j < i$, as these arcs are also removed. But no other in-neighbor of v^i remains, a contradiction.

Let us also argue that using the gadget P instead of the palette clique does not affect the k-colorability of the graph. This is not hard to see because, following Properties 3 and 4 we can assume that any k-coloring of P will give color i to all vertices of $\{v^i\} \cup \{v^i_{\ell,in}, v^i_{\ell,out} \mid \ell \in [M]\}$. The important observation is now that, for all $\ell \in [M]$ there will always exist a monochromatic path from v^i to $v^i_{\ell,out}$ and from $v^i_{\ell,in}$ to v^i. We now note that, if we fix a coloring of the non-palette part of the graph, this coloring contains a monochromatic cycle involving vertex v_i of the original palette if and only if the same coloring gives a monochromatic cycle in the new graph going through v^i.

□

6.5 Treewidth

In this section we consider the complexity of Digraph Coloring with respect to parameters measuring the acyclicity of the underlying graph, namely, treewidth and treedepth. Before we proceed let us recall that in all graphs G we have $\chi(G) \leq tw(G)+1 \leq \mathsf{td}(\mathsf{G})+1$. This means that if our goal is simply to obtain an fpt-algorithm then parameterizing by treewidth implies that the graph's chromatic number (and therefore also the digraph's dichromatic number) is bounded. We first present an algorithm with complexity $k^{tw}(tw!)$ which, using the above argument, proves that Digraph Coloring is FPT parameterized by treewidth.

Theorem 6.9. *There is an algorithm which, given a digraph D on n vertices and a tree decomposition of its underlying graph of width tw decides if D is k-colorable in time $k^{tw}(tw!)n^{O(1)}$.*

Proof. The proof uses standard techniques so we sketch some details. In particular we assume that we are given a nice tree decomposition on which we will perform dynamic

programming. Before we proceed, let us slightly recast the problem. We will say that a digraph $D = (V, E)$ is k-colorable if there exist two functions c, σ such that (i) $c : V \to [k]$ partitions V into k sets (ii) σ is an ordering of V (iii) for all arcs $uv \in E$ we have either $c(u) \neq c(v)$ or $\sigma(u) < \sigma(v)$. It is not hard to see that this reformulation is equivalent to the original problem. Indeed, if we have a k-coloring, since each color class is acyclic, we can find a topological ordering σ_i of the graph $G[V_i]$ induced by each color class and then concatenate them to obtain an ordering of V. For the converse direction, the existence of c, σ implies that if we look at vertices of each color class, σ must induce a topological ordering, hence each color class is acyclic.

Now, let D be a digraph and S be a subset of its vertices. Let (c, σ) be a pair of coloring and ordering functions that prove that D is k-colorable. Then, we will say that the signature of solution (c, σ) for set S is the pair (c_S, σ_S) where $c_S : S \to [k]$ is defined as $c_S(u) = c(u)$ and $\sigma_S : S \to [|S|]$ is an ordering function such that for all $u, v \in S$ we have $\sigma_S(u) < \sigma_S(v)$ if and only if $\sigma(u) < \sigma(v)$. In other words, the signature of a solution is the restriction of the solution to the set S.

Given a rooted nice tree decomposition of D, let B_t be a bag of the decomposition and denote by $B_t^{\downarrow}$ the set of vertices of D which are contained in B_t and bags in the sub-tree rooted at B_t. Our dynamic programming algorithm stores for each B_t a collection of all pairs (c, σ) such that there exists a k-coloring of $D[B_t^{\downarrow}]$ whose signature is (c, σ). If we manage to construct such a table for each node, it will suffice to check if the collection of signatures of the root is empty to decide if the graph is k-colorable.

The table is easy to initialize for Leaf nodes, as the only valid signature contains the empty coloring and ordering function. For an Introduce node that adds u to a bag containing B_t we consider all signatures (c, σ) of contained in the table of the child bag. For each such signature we construct a signature (c', σ') which is consistent with (c, σ) but also colors u and places it somewhere in the ordering (we consider all such possibilities). For each (c', σ') we delete this signature if u has a neighbor in the bag that is assigned the same color by c' but such that their arc violates the topological ordering σ'. We keep all other produced signatures. To see that this is correct observe that u has no neighbors in $B_t^{\downarrow} \setminus B_t$, because all bags are separators, so if we produce an ordering of $B_t^{\downarrow}$ consistent with σ' the only arcs incident on u that could violate it are contained in the bag (and have been checked). For Forget nodes the table is easily updated by keeping only the restrictions of valid signatures to the new bag. Finally, for Join nodes we keep a signature (c, σ) if and only if it is valid for both sub-trees. Again this is correct because nodes of one sub-tree not contained in the bag do not have neighbors in the other sub-tree, so as long as we produce an ordering consistent with σ we can concatenate we cannot violate the topological ordering condition.

For the running time observe that the size of the DP table is $k^{tw}(tw!)$, because we consider all colorings and all orderings of each bag. In Introduce nodes we spend polynomial time for each entry of the child node (checking all placements of the new vertex), while computation in Join nodes can be performed in time linear in the size of the table. The total running time is $k^{tw}(tw!)n^{O(1)}$. □

As we explained, even though Theorem 6.9 implies that Digraph Coloring is

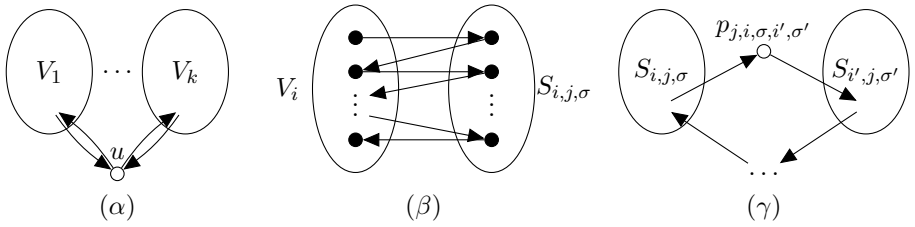


Figure 6.6: (α): For each set of variables X_i, $i \in [\log n]$, we create an independent set V_i. We connect all vertices of $\bigcup_{i\in[\log n]} V_i$ to a universal vertex u. (β): The set $S_{i,j,\sigma}$ exists only if the truth assignment σ of X_i satisfies the clause c_j. For each truth assignment σ of X_i we have defined an ordering of the vertices in V_i (unique for each assignment). We create a path that uses, alternatively, vertices from V_i and $S_{i,j,\sigma}$ and respects the ordering we defined for σ. (γ): The cycles created by the sets $S_{i,j,\sigma}$ (for a clause c_j) and the connector vertices p.

FPT parameterized by treewidth, the complexity it gives is significantly worse than the complexity of Coloring, which is essentially k^{tw}. Our main result in this section is to show that this is likely to be inevitable, even if we focus on the more restricted case of treedepth and 2 colors.

Theorem 6.10. *If there exists an algorithm which decides if a given digraph on n vertices and (undirected) treedepth td is 2-colorable in time $\mathsf{td}^{\mathsf{o}(\mathsf{td})}\mathsf{n}^{\mathsf{O}(1)}$, then the ETH is false.*

Proof. Suppose we are given a 3-SAT formula ϕ with n variables and m clauses. We will produce a digraph G such that $|V(G)| = 2^{O(n/\log n)}m$ and $\mathsf{td}(\mathsf{G}) = \mathsf{O}(\mathsf{n}/\log \mathsf{n})$ and G is 2-colorable if and only if ϕ is satisfiable. Before we proceed, observe that if we can construct such a graph the theorem follows, as an algorithm with running time $O^*(\mathsf{td}^{\mathsf{o}(\mathsf{td})})$ for 2-coloring G would decide the satisfiability of ϕ in time $2^{o(n)}$.

To simplify the presentation we assume without loss of generality that n is a power of 2 (otherwise adding dummy variables to ϕ can achieve this while increasing n be a factor of at most 2). We begin the construction of G by creating $\log n$ independent sets $V_1, \ldots, V_{\log n}$, each of size $\lceil \frac{2en}{\log^2 n} \rceil$. We add a vertex u and connect it with arcs in both directions to all vertices of $\cup_{i\in[\log n]} V_i$ (Fig. 6.6(α)). We also partition the variables of ϕ into $\log n$ sets $X_1, \ldots, X_{\log n}$ of size at most $\lceil \frac{n}{\log n} \rceil$.

The main idea of our construction is that the vertices of V_i will represent an assignment to the variables of X_i. Observe that all vertices of V_i are forced to obtain the same color (as all are forced to have a distinct color from u), therefore the way these vertices represent an assignment is via their topological ordering in the DAG they induce together with other vertices of the graph which obtain the same color.

To continue our construction, for each $i \in [\log n]$ we do the following: we enumerate all the possible truth assignments of the variables of X_i and for each such truth assignment $\sigma : X_i \to \{0,1\}^{|X_i|}$ we define (in an arbitrary way) a distinct ordering $\rho(\sigma)$ of the vertices of V_i. We will say that the ordering $\rho(\sigma)$ is the *translation* of assignment σ.

Note that there are $|V_i|! \geq (\frac{2en}{\log^2 n})! \geq (\frac{2n}{\log^2 n})^{\frac{2en}{\log^2 n}} = 2^{\frac{2en}{\log^2 n}(1+\log n - 2\log\log n)} > 2^{\lceil \frac{n}{\log n} \rceil}$ for n sufficiently large, so it is possible to translate truth assignments to X_i to orderings of V_i injectively. Note that enumerating all assignments for each group takes time $2^{O(n/\log n)} = 2^{o(n)}$.

Consider now a clause c_j of ϕ and suppose some variable of the group X_i appears in c_j. For each truth assignment σ to X_i which satisfies c_j we construct an independent set $S_{j,i,\sigma}$ of size $|X_i| - 1$, label its vertices $s^{\ell}_{j,i,\sigma}$, for $\ell \in [|X_i| - 1]$. For each ℓ we add an arc from $\rho(\sigma)^{-1}(\ell)$ to $s^{\ell}_{j,i,\sigma}$ and an arc from $s^{\ell}_{j,i,\sigma}$ to $\rho(\sigma)^{-1}(\ell + 1)$. In other words, the ℓ-th vertex of $S_{j,i,\sigma}$ has an incoming arc from the vertex of V_i which is ℓ-th according to the ordering $\rho(\sigma)$ which is the translation of assignment σ and an outgoing arc to the vertex of V_i which is in position $(\ell + 1)$ in the same ordering (these paths are presented at Fig. 6.6 (β)). Observe that this implies that if all vertices of V_i and of $S_{j,i,\sigma}$ are given the same color, then the topological ordering of the induced DAG will agree with $\rho(\sigma)$ on the vertices of V_i.

To complete the construction, for each clause c_j we do the following: take all independent sets $S_{j,i,\sigma}$ which we have constructed for c_j and order them in a cycle in some arbitrary way. For two sets $S_{j,i,\sigma}, S_{j,i',\sigma'}$ which are consecutive in this cycle add a new "connector" vertex $p_{j,i,\sigma,i',\sigma'}$, all arcs from $S_{j,i,\sigma}$ to this vertex, and all arcs from this vertex to $S_{j,i',\sigma'}$ (Fig. 6.6(γ)). Finally, we connect each connector vertex $p_{j,i,\sigma,i',\sigma'}$ we have constructed to an arbitrary vertex of V_1 with a digon. This completes the construction.

Let us argue that if ϕ is satisfiable, then G is 2-colorable. We color u with color 2, all the vertices in V_i for $i \in [\log n]$ with 1 and all connector vertices $p_{i,j,\sigma,i',\sigma'}$ with 2. For each clause c_j there exists a group X_i that contains a variable of c_j such that the supposed satisfying assignment of ϕ, when restricted to X_i gives an assignment $\sigma : X_i \to \{0,1\}^{|X_i|}$ which satisfies c_j. Therefore, there exists a corresponding set $S_{j,i,\sigma}$. Color all vertices of this set with 1. After doing this for all clauses, we color all other vertices with 2. We claim this is a valid 2-coloring. Indeed, the graph induced by color 2 is acyclic, as it contains u (but none of its neighbors) and for each c_j, all but one of the sets $S_{j,i,\sigma}$ and the vertices $p_{j,i,\sigma,i',\sigma'}$. Since these sets have been connected in a directed cycle throught connector vertices, and for each c_j we have colored one of these sets with 1, the remaining sets induce a DAG. For the graph induced by color 1 consider for each V_i the ordering $\rho(\sigma)$, where σ is the satisfying assignment restricted to V_i. Every vertex outside V_i which received color 1 and has arcs to V_i, has exactly one incoming and one outgoing arc to V_i. Furthermore, the directions of these arcs agree with the ordering $\rho(\sigma)$. Hence, since $\cup_{i\in[\log n]} V_i$ touches all arcs with both endpoints having color 1 and all such arcs respect the orderings of V_i, the graph induced by color 1 is acyclic.

For the converse direction, suppose we have a 2-coloring of G. Without loss of generality, u has color 2 and $\cup_{i\in[\log n]} V_i$ has color 1. Furthermore, all connectors $p_{j,i,\sigma,i',\sigma'}$ also have color 2. Consider now a clause c_j. We claim that there must be a group $S_{j,i,\sigma}$ such that $S_{j,i,\sigma}$ does not use color 2. Indeed, if all such groups use color 2, since they are linked in a directed cycle with all possible arcs between consecutive groups and connectors, color 2 would not induce a DAG. So, for each c_j we find a group $S_{j,i,\sigma}$ that is fully colored 1 and infer from this the truth assignment σ for the group X_i. Doing

this for all clauses gives us an assignment that satisfies every clause. However, we need to argue that the assignment we extract is consistent, that is, there do not exist $S_{j,i,\sigma}$ and $S_{j',i,\sigma'}$ which are fully colored 1 with $\sigma \neq \sigma'$. For the sake of contradiction, suppose that two such sets exist, and recall that $\rho(\sigma) \neq \rho(\sigma')$. We now observe that if $S_{j,i,\sigma} \cup V_i$ only uses color 1, then any topological ordering of V_i in the graph induced by color 1 must agree with $\rho(\sigma)$, which is a total ordering of V_i. In a similar way, the ordering of V_i must agree with $\rho(\sigma')$, so if $\sigma \neq \sigma'$ we get a contradiction.

Finally, let us argue about the parameters of G. For each clause c_j of ϕ we construct an independent set of size $O(n/\log^2 n)$ for each satisfying assignment of a group X_i containing a variable of c_j. There are at most 3 such groups, and each group has at most $2^{n/\log n}$ satisfying assignments for c_j, so $|V(G)| = 2^{O(n/\log n)}m$.

For the treedepth, recall that deleting a vertex decreases treedepth by at most 1. We delete u and all of $\cup_{i\in[\log n]}V_i$ which are $O(n/\log n)$ vertices in total. It now suffices to prove that in the remainder all components have treedepth $O(n/\log n)$. In the remainder every component is made up of the directed cycle formed by sets $S_{j,i,\sigma}$ and connectors $p_{j,i,\sigma,i',\sigma'}$. We first delete a vertex $p_{j,i,\sigma,i',\sigma'}$ to turn the cycle into a directed "path" of length $L = 2^{O(n/\log n)}$. We now use the standard argument which proves that paths of length L have treedepth $\log L$, namely, we delete the $p_{j,i,\sigma,i',\sigma'}$ vertex that is closest to the middle of the path and then recursively do the same in each component. This shows that the remaining graph has treedepth logarithmic in the length of the path, therefore at most $O(n/\log n)$. □

6.6 Coloring Tournaments

In this section we propose an algorithm that decides if a given tournament T is 2-colorable in time $O^*(\sqrt[3]{6}^n)$. Our algorithm starts by removing, arbitrarily, as many disjoint triangles from the tournament as possible and then considers all the proper partial colorings of the tournament induced on these triangles. Then we use a recursive algorithm in order to determine if any of these partial colorings can be extended to a proper 2-coloring for the whole tournament. Finally, we use the same idea in order to decide if a tournament is k-colorable.

This algorithm searches for two types of triangles in the tournament - triangles that contain one uncolored vertex and two vertices with the same color and triangles that contain only one colored vertex. For the first type, it is easy to see that we know which color we have to assign to the uncolored vertex. However, for the second type, the algorithm calls itself in order to decide if any of the possible colorings is extendable to this triangle.

Before we continue to the proof, let us recall that any tournament T that has a directed cycle must contain a triangle. Therefore, in the Algorithm 7 we know that the graph $T[V \setminus V_1]$ where V_1 is the set of (initially) removed vertices, is acyclic as we could not find any other triangles in it.

Now, let us prove that the Algorithm 8 does what we claim.

Algorithm 7 [2-COL(T) decision function]

Input: A tournament $T = (V, E)$.
Output: Is $\overrightarrow{\chi}(T) = 2$ or not?

```
1: V_1 ← ∅, V_2 ← V
2: IsTwoDC ← False
3: while there is a triangle {v_1, v_2, v_3} in V_2 do
4:     V_1 ← V_1 ∪ {v_1, v_2, v_3}
5:     V_2 ← V_2 \ {v_1, v_2, v_3}
6: end while
7: for all 2-coloring C : V_1 → {1, 2} that are proper do
8:     IsTwoDC ← IsTwoDC ∨ Ext 2-DCN(T,V_1,C)
9: end for
10: return IsTwoDC
```

Lemma 6.11. *Given a tournament $T = (V, E)$, a set of vertices $S \subseteq V$ such that $T[V \setminus S]$ is acyclic and a function $\mathcal{C} : S \to \{1, 2\}$ of S, Algorithm 8 applied to $V_C = S$ decides if we can find a function $\mathcal{C}^* : V \to \{1, 2\}$ that gives a proper 2-coloring for the tournament T such that $\mathcal{C}^*(v) = \mathcal{C}(v)$ for all $v \in S$.*

Proof. First note that if the function cannot be extended the algorithm will return **False**. Indeed, the algorithm changes the variable *Ext* to true only if there exists an extension $\mathcal{C}^*$ of $\mathcal{C}$ that is a proper coloring for the tournament. Such an extension can not exists since we have assume that $\mathcal{C}$ is not extendable. So we have to prove that if the given function $\mathcal{C}$ can be extended in order to give a proper coloring of the whole tournament then the algorithm will return **True**. For the rest of the proof let us call the triangles that contain one uncolored vertex and two vertices of the same color as *type one* and the triangles with two uncolored vertices as *type two*. Assume $\mathcal{C}$ is extendable (i.e., there is an extension $\mathcal{C}^*$ that gives a proper coloring for the tournament); the algorithm first checks if there exists a triangle of *type one* and gives to the uncolored vertices the other color (in line 5). It is clear that this is the only option for these vertices so that the new color function remains extendable. After that the algorithm checks for triangles of *type two*. In this case we know that the two uncolored vertices cannot have both the same color as the third vertex; so we have a total of $2^2 - 1 = 3$ cases. After that the algorithm checks (between lines 10 and 13) if any of these possibilities can be can be extended and gives us a proper coloring (by calling itself in line 12).

As we mentioned, the algorithm tries to extend all the possible colorings (except those that are not proper) so at some point we have an extendable function $\mathcal{C}$ and either we do not have any uncolored vertices or we do not have any triangles of *type one* or *two*.

Case 1. Suppose $V_{NC} = \emptyset$ when line 16 of Algorithm 2 is executed. Then $\mathcal{C}$ is a proper coloring of V which means that after the check in line 18 we change the value of the variable *Ext* to **True**.

It remains to show that in the second case if we colored the remaining uncolored

Algorithm 8 [Ext 2-COL(T,V_C,$\mathcal{C}$) decision function]

Input: A tournament $T = (V, E)$, a set of vertices $V_C \subseteq V$ and a function $\mathcal{C} : V_C \to \{1, 2\}$.

Output: Can we find a proper 2-coloring for T by extending $\mathcal{C}$?

1: $V_{NC} \leftarrow V \setminus V_C$
2: $Ext \leftarrow$ **False**
3: **while** there is a triangle $\{v_1, v_2, v_3\}$ such that $v_1 \in V_{NC}$, $v_2, v_3 \in V_C$ and $\mathcal{C}(v_2) = \mathcal{C}(v_3)$ **do**
4: $V_C \leftarrow V_C \cup \{v_1\}$, $V_{NC} \leftarrow V_{NC} \setminus \{v_1\}$
5: set $\mathcal{C}(v_1)$ to be the color that is not $\mathcal{C}(v_2) = \mathcal{C}(v_3)$
6: **end while**
7: **if** $\mathcal{C}$ is a proper coloring for V_C **then**
8: **while** there is a triangle $\{v_1, v_2, v_2\}$ such that $v_1, v_2 \in V_{NC}$ and $v_3 \in V_C$ **do**
9: $V_C \leftarrow V_C \cup \{v_1, v_2\}$, $V_{NC} \leftarrow V_{NC} \setminus \{v_1, v_2\}$
10: **for all** the pairs $\{Col_1, Col_2\} \neq \{\mathcal{C}(v_3), \mathcal{C}(v_3)\}$ **do**
11: set $\mathcal{C}(v_1) \leftarrow Col_1$ and $\mathcal{C}(v_2) \leftarrow Col_2$
12: $Ext \leftarrow Ext \vee$ Ext 2-DNC(T,V_C,$\mathcal{C}$)
13: **end for**
14: **end while**
15: **end if**
16: for all $v \in V_{NC}$ set $\mathcal{C}(v)$ to be 1
17: **if** $\mathcal{C}$ is a proper coloring for V **then**
18: $Ext \leftarrow$ **True**
19: **end if**
20: **return** Ext

vertices with any color we have a proper coloring for T.

Case 2. In this case we do not have any triangles of *type one* or *two*. This combined with the assumption that the coloring is extendable implies that by coloring the remaining vertices with any color we end up with a coloring that does not have any monochromatic triangle. It remains to show the following claim:

Claim 6.12. *Let $T = (V, E)$ be a tournament and $C : V \to \{1, 2\}$ be a function that is a 2-coloring of T such that there is no monochromatic triangle. Then C is a proper coloring.*

Proof. Assume that C does not give a proper coloring. Then there must exist a monochromatic cycle S with length grater than 3. Note that S induces a tournament. But any tournament which contains a directed cycle contains a triangle. This gives a contradiction since there are no monochromatic triangles in T. □

So our coloring is a proper 2-coloring; thus the algorithm will change the value of the variable Ext to **True** in line 18 and due to the logic *or* in line 12 this **True** will be kept until the algorithm terminates. □

Finally we are going to prove that Algorithm 7 decides if a tournament is 2-colorable and that it runs in $O^*(\sqrt[3]{6}^n)$ time.

Theorem 6.13. *Given a tournament $T = (V, E)$, Algorithm 7 decides if T is 2-colorable.*

Proof. It is easy to see that Algorithm 7 tries to extend any proper coloring of V_1. Now, in order to use Lemma 6.11 we need to observe that we have no triangles in V_2. Since a tournament without triangles is acyclic, it follows that V_2 is acyclic. So, from lemma 6.11 we know that if one of these colorings can be extended then the Algorithm 8 will return **True**. Thus, Algorithm 7 returns **True** if the tournament is 2-colorable and **False** otherwise. □

Theorem 6.14. *Let $T = (V, E)$ be a tournament. Then we can decide if the dichromatic number of T is two in time $O^*(\sqrt[3]{6}^n)$.*

Proof. Observe that in Algorithm 7 all the steps are polynomial except the number of the proper ways to color the set V_1 and the time Algorithm 8 needs. It is easy to see that the number of proper ways to color V_1 is at most $6^{\frac{|V_1|}{3}}$ since for every triangle in V_1 we know that we have six possible choices to color it (all except the two that give to every vertex the same color). This means that we call Algorithm 8 at most $6^{\frac{|V_1|}{3}}$ times. The running time of the second algorithm depends on the number of times that it will call itself. Now we can see that for the remaining vertices ($V_2 = V \setminus V_1$), in the worst case, we need to check three different colorings (see proof of lemma 6.11) for two vertices at a time. Thus, the running time of Algorithm 8 is $lO^*(3^{\frac{|V_2|}{2}})$. So, we can decide if T is 2-colorable in time

$$O^*(6^{\frac{|V_1|}{3}} \cdot 3^{\frac{|V_2|}{2}}) = O^*(\sqrt[3]{6}^{|V_1|+|V_2|}) = O^*(\sqrt[3]{6}^n)$$

□

Note that, Algorithm 7 can be modified in order to decide if a tournament is k-colorable. The difference is that we need to consider all the proper ways, the vertices of the type one triangles can be colored (for the definition of type one triangles see the proof of Lemma 6.11). Since, in this case, we consider k colors this affects the running time of the algorithm. The the number of proper ways to color V_1 is at most $(k^3 - k)^{\frac{|V_1|}{3}}$. The worst case for the vertices of V_2 is to be considered in triangles of type two (for the definition of type one triangles see the proof of Lemma 6.11); this gives us $k^2 - 1$ possible colorings for two vertices at a time. Therefore, the total running time is $O^*(\sqrt[3]{k^3 - k}^n)$. This gives us the following corollary.

Corollary 6.15. *Let $T = (V, E)$ be a tournament. We can decide if the dichromatic number of T is k in time $O^*(\sqrt[3]{k^3 - k}^n)$.*

6.7 Conclusions

In this chapter we have strengthened known results about the complexity of Digraph Coloring on digraphs which are close to being DAGs, precisely mapping the threshold of tractability for directed feedback vertex set and feedback arc set; and we precisely bounded the complexity of the problem parameterized by treewidth, uncovering an important discrepancy with its undirected counterpart. One question for further study is to settle the degree bound for which k-Digraph Coloring is NP-hard for directed feedback vertex set of size k, and more generally to map out how the tractability threshold for the degree evolves for larger values of the directed feedback vertex set from $4k - \Theta(1)$ to $2k + \Theta(1)$, which is the correct threshold when the size of directed feedback vertex set is unbounded. With regards to undirected structural parameters, it would be interesting to investigate whether a $\mathrm{vc}^{o(\mathrm{vc})}$ algorithm exists for 2-Digraph Coloring, where vc is the input graph's vertex cover, as it seems challenging to extend our hardness result to this more restricted case.

Chapter 7

Locally irregular induced subgraphs

7.1 Introduction

A graph G is said to be *locally irregular*, if every two adjacent vertices of G have different degrees. In this chapter, we introduce and study the problem of finding a largest locally irregular induced subgraph of a given graph. This problem is equivalent to identifying what is the minimum number of vertices that must be deleted from G, so that what remains is a locally irregular graph.

The notion of locally irregular graphs was first introduced in [21]. The most interesting aspect of locally irregular graphs, comes from their connection to the so-called 1-2-3 Conjecture, proposed in [126]. Formally, the 1-2-3 Conjecture states that for almost every graph, we should be able to place weights from $\{1, 2, 3\}$ on the edges of that graph, so that the coloring, that assigns a color to each vertex equal to the sum of the weights on its adjacent edges, is a proper vertex-coloring of the graph. An obvious connection is that this conjecture holds for locally irregular graphs. Indeed, placing weight equal to 1 to all the edges of a locally irregular graph, suffices to produce a proper vertex-coloring, as each vertex receives a color equal to its degree.

The problem we introduce belongs in a more general and well studied family of problems, which is about identifying a largest induced subgraph of a given graph that verifies a specific property Π. That is, given a graph $G = (V, E)$ and an integer k, is there a set $V' \subseteq V$ such that $|V'| \leq k$ and $G[V \setminus V']$ has the specified property Π? This generalised problem is indeed classic in graph theory, and it is known as the Induced Subgraph with Property Π (ΠSP for short) problem in [94]. In our case, the property Π is "the induced subgraph is locally irregular".

Related Work In [140] the authors showed that ΠSP is a hard problem for any property Π that is *hereditary*, *i.e.*, all induced subgraphs of G verify Π if G itself verifies that property. However, it remains interesting (one could say that it actually becomes more interesting) even if the property Π is not hereditary. Recently, the authors of [25] studied

the problem for Π being "all vertices of the induced subgraph have odd degree", which clearly is not a hereditary property. Nevertheless, they showed that this is an NP-hard problem, and they gave an fpt-algorithm that solves the problem when parameterised by the rank-width. Also, the authors of [5, 14, 159] studied the ISPΠ problem, where Π is the rather natural property "the induced subgraph is d-regular", where d is an integer given in the input (recall that a graph is said to be *d-regular* if all of its vertices have the same degree d). In particular, in [14] it is shown that finding a largest (connected) induced subgraph that is d-regular, is NP-hard to approximate, even when restricted on bipartite or planar graphs. The authors of [14] also provide a linear-time algorithm to solve this problem for graphs with bounded treewidth. In contrast, the authors of [5] take a more practical approach, as they focus on solving the problem for the particular values of $d = 1$ and $d = 2$, by using bounds from quadratic programming, Lagrangian relaxation and integer programming.

It is quite clear that, in some sense, the property that interests us lies on the opposite side of the one studied in [5, 14, 159]. However, both properties, "the induced subgraph is regular" and "the induced subgraph is locally irregular" are not hereditary. This means that we do not get an NP-hardness result directly from [140]. Furthermore, the ISPΠ problem always admits an fpt-algorithm, when parameterised by the size of the solution, if Π is a hereditary property (proven in [46, 131]), but for a non-hereditary one, this is not always true. Indeed in [159], the authors proved that when considering Π as "the induced subgraph is regular", the ISPΠ problem is W[1]-hard when parameterised by the size of the solution.

As we mentioned earlier, the 1-2-3 Conjecture seems to have some very interesting links to locally irregular graphs. There have been some steps towards proving that conjecture, which involve edge-decomposing a graph into a constant number of locally irregular subgraphs, *i.e.*, given G, find an edge-coloring of G using a constant number of colors, such that each color induces a locally irregular subgraph of G. This is the main motivation behind [21], and it seems to remain interesting enough to attract more attention [27, 143, 171].

Note that the class of locally irregular graphs can be seen as an antonym to that of *regular*,*i.e.*, graphs such that all of their vertices have the same degree. It is important to state here that there exist several alternative such notions. This is mainly due to the very well known fact that there are no non-trivial *irregular* graphs, *i.e.*, graphs that do not contain two vertices (not necessarily adjacent) with the same degree (see [49]). Thus, the literature has plenty of slightly different definitions of irregularity (see for example [6, 49, 50, 89, 170]). One way to deal with the nonexistence of irregular graphs, is to define a notion of *local* irregularity. Intuitively, instead of demanding for all vertices of a graph to have different degrees, we are now considering each vertex v separately, and request that the vertices "around" v to verify some properties of irregularity. For example, the authors of [7] study graphs G such that for every vertex v of G, no two neighbours of v have the same degree. For an overview of other interesting notions of irregularity (local or otherwise), we refer the reader to [8].

Our Contribution We begin in Section 7.2 by providing the basic notations, definitions and lemmas that are going to be used throughout this chapter. In Section 7.3, we deal with the complexity of the introduced problem. In particular, we show that the problem belongs in P if the input graph is a path, cycle, tree, complete bipartite or complete graph. We then prove that finding the maximum induced locally irregular subgraph of a given graph G is NP-hard, even if G is restricted to being a subcubic planar bipartite, or a cubic bipartite graph.

As the problem we introduce seems to be computationally hard even for rather restricted families of graphs, we proceed by investigating its approximability. Unfortunately, we prove in Section 7.4 that for any bipartite graph G of order n and $k \geq 1$, there can be no polynomial time algorithm that finds an approximation of $\mathrm{I}(G)$ within ratio $O(n^{1-\frac{1}{k}})$, unless P=NP. Nevertheless, we do manage to give a (simple) d-approximation algorithm for d-regular bipartite graphs.

We then decide to look into its parameterised complexity. In Section 7.5, we present two algorithms that compute $\mathrm{I}(G)$, each one considering different parameters. The first considers the size of the solution k and the maximum degree Δ of G, and and has running time $(2\Delta)^k n^{O(1)}$, while the second considers the treewidth tw and Δ of G, and has running time $\Delta^{3tw} n^{O(1)}$. Unfortunately, these algorithms can be considered as being FPT only if Δ is part of the parameter. In Section 7.5.3, we present two linear fpt-reductions which prove that the problem is W[2]-hard when parameterised only by the size of the solution and W[1]-hard when parameterised only by the treewidth. These reductions also show that we can not even have an algorithm that computes $\mathrm{I}(G)$ in time $f(k)n^{o(k)}$ or $O^*(f(tw)n^{o(tw)})$, unless the ETH fails. The O^* notation is used to suppress polynomial factors in regards to n and tw.

7.2 Preliminaries

Let $G = (V, E)$ be a graph. We say that G is *locally irregular* if for every edge $uv \in E$, we have $d(u) \neq d(v)$. Now, let $S \subseteq V$ be such that $G[V \setminus S]$ is a locally irregular graph; any set S that has this property is said to be an *irregulator of* G. For short, we say that S is an $ir(G)$. Moreover, let $\mathrm{I}(G)$ be the minimum order that any $ir(G)$ can have. We say that S is a *minimum* irregulator of G, for short S is an $ir^*(G)$, if S is an $ir(G)$ and $|S| = \mathrm{I}(G)$.

We also define the following notion, which generalises $ir(G)$. Let $G = (V, E)$ be a graph, $S, X \subseteq V$ and let $G' = G[V \setminus S]$. Now, let $S \subseteq V$ be such that, for each two neighbouring vertices u, v in $X \setminus S$, we have that $d_{G'}(u) \neq d_{G'}(v)$; any set S that has this property is said to be an *irregulator of* X *in* G, for short $ir(G, X)$. We define the notions of $ir^*(G, X)$ and $\mathrm{I}(G, X)$ analogously to the previous definitions.

We now provide some lemmas and an observation that will be useful throughout this chapter. In the three lemmas below, we investigate the relationship between $\mathrm{I}(G)$ and $\mathrm{I}(G, X)$.

Lemma 7.1. *Let $G = (V, E)$ be a graph and let $X \subseteq V$. Then $\mathrm{I}(G, X) \leq \mathrm{I}(G)$.*

Proof. Let S be an $ir^*(G)$, $G' = G[V \setminus S]$ and $X' = X \setminus S$. Observe that for each pair of vertices u, v such that $u \in X'$ and $v \in N_{G'}(u) \cap X'$, we have that $d_{G'}(u) \neq d_{G'}(v)$, since S is an $ir^*(G)$. It follows that S is also an irregulator of X in G, *i.e.* S is an $ir(G, X)$, and thus we have that $\mathrm{I}(G, X) \leq |S| = \mathrm{I}(G)$. □

Lemma 7.2. *Let $G = (V, E)$ be a graph and $S, X \subseteq V$ such that S is an $ir^*(G, X)$. Then, $S \subseteq N[X]$ and $\mathrm{I}(G, X) = \mathrm{I}(G[N[X]], X)$.*

Lemma 7.3. *Let $G = (V, E)$ be a graph, and $X_1, \ldots, X_n \subseteq V$ such that $N[X_i] \cap N[X_j] = \emptyset$ for every $1 \leq i < j \leq n$. Then $\sum_{i=1}^{n} \mathrm{I}(G, X_i) \leq \mathrm{I}(G)$.*

In this final lemma, we show that an irregulator of a graph is also an irregulator of any subset of the vertices of the graph, even if we only consider the neighbourhood of that subset.

Lemma 7.4. *Let $G = (V, E)$ be a graph, X be a subset of V and S be an $ir(G)$. The set $S \cap N[X]$ is an $ir(G, X)$ and an $ir(G[N[X]], X)$.*

The proofs of the above lemmas are included in the appendix.

The following, almost trivial, observation, will be useful throughout the rest of the chapter.

Observation 1. *Let $G = (V, E)$ be a graph and S be an $ir(G)$. Then, for each edge $uv \in E$, if $d(u) = d(v)$, then S contains at least one vertex in $N[\{u, v\}]$. Additionally, for a set $X \subseteq V$, let S^* be an $ir(G[N[X]], X)$. Then for each edge $uv \in E(G[X])$, if $d(u) = d(v)$, then S^* contains at least one vertex in $N[\{u, v\}]$.*

7.3 (Classic) complexity

In this section, we deal with the complexity of the problem we introduced. In Section 7.3.1 we present all the families of graphs for which we prove that $\mathrm{I}(G)$ is computed in polynomial time. Then in Section 7.3.2, we prove that, in general, computing $\mathrm{I}(G)$ is NP-hard.

7.3.1 Polynomial Cases

Theorem 7.5. *Let G be a graph. If $G = K_n$, then $\mathrm{I}(G) = n - 1$. Also, if $G = K_{n,m}$ with $0 \leq n \leq m$, then $\mathrm{I}(G) \leq 1$ with the equality holding if and only if $n = m$.*

Proof. Let $G = (V, E)$. Assume that $G = K_n$, and let S be an $ir(G)$ with $|S| < n - 1$. Then $G' = G[V \setminus S]$ is a complete graph of order $n' > n - (n - 1) = 1$, and for any $n' \geq 2$, we have that $K_{n'}$ is not locally irregular, leading to a contradiction.

Observe that $K_{n,m}$, with $0 \leq n < m$, is locally irregular, and thus $\mathrm{I}(K_{n,m}) = 0$ in this case. Assume now that $G = K_{n,n}$ with $n \geq 1$. We have that $\mathrm{I}(G) \geq 1$ as $K_{n,n}$ is not locally irregular. Let L, R be the two bipartitions of V, with $|L| = n$ and $|R| = n$. Consider the set $S = \{v\}$, where v is any vertex of L. Clearly, after the deletion of v, the graph $G' = G[V \setminus S]$ is isomorphic to $K_{n-1,n}$ which is locally irregular. □

Theorem 7.6. *Let P_n be the path on n vertices, then*

$$\mathrm{I}(P_n) = \begin{cases} \lfloor \frac{n}{4} \rfloor, & \text{if } n \not\equiv 2 \bmod 4 \\ \lfloor \frac{n}{4} \rfloor + 1, & \text{if } n \equiv 2 \bmod 4 \end{cases}$$

Proof. We begin our proof by examining the cases of P_1, P_2, P_3, and P_4. Observe first of all; that P_1 and P_3 are locally irregular graphs. It follows that $\mathrm{I}(P_1) = \mathrm{I}(P_3) = 0$.

On the other hand, it is also easy to check that P_2 is not locally irregular, but that deleting any one of its vertices suffices to turn it into P_1 (which is locally irregular). It follows that $\mathrm{I}(P_2) = 1$. We now show that $\mathrm{I}(P_4) = 1$. Let $P_4 = v_1v_2v_3v_4$ and note that P_4 is not locally irregular (we have that $d(v_2) = d(v_3) = 2$). Moreover, deleting either v_1 or v_4 from P_4, results in the graph P_3, which is locally irregular. Thus $\mathrm{I}(P_4) = 1$. Observe moreover that any path on more than 4 vertices is not locally irregular.

We are now ready to continue with the proof. Let $n, k, d \in \mathbb{N}$, with $n \geq 5$, $n \equiv k \bmod 4$, $d = \lfloor \frac{n}{4} \rfloor$ and $G = P_n = v_1 \dots v_n$. We have the following two cases:

- $k \neq 2$. Consider the set $S = \{v_i : i \equiv 0 \bmod 4\}$. We have that $|S| = d$. Also, observe that the graph $G[V(G) \setminus S]$ has d connected components, each one of which is isomorphic to P_3, which are locally irregular, and a connected component isomorphic to P_k, where $k \in \{0, 1, 3\}$, which is also locally irregular (the graph P_0 is the null graph). It follows that S is an $ir(P_n)$ and that $\mathrm{I}(P_n) \leq |S| = d$. All that is left to show is that $\mathrm{I}(P_n) \geq d$. Let us assume that there exists a set S_0 that is an $ir(P_n)$ and $|S_0| < d$. Now observe that $G[V(G) \setminus S_0]$ contains at least one connected component isomorphic to P_m, with $m \geq 4$ This is a contradiction, since P_m is not locally irregular.

- $k = 2$. Consider the set $S = \{v_i : i \equiv 0 \bmod 4\} \cup \{v_n\}$. We have that $|S| = d + 1$. Similarly to the previous case, we have that $G[V(G) \setminus S]$ contains d connected components isomorphic to P_3 and one connected component isomorphic to P_1. Thus S is an $ir(P_n)$ and $\mathrm{I}(P_n) \leq |S| \leq d + 1$. All that is left to show is that $\mathrm{I}(P_n) \geq d + 1$. Observe that the arguments supporting that $\mathrm{I}(P_n) > d$ are the same as the previous case. So, we assume that there exists a set S_0 that is an $ir(P_n)$ and $|S_0| = d$. Observe that all the connected components of $G[V(G) \setminus S_0]$ are paths. Also, if there exists a connected component isomorphic to a P_m, with $m \geq 4$, then $G[V(G) \setminus S_0]$ is not locally irregular. So we may assume that all the connected components of $G[V(G) \setminus S_0]$ are isomorphic to a paths on at most 3 vertices. It follows that one of these components must be isomorphic to P_2, and P_2 is not locally irregular. This is a contradiction.

□

Corollary 7.7. *Let C_n be the cycle on $n \geq 3$ vertices, then* $\mathrm{I}(C_n) = \mathrm{I}(P_{n-1}) + 1$.

To explain the above statement, observe that for every vertex v belonging to the cycle $G = C_n$, we have that $d(v) = 2$. Thus, we know that $\mathrm{I}(C_n) \geq 1$ and that any S

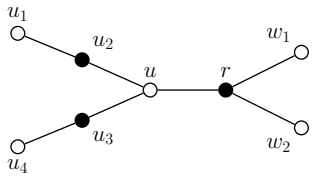


Figure 7.1: The gadget used in the proof of Theorem 7.9. The white and black vertices are used to denote vertices belonging to different bipartitions.

that is an $ir(G)$ contains at least one vertex, say vertex v. The statement follows by observing that the graph $G[V(G) \setminus v]$ is isomorphic to P_{n-1}.

Another interesting case is when G is a tree. In this case, we can calculate $\mathrm{I}(G)$ in polynomial time as a direct consequence of Theorem 7.17 (see Section 7.5). Indeed, in that theorem we provide an algorithm that computes $\mathrm{I}(G)$ in time $\Delta^{2tw} n^{O(1)}$. The next corollary follows directly from the fact that trees have $tw = 1$ (by definition) and $\Delta = O(n)$.

Corollary 7.8. *Let T be a tree. There exists a polynomial time algorithm that computes $\mathrm{I}(T)$.*

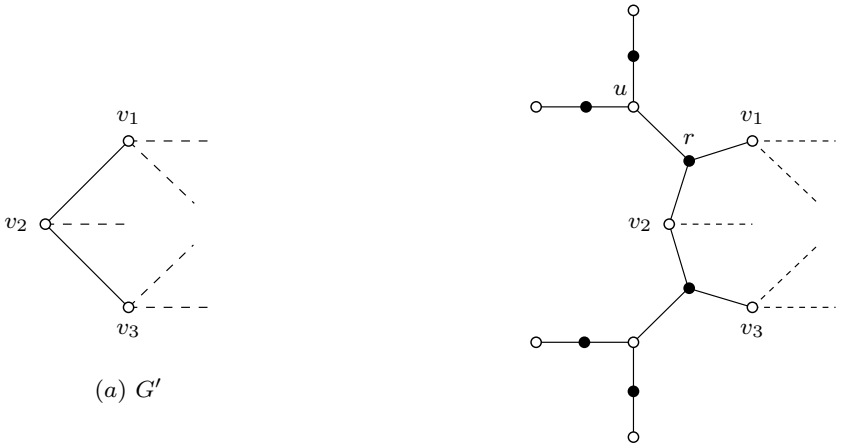
7.3.2 NP-Hard Cases

We now show that finding a minimum irregulator of a graph is NP-hard. Interestingly, this remains true even for quite restricted families of graphs, such as cubic (*i.e.*, 3-regular) bipartite graphs and subcubic (*i.e.*, of maximum degree at most 3) planar bipartite graphs.

Theorem 7.9. *Let G be graph and $k \in \mathbb{N}$. Deciding if $\mathrm{I}(G) \leq k$ is NP-complete, even when G is a planar bipartite graph with maximum degree $\Delta \leq 3$.*

Proof. Since the problem is clearly in NP, we focus on proving it is also NP-hard. The reduction is from the Vertex Cover problem, which remains NP-complete when restricted to planar cubic graphs [156]. In that problem, a planar cubic graph G and an integer $k \geq 1$ are given as an input. The question is, whether there exists a vertex cover of G of order at most k. That is, whether there exists a set $VC \subseteq V(G)$ such that for every edge $uv \in E(G)$, at least one of u and v belongs in VC and $|VC| \leq k$.

Let G' be a planar cubic graph and $k \geq 1$ given as input for Vertex Cover. Let $|E(G')| = m$. We construct a planar bipartite graph G as follows; we start with the graph G', and modify it by using multiple copies of the gadget, illustrated in Figure 7.1. Note that we follow the naming convention illustrated in Figure 7.1 whenever we talk about the vertices of our gadgets. When we say that we *attach* a copy H of the gadget to the vertices v and v' of G', we mean that we add H to G', and we identify the vertices w_1 and w_2 to the vertices v and v' respectively. Now, for each edge $vv' \in E(G')$, attach one copy H of the gadget to the vertices v and v', and then delete the edge vv' (see Figure 7.2). Clearly this construction is achieved in linear time (we have added m copies



(a) G' (b) G

Figure 7.2: The construction in the proof of Theorem 7.9. The graph G' is the initial planar cubic graph, and G is the graph built during our reduction. In G, the white and black vertices are used to denote vertices belonging to different bipartitions.

of the gadget). Note also that the resulting graph G has $\Delta(G) = 3$ and that the planarity of G' is preserved since G is constructed by essentially subdividing the edges of G' and adding a tree pending from each new vertex. Also, G is bipartite. Indeed, observe that after removing the edges of $E(G')$, the vertices of $V(G')$ form an independent set of G. Furthermore, the gadget is bipartite, and the vertices w_1, w_2 (that have been identified with vertices of $V(G')$) belong to the same bipartition (in the gadget). Finally, for any $1 \leq i \leq m$, let H_i be the i^{th} copy of the gadget attached to vertices of G'. We use the vertices r^i and u^i to denote the copies of the vertices r and u (respectively) that also belong to H_i.

We are now ready to show that the minimum vertex cover of G' has size k' if and only if $\mathrm{I}(G) = k'$.

Let VC be a minimum vertex cover of G' and $|VC| = k'$. We show that the set $S = VC$ is an $ir(G)$. Let $G^* = G[V(G) \setminus S]$. First, note that S contains only vertices of G'. Thus, for each i, the vertices of H_i except from r^i, which also remain in G^*, have the same degree in G' and in G^*. Also note that each vertex of G' is adjacent only to copies of r. It follows that it suffices to only consider the vertices r^i to show that VC is an $ir(G)$. Now, for any $1 \leq i \leq m$, consider the vertex r^i. Since VC is a vertex cover of G', for each edge $vv' \in E(G')$, VC contains at least one of v and v'. It follows that $d_{G^*}(r^i) \leq 2$. Note also that $N_{G^*}(r^i)$ contains the vertex $u^i \in V(H_i)$ and possibly one vertex $v \in V(G')$.

Also, since we only delete vertices in $V(H_i) \cap V(G')$, we have that $d_{G^*}(u^i) = 3 > d_{G^*}(r^i)$. In the case where $N_{G^*}(r^i)$ also contains a vertex $v \in V(G')$, the vertex v is adjacent only to vertices which do not belong in $V(G')$. Thus, $d_{G^*}(v) = d_G(v) = 3 > d_{G^*}(r^i)$. It follows that r^i has a different degree from all of its neighbours and that VC

is an $ir(G)$.

Now, we prove that if $\mathrm{I}(G) = k'$ then there exists a vertex cover of size at most k'. Assume that $\mathrm{I}(G) = k'$ and let S be an $ir^*(G)$. Observe that since S is an $ir^*(G)$, S contains at least one vertex of H_i (for each $1 \leq i \leq m$). Let $X_i = V(H_i) \cap V(G')$. To construct a vertex cover VC of G' with $|VC| \leq k'$, we work as follows. For each $1 \leq i \leq m$:

1. for each vertex $v \in X_i$, if $v \in S$ then put v in VC. Then,
2. if $S \cap X_i = \emptyset$, put any one of the two vertices of X_i in VC.

Observe now that any vertex that is added to VC during step 1. of the above procedure, also belongs to S and any vertex that is added during step 2. of the above procedure corresponds to at least one vertex in S. It follows that $|VC| \leq k'$. Also note that VC contains at least one vertex of X_i, for each i, and that for each $uv \in E(G')$, there exists an i such that $V(X_i) = \{u, v\}$. Thus VC is indeed a vertex cover of G'.

Therefore G' has a minimum vertex cover of size k' if and only if $\mathrm{I}(G) = k'$. To complete the proof note that deciding if $\mathrm{I}(G) = k' < k$ for a given k, answers the question whether G' has a vertex cover of size less than k or not. □

In the following theorem we show that calculating $\mathrm{I}(G)$ is NP-hard even if G is a cubic bipartite graph.

Theorem 7.10. *Let G be graph and $k \in \mathbb{N}$. Deciding if $\mathrm{I}(G) \leq k$ is NP-complete even in cubic bipartite graphs.*

The proof of this theorem is included in the appendix.

7.4 (In)approximability

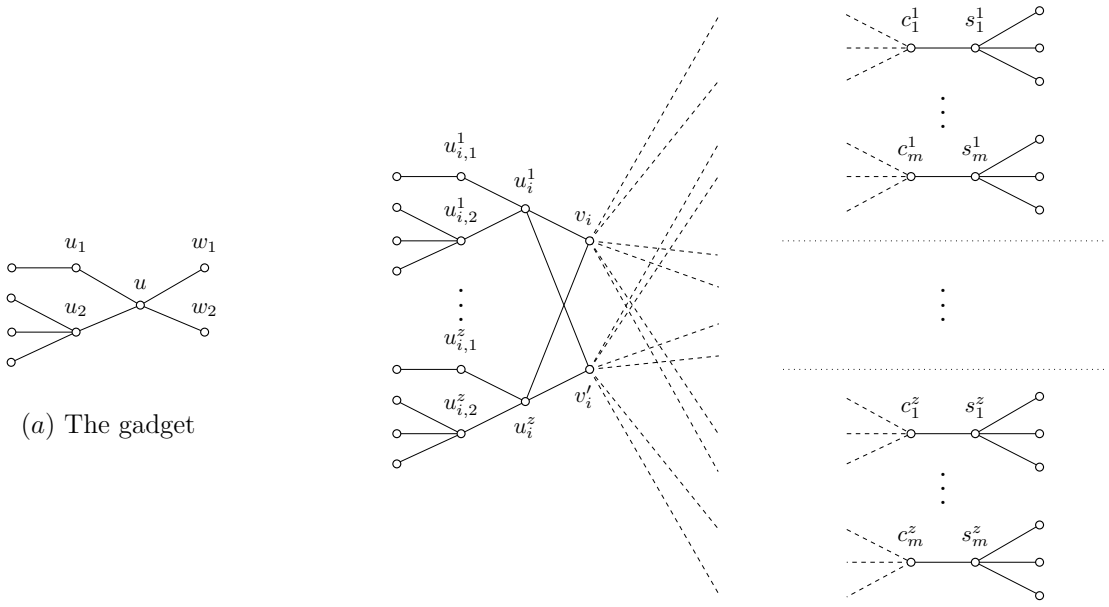
In the previous section we showed that computing $\mathrm{I}(G)$ is NP-hard, even for graphs G belonging to quite restricted families of graphs. So the natural question to pose next, which we investigate in this section, is whether we can approximate $\mathrm{I}(G)$. Unfortunately, most of the results we present below are once again negative.

We start with a corollary that follows from the proof of Theorem 7.9 and the inapproximability of Vertex Cover in cubic graphs [56]:

Corollary 7.11. *Given a graph G, it is NP-hard to approximate $\mathrm{I}(G)$ to within a ratio of $\frac{100}{99}$, even if G is bipartite and $\Delta(G) = 3$.*

Now, we are going to show that there can be no algorithm that approximates $\mathrm{I}(G)$ to within any decent ratio in polynomial time, unless P=NP, even if G is a bipartite graph (with no restriction on its maximum degree).

Theorem 7.12. *Let G be a bipartite graph of order N and $k \in \mathbb{N}$ be a constant such that $k \geq 1$. It is NP-hard to approximate $\mathrm{I}(G)$ to within $O(N^{1-\frac{1}{k}})$.*



(*a*) The gadget

(*b*) The constructed graph G for $n^{k'} = z$

Figure 7.3: The construction in the proof of Theorem 7.12. In subfigure (*b*), we illustrate how each pair of literal vertices is connected to the rest of the graph. Whenever there is an upper index $1 \leq l \leq n^{k'}$ on a vertex, it is used to denote the l^{th} copy of that vertex. The dashed lines are used to represent the edges between the literal and the clause vertices.

Proof. The proof is by a *gap producing reduction* from 2-Balanced 3-SAT, which was proven to be NP-complete in [29]. In that problem, a 3CNF formula F is given as an input, comprised by a set C of clauses over a set of Boolean variables X. In particular, we have that each clause contains exactly 3 literals, and each variable $x \in X$ appears in F exactly twice as a positive and twice as a negative literal. The question is, whether there exists a truth assignment to the variables of X satisfying F.

Let F be a 3CNF formula with m clauses $C_1, \ldots, C_m$ and n variables $x_1, \ldots, x_n$ that is given as input to the 2-Balanced 3-SAT problem. Let $2k = k' + 1$. Based on the instance F, we are going to construct a bipartite graph $G = (V, E)$ where $|V| = O(n^{k'+1})$ and

- $\mathrm{I}(G) \leq n$ if F is satisfiable
- $\mathrm{I}(G) > n^{k'}$ otherwise.

To construct $G = (V, E)$, we start with the following graph: for each literal x_i ($\neg x_i$ resp.) in F, add a *literal vertex* v_i (v'_i resp.) in V, and for each clause C_j of F, add a *clause vertex* c_j in V. Next, for each $1 \leq j \leq m$, add the edge $v_i c_j$ ($v'_i c_j$ resp.) if the literal x_i ($\neg x_i$ resp.) appears in C_j according to F. Observe that the resulting graph

is bipartite, for each clause vertex c we have $d(c) = 3$ and for each literal vertex v we have $d(v) = 2$ (since in F, each variable appears twice as a positive and twice as a negative literal). To finish the construction of G, we make use of the gadget shown in Figure 7.3(a), as well as some copies of S_5, the star on 5 vertices. When we say that we *attach* a copy H of the gadget to the vertices v_i and v'_i (for some $1 \leq i \leq n$), we mean that we add H to G, and we identify the vertices w_1 and w_2 to the vertices v_i and v'_i respectively. Now:

- for each $1 \leq i \leq n$, we attach $n^{k'}$ copies of the gadget to the vertices v_i and v'_i of G. For convenience, we give unique names to the vertices corresponding to each gadget added that way. So, the vertex u_i^l (for $1 \leq l \leq n^{k'}$ and $1 \leq i \leq n$) is used to represent the vertex u of the l^{th} copy of the gadget attached to v_i and v'_i, and $u_{i,1}^l$ ($u_{i,2}^l$ resp.) is used to denote the vertex u_1 (u_2 resp.) of that same gadget. Then,

- for each $1 \leq j \leq m$, we add $n^{k'} - 1$ copies of the clause vertex c_j to G, each one of these copies being adjacent to the same literal vertices as c_j. For $1 \leq l \leq n^{k'}$, the vertex c_j^l is the l^{th} copy of c_j. Finally,

- for each $1 \leq j \leq m$ and $1 \leq l \leq n^{k'}$, we add a copy of the star on 5 vertices S_5 to G and identify any degree-1 vertex of S_5 to c_j^l. Let s_j^l be the neighbour of c_j^l that also belongs to a copy of S_5.

Observe that the resulting graph G (illustrated in Figure 7.3(b)) remains bipartite and that this construction is achieved in polynomial time in regards to $n + m$.

From the construction of G, we know that for every $1 \leq i \leq n$, $d(v_i) = d(v'_i) = \Theta(n^{k'})$. So, for sufficiently large n, the only pairs of adjacent vertices of G that have the same degrees are either the vertices u_i^l and $u_{i,2}^l$, or the vertices c_j^l and s_j^l (for every $1 \leq i \leq n$, $1 \leq l \leq n^{k'}$ and $1 \leq j \leq m$).

First, let F be a satisfiable formula and let t be a satisfying assignment of F. Also, let S be the set of literal vertices v_i (v'_i resp.) such that the corresponding literals x_i ($\neg x_i$ resp.) are assigned value *true* by t. Clearly $|S| = n$. We also show that S is an $ir(G)$. Consider the graph $G' = G[V \setminus S]$. Now, for any $1 \leq i \leq n$, we have that either v_i or v'_i, say v_i, belongs to the vertices of G'. Now for every $1 \leq l \leq n^k$, we have that $d_{G'}(u_i^l) = 3$, while $d_{G'}(u_{i,1}^l) = 2$ and $d_{G'}(u_{i,2}^l) = 4$ (since none of the neighbours of $u_{i,1}^l$ and $u_{i,2}^l$ belongs to S). Also, for every $1 \leq j \leq m$ and $1 \leq l \leq n^{k'}$, since t is a satisfying assignment of F, $N(c_j^l)$ contains at least one vertex in S. It follows that $d_{G'}(c_j^l) = 3 < 4 = d_{G'}(s_j^l)$. Finally, since S does not contain any neighbours of v_i, we have that $d_{G'}(v_i) = d_G(v_i) = O(n^{k'})$. It follows that S is an $ir(G)$ and thus that $\mathrm{I}(G) \leq n$.

Now let F be a non-satisfiable formula and assume that there exists an S that is an $ir(G)$ with $|S| \leq n^{k'}$. As usual, let $G' = G[V \setminus S]$. Then:

1. For every $1 \leq j \leq m$, there exists a literal vertex v such that $v \in N(c_j^l)$ for every $1 \leq l \leq n^{k'}$. Assume that this is not true for a specific j. Then, since

$d_G(c_j^l) = d_G(s_j^l) = 4$, for every $1 \leq l \leq n^{k'}$, we have that S contains at least one vertex in $N[\{c_j^l, s_j^l\}]$, which does not belong to the literal vertices. That is, S contains at least one (non-literal) vertex for each one of the $n^{k'}$ copies of c_j. Observe also that even if this is the case, S would also have to contain at least one more vertex to, for example, stop $u_{i,2}^1$ and u_i^1, from having the same degree in G'. It follows that $|S| > n^{k'}$, which is a contradiction.

2. For every $1 \leq i \leq n$, S does not contain both v_i and v_i'. Assume this is not true for a specific i. Then, for every $1 \leq l \leq n^{k'}$, we have that $d_{G'}(u_i^l) = d_{G'}(u_{i,1}^l) = 2$, unless S also contains an additional vertex of the gadgets attached to v_i and v_i', for each one of the $n^{k'}$ such gadgets. It follows that $|S| \geq n^{k'}$. Since we have also assumed that for a specific i, both v_i and v_i' belong to S, we have that $|S| > n^{k'}$, a contradiction.

3. For every $1 \leq i \leq n$, S contains at least one of v_i and v_i'. Assume this is not true for a specific i. Then, for every $1 \leq l \leq n^{k'}$, we have that $d_{G'}(u_i^l) = d_{G'}(u_{i,2}^l) = 4$, unless S also contains an additional vertex of the gadgets attached to v_i and v_i', for each one of the $n^{k'}$ such gadgets. Even if this is the case, S would also have to contain at least one more vertex to, for example, stop c_1^1 and S_1^1 from having the same degree in G'. It follows that $|S| > n^{k'}$, which is a contradiction.

So from items 2. and 3. above, it follows that for each $1 \leq i \leq n$, S contains exactly one of v_i and v_i'. Now consider the following truth assignment: we assign the value *true* to every variable x_i if the corresponding literal vertex v_i belongs in S, and value *false* to every other variable. Now, from item 1. above, it follows that each clause C_j contains either a positive literal x_i which has been set to *true*, or a negative literal $\neg x_i$ which has been set to *false*. Thus F is satisfied, which is a contradiction.

Up to this point, we have shown that there exists a graph $G = (V, E)$ with $|V(G)| = N = O(n^{k'+1})$ where

- $\mathrm{I}(G) \leq n$ if F is satisfiable
- $\mathrm{I}(G) > n^{k'}$ otherwise.

Therefore, we have that $\mathrm{I}(G)$ is not $O(n^{k'-1})$ approximable in polynomial time unless P=NP.

Now, since $N = |V(G)| = \Theta(n^{k'+1})$ and $2k = k' + 1$ we have $O(n^{k'-1}) = O(N^{\frac{k'-1}{k'+1}}) = O(N^{1-\frac{2}{k'+1}}) = O(N^{1-\frac{1}{k}})$. This ends the proof of this theorem. □

Now, we consider the case where G is regular bipartite graph. Below we present an upper bound to the size of $\mathrm{I}(G)$. This upper bound is then used to obtain a (simple) Δ-approximation of an optimal solution.

Theorem 7.13. *For any d-regular bipartite graph $G = (L, R, E)$ of order n we have that $\mathrm{I}(G) \geq n/2d$.*

Proof. Let S be an $ir^*(G)$ and $G' = G[(L \cup R) \setminus S]$. We distinguish two cases according to if S is a subset of one of the bipartitions L or R, or if S contains at least one vertex from each bipartition.

Let us first deal we the first case and assume, w.l.o.g, that $S \subseteq L$. If $|S| = |L|$ then the theorem holds (since $|L| = |R|$). Therefore, we consider the case $|S| < |L|$. Observe that any vertex $v \in R$ must have $d_{G'}(v) < d$. Indeed, since $S \subseteq L$, we know that for any vertex $u \in N_{G'}(v)$, we have $d_{G'}(v) = d$ and S is an irregulator. It follows that $N(S) = R$ so $d|S| \geq n/2$ which gives us $\mathrm{I}(G) \geq n/2d$.

Now, we consider the second case (S contains at least one vertex from each bipartition). Let $L_S = S \cap L$ and $R_S = S \cap R$. We partition L (R respectively) in to three sets: the $L_S = S \cap L$ ($R_S = S \cap R$ resp.), the $L_{d-1} = \{u \mid u \in L \setminus S \text{ and } d_{G'}(u) < d\}$ ($R_{d-1} = \{u \mid u \in R \setminus S \text{ and } d_{G'}(u) < d\}$) and the $L_d = L \setminus (L_S \cup L_{d-1})$ ($R_d = L \setminus (R_S \cup R_{d-1})$ resp.). Note that for all $u \in L_d \cup R_d$ we have $d_{G'}(u) = d$. Therefore all the vertices in L_d have exactly d neighbours in R_{d-1} (in both G and G') and all the vertices in R_d have exactly d neighbours in L_{d-1}. Furthermore, since $d_{G'}(u) < d$ for all $u \in L_{d-1} \cup R_{d-1}$, we know that each $u \in L_{d-1}$ has at least one neighbour in R_S and each $u \in R_{d-1}$ has at least one neighbour in L_S. Now we are going to find some upper bounds on the number of vertices in L_{d-1} and L_d.

Since each vertex $u \in L_{d-1}$ has at least one neighbour in R_S, we have that $|L_{d-1}| \leq d|R_S|$. Similarly we can show that $|R_{d-1}| \leq d|L_S|$.

Let E^* be the set of edges between L_d and R_{d-1}. Since, any vertex of L_d have exactly d neighbours in R_{d-1} we know that $|L_d| = |E^*|/d$. Since each vertex of R_{d-1} has at least one neighbours in L_S, it has at most $d-1$ edges in the E^*. Therefore $|E^*| \leq (d-1)|R_{d-1}| \leq d(d-1)|L_S|$. This gives us that $|L_d| \leq (d-1)|L_S|$.

Now, observe that $|L| = |L_S| + |L_{d-1}| + |L_d| \leq |L_S| + d|R_S| + (d-1)|L_S| = d(|L_S| + |R_S|)$. So, since $S = L_S \cup R_S$, we have that $\mathrm{I}(G) \geq n/2d$. □

Now recall that in any bipartite graph G, any bipartition of G is a vertex cover of G. Also observe that any vertex cover of a graph G, is also an irregulator of G. Indeed, deleting the vertices of any vertex cover of G, leaves us with an independent set, which is locally irregular. The next corollary follows from these observations and Theorem 7.13:

Corollary 7.14. *For any d-regular bipartite graph $G = (L, R, E)$, any of the sets L and R is a d-approximation of $ir^*(G)$.*

7.5 Parameterised complexity

As the problem of computing a minimal irregulator of a given graph G seems to be rather hard to solve, and even to approximate, we focused our efforts towards finding parameterised algorithms that can solve it. In Section 7.5.1 we present an fpt-algorithm that calculates $\mathrm{I}(G)$ when parameterised by the size of the solution and Δ, the maximum degree of the graph. Then, we turn our attention towards graphs that are "close to being trees", that is graphs of bounded treewidth. Indeed, in Section 7.5.2 we provide an fpt-algorithm that finds a minimum irregulator of G, when parameterised by the treewidth

of the input graph and by Δ. Observe that both of our algorithms have to consider Δ as part of the parameter if they are to be considered as FPT. The natural question to ask at this point is whether we can have an fpt-algorithm, when parameterised only by the size of the solution, or the treewidth of the input graph. In Section 7.5.3, we give a strong indication towards the negative answer for both cases, proving that, in some sense, the algorithms provided in Sections 7.5.1 and 7.5.2 are optimal.

7.5.1 FPT by the size of the solution and Δ

Let us first present the following lemma:

Lemma 7.15. *Let $G = (V, E)$ be a graph such that, G is not locally irregular, and S be an $ir^*(G)$. Furthermore let $G_v = (V', E')$ be the graph $G[V \setminus \{v\}]$ for a vertex $v \in S$. Then $\mathrm{I}(G_v) = \mathrm{I}(G) - 1$.*

Proof. First observe that $S' = S \setminus \{v\}$ must be an $ir(G_v)$ as $G_v[V' \setminus S'] = G[V \setminus S]$. It follows that $\mathrm{I}(G_v) \leq \mathrm{I}(G) - 1$. Assume that $\mathrm{I}(G_v) < \mathrm{I}(G) - 1$. Then these exists an S'' such that $|S''| < \mathrm{I}(G) - 1$ and S'' is an $ir(G_v)$. Since $G_v[V' \setminus S''] = G[V \setminus (S'' \cup \{v\})]$, we have that $S'' \cup \{v\}$ is an $ir(G)$ and $|S'' \cup \{v\}| = |S''| + 1 < \mathrm{I}(G)$. This is a contradiction. □

We are now ready to show the following:

Theorem 7.16. *For a given graph $G = (V, E)$ with $|V| = n$ and maximum degree Δ, and for $k \in \mathbb{N}$, there exists an algorithm that decides if $\mathrm{I}(G) \leq k$ in time $(2\Delta)^k n^{O(1)}$.*

Proof. In order to decide if $\mathrm{I}(G) \leq k$ we are going to use a recursive algorithm. The algorithm has input (G, k), where $G = (V, E)$ is a graph and $k \geq 0$ is an integer. The basic idea of this algorithm, is to take advantage of Observation 1. We present the exact procedure in Algorithm 9.

Now, let us argue about the correctness and the efficiency of this algorithm. We claim that for any graph $G = (V, E)$ and any integer $k \geq 0$, Algorithm 9 returns yes if $\mathrm{I}(G) \leq k$ and no otherwise. Furthermore, the number of steps that the algorithm requires, is $f(k, n) = (2\Delta)^k n^{O(1)}$, where $n = |V|$. We prove this by induction on k.

Base of the induction ($k = 0$)**:** Here, we only need to check if G is locally irregular. Algorithm 9 does this in line 1 and returns yes if it is (line 2) and no otherwise (line 4). Furthermore, we can check if G is locally irregular in polynomial time. So, the claim is true for the base.

Induction hypothesis ($k = k_0 \geq 0$)**:** We assume that we have a $k_0 \geq 0$ such that Algorithm 9 can decide if any graph G with n vertices and maximum degree Δ has $\mathrm{I}(G) \leq k_0$ in $f(k_0, n) = (k_0 + 1)(2\Delta)^{k_0} n^{O(1)}$ steps.

Induction step ($k = k_0 + 1$)**:** Let $G = (V, E)$ be a graph. If G is locally irregular then $\mathrm{I}(G) = 0$ and Algorithm 9 answers correctly (in line 2). Assume that G is not locally irregular; then there exist an edge $vu \in E$ such that $d_G(v) = d_G(v)$. Now, let S be an $ir^*(G)$. It follows from Observation 1 that S must include at least one vertex $w \in N_G[\{v, u\}]$. Since Algorithm 9 considers all the vertices in $N_G[\{v, u\}]$, at some point it also considers the vertex $w \in S \cap N_G[\{v, u\}]$. Now, observe that for any

Algorithm 9 [IsIrregular(G, k) decision function]

Input: A graph $G = (V, E)$ and an integer $k \geq 0$.
Output: Is $\mathrm{I}(G) \leq k$ or not?

```
1: if G is irregular then
2:     return yes
3: else if k = 0 then
4:     return no
5: else                                          ▷ k > 0 and G is not irregular
6:     ans ← no
7:     find an edge vu ∈ E such that d_G(v) = d_G(u)
8:     for all w ∈ N_G[{u, v}] do
9:         set G_w = G[V \ {w}]
10:        if IsIrregular(G_w, k − 1) returns yes then
11:            ans ← yes
12:        end if
13:    end for
14:    return ans
15: end if
```

$x \in S$, the set $S_x = S \setminus \{x\}$ is an $ir^*(G_x)$, where $G_x = G[V \setminus \{x\}]$. Furthermore, by Lemma 7.15, we have $\mathrm{I}(G_x) \leq k - 1 = k_0$ iff $\mathrm{I}(G) \leq k$. By the induction hypothesis, we know that the algorithm answers correctly for all the instances (G_x, k_0). Thus, if $\mathrm{I}(G) \leq k = k_0 + 1$, there must exist one instance (G_w, k_0), where $w \in S \cap N_G[\{v, u\}]$, for which the Algorithm 9 returns yes. Therefore the algorithm answers for $(G, k_0 + 1)$ correctly. Finally, this process request $n^{O(1)}$ steps in order to check if the graph is locally irregular and $2\Delta f(k-1, n-1)$ steps (by induction hypothesis) in order to check if for any graph G_x we have $\mathrm{I}(G_x) \leq k - 1 = k_0$ (where $x \in N[\{u, v\}]$). So, the algorithm decides in $n^{O(1)} + 2\Delta f(k-1, n-1) \leq n^{O(1)} + 2\Delta k(2\Delta)^{k-1}(n-1)^{O(1)} \leq n^{O(1)} + k(2\Delta)^k n^{O(1)} \leq (k+1)(2\Delta)^k n^{O(1)}$ steps. Finally, note that $k \leq n - 1$, and the result follows. □

7.5.2 FPT by Treewidth and Δ

Theorem 7.17. *For a given a graph $G = (V, E)$ and a nice tree decomposition of G, there exists an algorithm that returns $\mathrm{I}(G)$ in time $\Delta^{3tw} n^{O(1)}$, where tw is the treewidth of the given decomposition and Δ is the maximum degree of G.*

Proof. As the techniques we are going to use are standard, we are sketching some of the introductory details. We are going to perform dynamic programming on the nodes of the given nice tree decomposition. For a node t of the given tree decomposition of G, we denote by B_t the bag of this node and by $B_t^{\downarrow}$ the set of vertices of the graph that appears in the bags of the nodes of the subtree with t as a root. Observe that $B_t \subseteq B_t^{\downarrow}$.

The idea behind our algorithm, is that for each node t we store all the sets $S \subseteq B_t^{\downarrow}$ such that S is an $ir(G, B_t^{\downarrow} \setminus B_t)$. We also store the necessary "conditions" (explained

more in what follows) such that if there exists a set S', where $S' \setminus S \subseteq V \setminus B_t^{\downarrow}$, that meets these conditions, then S' is an $ir(G, B_t^{\downarrow})$. Observe that if we manage to do such a thing for every node of the tree decomposition, then we can find $\mathrm{I}(G)$. To do so, it suffices to check the size of all the irregulators we stored for the root r of the tree decomposition, which also meet the conditions we have set. In that way, we can find a set S that is an $ir(G, B_r^{\downarrow} \setminus B_r)$, satisfies our conditions and is of minimum order, and since $B_r^{\downarrow} = V$, this set S is a minimum irregulator of G and $\mathrm{I}(G) = |S|$.

Let us now present the actual information we are keeping for each node. Assume that t is a node of the tree decomposition and $S \subseteq B_t^{\downarrow}$ is an irregulator of $B_t^{\downarrow} \setminus B_t$ in G, *i.e.*, S is an $ir(G, B_t^{\downarrow} \setminus B_t)$. For this S we want to remember which vertices of B_t belong to S as well as the degrees of the vertices $v \in B_t \setminus S$ in $G[B_t^{\downarrow} \setminus B_t]$. This can be done by keeping a table D of size $tw+1$ where, if $v \in B_t \setminus S$ we set $D(v) = d_{G[B_t^{\downarrow} \setminus B_t]}(v)$ and if $v \in B_t \cap S$ we set $D(v) = \emptyset$ (slightly abusing the notation, by $D(v)$ we mean the position in the table D that corresponds to the vertex v). Like we have already said, we are going to keep some additional information about the conditions that could allow these sets to be extended to irregulators of $B_t^{\downarrow}$ in G if we add vertices of $V \setminus B_t^{\downarrow}$. For that reason, we are also going to keep a table with the "*target degree*" of each vertex; in this table we assign to each vertex $v \in B_t \setminus S$ a degree d_v such that, if there exists S' where $S' \setminus S \subseteq V \setminus B_t^{\downarrow}$ and for all $v \in B_t \setminus S$ we have $d_{G[V \setminus S']}(v) = d_v$, then S is an $ir(G, B_t^{\downarrow})$. This can be done by keeping a table T of size $tw+1$ where for each $v \in B_t \setminus S$ we set $T(v) = i$, where i is the target degree, and for each $v \in B_t \cap S$ we set $T(v) = \emptyset$. Such tables T will be called *valid* for S in B_t. Finally, we are going to keep the set $X = S \cap B_t$ and the value $min = |S|$. Note that the set X does not gives us any extra information, but we keep it as it will be useful to refer to it directly.

To sum up, for each node t of the tree decomposition of G, we keep a set of quadruples (X, D, T, min), each quadruple corresponding to a valid combination of a set S that is an $ir(G, B_t^{\downarrow} \setminus B_t)$ and the target degrees for the vertices of $B_t \setminus S$. Here it is important to say that when treating the node B_t, for every two quadruples (X_1, D_1, T_1, min_1) and (X_2, D_2, T_2, min_2) such that for all $v \in B_t$ we have that $D_1(v) = D_2(v)$ and $T_1(v) = T_2(v)$ (this indicates that $X_1 = X_2$ as well), then we are only going to keep the quadruple with the minimum value between min_1 and min_2 as we prove that this is enough in order to find $\mathrm{I}(G)$.

Claim 7.18. *Assume that for a node t, we have two sets S_1 and S_2 that are both $ir(G, B_t^{\downarrow} \setminus B_t)$, and that T is a target table that is common to both of them. Furthermore, assume that $(X_1, D_1, T, |S_1|)$ and $(X_2, D_2, T, |S_2|)$ are the quadruples we have to store for S_1 and S_2 respectively (both respecting T), with $D_1(v) = D_2(v)$ for every $v \in B_t$. Then for any set $S \subseteq V \setminus B_t^{\downarrow}$ such that $d_{G[V \setminus (S_1 \cup S)]}(v) = T(v)$ for all $v \in B_t$, we also have that $d_{G[V \setminus (S_2 \cup S)]}(v) = T(v)$ for all $v \in B_t$.*

Proof. Assume that we have such an S for S_1, let v be a vertex in B_t and $H = G[v \cup ((V \setminus B_t^{\downarrow}) \setminus S)]$ (observe that H does not depend on S_1 or S_2). Since $d_{G[V \setminus (S_1 \cup S)]}(v) = T(v)$, we know that in the graph H, v has exactly $T(v) - D_1(v)$ neighbours (as $D_1(v) =$

$d_{G[B_t^{\downarrow} \setminus S_1)]}(v)$). Now, since $D_1(v) = D_2(v) = d_{G[B_t^{\downarrow} \setminus S_2]}(v)$ we have that $d_{G[V \setminus S_2 \cup S]}(v) = T(v)$. Therefore, the claim holds. □

Simply put, Claim 7.18 states that for any two quadruples $Q_1 = (X, D, T, min_1)$ and $Q_2 = (X, D, T, min_2)$, any extension S of S_1 is also an extension of S_2 (where S_1 and S_2 are the two sets that correspond to Q_1 and Q_2 respectively). Therefore, in order to find the minimum solution, it is sufficient to keep the quadruple that has the minimum value between min_1 and min_2.

Now we are going to explain how we create all the quadruples (X, D, T, min) for each type of node in the tree decomposition. First we have to deal with the Leaf Nodes. For a Leaf node t we know that $B_t = B_t^{\downarrow} = \emptyset$. Therefore, we have only one quadruple (X, D, T, min), where the size of both D and T is zero (so we do not need to keep any information in them), $S = \emptyset$ and $min = |S| = 0$.

Now let t be an Introduce node; assume that we have all the quadruples (X, D, T, min) for its child c and let v be the introduced vertex. By construction, we know that v is introduced in B_t and thus it has no neighbours in $B_t^{\downarrow} \setminus B_t$. It follows that if $S \subseteq B_c^{\downarrow}$ is an irregulator for $B_c^{\downarrow} \setminus B_c$, then both S and $S \cup \{v\}$ are irregulators for $B_t^{\downarrow} \setminus B_t$ in G. Furthermore, there is no set $S \subseteq B_t^{\downarrow} \setminus \{v\}$ that is an irregulator of $B_t^{\downarrow} \setminus B_t$ and is not an irregulator of $B_c^{\downarrow} \setminus B_c$. So, we only need to consider two cases for the quadruples we have to store for c; if v belongs in the under-construction irregulator of $B_t^{\downarrow} \setminus B_t$ in G or not.

Case 1 *(v is in the irregulator)*. Observe that for any S that is an $ir(G, B_c^{\downarrow} \setminus B_c)$, which is stored in the quadruples of B_c, for every $u \in B_c \setminus S$, we have that $d_{G[B_c^{\downarrow} \setminus S]}(u) = d_{G[B_t^{\downarrow} \setminus (S \cup \{v\})]}(u)$. Moreover, for any target table T which is valid for S in c, the target table T' is valid for $S \cup \{v\}$ in t, where T' is almost the same as T, the only difference being that T' also contains the information about v, *i.e*, $T'(v) = \emptyset$. So, for each quadruple (X, D, T, min) in c, we need to create one quadruple $(X \cup \{v\}, D', T', min + 1)$ for t, where D' is the almost the same as D, except that it also contains the information about v, *i.e.*, $D'(v) = \emptyset$.

Case 2 (v *is not in the irregulator*). Let $q = (X, D, T, min)$ be a stored quadruple of c and S be the corresponding $ir(G, B_c^{\downarrow} \setminus B_c)$. We first explain how to construct D' of t, based on q. Observe that the only change between $G[B_c^{\downarrow} \setminus S]$ and $G[B_t^{\downarrow} \setminus S]$, is that in the latter there exist some new edges from v to some of the vertices of B_c. Therefore, for each vertex $u \in B_c \setminus X$ we set $D'(u) = D(u) + 1$ if $u \in N[v]$ and $D'(u) = D(u)$ otherwise. Finally, for the introduced vertex v, we set $D'(v) = |N(v) \cap (B_c \setminus X)|$. We now treat the target degrees for t. Observe that the target degrees for each vertex in $B_t \setminus \{v\}$ are the same as in T, since v only has edges incident to vertices in B_t. Now, we only need to decide which are the valid targets for v. Since $d_{G[B_t^{\downarrow} \setminus S]}(v) = D'(v)$, we know that for every target t', we have that $D'(v) \leq t' \leq \Delta$. Furthermore, we can not have the target degrees of v to be the same as the targets of one of its neighbours in B_c (these values are stored in T), as, otherwise, any valid target table T' of t would lead to adjacent vertices in B_t having the same degree. Let $\{t_1, \ldots, t_k\} \subset \{D(v), \ldots, \Delta\}$ be an enumeration of all the valid targets for v (i.e. $t_i \neq T(u)$ for all $u \in N[v] \cap B_c \setminus X$). Then,

for each quadruple (X, D, T, min) in c, and for each $i = 1, \ldots, k$, we need to create the quadruple (X, D', T_i, min), such that $T_i(u) = T(u)$ for all $u \in B_c$ and $T_i(v) = t_i$. In total, we have $k \leq \Delta$ such quadruples.

Now, let us explain how we deal with the Join nodes. Assume that t is a Join Node with c_1 and c_2 as its two children in the tree decomposition. Here, it is important to mention that $B_{c_1} = B_{c_2}$ and $(B_{c_1}^{\downarrow} \setminus B_{c_1}) \cap (B_{c_2}^{\downarrow} \setminus B_{c_2}) = \emptyset$. Assume that there exists an irregulator S of $B_t^{\downarrow} \setminus B_t$ in G, a valid target table T of S, and let (X, D, T, min) be the quadruple we need to store in t for this pair (S, T). Observe that this pair (S, T) is valid for both c_1 and c_2, so we must already have stored at least one quadruple in each node. Let $X \subseteq B_t$ and a target table T such that (X, D_1, T, min_1) and (X, D_2, T, min_2) are stored for c_1 and c_2 respectively. We create the quadruple (X, D, T, min) for t by setting $D(u) = D_1(u) + D_2(u) - d_{G[B_t \setminus X]}(u)$ for all $u \in B_t \setminus X$, $D(u) = \emptyset$ for all $u \in X$ and $min = min_1 + min_2 - |X|$. Observe that these are the correct values for the $D(u)$ and min, as otherwise we would count $d_{G[B_t \setminus X]}(u)$ and $|X|$ twice. Finally, we need to note that we do not store any quadruple (X, D, T, min) we create for the Join Note such that $D(u) > T(u)$ for a vertex $u \in B_t \setminus X$. This is because for such quadruples, the degree of vertex u will never be equal to any of the target degrees we have set, as it can only increase when we consider any of the ancestor (*i.e.* parent, grantparent etc.) nodes of t.

Finally, we need to treat the Forget nodes. Let t be a Forget node, c be the its child and v be the forgotten vertex. Assume that we have to store in t a quadruple (X, D, T, min). Then, since $X = B_t \cap S$ for an irregulator S of B_t in G, we know that in c we must have already stored a quadruple (X', D', T', min') such that, $X' = S \cap B_c$, $D'(u) = D(u)$ for all $u \in B_c$, $T'(u) = T(u)$ for all $u \in B_c$ and $min' = min$. Therefore, starting from the stored quadruples in c, we can create all the quadruples of t. For each quadruple (X', D', T', min') in c, we create at most one quadruple (X, D, T, min) for t by considering two cases; the forgotten vertex v_f belongs to X' or not.

Case 1 *(v belongs to X').* Then the quadruple (X, D, T, min) is almost the same as (X', D', T', min'), with the following differences: $X = X' \setminus \{v\}$, $min = min'$, $D(u) = D'(u)$ and $T(u) = T'(u)$ for all $u \in B_t$ and the tables D and T do not include any information for v as this vertex does not belong to B_t anymore.

Case 2 *(v does not belong to X').* First we check if $D'(v_f) = T'(v_f)$ or not. This is important because the degree of the v will never again be considered by our algorithm, and thus its degree will remain unchanged. So, if $D'(v_f) = T'(v_f)$, we create the quadruple (X, D, T, min) where $X = X'$, $min = min'$, $D(u) = D'(u)$ and $T(u) = T'(u)$ for all $u \in B_t$ and the tables D and T do not include any information for v.

For the running time, observe that the number of nodes of a nice tree decomposition is $O(tw \cdot n)$ and all the other calculations are polynomial in $n+m$. Thus we only need to count the different quadruples in each node. Now, for each vertex v, we either include it in X or we have $\Delta+1$ options for the value $D(u)$ and $\Delta+1-i$ for the value $T(u)$ if $D(u) = i$. Also, for sufficiently large Δ, we have that $1 + \sum_{i=0}^{\Delta}(\Delta + 1 - i) < \Delta^2$. Furthermore, the set X and the value min do not increase the number of quadruples because $X = \{u \mid D(u) = \emptyset\}$ and from all quadruples (X, D_1, T_1, min_1), (X, D_2, T_2, min_2) such that $D_1(u) = D_2(u)$ and $T_1(u) = T_2(u)$ for all $u \in B_t$, we only keep one of them (by

Claim 7.18).

Finally we need to consider the Join Nodes. Recall that for a Join Node t if we have the quadruples (X, D_1, T, min_1) and (X, D_2, T, min_2) in the first and second child respectively then we create the quadruple (X, D, T, min) (for t) by setting $D(u) = D_1(u) + D_2(u) - d_{G[B_t \setminus X]}(u)$ for all $u \in B_t \setminus X$, $D(u) = \emptyset$ for all $u \in X$ and $min = min_1 + min_2 - |X|$ (we discard this quadruple if $D(u) > T(u)$ for any $u \in B_t$). So, for a vertex for each vertex $u \in B_t$, we either include it in X or we have $\Delta + 1$ options for the value $T(u)$ and $T(u)$ options for the values $D_1(u)$ and $D_2(u)$. This give us in total $1 + \sum_{i=0}^{\Delta} i^2 < \Delta^3$ options, for sufficiently large Δ. Since each bag has at most $tw + 1$ such vertices this give us at most Δ^{3tw} combinations to check.

In total, the number of different quadruples in each node is Δ^{2tw}, and the algorithm decides in $\Delta^{3tw} n^{O(1)}$ time. □

It is worth noting that the algorithms of Theorem 7.16 and 7.17 can be used in order to also return an $ir^*(G)$.

7.5.3 W-Hardness

In this section we show a strong indication that there can be no *fpt*-algorithm that calculates an optimal irregulator of the input graph G, when parameterised by just the size of the solution or the treewidth of G. To do so, we present two linear-fpt reductions. The first is from the Dominating Set problem, when parameterised by the size of the solution, and the second is from the List Coloring problem, when parameterised by the treewidth of the graph.

Theorem 7.19. *Let G be a graph and $k \in \mathbb{N}$. Deciding if $\mathrm{I}(G) \leq k$ is W[2]-hard, when parameterised by k.*

Proof. The reduction is from the Dominating Set problem, which was shown to be W[2]-complete when parameterised by the size of the solution (*e.g.* in [65]). In that problem, a graph $H = (V, E)$ and an integer k are given as input. The question asked, is whether there exists a set $D \subseteq V$ of order at most k (called a *dominating set of H*), such that $V = N[D]$.

Let $H = (V, E)$ be a graph and $k \in \mathbb{N}$. We construct a graph $G = (V', E')$ such that H has a dominating set of order at most k if and only if G has an irregulator of order at most k. We begin by defining an arbitrary enumeration of the vertices of V. That is, $V = \{v_1, \ldots, v_n\}$. The graph G is built starting from a copy of the graph H. To avoid any confusion in what is to follow, we always use H to denote the original graph, and $G|_H = G[\{v'_1, \ldots, v'_n\}]$ to denote the copy of H that lies inside G (where the indices of the v'_is' are the same as the indices of the corresponding v_is'). Then, for each $1 \leq i \leq n$, we attach the necessary number of pending vertices (meaning vertices of degree 1) to the vertex v'_i, so that the degree of v'_i becomes equal to $i \cdot n$. Finally, for each v'_i, let u'_i be one of its newly attached pending vertices, and attach the necessary number of new pending vertices to u'_i, so that its degree becomes equal to that of v'_i. The resulting graph is G. To be clear, for every vertex v of G, we either have that $v = v'_i$ or $v = u'_i$,

or that v is a vertex pending from v'_i or u'_i (for some $1 \leq i \leq n$). Note also that for each $1 \leq i \leq n$, we have that $d_G(v'_i) = d_G(u'_i) = i \cdot n$.

Now let D be a dominating set of H, with $|D| = m \leq k$, and let D' be the subset of V' that corresponds to the vertices of D. That is, $D' = \{v'_i \in V' : v_i \in D\}$. We claim that the graph $G' = G[V' \setminus D']$ is locally irregular. Indeed, for every $1 \leq i \leq n$, let $\alpha(i)$ be the number of neighbours of v_i that belong to D. Observe that since D is a dominating set of H, we have that $1 \leq \alpha(i) \leq n-1$. Now, for every vertex v'_i in V', we have that either $v'_i \in D'$, in which case v'_i does not belong to G', or $d_{G'}(v'_i) = d_G(v'_i) - \alpha(i) < d_{G'}(u'_i)$. Moreover, for every $1 \leq i < j \leq n$, if $v'_i, v'_j \notin D'$, we have that $d_G(v'_j) - d_G(v'_i) \geq n$, and thus $d_{G'}(v'_j) - d_{G'}(v'_i) = d_G(v'_j) - \alpha(j) - d_G(v'_i) + \alpha(i) \geq n + \alpha(i) - \alpha(j) \geq 2$. Finally, every pending vertex l of G' is attached to either u'_i or v'_i, which have degree (in G') strictly larger than 1. It follows that D' is an irregulator of G with $|D'| = m \leq k$, and thus $\mathrm{I}(G) \leq k$.

For the other direction, assume that $\mathrm{I}(G) \leq k$ and let S be an $ir(G)$, with $|S| = k$, and $G' = G[V' \setminus S]$. For each $1 \leq i \leq n$, let $S_i = N[v'_i] \cup N(u'_i)$. We claim that for every i, we have $S \cap S_i \neq \emptyset$. Assume that this is not true, *i.e.*, that there exists an i_0 such that $S_{i_0} \cap S = \emptyset$. Then, by deleting the vertices of S from G, the degrees of v'_{i_0} and u'_{i_0} remain unchanged. Formally, we have that $d_{G'}(v'_{i_0}) = d_G(v'_{i_0}) = d_G(u'_{i_0}) = d_{G'}(u'_{i_0})$. This is a contradiction since S is an irregulator of G. Now, we consider the set S', defined as follows:

- Start with $S' = S$.
- For each i, while there exists a vertex $v \in S_i \cap S'$ such that $d_G(v) = 1$ or $v = u'_i$, remove v from S' and add v'_i to S'.

Clearly, we have that S' only contains vertices from $V(G|_H)$ and that $|S'| \leq |S| = k$. Also, from the construction of S', for every i, we have that $S_i \cap S' \neq \emptyset$. It follows that for every vertex v'_i, we either have $v'_i \in S'$ or there exists a vertex $v \in N(v'_i) \cap V(G|_H)$ such that $v \in S'$. Going back to H, let $D = \{v_i : v'_i \in S'\}$. It is clear that D is a dominating set of H of order at most k. This finishes our reduction.

Finally, note that throughout the above described reduction, the value of the parameter of the two problems is the same (in both of them, the parameter has the value k). Moreover, the construction of the graph G is achieved in polynomial time in regards to n. These observations conclude our proof. □

Theorem 7.20. *Let G be a graph with treewidth tw, and $k \in \mathbb{N}$. Deciding if $\mathrm{I}(G) = k$ is W[1]-hard when parameterised by tw.*

Proof. We present a reduction from the List Coloring problem: the input consists of a graph $H = (V, E)$ and a list function $L : V \to \mathcal{P}(\{1, \ldots, k\})$ that specifies the available colors for each vertex $u \in V$. The goal is to find a proper coloring $c : V \to \{1, \ldots, k\}$ such that $c(u) \in L(u)$ for all $u \in V$. When such a coloring exists, we say that (H, L) is a *yes-instance* of List Coloring. This problem is known to be W[1]-hard when parameterised by the treewidth of H [75].

Now, starting from an instance (H, L) of List Coloring, we construct a graph $G = (V', E')$ (see Figure 7.4 (a)) such that:

- $|V'| = O(|V|^6)$,
- $tw(G) = tw(H)$ and
- $\mathrm{I}(G) = nk$ if and only if (H, L) is a yes-instance of List Coloring.

Before we start with the construction of G, let us give the following observation.

Observation 2. *Let (H, L) be an instance of List Coloring where $H = (V, E)$ and there exists a vertex $u \in V$ such that $|L(u)| > d(u)$. Then the instance $(H[V \setminus \{u\}], L')$, where $L'(v) = L(v)$ for all $v \in V \setminus \{u\}$, is a yes-instance of List Coloring if and only if (H, L) is a yes-instance of List Coloring.*

Indeed, observe that for any vertex $u \in V$, by any proper coloring c of H, $c(u)$ only has to avoid $d(u)$ colors. Since $|L(u)| > d(u)$, we always have a spare color to use on u that belongs in $L(u)$. From the previous observation, we can assume that in our instance, for all $u \in V$, we have $|L(u)| \leq d(u)$. Furthermore, we can deduce that $k \leq n(n-1)$ as the degree of any vertex is at most $n-1$. Finally, let us denote by $\overline{L}(u)$ the set $\{0, 1, \ldots, k\} \setminus L(u)$. It is important to note here that for every $u \in V$, the list $L(u)$ contains at least one element belonging in $\{1, \ldots, k\}$. It follows that $\overline{L}(u)$ also contains at least one element, the color 0. To sum up, we have that $1 \leq |\overline{L}(u)| \leq k$.

Now, we present the three gadgets we are going to use in the construction of G. First, we have the "forbidden color gadget" H_i, which is a star with i leaves (see Figure 7.4(c)). When we say that we attach a copy of H_i on a vertex v of a graph G, we mean that we add H_i to G and we identify the vertices v and w_2 (where here and in what follows, we are using the naming illustrated in Figure 7.4 when talking about the vertices w_1, w_2, w_3, v_1 and v_2). The second, is the "degree gadget", which is presented in Figure 7.4(b). Finally, we have the "horn gadget", which is a path on three vertices (see Figure 7.4(d)). We define the operation of attaching these two gadgets on a vertex v of a graph G similarly to how we defined this operation for the forbidden color gadget (each time using the appropriate w_1 or w_3, according to if it is a degree or a horn gadget respectively).

In order to construct G, we start from a copy of H. Let us use $G|_H$ to denote the copy of H that lies inside of G and, for each vertex $u \in V$, let u' be its copy in V'. We will call the set of these vertices U. That is, $U = \{v \in V(G|_H)\}$. Then, we are going to attach several copies of each gadget to u', for each vertex $u' \in U$. We start by attaching k copies of the degree gadget to each vertex $u' \in U$. Then, for each $u \in V$ and each $i \in \overline{L}(u)$, we attach one copy of the forbidden color gadget H_{2n^3-i} to the vertex u'. Finally, for each $u' \in U$, we attach to u' as many copies of the horn gadget as are needed, in order to have $d_G(u') = 2n^3$.

Before we continue, observe that, for sufficiently large n, we have attached more than n^3 horn gadgets to each vertex of U. Indeed, before attaching the horn gadgets, each vertex $u' \in U$ has $d_G(u) \leq n-1$ neighbours in U, k neighbours from the degree gadgets and at most $k < n^2$ neighbours from the forbidden color gadgets (recall that

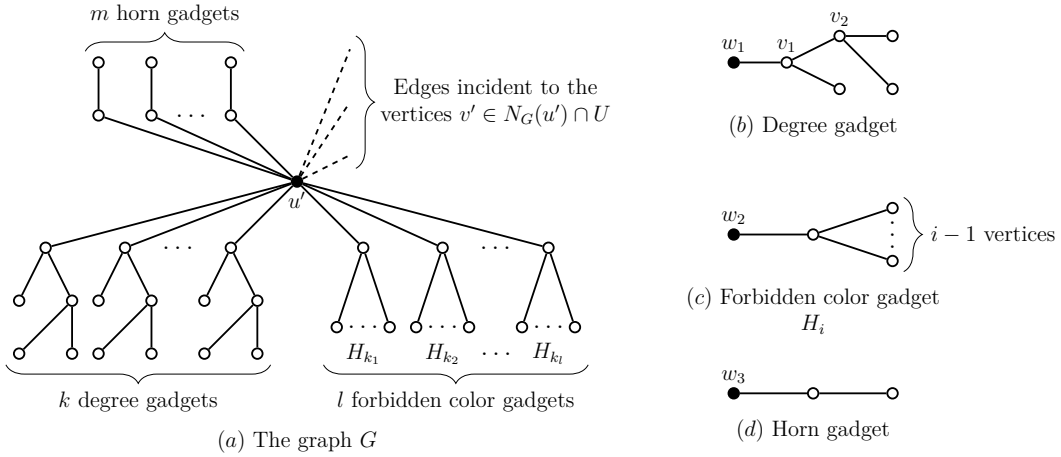


Figure 7.4: In (a) we illustrate the construction of G, as it is described in the proof of Theorem 7.20. The black vertex represents every vertex that belongs in U. For the specific vertex u' shown in the figure, we have that $\overline{L}(u) = \{c_1, \ldots, c_l\}$ and $k_i = n^3 - c_i$ for all $i = 1, \ldots, l$. We also have that $m = 2n^3 - d_G(u) - k - l$.

$|\overline{L}(u)| \leq k$). We now show that $|V'| = O(n^6)$. For that purpose, let us calculate the number of vertices in all the gadgets attached to a single vertex $u' \in U$. First, we have $5k < 5n^2$ vertices in the degree gadgets. Then, we have less than $4n^3$ vertices in the horn gadgets (as we have less that $2n^3$ such gadgets). Finally, we have at most $k < n^2$ forbidden color gadgets, each one of which containing at most $2n^3$ vertices. So, for each vertex $u' \in U$, we have at most $2n^5 + 4n^3 + 5n^2$ vertices in the gadgets attached to u'. Therefore, we have $|V'| = O(n^6)$.

Before we prove that $\mathrm{I}(G) \leq nk$ if and only if (H, L) is a yes-instance of List Coloring, we need to argue about two things. First, about the treewidth of the graph G and second, about the minimum value of $\mathrm{I}(G)$. Since our construction only attaches trees to each vertex of $G|_H$ (and recall that a tree has a treewidth of 1 by definition), we know that $tw(G) = tw(G|_H) = tw(H)$. As for $\mathrm{I}(G)$, we show that it has to be at least equal to nk. For that purpose we have the following two claims.

Claim 7.21. *Let S be an $ir(G)$ and $S \cap U \neq \emptyset$. Then $|S| > n^3$.*

Proof. Let $u' \in S \cap U$. By construction, G contains more than n^3 horn gadgets that are attached to u'. Therefore, by deleting u', we create more than n^3 copies of the P_2 graph, each one of which forces us to include at least one of its vertices in S. Hence, $|S| > n^3$. □

Claim 7.22. *Let S be an $ir(G)$ and $S \cap U = \emptyset$. Then $|S| \geq nk$. In particular, S includes at least one vertex from each copy of the degree gadget used in the construction of G.*

Proof. Let D be a copy of the degree gadget, attached to some vertex $u' \in U$. Observe that we have $d_G(v_1) = d_G(v_2)$. It follows by Observation 1, that S contains at least one

vertex v in $N[\{v_1, v_2\}]$, and since $u' \notin S$, this v is a vertex other than w_1. The result follows from the fact that the same arguments hold for any degree gadget attached to any vertex of U (recall that $|U| = n$ and we have attached k copies of the degree gadget to each one of the vertices of U). Hence, $|S| \geq nk$. □

By the previous two claims, we conclude that $\mathrm{I}(G) \geq nk$. We are ready to show that, if (H, L) is a yes-instance of List Coloring, then there exists a set $S \subseteq V'$ such that S is an $ir(G)$ and $|S| = nk$. Let c be a proper coloring of H such that $c(u) \in L(u)$ for all $u \in V$. We construct an $ir(G)$ as follows. For each $u \in V$, we partition (arbitrarily) the k degree gadgets attached to the vertex u' to $c(u)$ "good" and $(k - c(u))$ "bad" degree gadgets. For each good degree gadget, we add the copy of the vertex v_1 of that gadget to S and for each bad degree gadget we add the copy of the vertex v_2 of that gadget to S. This process creates a set S of size nk, as it includes k distinguished vertices for each vertex $u' \in U$.

Now we need to show that S is an $ir(G)$. Let $G' = G[V' \setminus S]$; observe that each vertex $u' \in U$ has degree $d_{G'}(u') = 2n^3 - c(u)$. Therefore, u' does not have the same degree as any of its neighbours that do not belong in U. Indeed, for every $v \in N_{G'}(u') \setminus U$, we have that $d_{G'}(v) \in \{1, 2\}$ (if v belongs to a bad degree or a horn gadget) or $d_{G'}(v) \in \{2n^3 - i : i \in \overline{L}(u)\}$ (if v belongs to a forbidden color gadget). Furthermore, since c is a proper coloring of H, for all $uv \in E$, we have that $c(u) \neq c(v)$. This gives us that for any edge $u'v' \in E'$ with $u', v' \in U$, we have that $d_{G'}(u') = 2n^3 - c(u) \neq 2n^3 - c(v) = d_{G'}(v')$.

So, we know that for every vertex $u' \in U$, there is no vertex $w \in N_{G'}(u')$ such that $d_{G'}(u') = d_{G'}(w)$. It remains to show that, in G', there exist no two vertices belonging to the same gadget, which have the same degrees. First of all, we have that S does not contain any vertex from any of the horn and forbidden color gadgets, nor from U. Thus any adjacent vertices belonging to these gadgets have different degrees. Last, it remains to check the vertices of the degree gadgets. Observe that for any copy of the degree gadget, S contains either v_1 or v_2. In both cases, after the deletion of the vertices of S, any adjacent vertices belonging to any degree gadget have different degrees. Therefore, S is an $ir(G)$ of order nk and since $\mathrm{I}(G) \geq nk$ we have that $\mathrm{I}(G) = nk$.

Now, for the opposite direction, assume that there exists a set $S \subseteq V'$ such that S is an $ir^*(G)$ and $|S| = nk$. Let $G' = (V'', E'')$ be the graph $G[V' \setminus S]$. It follows from Claim 7.21 and Claim 7.22, that $S \cap U = \emptyset$ and that S contains exactly one vertex from each copy of the degree gadget in G and no other vertices. Consider now the coloring c of H defined as $c(u) = 2n^3 - d_{G'}(u')$. We show that c is a proper coloring for H and that $c(u) \in L(u)$. First, we have that c is a proper coloring of H. Indeed, for any edge $uv \in E$, there exists an edge $u'v' \in E''$ (since $S \cap U = \emptyset$). Since G' is locally irregular we have that $d_{G'}(u') \neq d_{G'}(v')$, an thus $c(u) \neq c(v)$. It remains to show that $c(u) \in L(u)$ for all $u \in V$. First observe that, during the construction of G, we attached exactly k degree gadgets to each $u' \in U$. It follows that $d_{G'}(u') = 2n^3 - j$ and $c(u) = j$ for a $j \in \{0, 1, \ldots, k\}$. It is sufficient to show that $j \notin \overline{L}(u)$. Since S contains only vertices from the copies of the degree gadgets, we have that each $u' \in U$ has exactly one neighbour of degree $2n^3 - i$ for each $i \in \overline{L}(u)$ (this neighbour is a vertex of the H_i forbidden color gadget that was attached to u'). Furthermore, for all $u' \in U$, since

G' is locally irregular, we have that $d_{G'}(u') \neq 2n^3 - i$ for all $i \in \overline{L}(u)$. Equivalently, $d_{G'}(u') = 2n^3 - j$ for any $j \in L(u)$. Thus, $c(u) \in L(u)$ for all $u \in V$. □

Note that the reductions presented in the proofs of Theorem 7.19 and Theorem 7.20 are linear fpt-reductions. Additionally we know that

- there is no algorithm that answers if a graph G of order n has a Dominating Set of size at most k in time $f(k)n^{o(k)}$ unless the ETH fails [146] and
- there is no algorithm that answers if an instance (G, L) of the List Coloring is a yes-instance in time $O^*(f(tw)n^{o(tw)})$ unless the ETH fails [75].

So, the following corollary holds.

Corollary 7.23. *Let G be a graph of order n and assume the ETH. For $k \in \mathbb{N}$, there is no algorithm that decides if $\mathrm{I}(G) \leq k$ in time $f(k)n^{o(k)}$. Furthermore, assuming that G has treewidth tw, there is no algorithm that computes $\mathrm{I}(G)$ in time $O^*(f(tw)n^{o(tw)})$.*

7.6 Conclusion

In this chapter we introduced the problem of identifying the largest locally irregular induced subgraph of a given graph. There are many interesting directions that could be followed for further research. An obvious one is to investigate whether the problem of calculating $\mathrm{I}(G)$ remains NP-hard for other, restricted families of graphs. The first candidate for such a family would be the one of chordal graphs. On the other hand, there are some interesting families, for which the problem of computing an optimal irregulator could be decided in polynomial time, such as split graphs. Also, it could be feasible to conceive approximation algorithms for regular bipartite graphs, which have a better approximation ratio than the (simple) algorithm we present. The last aspect we find intriguing, is to study the parameterised complexity of calculating $\mathrm{I}(G)$ when considering other parameters, like the size of the minimum vertex cover of G, with the goal of identifying a parameter that suffices, by itself, in order to have an fpt-algorithm. Finally, it is worth investigating whether calculating $\mathrm{I}(G)$ could be done in FPT time (parameterised by the size of the solution) in the case where G is a planar graph.

Chapter 8

Crosswords Puzzle

8.1 Introduction

Crossword puzzles are one-player games where the goal is to fill a (traditionally two-dimensional) grid with words. Since their first appearance more than 100 years ago, crossword puzzles have rapidly become popular. Nowadays, they can be found in many newspapers and magazines around the world like the *New York Times* in the USA, or *Le Figaro* in France. Besides their obvious recreational interest, crossword puzzles are valued tools in education [64] and medicine. In particular, crossword puzzles participation seems to delay onset of accelerated memory decline [168]. They are also helpful for developing and testing computational techniques; see for example [173]. In fact, both the design and the completion of a puzzle are challenging. In this manuscript, we are interested in the task of solving a specific type of crossword puzzle.

There are different kinds of crossword puzzles. In the most famous ones, some clues are given together with the place where the answers should be located. A solution contains words that must be consistent with the given clues, and the intersecting pairs of words are constrained to agree on the letter they share. *Fill-in* crossword puzzles do not come with clues. Given a list of words and a grid in which some slots are identified, the objective is to fill all the slots with the given words. The list of words is typically succinct and provided explicitly.

In a variant of fill-in crossword puzzle currently proposed in a French TV magazine [148], one has to find up to 14 words and place them in a grid (the grid is the same for every instance, see Figure 8.1 for an illustration). The words are not explicitly listed but they must be *valid* (for instance, belong to the French language). In an instance of the game, some specified letters have a positive weight; the other letters have weight zero. The objective is to find a solution whose weight – defined as the total sum of the letters written in the grid – is at least a given threshold.

In this chapter we preset with a theoretical study of this fill-in crossword puzzle (the grid is not limited to the one of Figure 8.1). We are mainly interested in two problems: Can the grid be entirely completed? How can the weight of a solution be maximized? Hereafter, these problems are called Crossword Puzzle Decision and Crossword

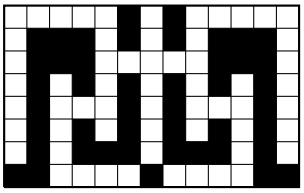


Figure 8.1: Place valid words in this grid. In a possible instance, letters S, U, I, V, R, E, and T have weight 7, 5, 4, 2, 6, 1, and 3, respectively. Any other letter has null weight. Try to obtain at least 330 points.

Puzzle Optimization (CP-Dec and CP-Opt in short), respectively.

CP-Dec is not new; see GP14 in [94]. The proof of NP-completeness is credited to a personal communication with Lewis and Papadimitriou. Thereafter, an alternative NP-completeness proof appeared in [77] (see also [139]). Other articles on crossword puzzles exist and they are mostly empirically validated techniques coming from Artificial Intelligence and Machine Learning; see for example [96, 150, 145, 9, 173, 172] and references therein.

Our Contribution Our goal in this chapter is to pinpoint the relevant structural parameters that make filling crossword puzzles intractable. We begin by examining the structure of the given grid. It is natural to think that, if the structure of the grid is tree-like, then the problem should become easier, as the vast majority of problems are tractable on graphs of small treewidth. We only partially confirm this intuition: by taking into account the structure of a graph that encodes the intersections between slots (the grid graph) we show in Section 8.3 that CP-Opt can be solved in polynomial time on instances of constant treewidth. However, our algorithm is not fixed-parameter tractable and, as we show, this cannot be avoided, even if one considers the much more restricted case where the problem is parameterized by the number of horizontal slots, which trivially bounds the grid graph's treewidth (Theorem 8.3). More devastatingly, we show that if we also impose the natural rule that words cannot be reused, the problem already becomes NP-hard when the grid graph is a matching for alphabets of size 3 (Theorem 8.5), or a union of stars for a binary alphabet (Theorem 8.4). Hence, a tree-like structure does not seem to be of much help in rendering crosswords tractable.

We then go on to consider CP-Opt parameterized by the total number of slots n. This is arguably a very natural parameterization of the problem, as in real-life crosswords, the size of the grid can be expected to be significantly smaller than the size of the dictionary. We show that in this case the problem does become fixed-parameter tractable (Corollary 8.7), but the running time of our algorithm is exponential in n^2. Our main result is to show that this disappointing dependence is likely to be best possible: even for a binary alphabet, an algorithm solving CP-Dec in time $2^{o(n^2)}$ would contradict the randomized ETH (Theorem 8.11). Note that all our positive results up to this point work for the more general CP-Opt, while our hardness results apply to CP-Dec.

Afterwards, in Section 8.5 we consider the approximability of CP-Opt. Here, it is

easy to obtain a $\frac{1}{2}$-approximation by only considering horizontal or vertical slots. We are only able to slightly improve upon this, giving a polynomial-time algorithm with ratio $\frac{1}{2} + O(\frac{1}{n})$. Our main result in this direction is to show that this is essentially best possible: obtaining an algorithm with ratio $\frac{1}{2} + \epsilon$ would falsify the Unique Games Conjecture (Theorem 8.13).

Before concluding, we explore in Section 8.6 the cases where CP-Dec can be resolved in polynomial time. We propose reductions from CP-Dec to some well-known problems that belong to P.

8.2 Preliminaries

We are given a dictionary $\mathcal{D} = \{d_1, \dots d_m\}$ whose words are constructed on an alphabet $\mathcal{L} = \{l_1, \dots l_\ell\}$, and a two-dimensional grid consisting of horizontal and vertical slots. A slot is composed of consecutive cells. Horizontal slots do not intersect each other; the same goes for vertical slots. However horizontal slots can intersect vertical slots. A cell is *shared* if it lies at the intersection of two slots. Unless specifically stated, n, m and ℓ denote the total number of slots, the size of $\mathcal{D}$, and the size of $\mathcal{L}$, respectively. Finally, let us mention that we consider only instances where the alphabet is of constant size, i.e., $\ell = O(1)$.

In a feasible solution, each slot S receives either a word of $\mathcal{D}$ of length $|S|$, or nothing (we sometimes say that a slot receiving nothing gets an *empty word*). Each cell gets at most one letter, and the words assigned to two intersecting slots must agree on the letter placed in the shared cell. All filled horizontal slots get words written from left to right (across) while all vertical slots get words written from top to bottom (down).

There is a weight function $w : \mathcal{L} \to \mathbb{N}$. The weight of a solution is the total sum of the weights of the letters placed in the grid. Observe that the weight of a solution is smaller than the total sum of the weights of its words because, in the former, the letters of the shared cells are counted only once.

The two main problems studied in this article are the following. Given a grid, a dictionary $\mathcal{D}$ on alphabet $\mathcal{L}$, and a weight function $w : \mathcal{L} \to \mathbb{N}$, the objective of Crossword Puzzle Optimization (CP-Opt in short) is to find a feasible solution of maximum weight. Given a grid and a dictionary $\mathcal{D}$ on alphabet $\mathcal{L}$, the question posed by Crossword Puzzle Decision (CP-Dec in short) is whether the grid can be completely filled or not?

Two cases will be considered: whether each word is used at most once, or if words can be assigned multiple times. In this article, we will sometimes suppose that some cells are pre-filled with some elements of $\mathcal{L}$. In this case, a solution is feasible if it is consistent with the pre-filled cells. Below we propose a first result when all the shared cells are pre-filled.

Proposition 8.1. *CP-Dec and CP-Opt can be solved in polynomial time if all the shared cells in the grid are pre-filled, whether word reuse is allowed or not.*

Proof. If word reuse is allowed, then for each combination of letters placed in these cells,

we greedily fill out the rest of each slot with the maximum value word that can still be placed there. This is guaranteed to produce the optimal solution. On the other hand, if word reuse is not allowed, we construct a bipartite graph, with elements of $\mathcal{D}$ on one side and the slots on the other, and place an edge between a word and a slot if the word can still be placed in the slot. If we give each edge weight equal to the value of its incident word reduced by the weight of the letters imposed by the shared cells of the slot, then an optimal solution corresponds to a maximum weight matching. □

One can associate a bipartite graph, hereafter called the *grid graph*, with each grid: each slot is a vertex and two vertices share an edge if the corresponding slots overlap. The grid (and then, the grid graph) is not necessarily connected.

Let us also note that so far we have been a bit vague about the encoding of the problem. Concretely, we could use a simple representation which lists for each slot the coordinates of its first cell, its size, and whether the slot is horizontal or vertical; and then supplies a list of all words in the dictionary and an encoding of the weight function. Such a representation would allow us to perform all the basic operations needed by our algorithms in polynomial time, such as deciding if it is possible to place a word d in a slot S, and which letter would then be placed in any particular cell of S. However, one drawback of this encoding is that its size may not be polynomially bounded in $n + m$, as some words may be exponentially long. We can work around this difficulty by using a more succinct representation: we are given the same information as above regarding the n slots; for each word we are given its total weight; and for each slot S and word d, we are told whether d fits exactly in S, and if yes, which letters are placed in the cells of S which are shared with other slots. Since the number of shared cells is $O(n^2)$ this representation is polynomial in $n + m$ and it is not hard to see that we are still able to perform any reasonable basic operation in polynomial time and that we can transform an instance given in the simple representation to this more succinct form. Hence, in the remainder, we will always assume that the size of the input is polynomially bounded in $n + m$.

8.3 When the Grid Graph is Tree-like

In this section we are considering instances of CP-Dec and CP-Opt where the grid graph is similar to a tree. First, we give an algorithm for both problems in cases where the grid graph has bounded treewidth and we are allowed to reuse words. We show that this algorithm is essentially optimal. Then, we show that CP-Dec and CP-Opt are much harder to deal with, in the case where we are not allowed to reuse words, by proving that the problems are NP-hard even for instances where the grid graph is just a matching. For the instances such that CP-Dec is NP-hard, we know that CP-Opt is NP-hard. That happens because we can assume that all the letters have weight equal to 1. Hence, a solution for CP-Dec is an optimal solution for CP-Opt.

8.3.1 Word Reuse

We propose a dynamic programming algorithm for CP-Opt and hence also for CP-Dec. Note that it can be extended to the case where some cells of the instance are pre-filled.

Theorem 8.2. *If we allow word reuse, then CP-Opt can be solved in time* $(m+1)^{tw}(n+m)^{O(1)}$ *on inputs where tw is the treewidth of the grid graph.*

Proof. As the techniques we are going to use are standard we are sketching some details. For more details on tree decomposition (definition and terminology) see [65, Chap. 7]. Assuming that we have a rooted nice tree decomposition of the grid graph, we are going to perform dynamic programming on the nodes of this tree decomposition. For a node B_t of the given tree decomposition of the grid graph we denote by $B_t^{\downarrow}$ the set of vertices of the grid graph that appears in the nodes of the subtree with B_t as a root. Since each vertex of the grid graph corresponds to a slot, we interchangeably mention a vertex of the grid graph and its corresponding slot. In particular, we say that a solution σ assigns words to the vertices of the grid graph, and $\sigma(v)$ denotes the word assigned to v.

For each node B_t of the tree decomposition we are going to keep all the triplets (σ, W, W_t) such that:

- σ is an assignment of words to the vertices of B_t;
- W is the weight of σ restricted to the vertices appearing in B_t;
- and W_m is the maximum weight, restricted to the vertices appearing in $B_t^{\downarrow}$, of an assignment consistent with σ.

In order to create all the possible triplets for all the nodes of the tree decomposition we are going to explore the nodes from leaves to the root. Therefore, each time we visit a node we assume that we have already created the triplets for all its children. Let us explain how we deal with the different types of nodes.

In the Leaf nodes we have no vertices so we keep an empty assignment (σ does not assign any word) and the weights W and W_m are equal to 0.

For an Introduce node B_t we need to take in consideration its child node. Assume that u is the introduced vertex; for each triplet (σ, W, W_m) of the child node we are going to create all the triplets (σ', W', W'_m) for the new node as follows. First we find all the words $d \in \mathcal{D}$ that fit in the corresponding slot of u and respect the assignment σ (i.e., if there are cells that are already filled under σ and d uses these cells then it must have the same letters). We create one triplet (σ', W', W'_m) for each such a d as follows:

- We set $\sigma'(u) := d$ and $\sigma'(v) := \sigma(v)$ for all $v \in B_t \setminus \{u\}$.
- We can easily calculate the total weight, W', of the words in B_t where the shared letters are counted only once under the assignment σ'.
- For the maximum weight W'_m we know that it is increased by the same amount as W; so we set $W'_m = W_m + W' - W$.

Observe that we do not need to consider the intersection with slots whose vertices appear in $B_t^{\downarrow} \setminus B_t$ as each node of a tree decomposition is a cut set.

Finally, we need to take in consideration that we can leave a slot empty. For this case we create a new word d_* which, we assume that, fits in all slots and d_* has weight 0. Because the empty word has weight 0, W' and W'_m are identical to W and W_m so for each triplet of the child node, we only need to extend σ by assigning d_* to u. In the case we assign the empty word somewhere we will consider that the cells of this slot are empty unless another word $d \neq d_*$ uses them.

For the Forget nodes we need to restrict the assignments of the child node to the vertex set of the Forget node, as it has been reduced by one vertex (the forgotten vertex), and reduce the weight W (which we can calculate easily). The maximum weight is not changed by the deletion.

However, if we restrict the assignments we may end up with several triplets (σ, W, W_m) with identical assignments σ. In that case we are keeping only the triplet with maximum W_m. Observe that we are allowed to keep only triplets with the maximum W_m because each node of a tree decomposition is a cut set so the same holds for the Forget nodes. Specifically, the vertices that appear in the nodes higher than a Forget node B_t of the tree decomposition do not have edges incident to vertices in $B_t^{\downarrow} \setminus B_t$ so we only care for the assignment in B_t.

Finally, we need to consider the Join nodes. Each Join node has exactly two children. For each possible assignment σ on the vertices of this Join node, we create a triplet iff this σ appears in a triplet of both children of the Join node.

Because W is related only to the assignment σ, it is easy to see that it will be the same as in the children of the Join node. So we need to find the maximum weight W_m. Observe that between the vertices that appear in the subtrees of two children of a Join node there are no edges except those incident to the vertices of the Join node. Therefore, we can calculate the maximum weight W_m as follows: first we consider the maximum weight of each child of the Join node reduced by W, we add all these weights and, in the end, we add again the W. It is easy to see that this way we consider the weight of the cells appearing in each subtree without those of the slots of the Join node and we add the weight of the words assigned to the vertices of the Join node in the end.

For the running time we need to observe that the number of nodes of a nice tree decomposition is $O(tw \cdot n)$ and all the other calculations are polynomial in $n + m$ so we only need to consider the different assignments for each node. Because for each vertex we have $|\mathcal{D}| + 1$ choices, the number of different assignments for a node is at most $(|\mathcal{D}| + 1)^{tw+1}$. □

It seems that the algorithm we propose for CP-Dec is essentially optimal, even if we consider a much more restricted case.

Theorem 8.3. *CP-Dec with word reuse is W[1]-hard parameterized by the number of horizontal slots of the grid, even for alphabets with two letters. Furthermore, under the ETH, no algorithm can solve this problem in time $m^{o(k)}$, where k is the number of horizontal slots.*

Proof. We perform a reduction from k-Independent Set, where we are given a graph $G = (V, E)$ with $|V|$ vertices and $|E|$ edges and are looking for an independent set of size k. This problem is well-known to be W[1]-hard and not solvable in $|V|^{o(k)}$ time under the ETH [65]. We assume without loss of generality that $|E| \neq k$. Furthermore, we can safely assume that G has no isolated vertices.

We first describe the grid of our construction which fits within an area of $2k-1$ lines and $2|E|-1$ columns. We construct:

1. k horizontal slots, each of length $2|E|-1$ (so each of these slots is as long horizontally as the whole grid). We place these slots in the unique way so that no two of these slots are in consecutive lines. We number these horizontal slots $1, \ldots, k$ from top to bottom.

2. $|E|$ vertical slots, each of length $2k-1$ (so each of these slots is long enough to cover the grid top to bottom). We place these slots in the unique way so that no two of them are in consecutive columns. We number them $1, \ldots, |E|$ from left to right.

Before we describe the dictionary, let us give some intuition about the grid. The main idea is that in the k horizontal slots we will place k words that signify which vertices we selected from the original graph. Each vertical slot represents an edge of E, and we will be able to place a word in it if and only if we have not placed words representing two of its endpoints in the horizontal slots.

Our alphabet has two letters, say $0, 1$. In the remainder, we assume that the edges of the original graph are numbered, that is, $E = \{e_1, \ldots, e_{|E|}\}$. The dictionary is as follows:

1. For each vertex v we construct a word of length $2|E|-1$. For each $i \in \{1, \ldots, |E|\}$, if the edge e_i is incident on v, then the letter at position $2i-1$ of the word representing v is 1. All other letters of the word representing v are 0. Observe that this means that if e_i is incident on v and we place the word representing v on a horizontal slot, the letter i will appear on the i-th vertical slot. Furthermore, the word representing v has a number of 1s equal to the degree of v.

2. We construct $k+1$ words of length $2k-1$. One of them is simply 0^{2k-1}. The remaining are $0^{2j-2}10^{2k-2j}$, for $j \in \{1, \ldots, k\}$, that is, the words formed by placing a 1 in an odd-numbered position and 0s everywhere else. Observe that if we place one of these k words on a vertical slot, a 1 will be placed on exactly one horizontal slot.

This completes the construction. We now observe that the k horizontal slots correspond to a vertex cover of the grid graph. Therefore, if the reduction preserves the answer, the hardness results for k-Independent Set transfer to our problem, since we preserve the value of the parameter.

We claim that if there exists an independent set of size k in G, then it is possible to fill the grid. Indeed, take such a set S and for each $v \in S$ we place the word representing v in a horizontal slot. Consider the i-th vertical slot. We will place in this slot one of the $k+1$ words of length $2k-1$. We claim that the vertical slot at this moment contains the letter 1 at most once, and if 1 appears it must be at an odd position (since these are the positions shared with the horizontal slots). If this is true, clearly there is a word we can place. To see that the claim is true, recall that since S is an independent set of k distinct vertices, there exists at most one vertex in S incident on e_i.

For the converse direction, recall that $|E| \neq k$. This implies that if there is a way to fill out the whole grid, then words representing vertices must go into horizontal slots and words of length $2k-1$ must go into vertical slots. By looking at the words that have been placed in the horizontal slots we obtain a collection of k (not necessarily distinct) vertices of G. We will prove that these vertices must actually be an independent set of size exactly k. To see this, consider the i-th vertical slot. If our collection of vertices contained two vertices incident on e_i, it would have been impossible to fill out the i-th vertical slot, since we would need a word with two 1s. Observe that the same argument rules out the possibility that our collection contains the same vertex v twice, as the column corresponding to any edge e_i incident on v would have been impossible to fill. □

8.3.2 No Word Reuse

If a word cannot be reused, then CP-Dec looks more challenging. Indeed, in the following theorem we prove that if reusing words is not allowed, then the problem becomes NP-hard even if the grid graph is acyclic and the alphabet size is 2. (Note that if the alphabet size is 1, the problem is trivial, independent of the structure of the graph.)

Theorem 8.4. *CP-Dec is NP-hard, even for instances where all of the following restrictions apply: (i) the grid graph is a union of stars (ii) the alphabet contains only two letters (iii) words cannot be reused.*

Proof. We show a reduction from 3-Partition. Recall that in 3-Partition we are given a collection of $3n$ distinct positive integers $x_1, \ldots, x_{3n}$ and are asked if it is possible to partition these integers into n sets of three integers (triples), such that all triples have the same sum. This problem has long been known to be strongly NP-hard [94] and NP-hardness when the integers are distinct was shown by Hulett et al. [117]. We can assume that $\sum_{i=1}^{3n} x_i = nB$ and that if a partition exists each triple has sum B. Furthermore, we can assume without loss of generality that $x_i > 6n$ for all $i \in \{1, \ldots, 3n\}$ (otherwise, we can simply add $6n$ to all numbers and adjust B accordingly without changing the answer).

Given an instance of 3-Partition as above, we construct a crossword instance as follows. First, the alphabet only contains two letters, say the letters $*$ and !. To construct our dictionary we do the following:

1. For each $i \in \{1, \ldots, 3n\}$, we add to the dictionary one word of length x_i that begins with ! and $n-1$ words of length x_i that begin with $*$. The remaining letters of these words are chosen in an arbitrary way so that all words remain distinct.

2. For each $i, j, k \in \{1, \ldots, 3n\}$ with $i < j < k$ we check if $x_i + x_j + x_k = B$. If this is the case, we add to the dictionary the word $*^{2i-2}!*^{2j-2i-1}!*^{2k-2j-1}!*^{6n-2k}$. In other words, we constructed a word that has $*$ everywhere except in positions $2i-1, 2j-1$, and $2k-1$. The length of this word is $6n-1$. Let f be the number of words added to the dictionary in this step. We have $f \leq \binom{3n}{3} = O(n^3)$.

We now also need to specify our grid. We first construct f horizontal slots, each of length $6n-1$. Among these f slots, we select n, which we call the "interesting" horizontal slots. For each interesting horizontal slot, we construct $3n$ vertical slots, such that the i-th of these slots has length x_i and its first cell is the cell in position $2i-1$ of the interesting horizontal slot. This completes the construction, which can clearly be carried out in polynomial time. Observe that the first two promised restrictions are satisfied as we have an alphabet with two letters and each vertical slot intersects at most one horizontal slot (so the grid graph is a union of stars).

We claim that if there exists a partition of the original instance, then we can place all the words of the dictionary on the grid. Indeed, for each $i, j, k \in \{1, \ldots, 3n\}$ such that $\{x_i, x_j, x_k\}$ is one of the triples of the partition, we have constructed a word of length $6n-1$ corresponding to the triple (i, j, k), because $x_i + x_j + x_k = B$. We place each of these n words on an interesting horizontal slot and we place the remaining words of length $6n-1$ on the non-interesting horizontal slots. Now, for every $i \in \{1, \ldots, 3n\}$ we have constructed n words, one starting with ! and $n-1$ starting with $*$. We observe that among the interesting horizontal slots, there is one that contains the letter ! at position $2i-1$ (the one corresponding to the triple containing x_i in the partition) and $n-1$ containing the letter $*$ at position $2i-1$. By construction, the vertical slots that begin in these positions have length x_i. Therefore, we can place all n words corresponding to x_i on these vertical slots. Proceeding in this way we fill the whole grid, fulfilling the third condition.

For the converse direction, suppose that there is a way to fill the whole grid. Then, vertical slots must contain words that were constructed in the second step and represent integers x_i, while horizontal slots must contain words constructed in the first step (this is a consequence of the fact that $x_i > 6n$ for all $i \in \{1, \ldots, 3n\}$). We consider the n interesting horizontal slots. Each such slot contains a word that represents a triple (i, j, k) with $x_i + x_j + x_k = B$. We therefore collect these n triples and attempt to construct a partition from them. To do this, we must prove that each x_i must belong to exactly one of these triples. However, recall that we have exactly n words of length x_i (since all integers of our instance are distinct) and exactly n vertical slots of this length. We conclude that exactly one vertical slot must have ! as its first letter, therefore x_i appears in exactly one triple and we have a proper partition. □

Actually, the problem remains NP-hard even in the case where the grid graph is a matching and the alphabet contains three letters. This is proved for grid graphs composed of $\mathcal{T}$s, where a $\mathcal{T}$ is a horizontal slot solely intersected by the first cell of a vertical slot.

Theorem 8.5. *CP-Dec is NP-hard, even for instances where all of the following restrictions apply: (i) each word can be used only once (ii) the grid is consisted only by $\mathcal{T}$s and (iii) the alphabet contains only three letters.*

The proof of this theorem is quite technical and is included in the appendix.

Remark. *In our construction each $\mathcal{T}$ has unique shape*[1] *so the problem remains NP-hard even in this case.*

Remark. *Theorem 8.3 can be adjusted to work also for the case where word reuse is not allowed. We simply need to add a suffix of length* $\log m$ *to all words of length* $2k-1$ *and add rows to the grid accordingly. Hence, under the ETH, no algorithm can solve this problem in time $m^{o(k)}$, where k is the number of horizontal slots.*

Finally, observe that by filling the slots of a vertex cover of the grid graph, all the shared cells are pre-filled. Since there are at most $(m+1)^k$ (where k is the size of the vertex cover) ways to assign words to these slots, by Proposition 8.1, we get the following corollary.

Corollary 8.6. *Given a vertex cover of size k of the grid graph we can solve CP-Dec and CP-Opt in time $(m+1)^k(n+m)^{O(1)}$. Furthermore, as vertex cover we can take the set of horizontal slots.*

Therefore, the bound given in Remark 8.3.2 for the parameter vertex cover is tight.

8.4 Parameterized by Total Number of Slots

In this section we consider a much more restrictive parameterization of the problem: we consider instances where the parameter is n, the total number of slots. Recall that in Theorem 8.3 (and Remark 8.3.2) we already considered the complexity of the problem parameterized by the number of *horizontal* slots of the instance. We showed that this case of the problem cannot be solved in $m^{o(k)}$ and that an algorithm with running time roughly $(m+1)^k$ is possible whether word reuse is allowed or not.

Since parameterizing by the number of horizontal slots is not sufficient to render the problem FPT, we therefore consider our parameter to be the total number of slots. This is, finally, sufficient to obtain a simple fpt-algorithm.

Corollary 8.7. *There is an algorithm that solves CP-Dec and CP-Opt in time $O^*((\ell+1)^{n^2/4})$, where n is the total number of slots and ℓ the size of the alphabet, whether word reuse is allowed or not.*

Proof. Since there are n slots in the instance, even if the grid is a complete bipartite graph, the instance contains at most $n^2/4$ cells which are shared between two slots. In time $(\ell+1)^{n^2/4}$ we consider all possible letters that could be placed in these cells. Finally, as we have shown in Proposition 8.1, each of these instances can be solved in polynomial time. □

[1] Two crosses are of the same shape if they are identical: same number of horizontal cells, same number of vertical cells, and same shared cell.

Even though the running time guaranteed by Corollary 8.7 is FPT for parameter n, we cannot help but observe that the dependence on n is rather disappointing, as our algorithm is exponential *in the square* of n. It is therefore a natural question whether an fpt-algorithm for this problem can achieve complexity $2^{o(n^2)}$, assuming the alphabet size is bounded. The main result of this section is to establish that this is likely to be impossible.

Overview Our hardness proof consists of two steps. In the first step we reduce 3-SAT to a version of the same problem where variables and clauses are partitioned into $O(\sqrt{n+m})$ groups, which we call Sparse 3-SAT. The key property of this intermediate problem is that interactions between groups of variables and groups of clauses are extremely limited. In particular, for each group of variables V_i and each group of clauses C_j, at most one variable of V_i appears in a clause of C_j. We obtain this rather severe restriction via a randomized reduction that runs in expected polynomial time. The second step is to reduce Sparse 3-SAT to CP-Dec. Here, every horizontal slot will represent a group of variables and every vertical slot a group of clauses, giving $O(\sqrt{n+m})$ slots in total. Hence, an algorithm for CP-Dec whose dependence on the total number of slots is subquadratic in the exponent will imply a sub-exponential time (randomized) algorithm for 3-SAT. The limited interactions between groups of clauses and variables will be key in allowing us to execute this reduction using a *binary* alphabet.

Let us now define our intermediate problem.

Definition 8.1. *In Sparse 3-SAT we are given an integer n which is a perfect square and a 3-SAT formula ϕ with at most n variables and at most n clauses, such that each variable appears in at most 3 clauses. Furthermore, we are given a partition of the set of variables V and the set of clauses C into $\sqrt{n}$ sets $V_1, \ldots, V_{\sqrt{n}}$ and $C_1, \ldots, C_{\sqrt{n}}$ of size at most $\sqrt{n}$ each, such that for all $i, j \in [\sqrt{n}]$ the number of variables of V_i which appear in at least one clause of C_j is at most one.*

Now, we are going to prove the hardness of Sparse 3-SAT, which is the first step of our reduction.

Lemma 8.8. *Suppose the randomized ETH is true. Then, there exists an $\epsilon > 0$ such that Sparse 3-SAT cannot be solved in time $2^{\epsilon n}$.*

The first step of our reduction will be to prove that Sparse 3-SAT cannot be solved in sub-exponential time (in n) under the randomized ETH, via a reduction from 3-SAT. To do this, we will need the following combinatorial lemma.

Lemma 8.9. *For each $\epsilon > 0$ there exists $C > 0$ such that for sufficiently large n we have the following. There exists a randomized algorithm running in expected polynomial time which, given a bipartite graph $G = (A, B, E)$ such that $|A| = |B| = n$ and the maximum degree of G is 3, produces a set $V' \subseteq A \cup B$ with $|V'| \geq 2(1-\epsilon)n$ and a coloring $c : V' \to [k]$ of the vertices of V' with k colors, where $k \leq C\sqrt{n}$, such that for all $i \in [k]$ we have $|c^{-1}(i)| \leq \sqrt{n}$ and for all $i, j \in [k]$ the graph induced by $c^{-1}(i) \cup c^{-1}(j)$ contains at most one edge.*

Proof. Let $k = C\lceil\sqrt{n}\rceil$, where C is a sufficiently large constant (depending only on ϵ) to be specified later. We color each vertex of the graph uniformly at random from a color in $[k]$, call this coloring c. Let $X_{i,j}$ be the set of edges which have as endpoints a vertex of color i and a vertex of color j.

Our algorithm is rather simple: initially, we set $V' = V$. Then, for each $i, j \in [k]$ we check whether $X_{i,j}$ contains at most one edge. If yes, we do nothing; if not, we select for each edge $e \in X_{i,j}$ an arbitrary endpoint and remove that vertex from V'. In the end we return the set V' that remains and its coloring. It is clear that this satisfies the property that $c^{-1}(i) \cup c^{-1}(j)$ contains at most one edge for the graph induced by V' for all $i, j \in [k]$, so what we need to argue is that (i) $|c^{-1}(i)| \leq \sqrt{n}$ for all i with high probability and (ii) that V' has the promised size with at least constant probability. If we achieve this it will be sufficient to repeat the algorithm a polynomial number of times to obtain the claimed properties with high probability, hence we will have an expected running time polynomial in n.

For the first part, fix an $i \in [k]$ and observe that $E[|c^{-1}(i)|] \leq \frac{2\sqrt{n}}{C}$. To prove that all $|c^{-1}(i)|$ are of size at most $4\sqrt{n}/C$ with high probability (and hence also at most $\sqrt{n}$ for C sufficiently large), we will use Chernoff's Inequality.

Proposition 8.10 (Chernoff's Inequality)**.** *Let X be a binomial random variable and $\epsilon > 0$. Then $P[|X - E[X]| > \epsilon E[X]] < 2e^{-\epsilon^2 E[X]/3}$*

We take $\epsilon = 1$. It follows that $P[|c^{-1}(i)| > 4\sqrt{n}/C] \leq 2e^{-2\sqrt{n}/3C}$. Now, taking the union bound, we obtain that almost surely for all color i, $|c^{-1}(i)| < 4\sqrt{n}/C$

The more interesting part of this proof is to bound the expected size of V'. Let e be an edge whose endpoints are colored with colors i and j. We say that e is *good* if no other edge in G has one endpoint colored i and the other colored j by the coloring c. Let u and v be the endpoints of e. The probability of another edge having endpoints of colors i and j in the graph $G - \{u, v\}$ is at most $\frac{2|E|}{C^2 n} \leq \frac{6}{C^2}$. The probability that at least one of the at most four edges incident to e has endpoints colored i and j is at most $\frac{4}{C\sqrt{n}}$. Thus, the probability that e is good is at least $1 - \frac{6}{C^2} - \frac{4}{C\sqrt{n}} > 1 - \frac{7}{C^2}$, if n is sufficiently large. Let X be the number of edges which are not good. Then, $E[X] \leq 7C^{-2}|E|$. By Markov's Inequality $P[X > 21C^{-2}|E|] < 1/3$. Thus, with probability at least $2/3$, our algorithm will remove at most $21C^{-2}|E| \leq 63C^{-2}n$ vertices. Since we have promised to remove at most $2\epsilon n$ vertices, it suffices to select any value $C \geq \frac{8}{\sqrt{\epsilon}}$. □

Now, we present the proof of Lemma 8.8

Proof. Suppose that the statement is false, therefore for any $\epsilon > 0$ we can solve Sparse 3-SAT in which the number of variables and clauses can be upper-bounded by N in expected time $2^{\epsilon N}$ using some supposed algorithm. Fix an arbitrary $\epsilon' > 0$. We will show how to solve an arbitrary instance of 3-SAT with n variables and m clauses in expected time $2^{\epsilon'(n+m)}$ using this supposed algorithm for Sparse 3-SAT. If we can do this for any arbitrary ϵ', this will contradict the randomized ETH.

Start with an arbitrary 3-SAT instance ϕ with n variables and m clauses. We first edit ϕ to ensure that each variable appears at most three times. In particular, if x appears $k > 3$ times, we replace each appearance of x with a fresh variable x_i, $i \in [k]$, and add the clauses $(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge \ldots \wedge (\neg x_k \vee x_1)$.

The number of variables in the new instance is at most $n+3m$. The number of clauses is at most $4m$. This is because every new clause and every new variable corresponds to an occurrence of an original variable in an original clause and there are at most $3m$ such occurrences.

We now have an instance ϕ' equivalent to ϕ with at most $n + 3m$ variables and at most $4m$ clauses, such that each variable appears at most 3 times. Let N be the smallest perfect square such that $N \geq n + 4m$. We have $N < 10(n + m)$. What we need now is to produce a partition of the vertices and clauses of ϕ'.

In order to produce this partition we invoke Lemma 8.9 on the incidence graph of ϕ', that is, the bipartite graph where we have variables on one side and clauses on the other, and edges signify that a variable appears in a clause. Add some dummy isolated vertices on each side so that both sides of the incidence graph contain N vertices. We invoke Lemma 8.9 by setting ϵ to be $\epsilon'/80$. We obtain a coloring of all but at most $\frac{\epsilon' N}{40} \leq \frac{\epsilon'(n+m)}{4}$ of the vertices of the incidence graph.

Let U be the set of variables and clauses that correspond to uncolored vertices of the incidence graph. Then, for each such variable we produce two formulas (one by setting it to True and one by setting it to False), and for each such clause, at most 3 formulas (one by setting each of the literals of the clause to True). We thus construct at most $3^{\epsilon'(n+m)/4} \leq 2^{\epsilon'(n+m)/2}$ new formulas, such that one of them is satisfiable if and only if ϕ was satisfiable. We will then use the supposed algorithm for Sparse 3-SAT to decide each of these formulas one by one.

Each new formula we have contains at most N variables and at most N clauses, and by Lemma 8.9 we have partitions of the variables and clauses into $C\sqrt{N}$ groups, where C is a constant (that depends on ϵ'). By setting $N' = \lceil C \rceil^2 N$ we can view these instances as instances of Sparse 3-SAT, because then the number of groups becomes equal to the square root of the upper bound on the number of variables and clauses, and by the properties of Lemma 8.9 there is at most one edge between each group of variables and each group of clauses. Since we suppose that for all $\epsilon > 0$ such instances can be solved in time $2^{\epsilon N'}$, by setting $\epsilon = \epsilon'/50\lceil C \rceil^2$ we can solve each formula in $2^{\epsilon'(n+m)/5}$. The total expected running time of our algorithm is at most $2^{\epsilon'(n+m)/2} \cdot 2^{\epsilon'(n+m)/5} \cdot (n+m)^{O(1)} \leq 2^{\epsilon'(n+m)}$, so we contradict the ETH. □

We are now ready to prove the main theorem of this section.

Theorem 8.11. *Suppose the randomized ETH is true. Then, there exists an $\epsilon > 0$ such that CP-Dec on instances with a binary alphabet cannot be solved in time $2^{\epsilon n^2} \cdot m^{O(1)}$. This holds also for instances where all slots have distinct sizes (so words cannot be reused).*

Proof. Suppose for the sake of contradiction that for any fixed $\epsilon > 0$, CP-Dec on instances with a binary alphabet can be solved in time $2^{\epsilon n^2} \cdot m^{O(1)}$. We will then

contradict Lemma 8.8. In particular, we will show that for any ϵ' we can solve SPARSE 3-SAT in time $2^{\epsilon' N}$, where N is the upper bound on the number of variables and clauses. Fix some $\epsilon' > 0$ and suppose that ϕ is an instance of SPARSE 3-SAT with at most N variables and at most N clauses, where N is a perfect square. Recall that the variables are given partitioned into $\sqrt{N}$ sets, $V_1, \ldots, V_{\sqrt{N}}$ and the clauses partitioned into $\sqrt{N}$ sets $C_1, \ldots, C_{\sqrt{N}}$. In the remainder, when we write $V(C_j)$ we will denote the set of variables that appear in a clause of C_j. Recall that the partition satisfies the property that for all $i, j \in [\sqrt{N}\ ]$ we have $|V_i \cap V(C_j)| \leq 1$. Suppose that the variables of ϕ are ordered $x_1, x_2, \ldots, x_N$.

We construct a grid as follows: for each group V_i we construct a horizontal slot and for each group C_j we construct a vertical slot, in a way that all slots have distinct lengths. More precisely, the i-th horizontal slot, for $i \in [\sqrt{N}]$ is placed on row $2i-1$, starts in the first column and has length $2\sqrt{N} + 2i$. The j-th vertical slot is placed in column $2j-1$, starts in the first row and has length $5\sqrt{N} + 2j$. (As usual, we number the rows and columns top-to-bottom and left-to-right). Observe that all horizontal slots intersect all vertical slots; in particular, the cell in row $2i-1$ and column $2j-1$ is shared between the i-th horizontal and j-th vertical slot, for $i, j \in [\sqrt{N}]$. We define $\mathcal{L}$ to contain two letters $\{0, 1\}$.

What remains is to describe the dictionary.

- For each $i \in [\sqrt{N}]$ and for each assignment function $\sigma : V_i \to \{0, 1\}$ we construct a word w_σ of length $2\sqrt{N} + 2i$. The word w_σ has the letter 0 in all positions, except positions $2j-1$, for $j \in [\sqrt{N}]$. For each such j, we consider σ restricted to $V_i \cap V(C_j)$. By the properties of SPARSE 3-SAT, we have $|V_i \cap V(C_j)| \leq 1$. If $V_i \cap V(C_j) = \emptyset$ then we place letter 0 in position $2j-1$; otherwise we set in position $2j-1$ the letter that corresponds to the value assigned by σ to the unique variable of $V_i \cap V(C_j)$.

- For each $j \in [\sqrt{N}]$ and for each *satisfying* assignment function $\sigma : V(C_j) \to \{0, 1\}$, that is, every assignment function that satisfies all clauses of C_j, we construct a word w'_σ of length $5\sqrt{N} + 2j$. The word w'_σ has the letter 0 in all positions, except positions $2i-1$, for $i \in [\sqrt{N}]$. For each such i, we consider σ restricted to $V_i \cap V(C_j)$. If $V_i \cap V(C_j) = \emptyset$ then we place letter 0 in position $2i-1$; otherwise we set in position $2i-1$ the letter that corresponds to the value assigned by σ to the unique variable of $V_i \cap V(C_j)$.

The construction is now complete. We claim that if ϕ is satisfiable, then it is possible to fill out the grid we have constructed. Indeed, fix a satisfying assignment σ to the variables of ϕ. For each $i \in [\sqrt{N}]$ let σ_i be the restriction of σ to V_i. We place in the i-th horizontal slot the word w_{σ_i}. Similarly, for each $j \in [\sqrt{N}]$ we let σ'_j be the restriction of σ to $V(C_j)$ and place $w'_{\sigma'_j}$ in the j-th vertical slot. Now if we examine the cell shared by the i-th horizontal and j-th vertical slot, we can see that it contains a letter that represents σ restricted to (the unique variable of) $V_i \cap V(C_j)$ or 0 if $V_i \cap V(C_j) = \emptyset$, and both the horizontal and vertical word place the same letter in that cell.

For the converse direction, if the grid is filled, we can extract an assignment σ for the variables of ϕ as follows: for each $x \in V_i$ we find a C_j such that x appears in some clause of C_j (we can assume that every variable appears in some clause). We then look at the cell shared between the i-th horizontal and the j-th vertical slot. The letter we have placed in that cell gives an assignment for the variable contained $V_i \cap V(C_j)$, that is x. Having extracted an assignment to all the variables, we claim it must satisfy ϕ. If not, there is a group C_j that contains an unsatisfied clause. Nevertheless, in the j-th vertical slot we have placed a word that corresponds to a *satisfying* assignment for the clauses of C_j, call it σ_j. Then σ_j must disagree with σ in a variable x that appears in C_j. Suppose this variable is part of V_i. Then, this would contradict the fact that we extracted an assignment for x from the word placed in the i-th horizontal slot.

Observe that the new instance has $n = 2\sqrt{N}$ slots. If there exists an algorithm that solves CP-Dec in time $2^{\epsilon n^2} m^{O(1)}$ for any $\epsilon > 0$, we set $\epsilon = \epsilon'/8$ (so ϵ only depends on ϵ') and execute this algorithm on the constructed instance. We observe that $m \leq 2\sqrt{N} \cdot 7^{\sqrt{N}}$, and that $2^{\epsilon n^2} \leq 2^{\epsilon' N/2}$. Assuming that N is sufficiently large, using the supposed algorithm for CP-Dec we obtain an algorithm for Sparse 3-SAT with complexity at most $2^{\epsilon' N}$. Since we can do this for arbitrary ϵ', this contradicts the randomized ETH. □

8.5 Approximability of CP-Opt

This section begins with a $(\frac{1}{2} + O(\frac{1}{n}))$-approximation algorithm which works when words can, or cannot, be reused. After that, we prove that under the unique games conjecture, an approximation algorithm with a significantly better ratio is unlikely.

Theorem 8.12. *CP-Opt is $(\frac{1}{2} + \frac{1}{2(\varepsilon n+1)})$-approximable in polynomial time, for all $\varepsilon \in (0, 1]$.*

Proof. Fix some $\varepsilon \in (0, 1]$. Let $k_v := \min(\lceil \frac{1}{\varepsilon} \rceil, n - h)$ and $r_v := \lceil \frac{n-h}{k_v} \rceil$, where h is the number of horizontal slots in the grid. Create r_v groups of vertical slots $G_1, \ldots, G_{r_v}$ such that $|G_i| \leq k_v$ for all $i \in [r_v]$ and $G_1 \cup \ldots \cup G_{r_v}$ covers the entire set of vertical slots. For each G_i, guess an optimal choice of words, i.e., identical to a global optimum, and complete this partial solution by filling the horizontal slots (use the aforementioned matching technique where the words selected for G_i are excluded from $\mathcal{D}$). Each slot of $\bigcup_{j \neq i} G_j$ gets the empty word.

Since $|G_i| \leq k_v$, guessing an optimal choice of words for G_i by brute force requires at most $(m+1)^{k_v}$ combinations. This is done r_v times (once for each G_i). The maximum matching runs in time $\mathcal{O}((m+n)^2 \cdot mn)$. In all, the time complexity of the algorithm is $\mathcal{O}((m+1)^{k_v} \cdot r_v \cdot (m+n)^2 \cdot mn) \leq \mathcal{O}((m+1)^{1/\varepsilon} \cdot \varepsilon n \cdot (m+n)^2 \cdot mn)$.

Assume that, given an optimal solution, W_H^* and W_V^* are the total weight of the words assigned to the horizontal and vertical slots, respectively, both including the shared cells. Furthermore, let W_S^* be the weight of the letters assigned to the shared cells in the optimal solution. Observe that the weight of the optimal solution is $W_H^* + W_V^* - W_S^*$ and the weight of our solution is at least $W_H^* + \frac{1}{r_v}(W_V^* - W_S^*)$.

We repeat the same process, but the roles of vertical and horizontal slots are interchanged. Fix a parameter $k_h := \min(\lceil \frac{1}{\varepsilon} \rceil, h)$. Create $r_h := \lceil \frac{h}{k_h} \rceil$ groups of horizontal slots $G_1, \ldots, G_{r_h}$ such that $|G_i| \leq k_h$ for all $i \in [r_h]$ and $G_1 \cup \ldots \cup G_{r_h}$ covers the entire set of horizontal slots. For each G_i, guess an optimal choice of words and complete this partial solution by filling the vertical slots. Each slot of $\bigcup_{j \neq i} G_j$ gets the empty word.

Using the same arguments as above, we can conclude that the time complexity is $O((m+1)^{1/\varepsilon} \cdot \varepsilon n \cdot (m+n)^2 \cdot mn)$ and that we return a solution of weight at least $W_V^* + \frac{1}{r_h}(W_H^* - W_S^*)$.

Finally, between the two solutions, we return the one with the greater weight. It remains to argue about the approximation ratio. We need to consider two cases: $W_H^* \geq W_V^*$ and $W_V^* > W_H^*$.

Suppose $W_H^* \geq W_V^*$. The first approximate solution has value $W_H^* + \frac{1}{r_v}(W_V^* - W_S^*) \geq \frac{1+1/r_v}{2}(W_H^* + W_V^* - W_S^*)$. If $k_v = n - h$ then $r_v = 1$ and our approximation ratio is 1. Otherwise, $k_v = \lceil \frac{1}{\varepsilon} \rceil$ and $r_v = \lceil \frac{n-h}{\lceil 1/\varepsilon \rceil} \rceil \leq \frac{n-h}{\lceil 1/\varepsilon \rceil} + 1 = \frac{n-h+\lceil 1/\varepsilon \rceil}{\lceil 1/\varepsilon \rceil}$. It follows that $\frac{1}{r_v} \geq \frac{\lceil 1/\varepsilon \rceil}{n-h+\lceil 1/\varepsilon \rceil}$. Use $n-h+\lceil 1/\varepsilon \rceil \leq n+\frac{1}{\varepsilon}$ and $\lceil 1/\varepsilon \rceil \geq 1/\varepsilon$ to get that $\frac{1}{r_v} \geq \frac{1/\varepsilon}{n+1/\varepsilon} = \frac{1}{\varepsilon n+1}$. Our approximation ratio is at least $\frac{1+1/(\varepsilon n+1)}{2}$.

Suppose $W_V^* > W_H^*$. The second approximate solution has value $W_V^* + \frac{1}{r_h}(W_H^* - W_S^*) > \frac{1+1/r_h}{2}(W_H^* + W_V^* - W_S^*)$. If $k_h = h$, then our approximation ratio is 1. Otherwise, $k_h = \lceil \frac{1}{\varepsilon} \rceil$ and, using the same arguments, our approximation ratio is at least $\frac{1+1/(\varepsilon n+1)}{2}$.

Note that $\frac{1+1/(\varepsilon n+1)}{2} \leq 1$. In all, we have a $\frac{1+1/(\varepsilon n+1)}{2}$-approximate solution in $\mathcal{O}((m+1)^{1/\varepsilon} \cdot \varepsilon n \cdot (m+n)^2 \cdot mn)$ for all $\varepsilon \in (0, 1]$. □

The previous approximation algorithm only achieves an approximation ratio of $\frac{1}{2} + O(\frac{1}{n})$, which tends to $\frac{1}{2}$ as n increases. At first glance this is quite disappointing, as someone can observe that a ratio of $\frac{1}{2}$ is achievable simply by placing words only on the horizontal or the vertical slots of the instance[2]. Nevertheless, we are going to show that this performance is justified, as improving upon this trivial approximation ratio would falsify the Unique Games Conjecture (UGC).

Using the version of the UGC given in Theorem 2.1 we are ready to present our hardness of approximation argument for the crossword puzzle.

Theorem 8.13. *Suppose that the Unique Games Conjecture is true. Then, for all ϵ with $\frac{1}{4} > \epsilon > 0$, there exists an alphabet Σ_ϵ such that it is NP-hard to distinguish between the following two cases of instances of the crossword problem on alphabet Σ_ϵ:*

- *(Yes case): There exists a valid solution that fills a $(1-\epsilon)$-fraction of all cells.*
- *(No case): No valid solution can fill more than a $(\frac{1}{2}+\epsilon)$-fraction of all cells.*

Moreover, the above still holds if all slots have distinct lengths (and hence reusing words is trivially impossible).

[2]This placement is done in a way that maximizes the weight, using the matching technique as in the proof of Proposition 8.1.

Proof. Fix an $\epsilon > 0$. We will later define an appropriately chosen value $\epsilon' \in (0, \epsilon)$ whose value only depends on ϵ. We present a reduction from a Unique Label Cover instance, as described in Theorem 2.1. In particular, suppose we have an instance $G = (V, E)$, with $|V| = n$, alphabet $[R]$, such that (under UGC) it is NP-hard to distinguish if there exists a set V' of size $(1 - \epsilon')n$ that satisfies all its induced constraints, or if all sets V' of size $\epsilon' n$ induce at least one violated constraint for any assignment. Throughout this proof we assume that n is sufficiently large (otherwise the initial instance is easy). In particular, let $n > \frac{20}{\epsilon}$.

We construct an instance of the crossword puzzle that fits in an $N \times N$ square, where $N = 4n + n^2$. We number the rows $1, \ldots, N$ from top to bottom and the columns $1, \ldots, N$ from left to right. The instance contains n horizontal and n vertical slots. For $i \in [n]$, the i-th horizontal slot is placed in row $2i$, starting at column 1, and has length $2n + n^2 + i$. For $j \in [n]$, the j-th vertical slot is placed in column $2j$, starts at row 1 and has length $3n + n^2 + j$. Observe that all horizontal slots intersect all vertical slots and in particular, for all $i, j \in [n]$ the cell in row $2i$, column $2j$ belongs to the i-th horizontal slot and the j-th vertical slot. Furthermore, each slot has a distinct length, as the longest horizontal slot has length $3n + n^2$ while the shortest vertical slot has length $3n + n^2 + 1$.

We define the alphabet as $\Sigma_\epsilon = [R] \cup \{*\}$. Before we define our dictionary, let us give some intuition. Let $V = \{v_1, \ldots, v_n\}$. The idea is that a variable $v_i \in V$ of the original instance will be represented by both the i-th horizontal slot and the i-th vertical slot. In particular, we will define, for each $\alpha \in [R]$ a pair of words that we can place in these slots to represent the fact that v_i is assigned with the value α. We will then ensure that if we place words on both the i-th horizontal slot and the j-th horizontal slot, where $v_i v_j \in E$, then the assignment that can be extracted by reading these words will satisfy the constraint $\pi_{(v_i, v_j)}$. The extra letter $*$ represents an indifferent assignment (which we need if $v_i v_j \notin E$).

Armed with this intuition, let us define our dictionary.

- For each $i \in [n]$, for each $\alpha \in [R]$ we define a word $d_{(i,\alpha)}$ of length $2n + n^2 + i$. The word $d_{(i,\alpha)}$ has the character $*$ everywhere except at position $2i$ and at positions $2j$ for $j \in [n]$ and $v_i v_j \in E$. In these positions the word $d_{(i,\alpha)}$ has the character α.
- For each $j \in [n]$, for each $\alpha \in [R]$ we define a word $d'_{(j,\alpha)}$ of length $3n + n^2 + j$. The word $d'_{(j,\alpha)}$ has the character $*$ everywhere except at position $2j$ and at positions $2i$ for $i \in [n]$ and $v_i v_j \in E$. In position $2j$ we have the character α. In position $2i$ with $v_i v_j \in E$, we place the character $\beta \in [R]$ such that the constraint $\pi_{(v_i, v_j)}$ is satisfied by assigning β to v_i and α to v_j. (Note that β always exists and is unique, as the constraints are permutations on $[R]$, that is, for each value α of v_j there exists a unique value β of v_i that satisfies the constraint).

This completes the construction. Suppose now that $V = \{v_1, \ldots, v_n\}$ and that we started from the Yes case of Unique Label Cover, that is, there exists a set $V' \subseteq V$ such that $|V'| \geq (1 - \epsilon')n$ and all constraints induced by V' can be simultaneously satisfied. Fix an assignment $\sigma : V' \to [R]$ that satisfies all constraints induced by V'.

For each $i \in [n]$ such that $v_i \in V'$ we place in the i-th horizontal slot (that is, in row $2i$) the word $d_{(i,\sigma(v_i))}$. For each $j \in [n]$ such that $v_j \in V'$ we place in the j-th vertical slot the word $d'_{(j,\sigma(v_j))}$. We leave all other slots empty. We claim that this solution is valid, that is, no shared cell is given different values from its horizontal and vertical slot. To see this, examine the cell in row $2i$ and column $2j$. If both of the slots that contain it are filled, then $v_i, v_j \in V'$. If $v_i v_j \notin E$ and $i \neq j$, then the cell contains $*$ from both words. If $i = j$, then the cell contains $\sigma(v_i)$ from both words. If $i \neq j$ and $v_i v_j \in E$, then the cell contains $\sigma(v_i)$. This is consistent with the vertical word, as the constraint $\pi_{(v_i,v_j)}$ is assumed to be satisfied by σ. We now observe that this solution covers at least $2(1-\epsilon')n^3$ cells, as we have placed $2(1-\epsilon')n$ words, each of length at least n^2+2n, that do not pairwise intersect beyond their first $2n$ characters.

Suppose now we started our construction from a No instance of Unique Label Cover. We claim that the optimal solution in the new instance cannot cover significantly more than half the cells. In particular, suppose a solution covers at least $(1+\epsilon')n^3+10n^2$ cells. We claim that the solution must have placed at least $(1+\epsilon')n$ words. Indeed, if we place at most $(1+\epsilon')n$ words, as the longest word has length n^2+4n, the maximum number of cells we can cover is $(1+\epsilon')n(n^2+4n) \leq (1+\epsilon')n^3+4(1+\epsilon')n^2 < (1+\epsilon')n^3+10n^2$. Let x be the number of indices $i \in [n]$ such that the supposed solution has placed a word in both the i-th horizontal slot and the i-th vertical slot. We claim that $x \geq \epsilon' n$. Indeed, if $x < \epsilon' n$, then the total number of words we might have placed is at most $(n-x)+2x < (1+\epsilon')n$, which contradicts our previous observation that we placed at least $(1+\epsilon')n$ words. Let $V' \subseteq V$ be defined as the set of $v_i \in V$ such that the solution places words in the i-th horizontal and vertical slot. Then $|V'| \geq \epsilon' n$. We claim that it is possible to satisfy all the constraints induced by V' in the original instance, obtaining a contradiction. Indeed, we can extract an assignment for each $v_i \in V'$ by assigning to v_i value α if the i-th horizontal slot contains the word $d_{(i,\alpha)}$. Note that the i-th horizontal slot must contain such a word, as these words are the only ones that have an appropriate length. Observe that in this case the i-th vertical slot must also contain $d'_{(i,\alpha)}$. Now, for $v_i, v_j \in V'$, with $v_i v_j \in E$ we see that $\pi_{(v_i,v_j)}$ is satisfied by our assignment, otherwise we would have a conflict in the cell in position $(2i, 2j)$. Therefore, in the No case, it must be impossible to fill more than $(1+\epsilon')n^3+10n^2$ cells.

The only thing that remains is to define ϵ'. Let C be the total number of cells in the instance. Recall that we proved that in the Yes case we cover at least $2(1-\epsilon')n^3$ cells and in the No case at most $(1+\epsilon')n^3+10n^2$ cells. So we need to define ϵ' such that $2(1-\epsilon')n^3 \geq (1-\epsilon)C$ and $(1+\epsilon')n^3+10n^2 \leq (\frac{1}{2}+\epsilon)C$. To avoid tedious calculations, we observe that $2n^3 \leq C \leq 2n^3+8n^2$. Therefore, it suffices to have $2(1-\epsilon')n^3 \geq 2(1-\epsilon)(n^3+4n^2)$ and $(1+\epsilon')n^3+10n^2 \leq (1+2\epsilon)n^3$. The first inequality is equivalent to $(\epsilon-\epsilon')n \geq 4(1-\epsilon)$ and the second inequality is equivalent to $(2\epsilon-\epsilon')n \geq 10$. Since we have assumed that $n \geq 20/\epsilon$, it is sufficient to set $\epsilon' = \epsilon/2$. □

8.6 Special Cases Solvable in Polynomial Time

In this section we give some instances of CP-Dec which can be solved in polynomial time when word reuse is not allowed. Hereafter, we will always silently assume that word reuse is not allowed. Motivated by the fact that we could not generalize the NP-hardness proof for the case where the grid graph is a matching and the alphabet contains only two letters, we start considering restricted cases, that can be solved in polynomial time, with the hope that we will show that we can answer this question in polynomial time. Although this specific case remains open, we were able to find other instances of the problem that can be solved in polynomial time. In the propositions that follow, the spacial cases we ask for may seem extremely restrictive, however, given our initial goal, it becomes easier to understand how we end up with such constraints.

We propose reductions from these instances to some well-known problems that belong to P. The first of these problems is 2-SAT which can be solved in linear time (see [80]). The second problem is the maximum matching problem. In a bipartite graph $G = (V, E)$ this problem can be solved in $O(\sqrt{|V|}(|E| + |V|)$ [113]. For general graphs, a much more involved algorithm by Micali and Vazirani matches the previous performance [152]. Finally, we will reduce some instances of CP-Dec to the Exact Matching problem. Karzanov in [127] proved that Exact Matching with 0-1 weights can be solved in polynomial time in complete balanced bipartite graphs. In general graphs, under the same weight restrictions, there exists an RNC algorithm by Mulmuley et al. [162], which implies that Exact Matching is solvable in randomized polynomial time – though we note that finding a deterministic polynomial time algorithm for Exact Matching is a notorious open problem.

If the grid graph of the crossword puzzle is a matching, then we will call *crosses* the pairs of slots that intersect. We will say that a pair of words (d_i, d_j), $i \neq j$ and $i, j \in [m]$, indicated by their indices (i, j), can be assigned to a cross $\mathcal{C}$ if d_i fits into the horizontal slot of this cross, d_j fits into the vertical slot and they have the same letter in the shared cell. In the sequel, pairs (i, j) and (j, i) for any $i \neq j$ and $i, j \in [m]$ will count as different pairs.

Proposition 8.14. *CP-Dec can solved in polynomial time for instances where all of the following conditions apply: (i) the grid consists of $n/2$ crosses, (ii) for any cross of the grid we have at most two pairs of words that can fit into it.*

We can recognize this kind of instances in polynomial time by counting the pairs that fit in each cross as follows. For each cross $\mathcal{C}_k$, $k \in [n/2]$,

- first we find the set $\mathcal{D}_{k,h}$ of words that fit into the horizontal slot of $\mathcal{C}_k$,
- for each word $d_i \in \mathcal{D}_{k,h}$ we find the set of words d_j, different from d_i, that can fit into the vertical slot of $\mathcal{C}_k$, and such that d_i and d_j agree on the shared cell of $\mathcal{C}_k$; let $\mathcal{D}_{k,i,v}$ be this set,
- let $P_k = \{(i, j) \mid d_i \in \mathcal{D}_{k,h} \text{ and } d_j \in \mathcal{D}_{k,i,v}\}$ be the set of pairs that can be assigned to $\mathcal{C}_k$,

- the number of pairs that fit in $\mathcal{C}_k$ is: $np(\mathcal{C}_k) = \sum_{i:d_i \in \mathcal{D}_{k,h}} |\mathcal{D}_{k,i,v}|$.

So we can recognize if we have this kind of instance by checking if $np(\mathcal{C}_k) \leq 2$ for all $k \in [n/2]$. Furthermore, the above process is polynomial and can give us the pairs of words that can be assigned to each cross. Now, we are going to present the proof of Proposition 8.14.

Proof. First let $\mathcal{D} = \{d_1, \ldots, d_m\}$ be the dictionary, $\mathcal{C}_k$, $k \in [n/2]$, be all the crosses of the grid and P_k, $k \in [n/2]$, the pairs of word indices that can be assigned to $\mathcal{C}_k$. Observe that if $P_k = 0$ for some $k \in [n/2]$, then we know that we can not completely fill the grid. Moreover, if $P_k = 1$, for some $k \in [n/2]$, then there exists only one pair (i, j) for the cross $\mathcal{C}_k$ so we can assign the only pair of words that fits into $\mathcal{C}_k$ and reduce the instance by removing $\mathcal{C}_k$ from the grid, and the words from the dictionary. Therefore, we can assume that $P_k = 2$ for all $k \in [n/2]$.

From the sets P_k, $k \in [n/2]$, we will construct an instance of 2-SAT that is satisfiable if and only if we can completely fill the given grid without reusing the words of the dictionary $\mathcal{D}$. Before we continue the construction, let us mention that 2-SAT can be solved in linear time [80].

We start by creating $\sum_{k \in [n/2]} |P_k| = n$ variables as follows; for each $k \in [n/2]$ and each $(i, j) \in P_k$ we create the variable $x_{k,i,j}$. Let X be the set of all the variables we created. Now, we will construct a CNF formula ϕ such that each clause contains at most two variables. First we add in ϕ $n/2$ clauses c_k, $k \in [n/2]$ as follows. For each cross $\mathcal{C}_k$, $k \in [n/2]$, we add the clause $c_k = (x_{k,i,j} \vee x_{k,i',j'})$ where (i, j) and (i', j') are in P_k.

After that, for all $i \in [m]$ let X_i be the set of variables related to the word d_i, i.e., $X_i := X \cap \{x_{k,i,j}, x_{k,j,i} \mid k \in [n/2] \text{ and } d_j \in \mathcal{D}\}$. If $|X_i| \geq 2$, then for any pair of variables $x, x' \in X_i$, $x \neq x'$, we add in ϕ a new clause $(\neg x \vee \neg x')$. That process creates $\sum_{i \in [m]} \binom{|X_i|}{2} < mn^2$ additional clauses as $|X_i| \leq n$. Therefore, the number of clauses in ϕ is $O(mn^2)$.

It remains to prove that ϕ is satisfiable if and only if we can completely fill the grid. Assume that ϕ is satisfiable and $f : X \to \{T, F\}$ is a satisfying assignment. Then, we claim that for each $k \in [n/2]$ there exists at least one variable $x_{k,i,j}$ such that $f(x_{k,i,j}) = T$. For $k \in [n/2]$, we select one such a variable, $x_{k,i,j}$, and we assign d_i horizontally and d_j vertically to the cross $\mathcal{C}_k$.

Let us argue why there is always such a variable for each $k \in [n/2]$. By construction, for each $k \in [n/2]$ we have a clause c_k that can be satisfied only if a variable $x_{k,i,j}$, such that $(i, j) \in P_k$, has the value T. So, we know that for each $k \in [n/2]$ we have one such a variable. Furthermore, since $(i, j) \in P_k$, the words d_i and d_j fits to the horizontal and vertical slots of the cross $\mathcal{C}_k$ respectively.

We need to prove that we have not assigned the same word to more than one crosses. Assume that we have assigned a word d_i to two different crosses. Then, there exist two variables x and x' in X_i such that $f(x) = f(x') = T$. This is a contradiction because for each such pair of variables we have a clause $(\neg x \vee \neg x')$.

For the reverse direction we will show the following. If the grid is completely filled

then the truth assignment f such that

$$f(x_{k,i,j}) = \begin{cases} T, \text{ if we have assigned the pair } (i,j) \in P_k \text{ to the cross } \mathcal{C}_k, \\ F, \text{ otherwise,} \end{cases}$$

is a satisfying assignment for the formula ϕ. First, observe that for each $k \in [n/2]$ the clause c_k is satisfied because in $\mathcal{C}_k$ we have assigned a pair $(i,j) \in P_k$ (these are the only pairs that can be assigned to $\mathcal{C}_k$) so the variable $x_{k,i,j}$ that appears positively in c_k takes the value T by f. In order to complete the proof we need to show that at most one of the variables in X_i is true. Observe that if two of them are true then we know that we have two pairs of words containing d_i and they are assigned to a cross. This is a contradiction because we can not reuse words. □

If the shared cell of a cross is the first cell of the vertical slot, then the cross is called $\mathcal{T}$ because the slots have the form of a capital T. The next proposition shows that under some restrictions, the crossword problem can be solved efficiently if the grid is only made of $\mathcal{T}$'s. We remark that the grid could have an unbounded number of different kinds of $\mathcal{T}$'s.

Proposition 8.15. *CP-Dec can be solved in polynomial time if the alphabet $\mathcal{L}$ has only 2 letters, l_1 and l_2, and the grid has the following properties: (i) it only consists of $\mathcal{T}$'s and (ii) all the horizontal slots have length ℓ_h and all the vertical slots have length ℓ_v, where $\ell_h \neq \ell_v$.*

Proof. Starting from the dictionary $\mathcal{D}$, we can create two disjoint sub-dictionaries $\mathcal{D}_h$ and $\mathcal{D}_v$ containing the words that can fit into the horizontal slots and vertical slots, respectively.

Now, let d_i, $i \in [|\mathcal{D}_h|]$ be the words in $\mathcal{D}_h$ and d'_i, $i \in [|\mathcal{D}_v|]$ be the words in $\mathcal{D}_v$. Furthermore, let m_1 and m_2 be the number of words in $\mathcal{D}_v$ that start with l_1 and l_2, respectively. Finally, let us call $\mathcal{T}_i$, for each $i \in [n/2]$, the $\mathcal{T}$'s of the grid.

Observe that, for any completely filled grid, we know the minimum and the maximum possible number of appearances of the symbol l_1 in the shared cells. In particular, since m_2 words of $\mathcal{D}_v$ do not start with l_1 we have at least $w_1 = \max\{0, n/2 - m_2\}$ appearances and since m_1 words of $\mathcal{D}_v$ start with l_1 we have at most $w_2 = \min\{n/2, m_1\}$ appearances. So we need to decide if there exists a way to fill completely the horizontal slots of the grid that forces w appearances of l_1 in the shared cells for a $w \in \{w_1, \ldots, w_2\}$.

Based on $\mathcal{D}_h$ and the $\mathcal{T}$'s we will construct an instance of the Exact Matching problem. First, we create a complete balanced bipartite graph $G = (V, U, E)$ where $V = \{v_1, \ldots, v_{|\mathcal{D}_h|}\}$ represents the words in $\mathcal{D}_h$, vertices $u_j \in U$ where $j \in [n/2]$ represent the $\mathcal{T}$'s of the grid and vertices $u_j \in U$ such that $j \in \{n/2+1, \ldots, |\mathcal{D}_h|\}$ are added so that G is balanced.

It remains to assign weights to the edges. For each edge $v_i u_j \in E$, if $i \in [|\mathcal{D}_h|]$, $j \in [n/2]$ and d_i contains symbol l_1 in the position of the shared cell of the horizontal slot of $\mathcal{T}_j$, then $v_i u_j$ has weight 1, otherwise its weight is 0.

Now it is not difficult to observe that G has a perfect matching of weight exactly w if and only if we can fill all the horizontal slots of the grid with words of $\mathcal{D}_h$ such that l_1 appears exactly w times in the shared cells. Finally, we can decide if G has a perfect matching of weight w, for any $w \in \{w_1, \ldots, w_2\}$, in polynomial time by using Theorem 1 in [127]. □

Remark. *The same proof works if instead of $\mathcal{T}$'s (where two slots intersect only in the first position of the vertical slot), one has crosses such that any two slots that intersect do so only at the same position of the vertical slot.*

At first sight, the case covered by Proposition 8.15 looks restricted but it necessitates to match three objects (1 horizontal word, 1 vertical word, and a cross) and 3D-matchings are, in general, hard problems [94].

Proposition 8.16. *CP-Dec can be solved in polynomial time if the instance has the following properties: (i) the grid graph is a matching and (ii) the number of different types of crosses*[3] *is a constant.*

Proof. First, we will prove that if we know the number of appearances of each letter in the shared cells of each type of crosses, then we can find a way to fill the grid (if there exists one) using the maximum matching problem.

Let $t \in \mathbb{N}$ be the number of different types of crosses and let $\mathcal{L} = \{l_1, \ldots, l_\ell\}$ be the alphabet. Furthermore, suppose that the given instance has a solution and that we are given values $a_{i,j}$, $i \in [\ell]$ and $j \in [t]$ which indicate the number of appearances of the letter l_i in the shared cell of type j crosses in this solution. In the end we will repeat the algorithm for all combinations of such values, so we can assume that these values are given to us in the input. We will therefore look for a solution that agrees with the given values $a_{i,j}$.

Now, we construct the following bipartite graph.

- For each pair $(i,j) \in [\ell] \times [t]$ we create a set $V_{i,j} = \{v_{i,j,k}, v'_{i,j,k} \mid k \in [a_{i,j}]\}$ of $2a_{i,j}$ vertices. Furthermore, let V be the set $\bigcup_{(i,j)\in[\ell]\times[t]} V_{i,j}$.

- For each word $d \in \mathcal{D}$ we create a vertex u_d. Let U be the set of these vertices.

- For all $d \in \mathcal{D}$ and $(i,j) \in [\ell] \times [t]$, if the word d fits in the horizontal slot of a cross of type j and it forces the letter l_i in the shared cell of this slot, then we add all the edges $u_d v_{i,j,k}$ where $k \in [a_{i,j}]$.

- Finally, for all $d \in \mathcal{D}$ and $(i,j) \in [\ell] \times [t]$, if the word d fits in the vertical slot of a cross of type j and it forces the letter l_i in the shared cell of this slot, then we add all the edges $u_d v'_{i,j,k}$ where $k \in [a_{i,j}]$.

[3] Two crosses are of the same type if they are identical: same number of horizontal cells, same number of vertical cells, and same shared cell.

Now, we claim that this graph has a matching that covers all the vertices in V if and only if we can completely fill the grid in a way that respects the given appearances of the letters. That is, for each letter $i \in [\ell]$ and each type $j \in [t]$ there are exactly $a_{i,j}$ crosses of type j that contain i in the shared cell in the solution.

In one direction, assume that we have an assignment of words in the grid that respects the given appearances of the letters. For each $(i,j) \in [\ell] \times [t]$ let $\mathcal{D}_{i,j}$ be the set of words that have been assigned to the crosses of type j and these words force the letter l_i in the shared cell. Observe that $\mathcal{D}_{i,j}$ has size $2a_{i,j}$ as it respects the appearances of the letters and for each $d \in \mathcal{D}_{i,j}$ the vertex u_d is adjacent to $v_{i,j,k}$, for all $k \in [a_{i,j}]$, if d has been assigned to a horizontal slot or is adjacent to $v'_{i,j,k}$, for all $k \in [a_{i,j}]$ if d has been assigned to a vertical slot.

Now we are going to create a matching of the graph. Starting from an empty set S, for each $(i,j) \in [\ell] \times [t]$ and each word d in $\mathcal{D}_{i,j}$, if d has been assigned to a horizontal slot, then we add in S an edge $u_d v_{i,j,k}$ for some k in $[a_{i,j}]$ that covers an uncovered vertex $v_{i,j,k}$; otherwise we add an edge $u_d v'_{i,j,k}$ for some k in $[a_{i,j}]$ that covers an uncovered vertex $v'_{i,j,k}$ (we know that there are enough vertices by the previous observations).

Observe that S is a matching because each time we add an edge incident to two uncovered vertices. Furthermore, because the size of $\mathcal{D}_{i,j}$ is $2a_{i,j}$ and we have exactly $a_{i,j}$ horizontal and $a_{i,j}$ vertical slots we know that we will cover all the $v_{i,j,k}$ with the corresponding vertices of the horizontal slots and all the $v'_{i,j,k}$ with the corresponding vertices of the vertical slots. Hence, S is a matching that covers all the vertices of V.

For the reverse direction, assume that S is a matching that covers all the vertices of V. Because V is an independent set we know that for all $v \in V$ there exists an edge $u_d v \in S$ for some word $d \in \mathcal{D}$ and $u_d \in U$. We will assign words to the slots of the grid as follows. For each edge $u_d v_{i,j,k}$ we assign the word d to an empty horizontal slot of a cross of type j. Because all $v_{i,j,k}$, $k \in [a_{i,j}]$, are adjacent to vertices u_d such that d fits in the horizontal slots of crosses of type j and forcing the letter l_i in the shared cells, we have filled all the horizontal slots in a way that respects the given appearances of the letters.

Now, we are going to fill the vertical slots. For each edge $u_d v'_{i,j,k}$ we assign the word d to an empty vertical slot of a cross of type j where the shared cell has the letter l_i. Because for a given pair $(i,j) \in [\ell] \times [t]$ we have $a_{i,j}$ vertices $v_{i,j,k}$ and $v'_{i,j,k}$ we know that we have forced exactly $a_{i,j}$ times the letter l_i in the shared cell of crosses of type j, so we have the exact number of vertical slots we want.

Finally, because the total number of vertices in V is the same as the number of slots in the grid, we know that we have completely filled the grid.

Now, note that we can decide in polynomial time if there exists such a matching (by finding a maximum matching).

In order to complete the proof we need to show that the different guesses for the number of appearances $a_{i,j}$, $(i,j) \in [\ell] \times [t]$ is polynomially bounded by the input size. Fix a type j and suppose it has ν_j crosses. Enumerating all the possible $a_{i,j}$'s is equivalent to choosing $\ell - 1$ positions in a row vector of size $\nu_j + \ell - 1$ (the chosen cells "separate" the $a_{i,j}$'s). Then, there are $\binom{\nu_j+\ell-1}{\ell-1}$ choices, which is upper bounded by $\binom{n+\ell-1}{\ell-1}$.

Finally, because we have t types of crosses, the total number of guesses is $O(n^{t\ell})$ which is polynomial as t and ℓ are constant. □

So far we have assumed that the size of the alphabet is constant. In contrast, the next two propositions hold without this hypothesis. Therefore, there are independent of Proposition 8.16.

Proposition 8.17. *CP-Dec can be solved in polynomial time when the grid consists of identical crosses. This holds even if the size of the alphabet is not constant.*

Proof. Let $\mathcal{D}$ be the dictionary of size m and let $n/2$ be the number of crosses in the grid. In order to answer the question we will create a graph G on m vertices such that G has a matching of size at least $n/2$ if and only if we can completely fill the grid. We construct $G = (V, E)$ as follows: V has exactly one vertex v_i for each word $d_i \in \mathcal{D}$ and E contains the edge v_iv_j if and only if at least one of the pairs of words (i, j) or (j, i) fits in a cross.

Now, observe that an edge of G gives us a pair of words that can fit into a cross. Therefore, each matching S represents pairs of words that (all of them) fit in crosses. Because the words are represented by vertices in the graph, the pairs we take from a matching are independent as each vertex is covered at most once by S. Finally, if the matching has at least $n/2$ edges then we know that we have enough pairs to fill the grid completely.

Conversely, if we can cover the grid completely, then there exist $n/2$ distinct pairs of words that fit in a cross. For each of these pairs we have an edge in G. Furthermore, the set of these edges is a matching as we have not used the same word twice so there are no two edges incident to the same vertex. Hence, G has a matching of size at least $n/2$.

Because we can find a maximum matching of a graph in polynomial time we know that we can decide if we can fill the whole grid in the same time. □

Proposition 8.18. *CP-Dec is polynomial time reducible to* 0-1 *Exact Matching if the grid has the following properties: (i) it is a matching and (ii) the number of different types of crosses is a constant. This holds even if the size of the alphabet is not constant.*

Proof. Let $t \in \mathbb{N}$ be the number of different types of crosses. Let a_k, for $k \in [t]$, be the number of crosses of type t in the grid. We assume w.l.o.g. that the number of words $|\mathcal{D}| = m$ is even. We first create a multi-graph where parallel edges are allowed, and we assign weights to the edges from the set $\{0, m^0, \ldots, m^{t-1}\}$ as follows:

- Let $G = (V, E)$ be a complete graph with m vertices and give weight 0 to each edge.
- For each $k \in [t]$, add a set of edges E_k such that $v_iv_j \in E_k$ if one of the pairs of words (i, j) or (j, i) fits in a cross of type $k \in [t]$. Give weight m^{k-1} to all the edges in E_k.

In order to distinguish the parallel edges, denote by $(v_iv_j)_k$ the edge $v_iv_j \in E_k$. Now, we claim that this graph has a perfect matching of weight $W = \sum_{k\in[t]} a_k m^{k-1}$ if and only if we can fill the grid completely.

Assume that the grid is filled and construct a perfect matching as follows. Start from an empty set S. For each cross, add to S the edge $(v_iv_j)_k$ where (i, j) is the pair of words allocated to the cross and k is the type of the cross. Since reusing words is not allowed, we know that S is a matching. Furthermore, because each cross of type k has an edge of E_k, we get that S has weight W. Finally, S can be extended to a perfect matching of the same weight because all the vertices have edges of weight 0 between them.

Conversely, assume that the graph has a perfect matching S of weight W. Observe that any perfect matching has exactly $m/2$ edges. Therefore we know that we have at most $m/2$ edges from each set E_k, for any $k \in [t]$. Now we are going to show that we need exactly a_k edges from each set E_k. In particular, we claim that for any $k \in \{1, \ldots, t\}$ we have that $|S \cap E_k| = a_k$. We prove this by induction.

Base Case (t)**:** Assume that there exists a perfect matching S with weight W such that $|S \cap E_t| = b_t \neq a_t$. Assume that $b_t < a_t$. Since $0 \leq b_t$, we get that $1 \leq a_t$. Even if we use a maximum number of edges of maximum weight (i.e., $m/2$ edges of weight m^{t-2}), we cannot compensate, i.e., S cannot have weight W because $m^{t-2} \cdot m/2 < m^{t-1}$. Now, we have to check the case $b_t > a_t$. Since S has weight at least $(a_t + 1)m^{t-1}$, we have $W \geq (a_t + 1)m^{t-1}$. That gives us:

$$\sum_{k\in[t]} a_k m^{k-1} \geq (a_t + 1)m^{t-1} \Rightarrow \sum_{k\in[t-1]} a_k m^{k-1} \geq m^{t-1} \Rightarrow m^{t-2} \sum_{k\in[t-1]} a_k \geq m^{t-1}$$

which is a contradiction since $\sum_{k\in[t-1]} a_k \leq m/2$. Therefore, for any perfect matching of weight W it must be $|S \cap E_t| = a_t$.

Induction Hypothesis $(\{k, k + 1, \ldots, t\},\ k > 1)$**:** We assume that, for a given $k > 1$, $|S \cap E_i| = a_i$ holds for all $i \in \{k, k + 1, \ldots, t\}$.

Induction Step $(k - 1)$**:** For every $l \in \{k, \ldots, t\}$ we know that $|S \cap E_l| = a_l$. Therefore, the set $S' = S \setminus \bigcup_{l=k}^{t}(S \cap E_l)$ has weight exactly $\sum_{l=1}^{k-1} a_l m^{l-1}$.

Assume that $k - 1 = 1$; then $\sum_{l=1}^{k-1} a_l m^{l-1} = a_1$. Since S' consists only of edges in $E_1 \cup E$, the edges in E_1 have weight 1 and the edges in E have weight 0, so we can conclude that S' contains exactly a_1 edges from E_1.

Now, we consider the case $k - 1 > 1$. Similarly to the base case, assume that $|S' \cap E_{k-1}| = b_{k-1}$ and $b_{k-1} < a_{k-1}$. In this case, even if we use a maximum number of edges (i.e., $m/2$) of maximum weight (i.e., m^{k-2}), we cannot compensate, i.e., $m^{k-2} \cdot m/2 < m^{k-1}$. Now we prove that, assuming $b_{k-1} > a_{k-1}$ leads to a contradiction. Indeed, in this case S' has weight at least $(a_{k-1} + 1)m^{k-2}$ which implies that:

$$\sum_{l\in[k-1]} a_l m^{l-1} \geq (a_{k-1} + 1)m^{k-2} \Rightarrow \sum_{l\in[k-2]} a_l m^{l-1} \geq m^{k-2} \Rightarrow m^{k-3} \sum_{l\in[k-2]} a_k \geq m^{t-1}$$

The last inequality contradicts the fact that $\sum_{l\in[k-2]} a_k \leq \sum_{l\in[t]} a_t \leq m/2$. Hence, $b_{k-1} = a_{k-1}$.

Finally, observe that for any $k \in [t]$ each edge in $S \cap E_k$ gives us a pair of words that fits into a cross of type k. Moreover, because S is a matching and $|S \cap E_k| = a_k$ for all $k \in [t]$, we can completely fill the grid.

Now, starting from the graph we created in the first reduction, we will create an instance of 0-1 Exact Matching, using the same technique as Papadimitriou and Yannakakis in their proof of Proposition 1 of [167]. We build the new graph G' by replacing each edge $(v_i v_j)_k$, $k \in [t]$, with a path, $p_{i,j,k} = \langle v_i, u^k_{i,j,1}, \dots u^k_{i,j,2m^{k-1}}, v_j \rangle$. Assign weight 0 to the edges $u^k_{i,j,2m^{k-1}} v_j$ and $u^k_{i,j,l} u^k_{i,j,l+1}$, where $l < 2m^{k-1}$ is odd. All the other edges of the path have weight 1.

Now, let $E_{i,j,k}$ be the edges of a path $p_{i,j,k}$. Observe that for any perfect matching $M \subseteq E'$ of G' and any path $p_{i,j,k}$, the set of edges $M \cap E_{i,j,k}$:

- either, contains edges of total weight 0 and it does not cover any endpoint of $p_{i,j,k}$,
- or, it contains edges of total weight m^{k-1} and it covers both v_i and v_j.

Based on this observation we can transform any perfect matching of G into a perfect matching of G' with the same weight and vice versa.

Finally, since G' has $O(m^t)$ vertices, the starting instance of CP-Dec is polynomial time reducible to 0-1 Exact Matching. □

We can also use the technique of Papadimitriou and Yannakakis [167] in the proof of Proposition 8.15, giving us Corollary 8.19 which more general than Proposition 8.15.

Corollary 8.19. *CP-Dec with alphabet of constant size is polynomial time reducible to* 0-1 *Exact Matching if the grid has the following properties: (i) it is a matching, (ii) all shared cells are at the same position of the vertical slots, (iii) all vertical slots have the same length l_v and (iv) there is no horizontal slot of length l_v.*

The details are omitted but observe that we don't need to construct a complete bipartite graph and we can allow weights up to m^k, where $m = O(|\mathcal{D}|)$ and $k = O(|\mathcal{L}|)$, before we apply the aforementioned technique.

8.7 Conclusion

We studied the parameterized complexity of some crossword puzzles under several different parameters and we gave some positive results followed by proofs which show that our algorithms are essentially optimal. Based on our results the most natural questions that arise are:

- What is the complexity of CP-Dec when the grid graph is a matching and the alphabet has size 2?
- Can Theorem 8.11 be strengthened by starting from ETH instead of randomized ETH?

- Can we beat the $(\frac{1}{2} + O(\frac{1}{n}))$-approximation ratio of CP-Opt if we restrict our instances?
- Can Theorem 2.1 be strengthened by dropping the UGC?

Finally, as a future work, we could consider a variation of the crossword puzzle problems where each word can be used a given number of times. This would be an intermediate case between word reuse and no word reuse.

Chapter 9

Conclusions

In this manuscript we have considered several optimization problems and studied many different techniques that can be applied to them. We were mainly interested in variants of well-known problems; four of them are problems defined by graphs. In particular, we studied Upper r-Tolerant Edge Cover, the Upper Feedback Vertex Set, the Digraph Coloring, and the Maximum Locally Irregular Induced Subgraph problem. For these problems, we studied their complexity in the general case and for special cases as well; some preliminary results were already known but we completed the picture. In addition we studied the approximability of these problems. We presented some approximation algorithms and some hardness of approximation results. Finally, we studied their parameterized complexity. We decided which parameter to use based on the nature of the problem. Furthermore, we proved the optimality of the presented algorithms by giving lower bounds based on standard assumptions such as ETH.

The other two problems are the Subset-Sums Ratio (SSR) problem and a crossword completion problem. Both problems were known to be NP-hard. We first investigated the relationship between SSR and the well-known Subset Sum problem. This results in a new FPTAS for SSR that uses existing algorithms for the Subset Sum problem. In addition, we generalized other existing techniques in order to present a framework that gives FPTASs for variations of SSR when certain conditions are met. For the second problem our main approach was to consider instances of the puzzle where the grid has certain properties. Moreover, we studied the approximability of the problem, gave an approximation algorithm for the problem with ratio slightly better than $\frac{1}{2}$, and showed that we should not expect to improve this ratio.

Within the results presented in this document, we would like to highlight some of them and explain why they deserve to be highlighted. The first of them is the fine-grained reduction, presented in Chapter 6, form of 3-SAT to Digraph Coloring which, according to ETH, gives us a lower bound on the execution time of 2-Digraph Coloring when parameterized by treedepth is rather surprising. As second we want to mention, the r-approximation algorithm in Chapter 5 for the Max Min FVS problem which runs in $n^{O(n/r^{3/2})}$ along with the proof that this time-approximation trade-off is essentially tight under the ETH. Moreover, the relationship we established between SSR and

Subset Sum in Chapter 3 appears to be important for SSR, as any improvement for Subset Sum will lead to improved FPTAS for SSR. Finally we want to mention, the reduction presented in Section 8 which gives us a bound on the approximation ratio of the crossword puzzle. This reduction is under the Unique Game Conjecture, which is used for the hardness of approximation reductions.

Future work for each problem has been given in the end of the corresponding chapters, however, we would like to mention again some of the most interesting questions and add a few more.

Regarding the SSR, it is would be interesting to investigate weather we can give a reduction to Partition problem instead of Subset Sum. Second, it would be interesting to study the constraints that we can apply to SSR and guarantee the existence of pseudo polynomial-time algorithms. This could give us a more precise idea of which problems in F-SSR admit FPTASs. Finally, there are no existing lower bound on the running time of exact and approximate solutions for SSR. Such lower bounds could also give us a measure of how far from optimal our algorithms are.

For Upper r-EC and Max-Min FVS, there are to few results regarding their parameterized complexity. In particular, the polynomial kernel we presented in Chapter 5 for Max-Min FVS shows that the problem belongs in FPT when parameterized by the size of the solution as it can be translated to an exhaustive algorithm that runs in $2^{k^3}n^{O(1)}$. However, this performance is rather disappointing. It would be interesting to see whether we can develop an algorithm with better asymptotic performance. A natural question for the Upper r-EC is whether the problem belongs to FPT when parameterized by treewidth since we know that, when $r = 1$, this holds. Similarly to these two problems, there are many others that could be studied under the Max-Min (Min-Max) framework. Regarding this framework, a much more general question that we could address is whether there is any problem such that the Max-Min (or Min-Max) version of the problem is easier than the "original" version.

For k-Digraph Coloring it would be nice to settle the degree for which the problem is NP-hard for graph with directed feedback vertex set of size k. In particular, we have shown that the problem is FPT when the given graph has DFVS of size k and maximum degree $4k - 3$ but becomes NP-hard when the given graph has DFVS of size k and maximum degree $4k - 1$. However, the case where the DFVS is of size k and maximum degree is $4k - 2$ remains open. Furthermore, it would be interesting to consider graphs with larger values of the directed feedback vertex set and try to map out how the tractability threshold for the degree evolves from $4k - \Theta(1)$ to $2k + \Theta(1)$.

The problem of identifying the largest locally irregular induced subgraph of a given graph was studied for the first time in this document. Some interesting directions for future investigation are whether the problem of calculating $\mathrm{I}(G)$ remains NP-hard for other, restricted families of graphs such as chordal graphs and split graphs. Furthermore, it is worth investigating whether calculating $\mathrm{I}(G)$ could be done in FPT time (parameterised by the size of the solution or other parameter) in the case where G is a planar graph. Additionally, there is a huge variety of problems related to the irregularity of a graph, which have not been sufficiently studied from an algorithmic point of view.

For the problem of filling crosswords, the most intriguing questions are whether we can give strengthen our inapproximability result by dropping the UGC and to study complexity of the problem when, the grid graph is a matching and the alphabet contains only two letters. Moreover, it would be interesting to consider other variants of the problem, such as an intermediate case where repetition of words is allowed at most k times, for a given k, or crosswords on three-dimensional grids.

Finally, we want to present some more general directions. The first is to study the parameterized complexity of same problems under other parameters. In this document we have give more attention to the size of the solution and the treewidth of a graph and less attention to parameters like feedback vertex set and vertex cover. However, there are many other structural parameters that can be studied. Second, since most of the problems we studied are difficult both in terms of approximation and parameterization, we may need to find other ways to deal with them. In such cases we can try to develop parameterized approximation algorithms with the ultimate goal of obtaining fpt-algorithms that return approximate solutions.

Bibliography

[1] Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. Seth-based lower bounds for subset sum and bicriteria path. In *SODA 2019*, pages 41–57. SIAM, 2019.

[2] Hassan AbouEisha, Shahid Hussain, Vadim V. Lozin, Jérôme Monnot, Bernard Ries, and Viktor Zamaraev. Upper domination: Towards a dichotomy through boundary properties. *Algorithmica*, 80(10):2799–2817, 2018.

[3] Pierre Aboulker, Édouard Bonnet, Eun Jung Kim, and Florian Sikora. Grundy coloring & friends, half-graphs, bicliques. In *STACS*, volume 154 of *LIPIcs*, pages 58:1–58:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[4] Pierre Aboulker, Nathann Cohen, Frédéric Havet, William Lochet, Phablo F. S. Moura, and Stéphan Thomassé. Subdivisions in digraphs of large out-degree or large dichromatic number. *The Electronic Journal of Combinatorics*, 26(3):P3.19, 2019.

[5] Agostinho Agra, Geir Dahl, Torkel Andreas Haufmann, and Sofia J. Pinheiro. The k-regular induced subgraph problem. *Discretete Applied Mathematics*, 222:14–30, 2017.

[6] Yousef Alavi, Alfred Boals, Gary Chartrand, Ortrud Oellermann, and Paul Erdős. K-path irregular graphs. *Congressus Numerantium*, 65, 01 1988.

[7] Yousef Alavi, Gary Chartrand, Fan R. K. Chung, Paul Erdös, Ronald L. Graham, and Ortrud R. Oellermann. Highly irregular graphs. *Journal of Graph Theory*, 11(2):235–249, 1987.

[8] Akhbar Ali, Gary Chartrand, and Ping Zhang. *Irregularity in Graphs*. Springer briefs in mathematics. Springer, 2021.

[9] Anbulagan and Adi Botea. Crossword puzzles as a constraint problem. In *Principles and Practice of Constraint Programming, 14th International Conference, CP 2008. Proceedings*, pages 550–554, 2008.

[10] Stephan Dominique Andres and Winfried Hochstättler. Perfect digraphs. *Journal of Graph Theory*, 79(1):21–29, 2015.

[11] Antonis Antonopoulos, Aris Pagourtzis, Stavros Petsalakis, and Manolis Vasilakis. Faster algorithms for k-Subset Sum and variations. In *Frontiers of Algorithmics - International Joint Conference, IJTCS-FAW 2021, Beijing, China, August 16-19, 2021, Proceedings*, volume 12874 of *Lecture Notes in Computer Science*, pages 37–52. Springer, 2021.

[12] Esther M. Arkin, Michael A. Bender, Joseph S. B. Mitchell, and Steven Skiena. The lazy bureaucrat scheduling problem. *Inf. Comput.*, 184(1):129–146, 2003.

[13] S. Arumugam, S. T. Hedetniemi, S. M. Hedetniemi, L. Sathikala, and S. Sudha. The covering chain of a graph. *Util. Math*, 98:183–196, 2015.

[14] Yuichi Asahiro, Hiroshi Eto, Takehiro Ito, and Eiji Miyano. Complexity of finding maximum regular induced subgraphs with prescribed degree. *Theoretical Computer Science*, 550:21–35, 2014.

[15] Stavros Athanassopoulos, Ioannis Caragiannis, Christos Kaklamanis, and Maria Kyropoulou. An improved approximation bound for spanning star forest and color saving. In Rastislav Královic and Damian Niwinski, editors, *Proc. of 34th MFCS*, volume 5734 of *LNCS*, pages 90–101. Springer, 2009.

[16] Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Subset sum in the absence of concentration. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 48–61. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[17] Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Dense subset sum may be the hardest. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[18] Sandip Banerjee and Sujoy Bhore. Algorithm and hardness results on liar's dominating set and k-tuple dominating set. In Charles J. Colbourn, Roberto Grossi, and Nadia Pisanti, editors, *Proc. of IWOCA 2019*, volume 11638 of *Lecture Notes in Computer Science*, pages 48–60. Springer, 2019.

[19] Nikhil Bansal, Parinya Chalermsook, Bundit Laekhanukit, Danupon Nanongkai, and Jesper Nederlof. New tools and connections for exponential-time approximation. *Algorithmica*, 81(10):3993–4009, 2019.

[20] Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. A complexity dichotomy for hitting connected minors on bounded treewidth graphs: the chair and the banner draw the boundary. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 951–970. SIAM, 2020.

[21] Olivier Baudon, Julien Bensmail, Jakub Przybyło, and Mariusz Wozniak. On decomposing regular graphs into locally irregular subgraphs. *European Journal of Combinatorics*, 49:90–104, 2015.

[22] Cristina Bazgan, Ljiljana Brankovic, Katrin Casel, Henning Fernau, Klaus Jansen, Kim-Manuel Klein, Michael Lampis, Mathieu Liedloff, Jérôme Monnot, and Vangelis Th. Paschos. The many facets of upper domination. *Theoretical Computer Science*, 717:2–25, 2018.

[23] Cristina Bazgan, Miklos Santha, and Zsolt Tuza. Efficient approximation algorithms for the SUBSET-SUMS EQUALITY problem. *J. Comput. Syst. Sci.*, 64(2):160–170, 2002.

[24] Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, and Yota Otachi. Grundy Distinguishes Treewidth from Pathwidth. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:19, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[25] Rémy Belmonte and Ignasi Sau. On the complexity of finding large odd induced subgraphs and odd colorings. *Algorithmica*, 83(8):2351–2373, 2021.

[26] Julien Bensmail, Ararat Harutyunyan, and Ngoc-Khang Le. List coloring digraphs. *Journal of Graph Theory*, 87(4):492–508, 2018.

[27] Julien Bensmail, Martin Merker, and Carsten Thomassen. Decomposing graphs into a constant number of locally irregular subgraphs. *European Journal of Combinatorics*, 60:124–134, 2017.

[28] Benjamin Bergougnoux, Édouard Bonnet, Nick Brettell, and O-joung Kwon. Close relatives of feedback vertex set without single-exponential algorithms parameterized by treewidth. *CoRR*, abs/2007.14179, 2020.

[29] Piotr Berman, Marek Karpinski, and Alex D. Scott. Approximation hardness of short symmetric instances of MAX-3SAT. *Electronic Colloquium on Computational Complexity*, (049), 2003.

[30] Alan A. Bertossi. Dominating sets for split and bipartite graphs. *Inf. Process. Lett.*, 19(1):37–40, 1984.

[31] Dietmar Berwanger, Anuj Dawar, Paul Hunter, Stephan Kreutzer, and Jan Obdrzálek. The dag-width of directed graphs. *J. Comb. Theory, Ser. B*, 102(4):900–923, 2012.

[32] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

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

[34] Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015.

[35] Drago Bokal, Gasper Fijavz, Martin Juvan, P. Mark Kayll, and Bojan Mohar. The circular chromatic number of a digraph. *Journal of Graph Theory*, 46(3):227–240, 2004.

[36] Marthe Bonamy, Lukasz Kowalik, Jesper Nederlof, Michal Pilipczuk, Arkadiusz Socala, and Marcin Wrochna. On directed feedback vertex set parameterized by treewidth. In Andreas Brandstädt, Ekkehard Köhler, and Klaus Meer, editors, *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, volume 11159 of *Lecture Notes in Computer Science*, pages 65–78. Springer, 2018.

[37] Édouard Bonnet, Nick Brettell, O-joung Kwon, and Dániel Marx. Generalized feedback vertex set problems on bounded-treewidth graphs: Chordality is the key to single-exponential parameterized algorithms. *Algorithmica*, 81(10):3890–3935, 2019.

[38] Édouard Bonnet, Michael Lampis, and Vangelis Th. Paschos. Time-approximation trade-offs for inapproximable problems. *Journal of Computer and System Sciences*, 92:171 – 180, 2018.

[39] Kellogg S. Booth and J. Howard Johnson. Dominating sets in chordal graphs. *SIAM J. Comput.*, 11(1):191–199, 1982.

[40] Nicolas Boria, Federico Della Croce, and Vangelis Th. Paschos. On the max min vertex cover problem. *Discrete Applied Mathematics*, 196:62–71, 2015.

[41] Nicolas Bourgeois, Bruno Escoffier, and Vangelis Th. Paschos. Approximation of min coloring by moderately exponential algorithms. *Inf. Process. Lett.*, 109(16):950–954, 2009.

[42] Arman Boyaci and Jérôme Monnot. Weighted upper domination number. *Electron. Notes Discret. Math.*, 62:171–176, 2017.

[43] Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 1073–1084, Philadelphia, PA, January 2017. SIAM.

[44] Karl Bringmann and Vasileios Nakos. Top-k-convolution and the quest for near-linear output-sensitive subset sum. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 982–995, New York, NY, USA, 2020. ACM.

[45] Karl Bringmann and Vasileios Nakos. A fine-grained perspective on approximating subset sum and partition. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1797–1815. SIAM, 2021.

[46] Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

[47] Parinya Chalermsook, Bundit Laekhanukit, and Danupon Nanongkai. Independent set, induced matching, and pricing: Connections and tight (subexponential time) approximation hardnesses. In *FOCS*, pages 370–379. IEEE Computer Society, 2013.

[48] Gerard Jennhwa Chang, Paul Dorbec, Hye Kyung Kim, André Raspaud, Haichao Wang, and Weiliang Zhao. Upper k-tuple domination in graphs. *Discrete Mathematics & Theoretical Computer Science*, 14(2):285–292, 2012.

[49] Gary Chartrand, Paul Erdös, and Ortrud Oellermann. How to define an irregular graph. *The College Mathematics Journal*, 19, 01 1988.

[50] Gary Chartrand, Michael Jacobon, Jenö Lehel, Ortrud Oellermann, Sergio Ruiz, and Farrokh Saba. Irregular networks. *Congressus Numerantium*, 64, 01 1986.

[51] Mustapha Chellali, Odile Favaron, Adriana Hansberg, and Lutz Volkmann. k-domination and k-independence in graphs: A survey. *Graphs and Combinatorics*, 28(1):1–55, 2012.

[52] Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):21:1–21:19, 2008.

[53] Ning Chen, Roee Engelberg, C. Thach Nguyen, Prasad Raghavendra, Atri Rudra, and Gyanit Singh. Improved approximation algorithms for the spanning star forest problem. *Algorithmica*, 65(3):498–516, 2013.

[54] Xujin Chen, Xiaodong Hu, and Wenan Zang. A min-max theorem on tournaments. *SIAM J. Comput.*, 37(3):923–937, 2007.

[55] Grant A. Cheston, Gerd Fricke, Stephen T. Hedetniemi, and David Pokrass Jacobs. On the computational complexity of upper fractional domination. *Discrete Applied Mathematics*, 27(3):195–207, 1990.

[56] Miroslav Chlebík and Janka Chlebíková. Complexity of approximating bounded variants of optimization problems. *Theor. Comput. Sci.*, 354(3):320–338, 2006.

[57] Mark Cieliebak and Stephan J. Eidenbenz. Measurement errors make the partial digest problem np-hard. In *LATIN 2004: Theoretical Informatics, 6th Latin American Symposium*, volume 2976 of *Lecture Notes in Computer Science*, pages 379–390, Berlin, Heidelberg, 2004. Springer.

[58] Mark Cieliebak, Stephan J. Eidenbenz, and Aris Pagourtzis. Composing equipotent teams. In *Fundamentals of Computation Theory, 14th International Symposium, FCT 2003*, volume 2751 of *Lecture Notes in Computer Science*, pages 98–108, Berlin, Heidelberg, 2003. Springer.

[59] Mark Cieliebak, Stephan J. Eidenbenz, Aris Pagourtzis, and Konrad Schlude. On the complexity of variations of equal sum subsets. *Nord. J. Comput.*, 14(3):151–172, 2008.

[60] Mark Cieliebak, Stephan J. Eidenbenz, and Paolo Penna. Noisy data make the partial digest problem NP-hard. In *Algorithms in Bioinformatics, Third International Workshop, WABI 2003*, volume 2812 of *Lecture Notes in Computer Science*, pages 111–123, Berlin, Heidelberg, 2003. Springer.

[61] 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. ACM, 1971.

[62] Derek G. Corneil and J. M. Keil. A dynamic programming approach to the dominating set problem on k-trees. *SIAM Journal on Algebraic Discrete Methods*, 8(4):535–543, 1987.

[63] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

[64] Edward K. Crossman and Sharyn M. Crossman. The crossword puzzle as a teaching tool. *Teaching of Psychology*, 10(2):98–99, 1983.

[65] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer International Publishing, 2015.

[66] Marek Cygan, Lukasz Kowalik, and Mateusz Wykurz. Exponential-time approximation of weighted set cover. *Inf. Process. Lett.*, 109(16):957–961, 2009.

[67] Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. On problems equivalent to (min, +)-convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, 2019.

[68] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011.

[69] Marek Cygan and Marcin Pilipczuk. Exact and approximate bandwidth. *Theor. Comput. Sci.*, 411(40-42):3701–3713, 2010.

[70] Peter Damaschke, Haiko Müller, and Dieter Kratsch. Domination in convex and chordal bipartite graphs. *Inf. Process. Lett.*, 36(5):231–236, 1990.

[71] Frank K. H. A. Dehne, Michael R. Fellows, Henning Fernau, Elena Prieto-Rodriguez, and Frances A. Rosamond. NONBLOCKER: parameterized algorithmics for minimum dominating set. In *Proc. of the 32nd SOFSEM*, volume 3831 of *LNCS*, pages 237–245. Springer, 2006.

[72] Marc Demange. A note on the approximation of a minimum-weight maximal independent set. *Computational Optimization and Applications*, 14(1):157–169, 1999.

[73] Marc Demange, Tinaz Ekim, and Cerasela Tanasescu. Hardness and approximation of minimum maximal matchings. *Int. J. Comput. Math.*, 91(8):1635–1654, 2014.

[74] Rodney G. Downey and Michael R. Fellows. Fixed parameter tractability and completeness. pages 191–225. Cambridge University Press, 1992.

[75] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.

[76] Pranjal Dutta and Mahesh Sreekumar Rajasree. Algebraic algorithms for variants of subset sum. In *Algorithms and Discrete Applied Mathematics - 8th International Conference, CALDAM 2022*, volume 13179 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2022.

[77] Jakob Engel, Markus Holzer, Oliver Ruepp, and Frank Sehnke. On computer integrated rationalized crossword puzzle manufacturing. In *Fun with Algorithms - 6th International Conference, FUN 2012, Venice, Italy, June 4-6, 2012. Proceedings*, pages 131–141, 2012.

[78] Bruno Escoffier, Vangelis Th. Paschos, and Emeric Tourniaire. Approximating MAX SAT by moderately exponential and parameterized algorithms. *Theor. Comput. Sci.*, 560:147–157, 2014.

[79] Hiroshi Eto, Tesshu Hanaka, Yasuaki Kobayashi, and Yusuke Kobayashi. Parameterized Algorithms for Maximum Cut with Connectivity Constraints. In Bart M. P. Jansen and Jan Arne Telle, editors, *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*, volume 148 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[80] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976.

[81] M. Farber. Domination, independent domination and duality in strongly chordal graphs. *Discrete Appl. Math.*, 7:115–130, 1984.

[82] Tomás Feder, Pavol Hell, and Carlos S. Subi. Complexity of acyclic colorings of graphs and digraphs with degree and girth constraints. *CoRR*, abs/1907.00061, 2019.

[83] Henning Fernau and David F. Manlove. Vertex and edge covers with clustering properties: Complexity and algorithms. *Journal of Discrete Algorithms*, 7:149–167, 2009.

[84] Henning Fernau and Daniel Meister. Digraphs of bounded elimination width. *Discret. Appl. Math.*, 168:78–87, 2014.

[85] J. F. Fink and M. S. Jacobson. Graph theory with applications to algorithms and computer science. chapter n-Domination in Graphs, pages 283–300. John Wiley & Sons, Inc., New York, NY, USA, 1985.

[86] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010.

[87] 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.

[88] Dimitris Fotakis, Michael Lampis, and Vangelis Th. Paschos. Sub-exponential approximation schemes for csps: From dense to almost sparse. In *STACS*, volume 47 of *LIPIcs*, pages 37:1–37:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[89] Alan M. Frieze, Ronald J. Gould, Michal Karonski, and Florian Pfender. On graph irregularity strength. *Journal of Graph Theory*, 41(2):120–137, 2002.

[90] Fabio Furini, Ivana Ljubic, and Markus Sinnl. An effective dynamic programming algorithm for the minimum-cost maximal knapsack packing problem. *European Journal of Operational Research*, 262(2):438–448, 2017.

[91] Andrei Gagarin and Vadim E. Zverovich. A generalised upper bound for the k-tuple domination number. *Discrete Mathematics*, 308(5-6):880–885, 2008.

[92] Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In *IPEC*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013.

[93] Robert Ganian, Petr Hlinený, Joachim Kneis, Daniel Meister, Jan Obdrzálek, Peter Rossmanith, and Somnath Sikdar. Are there any good digraph width measures? *J. Comb. Theory, Ser. B*, 116:250–286, 2016.

[94] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[95] George Gens and Eugene Levner. Computational complexity of approximation algorithms for combinatorial problems. In *Mathematical Foundations of Computer Science 1979, Proceedings, 8th Symposium, Olomouc, Czechoslovakia, September 3-7, 1979*, volume 74 of *Lecture Notes in Computer Science*, pages 292–300. Springer, 1979.

[96] Matthew L. Ginsberg, Michael Frank, Michael P. Halpin, and Mark C. Torrance. Search lessons learned from crossword puzzles. In *Proceedings of the 8th National Conference on Artificial Intelligence. Boston, Massachusetts, USA, July 29 - August 3, 1990, 2 Volumes*, pages 210–215, 1990.

[97] Laurent Gourvès, Jérôme Monnot, and Aris Pagourtzis. The lazy bureaucrat problem with common arrivals and deadlines: Approximation and mechanism design. In *FCT*, volume 8070 of *Lecture Notes in Computer Science*, pages 171–182. Springer, 2013.

[98] Laurent Gourvès, Jérôme Monnot, and Lydia Tlilane. Subset sum problems with digraph constraints. *J. Comb. Optim.*, 36(3):937–964, 2018.

[99] Frank Gurski, Dominique Komander, and Carolin Rehs. Acyclic coloring of special digraphs. *CoRR*, abs/2006.13911, 2020.

[100] Tesshu Hanaka, Hans L. Bodlaender, Tom C. van der Zanden, and Hirotaka Ono. On the maximum weight minimal separator. *Theoretical Computer Science*, 796:294 – 308, 2019.

[101] Frank Harary and Teresa W. Haynes. Double domination in graphs. *Ars Comb.*, 55, 2000.

[102] Ararat Harutyunyan. Brooks-type results for coloring of digraphs. *PhD Thesis*, Simon Fraser University, 2011.

[103] Ararat Harutyunyan, Mehdi Khosravian Ghadikolaei, Nikolaos Melissinos, Jérôme Monnot, and Aris Pagourtzis. On the complexity of the upper r-tolerant edge cover problem. In Luís Soares Barbosa and Mohammad Ali Abam, editors, *Topics in Theoretical Computer Science - Third IFIP WG 1.8 International Conference, TTCS 2020, Tehran, Iran, July 1-2, 2020, Proceedings*, volume 12281 of *Lecture Notes in Computer Science*, pages 32–47. Springer, 2020.

[104] Ararat Harutyunyan, Mark Kayll, Bojan Mohar, and Liam Rafferty. Uniquely d-colorable digraphs with large girth. *Canad. J. Math.*, 64(6):1310–1328, 2012.

[105] Ararat Harutyunyan, Tien-Nam Le, Stéphan Thomassé, and Hehui Wu. Coloring tournaments: From local to global. *J. Comb. Theory, Ser. B*, 138:166–171, 2019.

[106] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Math*, 182:105–142, 1999.

[107] Teresa W. Haynes, Stephen T. Hedetniemi, and Peter J. Slater. *Fundamentals of domination in graphs*, volume 208 of *Pure and applied mathematics*. Dekker, 1998.

[108] Michael A. Henning and Dinabandhu Pradhan. Algorithmic aspects of upper paired-domination in graphs. *Theor. Comput. Sci.*, 804:98–114, 2020.

[109] Michael A. Henning and Anders Yeo. On upper transversals in 3-uniform hypergraphs. *Electron. J. Comb.*, 25(4):P4.27, 2018.

[110] Michael A. Henning and Anders Yeo. Upper transversals in hypergraphs. *Eur. J. Comb.*, 78:1–12, 2019.

[111] Michael A. Henning and Anders Yeo. Bounds on upper transversals in hypergraphs. *J. Comb. Optim.*, 39(1):77–89, 2020.

[112] Winfried Hochstättler, Felix Schröder, and Raphael Steiner. On the complexity of digraph colourings and vertex arboricity. *Discret. Math. Theor. Comput. Sci.*, 22(1), 2020.

[113] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[114] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, 1974.

[115] Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *J. ACM*, 23(2):317–327, 1976.

[116] Joseph Douglas Horton and Kyriakos Kilakos. Minimum edge dominating sets. *SIAM J. Discrete Math.*, 6(3):375–387, 1993.

[117] Heather Hulett, Todd G. Will, and Gerhard J. Woeginger. Multigraph realizations of degree sequences: Maximization is easy, minimization is hard. *Oper. Res. Lett.*, 36(5):594–596, 2008.

[118] Paul Hunter and Stephan Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theor. Comput. Sci.*, 399(3):206–219, 2008.

[119] Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, 1975.

[120] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[121] Ken Iwaide and Hiroshi Nagamochi. An improved algorithm for parameterized edge dominating set problem. *J. Graph Algorithms Appl.*, 20(1):23–58, 2016.

[122] Michael S. Jacobson and Kenneth Peters. Chordal graphs and upper irredundance, upper domination and independence. *Discret. Math.*, 86(1-3):59–69, 1990.

[123] Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. In *CIAC*, volume 10236 of *Lecture Notes in Computer Science*, pages 345–356, 2017.

[124] Ce Jin and Hongxun Wu. A Simple Near-Linear Pseudopolynomial Time Randomized Algorithm for Subset Sum. In *2nd Symposium on Simplicity in Algorithms (SOSA 2019)*, volume 69, pages 17:1–17:6, 2018.

[125] Thor Johnson, Neil Robertson, Paul D. Seymour, and Robin Thomas. Directed tree-width. *J. Comb. Theory, Ser. B*, 82(1):138–154, 2001.

[126] Michał Karoński, Tomasz Łuczak, and Andrew Thomason. Edge weights and vertex colors. *Journal of Combinatorial Theory*, 91:151–157, 05 2004.

[127] Alexander V. Karzanov. Maximum matching of given weight in complete and complete bipartite graphs. *Kibernetika*, 23(1):7–11, (1987) (English translation in CYBNAW23(1), 8–13 (1987)).

[128] Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Improved (in-) approximability bounds for d-scattered set. In *WAOA*, volume 11926 of *Lecture Notes in Computer Science*, pages 202–216. Springer, 2019.

[129] Hans Kellerer, Renata Mansini, Ulrich Pferschy, and Maria Grazia Speranza. An efficient fully polynomial approximation scheme for the subset-sum problem. *J. Comput. Syst. Sci.*, 66(2):349–370, 2003.

[130] Kaveh Khoshkhah, Mehdi Khosravian Ghadikolaei, Jérôme Monnot, and Florian Sikora. Weighted upper edge cover: Complexity and approximability. *J. Graph Algorithms Appl.*, 24(2):65–88, 2020.

[131] Subhash Khot and Venkatesh Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289(2):997–1008, 2002.

[132] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

[133] Ralf Klasing and Christian Laforest. Hardness results and approximation algorithms of k-tuple domination in graphs. *Inf. Process. Lett.*, 89(2):75–83, 2004.

[134] Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Trans. Algorithms*, 15(3):40:1–40:20, 2019.

[135] Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 184–192. IEEE, 2021.

[136] Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012.

[137] Michael Lampis. Finer tight bounds for coloring on clique-width. In *ICALP*, volume 107 of *LIPIcs*, pages 86:1–86:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[138] Michael Lampis, Georgia Kaouri, and Valia Mitsou. On the algorithmic effectiveness of digraph decompositions and complexity measures. *Discret. Optim.*, 8(1):129–138, 2011.

[139] Michael Lampis, Valia Mitsou, and Karolina Soltys. Scrabble is PSPACE-complete. *J. Inf. Process.*, 23(3):284–292, 2015.

[140] John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

[141] Zhentao Li and Bojan Mohar. Planar digraphs of digirth four are 2-colorable. *SIAM J. Discret. Math.*, 31(3):2201–2205, 2017.

[142] Chung-Shou Liao and Gerard J. Chang. k-tuple domination in graphs. *Inf. Process. Lett.*, 87(1):45–50, 2003.

[143] Carla Negri Lintzmayer, Guilherme Oliveira Mota, and Maycon Sambinelli. Decomposing split graphs into locally irregular graphs. *Discrete Applied Mathematics*, 292:33–44, 2021.

[144] Richard J. Lipton, Evangelos Markakis, Elchanan Mossel, and Amin Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the 5th ACM Conference on Electronic Commerce (EC-2004)*, pages 125–131, New York, NY, USA, 2004. ACM.

[145] Michael L. Littman, Greg A. Keim, and Noam M. Shazeer. Solving crosswords with PROVERB. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA*, pages 914–915, 1999.

[146] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the European Association for Theoretical Computer Science*, 105:41–72, 2011.

[147] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018.

[148] Télé Magazine. Publications Grand Public.

[149] David F. Manlove. On the algorithmic complexity of twelve covering and independence parameters of graphs. *Discrete Applied Mathematics*, 91(1-3):155–175, 1999.

[150] Gary Meehan and Peter Gray. Constructing crossword grids: Use of heuristics vs constraints. In *In: Proceedings of Expert Systems 97: Research and Development in Expert Systems XIV, SGES*, pages 159–174, 1997.

[151] Nikolaos Melissinos and Aris Pagourtzis. A faster FPTAS for the subset-sums ratio problem. In *Computing and Combinatorics - 24th International Conference, COCOON 2018*, volume 10976 of *Lecture Notes in Computer Science*, pages 602–614, Cham, 2018. Springer.

[152] Silvio Micali and Vijay V. Vazirani. An o(sqrt(|v|) |e|) algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27. IEEE Computer Society, 1980.

[153] Marcelo Garlet Millani, Raphael Steiner, and Sebastian Wiederrecht. Colouring non-even digraphs. *CoRR*, abs/1903.02872, 2019.

[154] Sounaka Mishra and Kripasindhu Sikdar. On the hardness of approximating some NP-optimization problems related to minimum linear ordering problem. *RAIRO Theor. Informatics Appl.*, 35(3):287–309, 2001.

[155] Pranabendu Misra, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Parameterized algorithms for even cycle transversal. In Martin Charles Golumbic, Michal Stern, Avivit Levy, and Gila Morgenstern, editors, *Graph-Theoretic Concepts in Computer Science - 38th International Workshop, WG 2012, Jerusalem, Israel, June 26-28, 2012, Revised Selcted Papers*, volume 7551 of *Lecture Notes in Computer Science*, pages 172–183. Springer, 2012.

[156] Bojan Mohar. Face covers and the genus problem for apex graphs. *J. Comb. Theory, Ser. B*, 82(1):102–117, 2001.

[157] Bojan Mohar. Circular colorings of edge-weighted graphs. *Journal of Graph Theory*, 43(2):107–116, 2003.

[158] Bojan Mohar. Eigenvalues and colorings of digraphs. *Linear Algebra and its Applications*, 432(9):2273 – 2277, 2010. Special Issue devoted to Selected Papers presented at the Workshop on Spectral Graph Theory with Applications on Computer Science, Combinatorial Optimization and Chemistry (Rio de Janeiro, 2008).

[159] Hannes Moser and Dimitrios M. Thilikos. Parameterized complexity of finding regular induced subgraphs. *Journal of Discrete Algorithms*, 7(2):181–190, 2009.

[160] Marcin Mucha, Jesper Nederlof, Jakub Pawlewicz, and Karol Wegrzycki. Equal-subset-sum faster than the meet-in-the-middle. In *27th Annual European Symposium on Algorithms, ESA 2019*, volume 144 of *LIPIcs*, pages 73:1–73:16, 2019.

[161] Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. A subquadratic approximation scheme for partition. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 70–88. SIAM, 2019.

[162] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Comb.*, 7(1):105–113, 1987.

[163] Danupon Nanongkai. Simple FPTAS for the subset-sums ratio problem. *Inf. Process. Lett.*, 113(19-21):750–753, 2013.

[164] Victor Neumann-Lara. The dichromatic number of a digraph. *J. Comb. Theory, Ser. B*, 33(3):265–270, 1982.

[165] C. Thach Nguyen, Jian Shen, Minmei Hou, Li Sheng, Webb Miller, and Louxin Zhang. Approximating the spanning star forest problem and its application to genomic sequence alignment. *SIAM J. Comput.*, 38(3):946–962, 2008.

[166] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.

[167] Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *J. ACM*, 29(2):285–309, 1982.

[168] Jagan A. Pillai, Charles B. Hall, Dennis W. Dickson, Herman Buschke, Richard B. Lipton, and Joe Verghese. Association of crossword puzzle participation with memory decline in persons who develop dementia. *Journal of the International Neuropsychological Society*, 17(6):1006–1013, 2011.

[169] Kirk Pruhs and Gerhard J. Woeginger. Approximation schemes for a class of subset selection problems. *Theor. Comput. Sci.*, 382(2):151–156, 2007.

[170] Jakub Przybyło. Irregularity strength of regular graphs. *Electronic Journal of Combinatorics*, 15, 06 2008.

[171] Jakub Przybyło. On decomposing graphs of large minimum degree into locally irregular subgraphs. *Electronic Journal of Combinatorics*, 23(2):2–31, 2016.

[172] Leonardo Rigutini, Michelangelo Diligenti, Marco Maggini, and Marco Gori. Automatic generation of crossword puzzles. *Int. J. Artif. Intell. Tools*, 21(3), 2012.

[173] Christopher D. Rosin. Nested rollout policy adaptation for Monte Carlo tree search. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 649–654, 2011.

[174] Sartaj Sahni. Algorithms for scheduling independent tasks. *J. ACM*, 23(1):116–127, 1976.

[175] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Springer-Verlag, 2003.

[176] P. J. Slater. Enclaveless sets and MK-systems. *J. Res. Nat. Bur. Stand.*, 82(3):197–202, 1977.

[177] Raphael Steiner and Sebastian Wiederrecht. Parameterized algorithms for directed modular width. In *CALDAM*, volume 12016 of *Lecture Notes in Computer Science*, pages 415–426. Springer, 2020.

[178] Shuichi Ueno, Yoji Kajitani, and Shin'ya Gotoh. On the nonseparating independent set problem and feedback set problem for graphs with no vertex degree exceeding three. *Discrete Mathematics*, 72(1):355 – 360, 1988.

[179] Nadav Voloch. Mssp for 2-d sets with unknown parameters and a cryptographic application. *Contemporary Engineering Sciences*, 10:921–931, 10 2017.

[180] Gerhard J. Woeginger. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (fptas)? *INFORMS J. Comput.*, 12(1):57–74, 2000.

[181] Gerhard J. Woeginger and Zhongliang Yu. On the equal-subset-sum problem. *Inf. Process. Lett.*, 42(6):299–302, 1992.

[182] Meirav Zehavi. Maximum minimal vertex cover parameterized by vertex cover. *SIAM Journal on Discrete Mathematics*, 31(4):2440–2456, 2017.

[183] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007.

[184] Igor E. Zvervich and Vadim E. Zverovich. An induced subgraph characterization of domination perfect graphs. *Journal of Graph Theory*, 20(3):375–395, 1995.

Appendix A

Missing Proofs

Proof of Theorem 3.11

Proof. Before we start the proof we have to remark two things. The first is that generally, it's not necessary that $S_1 \subseteq \{1, \ldots, n\}$ and $S_2 \subseteq \{n+1, \ldots, 2 \cdot n\}$ but they may be reversed, so when we mention a feasible solution, by convention we regard that $m \in S_1$. Secondly, in the case-algorithms inside of Algorithm 4 we construct two variables p and p' such that:

- If $m \in \{1, \ldots, n\}$ then $(p, p') = (0, n)$
- If $m \in \{n+1, \ldots, 2 \cdot n\}$ then $(p, p') = (n, 0)$

With the use of these variables we prove some properties for the indices of the feasible solutions.

Lemma A.1. *Let (S_1, S_2) be a feasible solution of the problem with input $(\{a_1, \ldots, a_{2 \cdot n}\}, m)$, the set $S = \{1, \ldots, n\}$ and (p, p') the variables we defined above, then:*

- $S_1 \subseteq \{i + p \mid i \in S\}$ (A.1)
- $S_2 \subseteq \{i + p' \mid i \in S\}$ (A.2)
- *for an index $j \in S_1$ then $j - p + p' \notin S_2$* (A.3)
- *for an index $j \in S_2$ then $j - p' + p \notin S_1$* (A.4)

Proof. Since (S_1, S_2) are in such order so $m \in S_1$ and, one of S_1, S_2 sets is subset of $\{1, \ldots, n\}$ while the other is subset of $\{n+1, \ldots, 2 \cdot n\}$, we have that
• If $m \in \{1, \ldots, n\}$ then $S_1 \subseteq \{1, \ldots, n\}$, $S_2 \subseteq \{n+1, \ldots, 2 \cdot n\}$ and $(p, p') = (0, n)$
• If $m \in \{n+1, \ldots, 2 \cdot n\}$ then $S_2 \subseteq \{1, \ldots, n\}$, $S_1 \subseteq \{n+1, \ldots, 2 \cdot n\}$ and $(p, p') = (n, 0)$
Without loss of generality we assume that $m \leq n$, this means that $p = 0$,

$$S_1 \subseteq \{1, \ldots, n\} = \{i + p \mid i \in S\}$$

and because $p' = n$

$$S_2 \subseteq \{n+1, \ldots, 2 \cdot n\} = \{i + p' \mid i \in S\} \ .$$

It remains to prove the relations between the indices of the two sets. By the definition of the problem for any feasible solution $(S_1, S_2) \nexists (i, j)$ such that $i \in S_1$, $j \in S_2$ and $i \equiv j \ (mod\ n)$. Because $p, p' \in \{0, n\}$ we have that $j \equiv j - p + p' \equiv j - p' + p \ (mod\ n)$ so both of the last two properties holds. □

Lemma A.2. *Let (S_1, S_2) be a feasible solution for the problem with input $(\{a_1, \ldots, a_{2\cdot n}\}, m)$ and S_{min}, S_{max} the sets as they are defined in Algorithm 4, then the following are true:*

$$\exists\ i \in S_{max} \text{ such that } i + p' \in S_2 \tag{A.5}$$

$$S_1 \subseteq \{i + p \mid i \in S_{min}\} \cup \{m\} \tag{A.6}$$

Proof. We start with the definitions of S_{min} and S_{max}.

$$S_{min} = \{i \mid i \in \{1, \ldots, n\} \text{ and } a_{i+p} \leq a_m\} \setminus \{m - p\}$$
$$S_{max} = \{i \mid i \in \{1, \ldots, n\} \text{ and } a_{i+p'} > a_m\} \setminus \{m - p + p'\}\ .$$

For any feasible solutions of the semi restricted version of 2-Set SSR we know that $a_m = \max_{i \in S_1} a_i$ and $a_m \leq \max_{i \in S_2} a_i$. Let $j \in S_2$ such that $a_j \geq a_m$ then by relation A.2, we have that $\exists\ i \in \{1, \ldots, n\}$ such that $i + p' = j$, $a_{i+p'} \geq a_m$ and $i \neq m - p + p'$ so $i \in S_{max}$ (by its definition). So relation A.5 holds. Now by considering the relation A.1, if $j \in S_1 \setminus \{m\}$ then $\exists\ i \in \{1, \ldots, n\}$ such that $i + p = j$, $a_{i+p} \leq a_m$ and $j \neq m$ which means that $i \in S_{min}$ so the relation A.6 holds. Thus the lemma holds. □

Now, let (S_1^*, S_2^*) be an optimal solution for the semi restricted version of 2-Set SSR with input $A = \{a_1, \ldots a_{2\cdot n}\}$ and m. Without lost of generality let

$$\max_{i \in S_1^*}\{a_i\} = a_{m_1} = a_m < a_{m_2} = \max_{j \in S_2^*}\{a_j\}$$

this means that the sets appear in the same order as if they were constructed from the algorithm. For this optimal solution we have two cases, either $a_{m_2} > Q$ or $a_{m_2} \leq Q$ (where $Q \leftarrow a_m + \sum_{i \in S_{min}} a_{i+p}$ as it is defined in case-algorithms of Algorithm 4).

Case 1 $(a_{m_2} > Q)$**.** In this case we return a solution with ratio equal to the optimal using Algorithm 5. By relation A.2 we know that there exists $m_0 \in \{1, \ldots, n\}$ such that the index $m_0 + p' = m_2$. With the additional knowledge that $a_{m_2} > Q$ we have that $m_0 \in S_{max}$. We claim that the pair of sets $(S, \{m_2\})$, where $S = \{i + p \mid i \in S_{min} \text{ or } i = m - p\} \setminus \{m_0 + p\}$ is an optimal solution for this case. In order to prove this claim we need to observe that for any feasible solution (S_1, S_2) for this case we have $S_1 \subseteq (\{i + p \mid i \in S_{min}\} \cup \{m\}) \setminus \{m_2 - p' + p\}$ and $S_2 \supseteq \{m_2\}$. So for any feasible solution for this case we have:

$$\begin{aligned}\mathcal{MR}(S_1, S_2, A) = \mathcal{R}(S_2, S_1, A) &\geq \mathcal{R}(\{m_2\}, S_1, A) \\ &\geq \mathcal{R}(\{m_2\}, (\{i + p \mid i \in S_{min}\} \cup \{m\}) \setminus \{m_2 - p' + p\}, A)\end{aligned}$$

which proves the claim. Due to this fact, Algorithm 5 returns $(S, \{m_2\})$ or a pair of sets with the same max ratio.

Case 2 $(a_{m_2} < Q)$**.** The first thing we have to prove in this case is the following lemma,

Lemma A.3. *If (S_1', S_2') is a feasible solution for this case and (S_1, S_2) is a pair of sets such that $S_1 \subseteq S_1'$ and $S_2 \subseteq S_2'$ then:*

$$-2 \cdot Q \leq \sum_{i \in S_1} a_i - \sum_{j \in S_2} a_j \leq Q$$

Proof. By relation A.6 it is obvious that $\sum_{i \in S_1} a_i \leq Q$ so we need prove that

$$\sum_{j \in S_2} a_j \leq 2 \cdot Q \ .$$

Let's assume that $\sum_{j \in S_2} a_j > 2 \cdot Q$ then because $Q \geq a_{m_2} = \max_{i \in S_2'}\{a_i\}$ and $S_2 \subseteq S_2'$, we have that S_2 should contain at least 3 indices and the same holds for the set S_2'. Let $m_0 \neq m_2$ be one of them, then because $Q \geq a_{m_0}$ and $\sum_{j \in S_2'} a_j \geq \sum_{j \in S_2} a_j > 2 \cdot Q$ we have that

$$\sum_{j \in S_2'} a_j > \sum_{j \in S_2' \setminus \{m_0\}} a_j > Q$$

and because $\sum_{i \in S_1'} a_i \leq Q$

$$1 \leq \mathcal{MR}(S_1', S_2' \setminus \{m_0\}, A) \leq \mathcal{MR}(S_1', S_2', A)$$

which is a contradiction. □

The next lemma shows that, for any feasible solution (S_1, S_2), the cell $T[n, d, 1]$, where $d = \sum_{i \in S_1} a_i - \sum_{i \in S_2} a_i$, is non empty. Furthermore the sets which are stored in this cell have max ratio at most $\mathcal{MR}(S_1, S_2, A)$.

Lemma A.4. *Let (S_1^*, S_2^*) is a feasible solution for this case and (S_1, S_2) is a pair of sets such that $S_1 \subseteq S_1^*$, $S_2 \subseteq S_2^*$ and $m \in S_1$. We define $m_0 = \max\{0, i \mid i + p \in S_1 \setminus \{m\}$ or $i + p' \in S_2\}$ and $d = \sum_{i \in S_1} a_i - \sum_{i \in S_2} a_i$.*

If $S_2 \cap \{i + p' \mid i \in S_{max}\} = \emptyset$	*then the cell* $T[m_0, d, 0] \neq (\emptyset, \emptyset, 0)$
else	*then the cell* $T[m_0, d, 1] \neq (\emptyset, \emptyset, 0)$

Proof. For any pair (S_1, S_2) (as these described at the lemma) we define the following set

$$S_{S_1,S_2} = \{0\} \cup \{i \bmod n \mid i + p \in S_1 \text{ or } i + p' \in S_2)\} \setminus \{m - p\} \ .$$

Note that $i \in \{1, \ldots, n\}$ if $i + p \in S_1$ (by relation A.1) and the same holds if $i + p' \in S_2$. Because of this, we know that $S_{S_1,S_2} \subseteq \{0, \ldots, n\} \setminus \{m - p\}$. We prove this lemma by using strong induction to the maximum element of the set S.

• If $\max\{S_{S_1,S_2}\} = 0$ (base case)

Because we have requested $m \in S_1$ and by the fact that $\max\{S_{S_1,S_2}\} = 0$ we can conclude that $S_2 \cap \{i + p' \mid i \in S_{max}\} = \emptyset$ and $(S_1, S_2) = (\{m\}, \emptyset)$, which is the pair of sets the algorithm stores in the cell $T[0, a_m, 0]$. This concludes the base case.

• Assuming that lemma's statement holds for all the indices k' which are smaller or

equal than a specific index $k < n$, we now prove it for $k+1$.
• Let $k+1 = \max\{S_{S_1,S_2}\}$ this means that $k+1+p \neq m$ and either $k+1+p \in S_1$ or $k+1+p' \in S_2$. So we have to check both cases.
Case A $(k+1+p \in S_1)$**.** In this case for the pair of sets $(S_1', S_2) = (S_1 \setminus \{k+1+p\}, S_2)$ meets the conditions of the induction because $\max\{S_{S_1',S_2}\} < \max\{S_{S_1,S_2} \setminus \{k+1\}\}$. So for $d = \sum_{i \in S_1'} a_i - \sum_{i \in S_2} a_i$ we know that:

$$\begin{array}{ll} \text{either the cell} & T[\max\{S_{S_1',S_2}\}, d, 0] \neq (\emptyset, \emptyset, 0) \text{ if } S_2 \cap \{i+p' \mid i \in S_{max}\} = \emptyset \\ \text{or the cell} & T[\max\{S_{S_1',S_2}\}, d, 1] \neq (\emptyset, \emptyset, 0) \text{ if } S_2 \cap \{i+p' \mid i \in S_{max}\} \neq \emptyset \end{array}$$

(respectively)

Algorithm 6 moves up all the cells (line 13). This means that the cell $T[k, d, 0]$ (resp. $T[k, d, 1]$) is non equal to $(\emptyset, \emptyset, 0)$ in the case $S_2 \cap S_{max} = \emptyset$ (resp. $S_2 \cap S_{max} \neq \emptyset$). Because $k+1 \in S_{min}$ (by relation A.6 and $k+1+p \in S_1$) then Algorithm 6 fills the cell $T[k+1, d+a_{k+1+p}, 0]$ (resp. $T[k+1, d+a_{k+1+p}, 1]$) in line 16. This proves the case A.
Case B $(k+1+p' \in S_2)$**.** Here we have two extra cases, either $k+1 \in S_{max}$ or not.
Case B.1 $(k+1 \notin S_{max})$**.** Like in the previous case, the pair of sets $(S_1, S_2') = (S_1, S_2 \setminus \{k+1+p'\})$ meets the conditions of the induction because $\max\{S_{S_1,S_2'}\} < \max\{S_{S_1,S_2} \setminus \{k+1\}\}$. So for $d = \sum_{i \in S_1} a_i - \sum_{i \in S_2'} a_i$ we know that:

$$\begin{array}{ll} \text{either the cell} & T[\max\{S_{S_1,S_2'}\}, d, 0] \neq (\emptyset, \emptyset, 0) \text{ if } S_2' \cap \{i+p' \mid i \in S_{max}\} = \emptyset \\ \text{or the cell} & T[\max\{S_{S_1,S_2'}\}, d, 1] \neq (\emptyset, \emptyset, 0) \text{ if } S_2' \cap \{i+p' \mid i \in S_{max}\} \neq \emptyset \end{array}$$

(respectively)

As we have said before, Algorithm 6 moves up all the cells (line 13). This means that the cell $T[k, d, 0]$ (resp. $T[k, d, 1]$) is non equal to $(\emptyset, \emptyset, 0)$ in the case of $S_2' \cap \{i+p' \mid i \in S_{max}\} = \emptyset$ (resp. $S_2' \cap \{i+p' \mid i \in S_{max}\} \neq \emptyset$) and because we know that $d - a_{k+1+p'} \geq -2 \cdot Q$ (by lemma A.3) in the line 22 the algorithm fills the wanted cell $T[k, d-a_{k+1+p'}, 0]$ (resp. $T[k, d-a_{k+1+p'}, 1]$).
Case B.2 $(k+1 \in S_{max})$**.** This case is similar to Case B.1 with the exception that the algorithm ensures that the algorithm fills the cell $T[k, d-a_{k+1+p'}, 1]$ in line 20 if either of the $T[k-1, d, 0]$ or $T[k-1, d, 1]$ are non-empty. □

To complete the proof we have to observe three things.
First, in any cell Algorithm 6 keeps the pair of sets with the greater total sum.
Second, For all the pairs of sets (S_1, S_2) which are stored we have $S_1 \subseteq \{i+p \mid i \in S_{min}\}$, $S_2 \subseteq \{i+p' \mid i \in \{1, \ldots, n\}\}$ and $\nexists(j, j')$ such that $j \in S_1$, $j' \in S_2$ and $j \equiv j' \ (mod\ n)$ (because we use only $i+p$ in S_1 or $i+p'$ in S_2 every time). Third, in order to store a pair of sets in any cell $T[i, d, 1]$ we have either add a $i+p'$ with $i \in S_{max}$ to the S_2 or use an other cell $T[i', d', 1]$ (which already have such an index in S_2) so these pairs are feasible solutions for this case.
With all that in mind we know that, if (S_1^*, S_2^*) be an optimal solution for the problem then for $d = \sum_{i \in S_1^*} a_i - \sum_{i \in S_2^*} a_i$ ($d \in \{-2 \cdot Q, \ldots, Q\}$ by the lemma A.3) a cell $T[i, d, 1] \neq$

$(\emptyset, \emptyset, 0)$ due to lemma A.4 and relation A.5 so the same holds for the $T[n, d, 1]$ (because the algorithm moves up all the cells). Let (S_1, S_2) pair of sets stored in that specific cell. As we mentioned earlier the pair (S_1, S_2) is a feasible solution such that:

$$\sum_{i \in S_1} a_i - \sum_{i \in S_2} a_i = \sum_{i \in S_1^*} a_i - \sum_{i \in S_2^*} a_i$$

and

$$\sum_{i \in S_1} a_i + \sum_{i \in S_2} a_i \geq \sum_{i \in S_1^*} a_i + \sum_{i \in S_2^*} a_i$$

Since the differences of the sums of the pairs are the same, it is easy to see that the pair with the smaller max ratio is the one with the greater total sum. Furthermore, we can not have smaller max ratio than the optimal so the stored pair is an optimal one. Thus the Algorithm 4 returns an optimal solution in both of cases. □

Proof of Lemma 5.1

Proof. Let S be a minimal fvs of $G-u$. We observe that $S \cup \{u\}$ is an fvs of G. If $S \cup \{u\}$ is minimal, we are done. If not, we delete vertices from it until it becomes minimal. We now note that the only vertex which may be deleted in this process is u, since all vertices of S have a private cycle in $G - u$ (that is, a cycle not intersected by any other vertex of S). Hence, the resulting set is a superset of S. □

Proof of Lemma 5.2

Proof. Before we prove the Lemma we note that the contraction operation, under the condition that $N(u) \cap N(v) = \emptyset$, preserves acyclicity in a strong sense: G is acyclic if and only if G/uv is acyclic. Indeed, if we contract an edge that is part of a cycle, this cycle must have length at least 4, and will therefore give a cycle in G/uv. Of course, contractions never create cycles in acyclic graphs.

Let $G' := G/uv$, w be the vertex of G' which has replaced u, v, $V' = V(G')$, and S be a minimal fvs of G'. We have two cases: $w \in S$ or $w \notin S$.

In case $w \in S$, we start with the set $S' = (S \setminus \{w\}) \cup \{u, v\}$. It is not hard to see that S' is an fvs of G. Furthermore, no vertex of $S' \setminus \{u, v\}$ is redundant: for all $z \in S \setminus \{w\}$, there is a cycle in $G'[(V' \setminus S) \cup \{z\}]$, therefore there is also a cycle in $G[(V \setminus S') \cup \{z\}]$. Furthermore, we claim that $S' \setminus \{u, v\}$ is not a valid fvs. Indeed, there must be a cycle contained (due to minimality) in $G_1 = G'[(V' \setminus S) \cup \{w\}]$. Therefore, if there is no cycle in $G_2 = G[(V \setminus S') \cup \{u, v\}]$, we get a contradiction, as G_1 can be obtained by G_2 by contracting the edge uv and contracting edges preserves acyclicity. We conclude that even if S' is not minimal, if we remove vertices until it becomes minimal, we will remove at most one vertex, so the size of the fvs obtained is at least $|S|$.

In case $w \notin S$, we will return the same set S. Let $F = V \setminus S, F' = V' \setminus S$. By definition, $G'[F']$ is acyclic. To see that $G[F]$ is also a forest, we note that $G'[F']$ is obtained from

$G[F]$ by contracting uv, and as we noted in the beginning, the contractions we use strongly preserve acyclicity. To see that S is minimal, take $z \in S$ and consider the graphs $G_1 = G[(V \setminus S) \cup \{z\}]$ and $G_2 = G'[(V' \setminus S) \cup \{z\}]$. We see that G_2 can be obtained from G_1 by contracting uv. But G_2 must have a cycle, by the minimality of S, so G_1 also has a cycle. Thus, S' is minimal in G. □

Proof of Theorem 6.1

Proof. We perform a reduction from NAE-3-SAT, a variant of 3-SAT where we are asked to find an assignment that sets at least one literal to True and one to False in each clause. First we remark that this problem remains NP-hard if all literals appear at most twice.

To see this, suppose that x appears $\ell \geq 4$ times in ϕ. We replace each appearance of x with a fresh variable x_i, $i \in [\ell]$ and add to the formula the clauses $(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \dots (\neg x_\ell \vee x_1)$. Repeating this for all variables that appear at least 4 times produces an equivalent instance ϕ' with $O(n + m)$ variables and clauses such that all literals appear at most 2 times. Furthermore, any satisfying assignment the formula forces exactly one true and one false literal in the new clauses.

We construct a digraph as follows: for each variable x_i we make a digon and label its vertices $x_i, \neg x_i$. We call this part of the digraph the assignment part. For each clause we make a directed cycle of size equal to the clause and associate each vertex of the cycle with a literal. We call this part the satisfaction part. Finally, for each vertex of the assignment part we connect it with digons with each vertex of the satisfaction part that represents the opposite literal.

The digraph we constructed has maximum degree 6; indeed, each literal has degree two in the assignment part and since each literal appears in at most two clauses, it has degree at most 4 in the satisfaction part. If there is a satisfying assignment then we give color 1 to all True literals of both parts and color 2 to everything else. Observe that all arcs connecting the two parts are bichromatic and if the assignment is satisfying all directed cycles are also bichromatic. For the converse direction, if there is a 2-coloring we can extract an assignment by setting to True all literals which have color 1 in the assignment part. Note that this implies that in the satisfaction part all literals which have color 1 have been set to True and all literals which have color 2 have been set to False, because of the digons connecting the two parts. But this implies that our assignment is satisfying because all cycles are bichromatic. □ □

Proof of the properties in Theorem 6.8

Property 1. Once we delete $\{v^i \mid i \in [k]\}$ we observe that for every vertex $v^i_{\ell,in}$ its only outgoing arcs are to vertices $v^j_{\ell,in}$ for $j < i$ or vertices $v^j_{\ell-1,in}$ for $j \geq i$. This shows that we have eliminated all directed paths from $v^i_{\ell,in}$ to $v^j_{\ell',out}$. Furthermore, this shows that no cycle can be formed using $v^i_{\ell,in}$ vertices, since all their outgoing arcs either move to a

previous row, or stay in the same row but decrease i. In a similar way, no directed cycle can be formed using only $v^i_{\ell,out}$ vertices, as all their outgoing arcs either move to a later row, or stay in the same row but decrease i.

Property 2. For a vertex v^i we have $2k-2$ arcs incident on it from the clique; the two arcs connecting it to $v^i_{1,in}, v^i_{1,out}$; two arcs connecting it to $v^j_{1,in}, v^j_{1,out}$ for $j > i$; two arcs connecting it to $v^j_{1,in}$ for $j < i$. This gives $2k-2+2i+2(k-i) = 4k-2$.

For a vertex $v^i_{1,in}$ we have one arc to v^j for $j \leq i$; two arcs to v^j for $j > i$; arcs to all v^j_{in}, v^j_{out} for $j \neq i$; arcs to $v^j_{2,in}$ for $j \leq i$. This gives $i + 2(k-i) + 2(k-1) + i = 4k-2$.

For a vertex $v^i_{1,out}$ we have arcs from v^j for $j \leq i$; arcs to $v^j_{1,in}, v^j_{1,out}$ for $j \neq i$; arcs to $v^j_{2,in}$ and $v^j_{2,out}$ for $j \geq i$; arcs to $v^j_{2,in}$ for all $j < i$. This gives $i+2(k-1)+2(k-i)+i = 4k-2$.

For a vertex $v^i_{\ell,in}$, $\ell \geq 2$ we have arcs to $v^j_{\ell-1,in}$, for $j \geq i$; to $v^j_{\ell-1,out}$ for $j \neq i$; to $v^j_{\ell,in}, v^j_{\ell,out}$ for $j \neq i$; from $v^j_{\ell+1,in}$ for $j \leq i$. This gives $(k-i+1)+(k-1)+2(k-2)+i = 4k-2$.

Finally, for a vertex $v^i_{\ell,out}$, $\ell \geq 2$ we have arcs from $v^j_{\ell-1,out}$ for $j \leq i$; to $v^j_{\ell,in}, v^j_{\ell,out}$ for $j \neq i$; to all $v^j_{\ell+1,in}$, for $j \neq i$; to $v^j_{\ell+1,out}$ for $j \geq i$. This gives $i + 2(k-1) + (k-1) + (k-i+1) = 4k-2$.

Property 3. We assign color i to v^i and to $\{v^i_{\ell,in}, v^i_{\ell,out} \mid \ell \in [M]\}$. We claim that there is no monochromatic cycle in P with this coloring. Indeed, if such a cycle exists, it must use v^i, as $\{v^i \mid i \in [k]\}$ is a directed feedback vertex set. But observe that with the coloring we gave, for each $\ell \in [M-1]$ the only out-neighbor of $v^i_{\ell,out}$ with color i is $v^i_{\ell+1,out}$ and $v^i_{M,out}$ has no out-neighbor of color i. Similar examination of $\{v^i_{\ell,in} \mid \ell \in [M]\}$ shows that the part of P colored i induces a directed path on $2M+1$ vertices with v^i in the middle.

Property 4. Since the vertices v^i induce a clique, we may assume without loss of generality that we are given a coloring c where $c(v^i) = i$. We prove the property by induction on ℓ. For $\ell = 1$, we will first prove that $c(v^i_{1,in}) = i$ by induction on i. For the base case we have that $v^1_{1,in}$ is connected with a digon with v^j for all $j > 1$, so $c(v^1_{1,in}) = 1$. Now, fix a j and suppose that for all $i < j$ we have $c(v^i_{1,in}) = i$. Then $v^j_{1,in}$ cannot receive any color $i < j$, because this would make a cycle with $v^i_{1,in}, v^i$. It can also not receive a color $i > j$ because it has a digon to all v^i for $i > j$. Hence, $c(v^j_{1,in}) = j$. Continuing on $\ell = 1$, we will prove by reverse induction on i that $c(v^i_{1,out}) = i$. For $c(v^k_{1,out})$ if we give this vertex any color $j < k$ then we get a cycle with $v^j, v^j_{1,in}$, so we must have $c(v^k_{1,out}) = k$. Now fix an i and suppose that for all $j > i$ we have $c(v^j_{1,out}) = j$. If we give $v^i_{1,out}$ a color $j > i$ this will make a cycle with $v_j, v^j_{1,out}, v^i_{1,out}, v^j_{1,in}$. But if we give $v^i_{1,out}$ a smaller color $j < i$, this will also make a cycle with $v^j, v^j_{1,in}$. Therefore, $c(v^i_{1,out}) = i$ for all i.

Suppose now that the property is true for row ℓ and we want to prove it for row $\ell+1$. We will use similar reasoning as in the previous case. We will also use the observation that for all i, there is a monochromatic path from v^i to $v^i_{\ell,out}$ and a monochromatic path

from $v^i_{\ell,in}$ to v^i. First, we show by induction on i that $c(v^i_{\ell+1,in}) = i$ for all i. For $v^1_{\ell+1,in}$ we observe that if we give this vertex color $j > 1$, then using the arcs from $v^j_{\ell,out}$ and to $v^j_{\ell,in}$ we have a monochromatic cycle of color j. Hence, $c(v^1_{\ell+1,in}) = 1$. Fix a j and suppose that for all $i < j$ we have $c(v^i_{\ell+1,in}) = i$. If we assign $c(v^j_{\ell+1,in})$ a color $i < j$, then we get a cycle using $v^i_{\ell,out}, v^j_{\ell+1,in}, v^i_{\ell+1,in}, v^i_{\ell,in}$. If we assign it a color $i > j$, then we get the cycle using $v^i_{\ell,out}, v^j_{\ell+1,in}, v^i_{\ell,in}$. So, for all i we have $c(v^i_{\ell+1,in}) = i$. To complete the proof, we do reverse induction to show that $c(v^i_{\ell+1,out}) = i$. For $c(v^k_{\ell+1,out})$ we cannot give this vertex color $j < k$ because this will give a cycle using $v^j_{\ell,out}, v^k_{\ell+1,out}, v^j_{\ell+1,in} v^j_{ell,in}$. Now, fix an i and assume that for $j > i$ we have $c(v^j_{\ell+1,out}) = j$. We cannot assign $v^i_{\ell+1,out}$ any color $j > i$ because this would give the cycle $v^j_{\ell,out}, v^j_{\ell+1,out}, v^i_{\ell+1,out}, v^j_{\ell+1,in}, v^j_{\ell,in}$. We can also not assign any color $j < i$ as this gives the cycle using $v^j_{\ell,out}, v^i_{\ell+1,out}, v^j_{\ell+1,in}, v^j_{\ell,in}$. We conclude that for all i we have $c(v^i_{\ell+1,out}) = i$.

Proof of Lemma 7.1

Proof. Let S be an $ir^*(G)$, $G' = G[V \setminus S]$ and $X' = X \setminus S$. Observe that for each pair of vertices u, v such that $u \in X'$ and $v \in N_{G'}(u) \cap X'$, we have that $d_{G'}(u) \neq d_{G'}(v)$, since S is an $ir^*(G)$. It follows that S is also an irregulator of X in G, *i.e.* S is an $ir(G, X)$, and thus we have that $\mathrm{I}(G, X) \leq |S| = \mathrm{I}(G)$. □

Proof of Lemma 7.2

Proof. Let S be an $ir^*(G, X)$, $S_1 = S \cap N[X]$ and $S_2 = S \setminus S_1$. It suffices to prove that S_1 is an $ir(G, X)$. Indeed, if $S_1 \subseteq S$ is an $ir(G, X)$, since S is an $ir^*(G, X)$, we can conclude that $S = S_1$ and that $S \subseteq N[X]$ (by definition of S_1).

Assume now that S_1 is not an $ir(G, X)$. Then there exists a pair of vertices u, v where uv is an edge in $G[X \setminus S_1]$ and $d_{G[V \setminus S_1]}(u) = d_{G[V \setminus S_1]}(v)$. Observe that $N[\{u, v\}] \subseteq N[X]$, and thus $N[\{u, v\}] \cap S_2 = \emptyset$. Therefore, $d_{G[V \setminus S]}(u) = d_{G[V \setminus S_1]}(u) = d_{G[V \setminus S_1]}(v) = d_{G[V \setminus S]}(v)$. This is a contradiction since S is an $ir^*(G, X)$.

Now, we prove that $\mathrm{I}(G, X) = \mathrm{I}(G[N[X]], X)$. Let S be an $ir^*(G, X)$. Since $S \subseteq N[X]$ and any vertex $v \in X \setminus S$ has $N(v) \subseteq N[X]$, we have that $d_{G[V \setminus S]}(v) = d_{G[N[X] \setminus S]}(v)$. Thus, S is an $ir(G[N[X]], X)$ and $\mathrm{I}(G, X) \geq \mathrm{I}(G[N[X]], X)$. Now for the opposite direction, let S' be an $ir^*(G[N[X]], X)$. We show that S' is also an $ir(G, X)$. Since for all $v \in X \setminus S'$, we have $d_{G[V \setminus S']}(v) = d_{G[N[X] \setminus S']}(v)$ (again because $N(v) \subseteq N[X]$) we have that S' is an $ir(G, X)$. Therefore, $\mathrm{I}(G, X) \leq \mathrm{I}(G[N[X]], X)$. □

Proof of Lemma 7.3

Proof. Let $X = \bigcup_{i=1}^{n} X_i$. For every $1 \leq i \leq n$, let S_i be an $ir^*(G, X_i)$ and $G'_i = G[V \setminus S_i]$, and let $S = \bigcup_{i=1}^{n} S_i$ and $G' = G[V \setminus S]$. Observe first that for every $i \neq j$, since

$N[X_i] \cap N[X_j] = \emptyset$, we have that $S_i \cap S_j = \emptyset$ as well. Thus, $|S| = \sum_{i=1}^{n} |S_i|$.

We now show that S is an $ir^*(G, X)$. Assume that there exists an S' such that $|S'| < |S|$ and S' is an $ir(G, X)$. Then, there exists a $k \leq n$ such that the set $S'_k = S' \cap N[X_k]$, is such that $|S'_k| < |S_k|$, as otherwise $|S'|$ can not be smaller than $|S|$. Observe that S'_k must be an $ir(G, X_k)$; this holds because for any vertex $u \in S' \setminus S'_k$, we know that $u \notin N[X_k]$. This is a contradiction since we have assumed that S_k is an $ir^*(G, X_k)$ and S'_k is an $ir(G, X_k)$ with size smaller than S_k. Therefore, S is an $ir^*(G, X)$, and the statement follows by Lemma 7.1. □

Proof of Lemma 7.4

Proof. Let $S_X = S \cap N[X]$, $G' = G[V \setminus S]$ $G^* = G[V \setminus S_X]$. Assume that S_X is not an $ir(G, X)$. Then there exist two adjacent vertices v, u such that $\{v, u\} \subset X \setminus S_X$ and $d_{G^*}(u) = d_{G^*}(v)$. Since $S_X = S \cap N[X]$ we have that $N_{G[S]}(u) = N_{G[S_X]}(u)$ and $N_{G[S]}(v) = N_{G[S_X]}(v)$. Therefore $d_{G'}(u) = d_{G^*}(u) = d_{G^*}(v) = d_{G'}(u)$ which is a contradiction since G' is locally irregular. It remains to show that S_X is an $ir(G[N[X]], X)$. Note that for any vertex $v \in X \setminus S_X$ $N[v]$ is included in both G and $G[N[X]]$. Therefore, $d_{G^*}(v) = d_{G[N[X] \setminus S_X]}(v)$ since we have remove the same vertices from $N[X]$. So S_X is an $ir(G[N[X]], X)$. □

Proof of Theorem 7.10

As the proof of Theorem 7.10 is rather technical, we first present the construction, as well as two lemmas that are going to be utilised in that proof.

Construction for Theorem 7.10 Assume that we have an instance (ϕ, X) of 2-Balanced 3-SAT comprised by a set $\{C_1, \ldots, C_m\}$ of clauses over the set of Boolean variables $\{x_1, \ldots, x_n\}$. We are going to create a cubic bipartite graph G. First we create a bipartite graph $F = (V, E)$ as follows: for each literal x_i ($\neg x_i$ resp.) in ϕ, add a *literal vertex* $v_{i,p}$ ($v_{i,n}$ resp.) in V, and for each clause C_j of ϕ, add a *clause vertex* c_j in V. Next, for each $1 \leq j \leq m$, add the edge $v_{i,p}c_j$ ($v_{i,n}c_j$ resp.) if the literal x_i ($\neg x_i$ resp.) appears in C_j according to ϕ. We create a copy $F' = (V', E')$ of F and we denote by $v'_{i,p}$, $v'_{i,n}$ and c'_j the copies of $v_{i,p}$, $v_{i,n}$ and c_j for all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$ respectively.

Finally, we are going to connect the two graphs using n copies of the gadget graph H presented in FigureA.1. In particular, for each $i \in \{1, \ldots, n\}$ we create a copy H_i of H and we attach it to the vertices related to the variable x_i. That is, we add H_i to the graph, and we identify $v_{i,p}$, $v_{i,n}$ with the vertices w_1 and w_2 and $v'_{i,p}$, $v'_{i,n}$ with the vertices w'_1 and w'_2 respectively. This results in a cubic bipartite graph which we call G.

Lemma A.5. *Let $H = (V, E)$ be the graph in Figure A.1 and let $V^* = V \setminus \{w_1, w_2, w'_1, w'_2\}$ then, the following holds:*

- $\mathrm{I}(H, V^*) = 3$

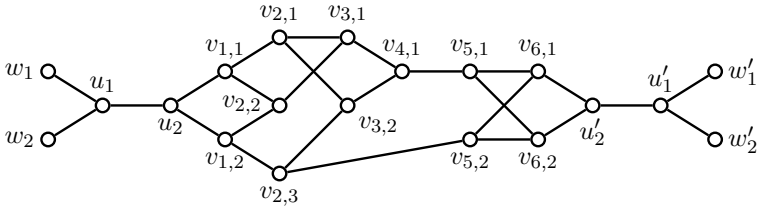


Figure A.1: The gadget used in the construction of Theorem 7.10

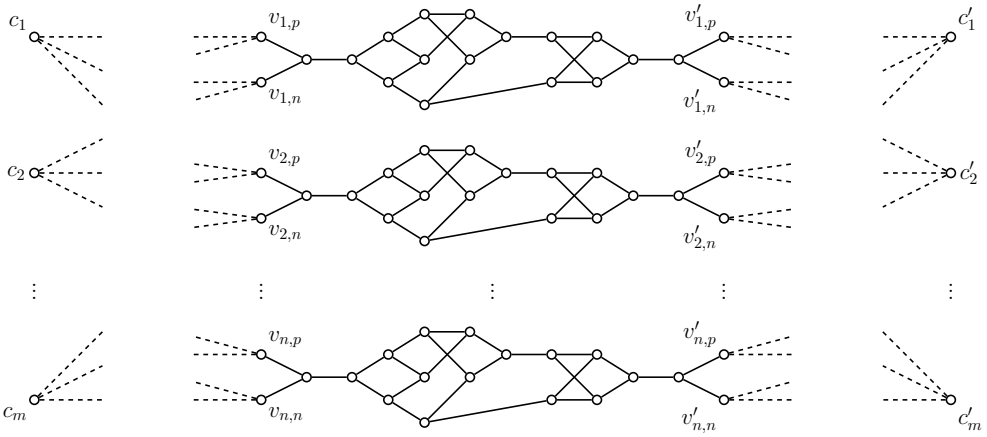


Figure A.2: The cubic bipartite graph G, constructed in the proof of Theorem 7.10

- *for any S that is an $ir^*(H, V^*)$ we have:*
  - $\{w_1, w_2, w'_1, w'_2\} \cap S = \emptyset$,
  - $\{u_1, u'_1\} \cap S = \emptyset$
  - $\{u_2, u'_2\} \cap S \neq \emptyset$
  - $\{u_2, u'_2\} \not\subseteq S$
- *for any S that is an $ir(H, V^*)$ and $|S| = 4$ we have:*
  - $|\{w_1, w_2, w'_1, w'_2\} \cap S| \leq 1$,
  - $|\{u_1, u'_1\} \cap S| \leq 1$

Proof. Consider the edges u_1u_2, $v_{3,1}v_{4,1}$ and $u'_2u'_1$. These edges have in common that both of their incident vertices have the same degree (equal to 3). It follows from Observation 1 that any $ir(H, V^*)$ contains at least one vertex in each one of the sets $S_1 = N(u_1u_2) = \{w_1, w_2, u_1, u_2, v_{1,1}, v_{1,2}\}$, $S_2 = N(v_{3,1}v_{4,1}) = \{v_{2,1}, v_{2,2}, v_{3,1}, v_{3,2}, v_{4,1}, v_{5,1}\}$ and $S_3 = N(u'_2u'_1) = \{v_{6,1}, v_{6,2}, u'_2, u'_1, w'_1, w'_2\}$. Finally, observe that the sets S_1, S_2 and S_3 are pairwise disjoint. It follows that $\mathrm{I}(H, V^*) \geq 3$. To show that $\mathrm{I}(H, V^*) = 3$, we provide the following list, which presents all the possible subsets of V that are irregulators of V^* in H and they have order 3:

- $\{u_2, v_{2,1}, v_{6,1}\}$
- $\{u_2, v_{2,1}, v_{6,2}\}$
- $\{u_2, v_{4,1}, v_{6,1}\}$
- $\{u_2, v_{4,1}, v_{6,2}\}$
- $\{v_{1,1}, v_{3,2}, u'_2\}$
- $\{v_{1,2}, v_{3,2}, u'_2\}$
- $\{v_{1,2}, v_{3,1}, u'_2\}$

It follows that $\mathrm{I}(H, V^*) = 3$. Note also that in all the sets presented in the above list, each set contains exactly one of the vertices u_2 and u'_2, and none of the vertices in $\{w_1, w_2, w'_1, w'_2, u_1, u'_1\}$. This suffices to validate the second item of the statement.

In the following list, we present all the possible subsets of V that are $ir(H, V^*)$ of order 4, validating the third item of the statement:

- $\{w_1, u_2, v_{2,1}, v_{6,1}\}$
- $\{w_1, u_2, v_{2,1}, v_{6,2}\}$
- $\{w_1, u_2, v_{4,1}, v_{6,1}\}$
- $\{w_1, u_2, v_{4,1}, v_{6,2}\}$
- $\{w_1, v_{2,2}, v_{2,3}, v_{6,2}\}$
- $\{w_1, v_{2,2}, v_{2,3}, v_{6,1}\}$
- $\{w_1, v_{2,2}, v_{2,1}, v_{6,2}\}$
- $\{w_1, v_{2,2}, v_{2,1}, v_{6,1}\}$
- $\{w_1, v_{2,2}, v_{4,1}, v_{6,2}\}$
- $\{w_1, v_{2,2}, v_{4,1}, v_{6,1}\}$
- $\{w_1, v_{2,3}, v_{2,1}, v_{6,2}\}$
- $\{w_1, v_{2,3}, v_{2,1}, v_{6,1}\}$
- $\{w_2, u_2, v_{2,1}, v_{6,1}\}$
- $\{w_2, u_2, v_{2,1}, v_{6,2}\}$
- $\{w_2, u_2, v_{4,1}, v_{6,1}\}$
- $\{w_2, u_2, v_{4,1}, v_{6,2}\}$
- $\{u_1, u_2, v_{2,1}, v_{6,2}\}$
- $\{u_1, u_2, v_{2,1}, v_{6,1}\}$
- $\{u_1, u_2, v_{4,1}, v_{6,2}\}$
- $\{u_1, u_2, v_{4,1}, v_{6,1}\}$
- $\{u_1, v_{1,1}, v_{3,2}, u'_2\}$
- $\{u_1, v_{1,2}, v_{3,2}, u'_2\}$
- $\{u_1, v_{1,2}, v_{3,1}, u'_2\}$
- $\{u_1, v_{3,2}, v_{3,1}, u'_2\}$
- $\{u_1, v_{3,1}, v_{5,2}, u'_2\}$
- $\{u_1, v_{3,1}, v_{5,2}, w'_1\}$
- $\{u_1, v_{3,1}, v_{5,2}, w'_2\}$
- $\{u_2, v_{2,2}, v_{2,3}, v_{6,2}\}$
- $\{u_2, v_{2,2}, v_{2,3}, v_{6,1}\}$
- $\{u_2, v_{2,2}, v_{2,1}, v_{6,2}\}$
- $\{u_2, v_{2,2}, v_{2,1}, v_{6,1}\}$
- $\{u_2, v_{2,2}, v_{4,1}, v_{6,2}\}$
- $\{u_2, v_{2,2}, v_{4,1}, v_{6,1}\}$
- $\{u_2, v_{2,3}, v_{2,1}, v_{6,2}\}$
- $\{u_2, v_{2,3}, v_{2,1}, v_{6,1}\}$
- $\{u_2, v_{2,3}, v_{4,1}, v_{6,2}\}$
- $\{u_2, v_{2,3}, v_{4,1}, v_{6,1}\}$
- $\{u_2, v_{2,3}, v_{4,1}, u'_1\}$
- $\{u_2, v_{2,1}, v_{4,1}, v_{6,2}\}$
- $\{u_2, v_{2,1}, v_{4,1}, v_{6,1}\}$
- $\{u_2, v_{2,1}, v_{6,2}, v_{6,1}\}$
- $\{u_2, v_{2,1}, v_{6,2}, u'_1\}$
- $\{u_2, v_{2,1}, v_{6,1}, u'_1\}$
- $\{u_2, v_{4,1}, v_{6,2}, v_{6,1}\}$
- $\{u_2, v_{4,1}, v_{6,2}, u'_1\}$
- $\{u_2, v_{4,1}, v_{6,1}, u'_1\}$
- $\{v_{1,1}, v_{1,2}, v_{3,2}, u'_2\}$
- $\{v_{1,1}, v_{1,2}, v_{3,1}, u'_2\}$
- $\{v_{1,1}, v_{1,2}, v_{5,1}, u'_2\}$
- $\{v_{1,1}, v_{1,2}, v_{5,1}, w'_1\}$
- $\{v_{1,1}, v_{1,2}, v_{5,1}, w'_2\}$
- $\{v_{1,1}, v_{3,2}, v_{3,1}, u'_2\}$
- $\{v_{1,1}, v_{3,2}, v_{5,1}, u'_2\}$
- $\{v_{1,1}, v_{3,2}, v_{5,1}, w'_1\}$
- $\{v_{1,1}, v_{3,2}, v_{5,1}, w'_2\}$
- $\{v_{1,1}, v_{3,2}, v_{6,2}, v_{6,1}\}$
- $\{v_{1,1}, v_{3,2}, v_{5,2}, u'_2\}$
- $\{v_{1,1}, v_{3,2}, v_{5,2}, w'_1\}$
- $\{v_{1,1}, v_{3,2}, v_{5,2}, w'_2\}$
- $\{v_{1,1}, v_{3,2}, u'_2, u'_1\}$
- $\{v_{1,1}, v_{3,1}, v_{5,2}, u'_2\}$
- $\{v_{1,1}, v_{3,1}, v_{5,2}, w'_1\}$
- $\{v_{1,1}, v_{3,1}, v_{5,2}, w'_2\}$
- $\{v_{1,1}, v_{5,1}, v_{6,2}, v_{5,2}\}$
- $\{v_{1,1}, v_{5,1}, v_{5,2}, v_{6,1}\}$
- $\{v_{1,1}, v_{5,1}, v_{5,2}, u'_2\}$
- $\{v_{1,1}, v_{5,1}, v_{5,2}, u'_1\}$
- $\{v_{1,1}, v_{5,1}, v_{5,2}, w'_1\}$
- $\{v_{1,1}, v_{5,1}, v_{5,2}, w'_2\}$
- $\{v_{1,1}, v_{3,2}, u'_2, w'_1\}$
- $\{v_{1,1}, v_{3,2}, u'_2, w'_2\}$
- $\{v_{2,2}, v_{2,3}, v_{6,2}, w_2\}$
- $\{v_{2,2}, v_{2,3}, v_{6,1}, w_2\}$
- $\{v_{2,2}, v_{2,1}, v_{6,2}, w_2\}$
- $\{v_{2,2}, v_{2,1}, v_{6,1}, w_2\}$
- $\{v_{2,2}, v_{4,1}, v_{6,2}, w_2\}$
- $\{v_{2,2}, v_{4,1}, v_{6,1}, w_2\}$
- $\{v_{1,2}, v_{3,2}, v_{3,1}, v_{6,2}\}$
- $\{v_{1,2}, v_{3,2}, v_{3,1}, v_{6,1}\}$
- $\{v_{1,2}, v_{3,2}, v_{3,1}, u'_2\}$
- $\{v_{1,2}, v_{3,2}, v_{5,1}, u'_2\}$
- $\{v_{1,2}, v_{3,2}, v_{5,1}, w'_1\}$
- $\{v_{1,2}, v_{3,2}, v_{5,1}, w'_2\}$
- $\{v_{1,2}, v_{3,2}, v_{5,2}, u'_2\}$
- $\{v_{1,2}, v_{3,2}, v_{5,2}, w'_1\}$
- $\{v_{1,2}, v_{3,2}, v_{5,2}, w'_2\}$
- $\{v_{1,2}, v_{3,2}, u'_2, u'_1\}$
- $\{v_{1,2}, v_{3,1}, v_{5,1}, u'_2\}$
- $\{v_{1,2}, v_{3,1}, v_{5,1}, w'_1\}$
- $\{v_{1,2}, v_{3,1}, v_{5,1}, w'_2\}$
- $\{v_{1,2}, v_{3,1}, v_{6,2}, v_{6,1}\}$
- $\{v_{1,2}, v_{3,1}, v_{5,2}, u'_2\}$
- $\{v_{1,2}, v_{3,1}, v_{5,2}, w'_1\}$
- $\{v_{1,2}, v_{3,1}, v_{5,2}, w'_2\}$
- $\{v_{1,2}, v_{3,1}, u'_2, u'_1\}$
- $\{v_{1,2}, v_{3,1}, u'_2, w'_1\}$
- $\{v_{1,2}, v_{3,1}, u'_2, w'_2\}$
- $\{v_{1,2}, v_{3,2}, u'_2, w'_1\}$
- $\{v_{1,2}, v_{3,2}, u'_2, w'_2\}$
- $\{v_{2,3}, v_{2,1}, v_{6,2}, w_2\}$
- $\{v_{2,3}, v_{2,1}, v_{6,1}, w_2\}$

□

At this point we would like to comment on the proof of Lemma A.5. Indeed, one could prove that lemma by an extensive case analysis. We stress however that, even taking advantage of Observation 1, that case analysis would be extremely long and uninteresting. So, instead, we decided to check all possible $ir(H, V^*)$ of order 3 and 4 with the help of a computer program (running a simple exhaustive algorithm), which gave us the above lists.

Before we continue, we are going to give some notation that we are going to use in what follows. First, we call V_C the set of all the clause vertices and V_X the set of all the literal vertices. Whenever it is not clear by the context, we use $v(i)$ in order to talk about the copy of a vertex $v \in V(H)$ that belongs to H_i. Similarly, we use V_i^* to denote the copy of V^* that is inside H_i. For all $i \in \{1, \ldots, n\}$, let X_i be the set of literal vertices $\{v_{i,p}, v_{i,n}, v'_{i,p}, v'_{1,n}\}$, U_i be the set that contains the copies of u_1 and u'_1 that belong in H_i and $N_i = \bigcup_{v \in X_i} N(v) \setminus U_i$ the set of clause vertices are adjacent to a literal vertex of X_i.

Now, assume that we have an irregulator S of G (where G is the graph described in the previous construction). By Lemma 7.4 we have that $S \cap V(H_i)$ is an $ir(H_i, V_i^*)$ (since $H_i = G[N[V_i^*]]$) for all $i \in \{1, \ldots, n\}$. Let $\mathbb{I} = \{j \mid j \in \{1, \ldots, n\} \text{ and } |V(H_j) \cap S| = 3\}$. Then, for all $j \in \mathbb{I}$, the set $S \cap V(H_j)$ is an $ir^*(H_j, V_j^*)$ which means that it has the same properties as any $ir^*(H, V^*)$. Furthermore, by Lemma A.5 we know that for all $v \in \{w_1, w_2, w'_1, w'_2, u_1, u'_1\}$ the copy of v in H_i is not included in $S \cap V(H_i)$. Additionally, one of the u_1, u'_1 has degree 2 and the other has degree 3 in the $G[V \setminus S]$. We call X_L the set of all the vertices $v_{j,p}$, $v_{j,n}$, $j \in \mathbb{I}$ such that the copy of u'_2 belonging to H_j is included to S. We call these the "left gadgets" and we denote by n_l the number of these gadgets. Similarly, we call X_R the set of all the vertices $v'_{j,p}$, $v'_{j,n}$, $j \in \mathbb{I}$ such that the copy of u_2 belonging to H_j is included to S. We call these the "right gadgets" and we denote by n_r the number of these gadgets. The gadgets H_i, $i \notin \mathbb{I}$, will be called *good* gadgets.

Now, we show that $\mathrm{I}(G) \geq 4n$, and we identify some additional properties of any minimum irregulator of G.

Lemma A.6. *Let G be the graph constructed as described above, starting from an instance (ϕ, X) of 2-Balanced 3-SAT with $|X| = n$. Then $\mathrm{I}(G) \geq 4n$. Furthermore, any set S that is an $ir^*(G)$, such that $|S| = 4n$, verifies the following:*

1. *there is no clause such that both its clause vertices c and c' belong to S.*
2. *$n_l + n_r = |S \cap V_C|$*
3. *$|S \cap V(H_i)| \leq 4$ for all $i \in \{1, \ldots, n\}$*
4. *any clause vertex $c \in S$ has exactly two neighbours in $X_L \cup X_R$ and one neighbour that belongs to a good gadget.*
5. *for any c and c', corresponding to the same clause in ϕ, if $c \in S$ ($c' \in S$ resp.) then there exists a variable vertex $v \in S$ such that $v \in N_G(c')$ ($v \in N_G(c)$).*

Proof. First we show that $\mathrm{I}(G) \geq 4n$. Let S be an $ir(G)$ and $G^* = G[V \setminus S]$. Let H_i be the copy of the gadget H that is attached to the literal vertices that correspond to the variable $x_i \in X$. Since $\mathrm{I}(H, V^*) = 3$, we have that $\mathrm{I}(G, V_i^*) = 3$ for all $i \in \{1, \ldots, n\}$. Observe that for any $i \in \mathbb{I}$, $|S \cap V(H_i)| = 3$ and $S \cap X_i = \emptyset$. We claim that S must include at least one vertex of the set N_i. Indeed, if this is not true then all the vertices in X_i have degree 3 in G^* and the same holds for one of the vertices in U_i which lead to a contradiction. It is also worth mentioning that for any $i \in \mathbb{I}$, for $(v, u) \in \{\{v_{i,p}v_{i,n}\} \times \{v'_{i,p}v'_{i,n}\}\}$ we have we have that $v \in X_L \iff u \notin X_R$ so $\{v, u\} \cap (X_L \cup X_R) \neq \emptyset$.

Before we continue, let us give some extra notation. Let $C_L \subseteq S$ (resp. $C_R \subseteq S$) be the set of clause vertices c such that $N_G(c) \cap X_L \neq \emptyset$ (resp. $N_G(c) \cap X_R \neq \emptyset$). Note that C_L is subset of the set of clause vertices of F ($C_L \subseteq V(F)$) and C_R is subset of the set of clause vertices of F' ($C_R \subseteq V(F')$). Furthermore, for $1 \leq l \leq 3$, let $C_{L,l}$ be the sets of vertices $c \in C_L$ such that $|N(c) \cap (\bigcup_{j \in \mathbb{I}} X_j)| = l$ and $C_{R,l}$ be the sets of vertices $c \in C_R$ such that $|N(c) \cap (\bigcup_{j \in \mathbb{I}} X_j)| = l$. In simple words, $C_{L,l}$ ($C_{R,l}$ resp.) are the clause vertices that belong to $S \cap F$ (to $S \cap F'$ resp.), which are adjacent to l left or right gadgets, with at least one of these being a left (right resp.) gadget. Finally, let $C'_{L,3}$ be the set $\{c'_i \mid c_i \in C_{L,3}\}$ and $C'_{R,3}$ be the set $\{c_i \mid c'_i \in C_{R,3}\}$. In Figure A.3 we illustrate all information given above.

We now show that $|S \cap V_C| \geq n_l + n_r$. To do that, we preset some relations between the sets $C_L \cup C_R$, $X_L \cup X_R$ and the edges between them in G. Let E^* be the set containing exactly these edges. First, we calculate a lower bound for $|E^*|$. Observe that all vertices in X_L belong to a left gadget and are neighbours of a copy of u_1. Additionally, $d_{G^*}(u_1) = 3$ for any u_1 that belongs to a left gadget. Therefore, for all $v \in X_L$, $d_{G^*}(v) \leq 2$ as otherwise G^* is not locally irregular. This means that each $v \in X_L$ has at least one neighbour in S, and thus that there are at least $|X_L| = 2n_l$ edges between S and X_L. Furthermore, all these edges must be incident to vertices in C_L (by the definition of C_L). In the same way we can show that between S and X_R are at least $|X_R| = 2n_r$ edges all of them being incident to vertices in C_R. Therefore, $|E^*| \geq 2(n_l + n_r)$.

Now we are going to calculate an upper bound for $|E^*|$. To do that, we have to split E^* into tree subsets, and treat them separately. For $1 \leq l \leq 3$, let E_l^* be the set of edges between $C_{L,l} \cup C_{R,l}$ and $X_L \cup X_R$. By the definition of $C_{L,1}$ and $C_{R,1}$ we know that each $c \in C_{L,1} \cup C_{R,1}$ has exactly one neighbour in $X_L \cup X_R$. It follows that $|E_1^*| \leq |C_{L,1} \cup C_{R,1}|$. Also, the vertices $c \in C_{L,2} \cup C_{R,2}$ have at most two neighbours in $X_L \cup X_R$. It follows that $|E_2^*| \leq 2|C_{L,2} \cup C_{R,2}|$. For the set $|E_3^*|$, an easy upper bound is $3|C_{L,3} \cup C_{R,3}|$. However, we are going to calculate a tighter upper bound. To do that, we use the following claim.

Claim A.7. *Both $C'_{R,3}$ and $C'_{L,3}$ are subsets of S.*

Proof. Let $c \in C'_{R,3}$. By the definition of $C'_{R,3}$, we know that there exists a $c' \in C_{R,3}$ and that c' is incident to a vertex $x \in X_R$ in G. Let H be the copy of the left gadget to which x belongs. Without loss of generality, assume that x is a copy of the vertex w'_1

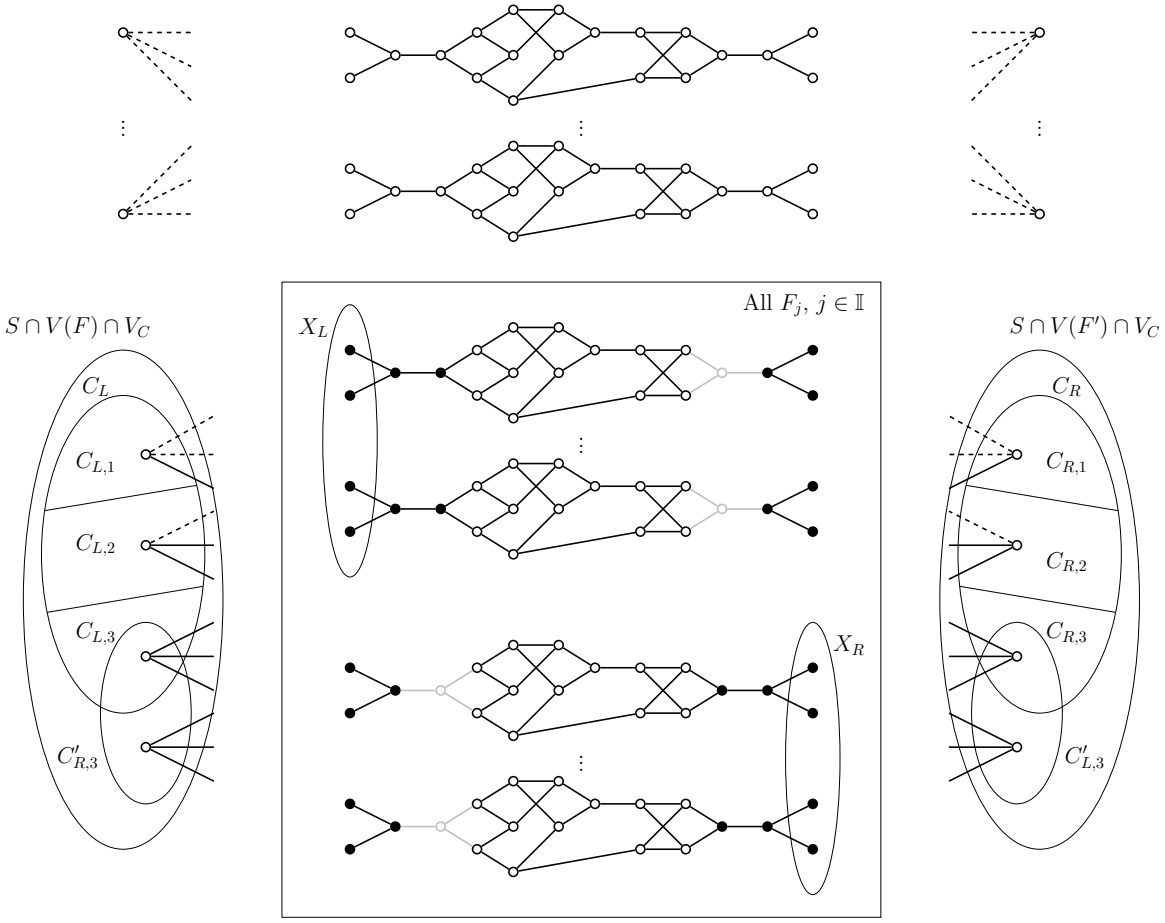


Figure A.3: An illustration of the subsets of C_L and C_R, defined in the proof of Theorem 7.10. The edges incident to clause vertices: are dashed if they are incident to vertices of F_i for some $i \notin \mathbb{I}$; are not dashed if they are incident to vertices of F_j for some $j \in \mathbb{I}$. We know that the black vertices do not belong to S and that the light-grey vertices belong to S.

of H. Therefore, in G, the vertex c is adjacent to the the copy of w_1 that belongs to H. Now, let z be the other clause vertex adjacent to w_1 in G. In order to decide if c belongs to S we need to consider two cases, the first being when $z \in S$ and the second when $z \notin S$. For the first case ($z \in S$), consider the vertex $u_1 \in H$. Since H is a left gadget, we have that $u_1 \notin S$, and that $d_{G^*}(u_1) = 2$. It follows that $c \in S$ as otherwise we would have that $d_{G^*}(w_1) = d_{G^*}(u_1) = 2$. So let $z \notin S$ (second case) and assume that $z \notin S$. It follows that $d_{G^*}(c) = 3$ as all of the neighbours of c in G are copies of vertices of X_R or of X_L (by the construction of $C'_{R,3}$) and none of these vertices belongs to S. Also, since $z \notin S$, we have that $d_{G^*}(w_1) = 3$ which is a contradiction since c is adjacent to w_1. So, for any $c \in C'_{R,3}$ we have that $c \in S$. With symmetric arguments we can show that any $c \in C'_{L,3}$ must belong to S. □

In order to calculate a tighter upper bound for E_3^* we are going to use the sets $C_{L,3} \cup C'_{R,3}$ and $C_{R,3} \cup C'_{L,3}$. Observe that the set $C_{R,3} \cup C'_{L,3}$ contains exactly the copies of the vertices in $C_{L,3} \cup C'_{R,3}$. Now, due to symmetry, we have that each pair $(c_j, c'_j) \in (C_{L,3} \cup C'_{R,3}) \times (C_{R,3} \cup C'_{L,3})$ has exactly three neighbours in $X_L \cup X_R$ (if c_j has $\ell \leq 3$ neighbours in X_L then c'_j has $3 - \ell$ neighbours in X_R and *vice versa*). Therefore, between the sets $C_{L,3} \cup C'_{R,3} \cup C_{R,3} \cup C'_{L,3}$ and $X_L \cup X_R$ we have exactly $3|C_{L,3} \cup C'_{R,3}|$ edges in G. Since $C_{L,3} \cup C_{R,3} \subseteq C_{L,3} \cup C'_{R,3} \cup C_{R,3} \cup C'_{L,3}$ we have $|E_3^*| \leq 3|C_{L,3} \cup C'_{R,3}|$.

Combining the above upper bounds, we get that $|E^*| \leq |C_{L,1} \cup C_{R,1}| + 2|C_{L,2} \cup C_{R,2}| + 3|C_{L,3} \cup C'_{R,3}|$.

By combining the lower and the upper bounds for $|E^*|$, we get the following inequality:

$$2(n_l + n_r) \leq |C_{L,1} \cup C_{R,1}| + 2|C_{L,2} \cup C_{R,2}| + 3|C_{L,3} \cup C'_{R,3}| \tag{A.7}$$

Since the sets $C_{L,3} \cup C'_{R,3}$ and $C_{R,3} \cup C'_{L,3}$ are both subsets of S (by Claim A.7), we have that $S \supseteq C_{L,1} \cup C_{L,2} \cup C_{L,3} \cup C'_{R,3} \cup C_{R,1} \cup C_{R,2} \cup C_{R,3} \cup C'_{L,3}$. Also, since the sets $C_{L,3} \cup C'_{R,3}$ and $C_{R,3} \cup C'_{L,3}$ have the same cardinalities, we get that:

$$|S \cap V_C| \geq |C_{L,1} \cup C_{R,1}| + |C_{L,2} \cup C_{R,2}| + 2|C_{L,3} \cup C'_{R,3}| \tag{A.8}$$

If we combine the two inequalities A.7 and A.8, we get that:

$$2(n_l + n_r) \leq 2|S \cap V_C| - |C_{L,1} \cup C_{R,1}| - |C_{L,3} \cup C'_{R,3}|$$

This gives us that $|S \cap V_C| \geq n_l + n_r$, from which follows that $|S| \geq 4n$.

Now, by taking a closer look at the previous inequalities, we can gain some extra information. Note that from this point we are considering only cases where we have an irregulator S such that $|S| = 4n$. Since for all $i \notin \mathbb{I}$ we have $|S \cap V(H_i)| \geq 4$, it follows that:

$$|S| = \sum_{i \notin \mathbb{I}} |S \cap V(H_i)| + \sum_{i \in \mathbb{I}} |S \cap V(H_i)| + |S \cap V_C| = \sum_{i \notin \mathbb{I}} |S \cap V(H_i)| + 3(n_l + n_r) + |S \cap V_C|.$$

Note that the number of the gadgets H_i with $i \notin \mathbb{I}$, is exactly $n - (n_l + n_r)$. Assume now that $|S \cap V_C| > (n_l + n_r)$. Then:

$$\begin{aligned}
|S| &= \sum_{i \notin \mathbb{I}} |S \cap V(H_i)| + 3(n_l + n_r) + |S \cap V_C| \\
&\geq 4(n - n_l - n_r) + 3(n_l + n_r) + |S \cap V_C| && \text{since } \forall i \notin \mathbb{I} : |S \cap V(H_i)| \geq 4 \\
&> 4(n - n_l - n_r) + 4(n_l + n_r) && \text{since } |S \cap V_C| > (n_l + n_r) \\
&= 4n
\end{aligned}$$

noindent which is a contradiction. Thus $|S \cap V_C| = (n_l + n_r)$. Now assume that there exists an $i \notin \mathbb{I}$ such that $|S \cap V(H_i)| > 4$. Then:

$$
\begin{aligned}
|S| &= \sum_{i \notin \mathbb{I}} |S \cap V(H_i)| + 3(n_l + n_r) + |S \cap V_C| \\
&\geq \sum_{i \notin \mathbb{I}} |S \cap V(H_i)| + 4(n_l + n_r) && \text{since } |S \cap V_C| = (n_l + n_r) \\
&> 4(n - n_l - n_r) + 3(n_l + n_r) + (n_l + n_r) && \text{since } \exists i \notin \mathbb{I} : |S \cap V(H_i)| > 4 \\
&= 4n
\end{aligned}
$$

This, again, is a contradiction. Therefore, $|S \cap V_C| = (n_l + n_r)$ and for all $i \notin \mathbb{I}$ we have $|S \cap V(H_i)| \leq 4$. In addition, for all $i \in \mathbb{I}$, we have that $|S \cap V(H_i)| = 3$. At this point we have shown the items 2. and 3. of the statement

Also, from eq. A.7, we can deduce that both $C_{L,1} \cup C_{R,1}$ and $C_{L,3} \cup C'_{R,3}$ are empty sets as otherwise $|S \cap V_C| > n_l + n_r$. Furthermore, $C'_{R,3}$ being empty means that $C_{R,3}$ is empty as well.

Now we are going to show that $S \cap V_C = C_{L,2} \cup C_{R,2}$ and that there no clause vertex $c \in C_{L,2}$ such that its copy c' belongs to $C_{R,2}$ (which suffices to prove items 4. and 1.). We first show that $S \cap V_C = C_{L,2} \cup C_{R,2}$. Assume that $S \cap V_C \supset C_{L,2} \cup C_{R,2}$. Then, $|S \cap V_C| > |C_{L,2} \cup C_{R,2}|$. Using this we can modify eq. A.7. This gives us $2(n_l+n_r) \leq 2|C_{L,2} \cup C_{R,2}| < 2|S \cap V_C|$ which contradicts the fact that $|n_l+n_r| = |S \cap V_C|$.

Assume now that there exists a set of clauses $S_c \subset C_{L,2}$ such that for each $c \in S_c$ we have that its copy c' belongs to $C_{R,2}$. Furthermore, let S_c be the largest such set and S'_c be the set $\{c' \mid c \in S_c\}$. We are going to calculate a new upper bound for the edges between $S \cap V_C$ and $X_L \cup X_R$ in G. Observe that, by construction, all the vertices in $C_{L,2} \cup C_{R,2}$ have exactly two neighbours in $\bigcup_{j \in \mathbb{I}} X_i$. Furthermore, the vertices in C_L have at least one neighbour in X_L and the vertices in C_R have at least one neighbour in X_R. So, we can conclude that all vertices in $S_c \cup S'_c$ have exactly one neighbour in $X_L \cup X_R$ and all vertices in $(C_{L,2} \cup C_{R,2}) \setminus (S_c \cup S'_c)$ have exactly two neighbours in $X_L \cup X_R$. In this case, the maximum number of edges between $S \cap V_C = C_{L,2} \cup C_{R,2}$ and $X_L \cup X_R$ in G is $|S_c \cup S'_c| + 2|(C_{L,2} \cup C_{R,2}) \setminus (S_c \cup S'_c)| = 2|S \cap V_C| - |S_c \cup S'_c|$. Since we have assumed that S_c is not empty we have $2(n_l + n_r) \leq 2|S \cap V_C| - |S_c \cup S'_c| < 2|S \cap V_C|$ which contradicts to the fact that $|S \cap V_C| = n_l + n_r$.

It remains to prove item 5. For that, consider a vertex $c \in S \cap V_C$. W.l.o.g. assume that $c \in C_L$. Recall that we have shown that $S \cap V_C = C_{L,2} \cup C_{R,2}$. Therefore, c' has two neighbours in left gadgets and one in a good gadget. Furthermore, since no literal vertex of a left gadget has been included in S we know that $d_{G^*}(c') \geq 2$. We show the following claim:

Claim A.8. $d_{G^*}(c') = 2$

Proof. Assume that c' has degree 3 in G^*. Now consider one of the neighbours of c' which belongs to a left gadget; let us call this neighbour v'. Since v' is a literal vertex of a left gadget and the vertex u'_1 of each left gadget has $d_{G^*}(u'_1) = 2$ we know that the degree of v' in G^* can be 0, 1 or 3 (because S is an $ir(G)$). Observe that u'_1 and c' do

not belong to S so v' has at least two neighbours in G^*. This means that $d_{G^*}(v')$ can be only 3. This is a contradiction because G^* is a locally irregular graph, and the edge $v'c'$ belongs to G^* and $d_{G^*}(v') = d_{G^*}(c') = 3$. □

Therefore c' can not have degree three in G^*, from which follows that one of its neighbours belong to S. This completes the proof of the lemma. □

Proof of Theorem 7.10

Proof. Since the problem is clearly in NP, we focus on proving it is also NP-hard. The reduction is from 2-Balanced 3-SAT, which was proven to be NP-complete in [29]. Starting with a formula ϕ over a set of n variables X we create the graph $G = (V, E)$ described above (illustrated in Figure A.2). We claim that $\mathrm{I}(G) = 4n$ if and only if ϕ is satisfiable.

Assume that ϕ is satisfiable and let $t : X \to \{T, F\}$ be a satisfying assignment. We construct an irregulator of G as follows: for each copy of H, select the vertices $u_1, v_{3,1}, v_{5,2}$. Then for each $i \in \{1, \ldots, n\}$ we select the vertex $v'_{i,p}$ if $t(x_i) = T$ and the vertex $v'_{i,n}$ otherwise. Let S be the set of all of these vertices and $G^* = G(V \setminus S)$. We claim that S is an $ir^*(G)$. First observe that $|S| = 4n$ since we have select exactly 4 vertices from each copy of H. Furthermore, observe that, for each $i \in \{1, \ldots, n\}$ we have that $S \cap V(H_i)$ is an $ir(G[V(H_i)], V_i^*)$. Thus, to show that S is an $ir(G)$, it suffices to show that there is no literal vertex in $G[V(G) \setminus S]$ that has the same degree as one of its neighbours (all the other pair of vertices either do not share an edge or they do not have the same degree because $S \cap V(H_i)$ is an $ir(G[V(H_i)], V_i^*)$). Observe that no literal vertex v, incident to a copy of u_1 in G, has been included in S. Therefore, any clause vertex c adjacent to any such literal vertex v, has degree 3 in G^*. Furthermore observe that for each such literal vertex v, the set S contains its neighbour u_1. Thus, any such literal vertex has degree 2 in G^* and only neighbours of degree 3. Now consider any literal vertex $v' \in V \setminus S$, incident to copies of u'_1. Since S does not include any vertex of V_C (recall that V_C is the set of all clause vertices in G) and any copy of u'_1, we have $d_{G^*}(v') = 3$. Furthermore, by the construction of S we have that $d_{G^*}(u'_1) = 2$ for all the copies of u'_1. It remains to show that the clause vertices $c' \in V_C \cap V(F')$ (the set of clause vertices to the right of the Figure A.2) have $d_{G^*}(c') \leq 2$. Since S contains all the literal vertices that correspond to all the true literals under t, and t is a satisfying assignment, we know that S contains least one neighbour of c' for all $c' \in V_C \cap V(F')$. Since these vertices do not exist in G^*, we have $d_{G^*}(c') \leq 2$ for all $c' \in V_C \cap V(F')$. Therefore, S is an $ir(G)$. Additionally, since $|S| = 4n$ and we know that $I(G) \geq 4n$, we have that S is an $ir^*(G)$.

For the reverse direction, let S be an $ir^*(G)$ of size $4n$. We show that ϕ is satisfiable. In particular we claim that

$$t(x_i) = \begin{cases} T & \text{if } \{v_{i,p}, v'_{i,p}\} \cap S \neq \emptyset, \\ F & \text{otherwise} \end{cases}$$

is a satisfying assignment of ϕ.

Since, by Lemma A.5, we have that for all i, $|\{v_{i,p}, v'_{i,p}, v_{i,n}, v'_{i,n}\} \cap S \cap H_i| \leq 1$, it suffices to show that for each clause C_j we have $N(\{c_j, c'_j\}) \cap S \neq \emptyset$. Assume that there exists a clause C_j such that $N(\{c_j, c'_j\}) \cap S = \emptyset$. We distinguish two cases: $\{c_j, c'_j\} \cap S \neq \emptyset$ and $\{c_j, c'_j\} \cap S = \emptyset$.

Case 1 ($\{c_j, c'_j\} \cap S \neq \emptyset$)**.** Let $c \in \{c_j, c'_j\} \cap S$; since the irregulator S has size $4n$, by Lemma A.6, we know that there exists a literal vertex $v \in N(\{c_j, c'_j\}) \cap S$. This is a contradiction.

Case 2 ($\{c_j, c'_j\} \cap S = \emptyset$)**.** Here, again, we are going to distinguish two cases; first if there exists a $k \in \mathbb{I}$ such that $N(\{c_j, c'_j\}) \cap V(H_k) \neq \emptyset$ and, second, if there is no such $k \in \mathbb{I}$.

Case 2(a) ($N(\{c_j, c'_j\}) \cap V(H_k) \neq \emptyset$)**.** W.l.o.g. let H_k be a left gadget. Then, c'_j is adjacent to one of the vertices $v'_{k,p}$ or $v'_{k,n}$; let us call v' the neighbour of c'_j in H_k. Since v' belongs to a left gadget and $c'_j \notin S$, we have $d_{G^*}(v') \geq 2$ which is a contradiction. Indeed, v' is a literal vertex of a left gadget and we know that $d_{G^*}(u'_1) = 2$ in any left gadget. Furthermore $d_{G^*}(c'_j) = 3$. It follows that v' has the same degree as either c'_j (if $d_{G^*}(v') = 3$) or the vertex u'_1 belonging to H_k (if $d_{G^*}(v') = 2$).

Case 2(b) ($N(\{c_j, c'_j\}) \cap V(H_k) = \emptyset$ **for all** $k \in \mathbb{I}$)**.** Let H_k be a gadget such that $N(\{c_j, c'_j\}) \cap V(H_k) \neq \emptyset$. By the hypothesis, $k \notin \mathbb{I}$. By Lemma 7.4 we have that $S_k = S \cap V(H_k)$ is an $ir(H_k, V_k^*)$. Furthermore, since $k \notin \mathbb{I}$, we have that $|S_k| = |S \cap V(H_k)| = 4$.

Therefore, we know that at most one of u_1 and u'_1 (of H_k) is included in S (since, by Lemma A.5, at most one of them is included in S_k). W.l.o.g. assume that $u_1 \notin S$. Let $v \in \{v_{k,p}, v_{k,n}\}$ be the neighbour of c_j in H_k. Since $N_{G^*}(v) \supseteq \{c_j, u_1\}$ and $d_{G^*}(c_j) = 3$, it follows that $N_{G^*}(v) = \{c_j, u_1\}$ and that there exists a $q \neq j$ such that $c_q \in N_G(v)$ and $c_q \in S$.

We are going to show that c'_j has a neighbour $v' \in S$. Due to symmetry we know that the copies c'_j and c'_q have a common neighbour v' in H_k. Furthermore, because of Lemma A.6 we know that two of the neighbours of c'_q belong to left gadgets and the third belongs to a good gadget. Also, the neighbour that belongs to a good gadget must belong to S. Since v' belongs to a good gadget and it is a neighbour of c'_q, it follows that $v' \in S$. This contradicts the starting assumption that $N_G\{c_j, c'_j\} \cap S = \emptyset$.

This ends the proof of the theorem. □

Proof of Theorem 8.5

In order to prove this theorem we need first to define a restricted version of Exactly-1 3-SAT.

Definition A.1 (Restricted Exactly 1 (3,2)-SAT)**.** *Assume that ϕ is a CNF formula where each clause has either three or two literals and each variable appears at most three times. We want to determine whether there exists a satisfying assignment so that each clause has exactly one true literal.*

Lemma A.9. *The Restricted Exactly-1 (3,2)-SAT is NP-complete.*

Proof. We show a reduction from Exactly-1 3-SAT which is known to be NP-complete [94] (lo4, one-in-three 3sat).

Let $I = (\phi, X)$ be an instance of Exactly-1 3-SAT with $|X| = n$ variables and m clauses. If there exists a variable x with $k > 3$ appearances, we replace each appearance with a fresh variable x_i, $i \in [k]$ and add to the formula the clauses $(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \dots (\neg x_k \vee x_1)$. We repeat this for all variables that appear more than three times. Let $I' = (\phi', X')$ be this new instance.

We claim that $I = (\phi, X)$ is a yes instance of Exactly-1 3-SAT iff $I' = (\phi', X')$ is a yes instance of Restricted Exactly-1 (3,2)-SAT.

Let $S : X \to \{T, F\}$ be a satisfying assignment for ϕ such that each clause of ϕ has exactly one true literal. It is not hard to see that $S' : X' \to \{T, F\}$ such that $S'(x) = S(x)$ if $x \in X$ and $S'(x_i) = S(x)$ if x_i replaces one appearance of $x \in X$, is a satisfying assignment for ϕ' such that each clause of ϕ' has exactly one true literal.

Conversely, let $S' : X' \to \{T, F\}$ be a satisfying assignment for ϕ' such that each clause of ϕ' has exactly one true literal. Let x_i, $i \in [k]$, be the variables replacing x. Because we have clauses $(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \dots (\neg x_k \vee x_1)$ we know that all the x_i, $i \in [k]$, must have the same value in order to guarantee that all of these clauses have exactly one true literal. Furthermore, is not hard to see that $S : X \to \{T, F\}$ where $S(x) = S'(x)$ if $x \in X'$ and $S(x) = S'(x_1)$ if x_1 replaces one appearance of x, then S is a satisfying assignment for ϕ such that each clause of ϕ has exactly one true literal. □

Now, let us give a construction that we are going to use.

Construction.
Let ϕ be an instance of Restricted Exactly 1 (3,2)-SAT with variables $X = \{x_1, \dots, x_n\}$ and clauses $C = \{c_1, \dots, c_m\}$. We will construct an instance of the crossword problem with alphabet $\mathcal{L} = \{s_1, s_2, s_3\}$ where each letter has weight 1. The dictionary $\mathcal{D}$ is as follows.

Let $nl_j \in \{2, 3\}$ be the number of literals in c_j. For each variable x_i, let $a_i \leq 3$ be the number of its appearances in ϕ. Then, we create $3a_i$ words, $d_{i,k,T}$, $d_{i,k,F}$ and $d_{i,k}$, for each $k \in [a_i]$ as follows.

- $d_{i,k,T}$ and $d_{i,k,F}$ have length $m + n + 3i + k$,
- the last letter of $d_{i,k,T}$ is s_k,
- the last letter of $d_{i,k,F}$ is $s_{k'}$ where $k' := k + 1$ when $k < a_i$, otherwise $k' := 1$,
- if the k-th appearance of x_i is positive then, $d_{i,k,T}$ starts with s_1 and $d_{i,k,F}$ starts with s_2,
- if the k-th appearance of x_i is negative then, $d_{i,k,T}$ starts with s_2 and $d_{i,k,F}$ starts with s_1,
- the word $d_{i,k}$ has length $m + i + 1$ and starts with s_k, and
- all the other letters of these words can be chosen arbitrarily.

Observe that the above process gives three words for each literal in ϕ.

For each clause c_j, $j \in [m]$, we construct nl_j distinct words d_j^t, $t \in [nl_j]$ of length $1+j$ such that one of them starts with the letter s_2, the other $nl_j - 1$ words start with s_1, and the unspecified letters can be chosen arbitrarily. Observe that we have enough positions in order to create $nl_j - 1$ distinct words starting with s_1, which indicates that we can create nl_j pairwise distinct words for each c_j.

In order to finish our construction we have to specify the grid. For each clause c_j and each literal l in c_j we construct two pairs of slots as follows. Let l be the k-th appearance of variable x_i, $k \in [a_i]$. The first pair of slots (type 1) consists of one horizontal slot $hSlot_{j,1}^{i,k}$ of length $m+n+3i+k$, and one vertical slot $vSlot_{j,1}^{i,k}$ of length $m+i+1$ such that, the last cell of the horizontal slot and the first cell of the vertical slot is the shared cell. The second pair of slots (type 2) consists of one horizontal slot $hSlot_{j,2}^{i,k}$ of length $m+n+3i+k$, and one vertical slot $vSlot_{j,2}^{i,k}$ of length $j+1$, that share their first cells. Here let us mention that the grid we constructed is consisted only by $\mathcal{T}$s.

Before we continue with the proof let us observe that in the instance of crossword puzzle we created the number of slots in the grid is equal to the number of words in the dictionary. Furthermore, we can specify in which slots each word can be assigned by considering the size of the words and slots. For any $i \in [n]$ and $k \in [a_i]$ the word $d_{i,k}$ can be assigned only to the vertical slots of the type 1 pairs of slots. For any $j \in [m]$ and $t \in [nl_j]$ the word d_j^t can be assigned only to the vertical slots of the type 2 pairs of slots. The rest of the words can be assigned to horizontal slots of any type.

Let us first prove the following property where $j(i,k)$ denotes the index of the clause where the k-th occurrence of x_i appears.

Property A.10. *For any given $i \in [n]$, slots $hSlot_{j(i,k),1}^{i,k}$ and $vSlot_{j(i,k),1}^{i,k}$ for $k \in [a_i]$ are all filled iff we have assigned either all the words of $\{d_{i,k,T} : k \in [a_i]\}$, or all the words of $\{d_{i,k,F} : k \in [a_i]\}$, to the slots $hSlot_{j(i,k),1}^{i,k}$, $k \in [a_i]$.*

Proof. In one direction, if we have assigned to slots $hSlot_{j(i,k),1}^{i,k}$, $k \in [a_i]$, all the words of $\{d_{i,k,T} : k \in [a_i]\}$ or all the words of $\{d_{i,k,F} : k \in [a_i]\}$, then all the letters $s_1, \ldots, s_{a_i}$ appear exactly once in the end of these a_i slots. Because the words of $\{d_{i,k}\colon\ k \in [a_i]\}$ start exactly with this set of letters, there is a unique way to assign them properly to the slots $vSlot_{j(i,k),1}^{i,k}$, $k \in [a_i]$.

Conversely, assume that all the type 1 pairs of slots of x_i are filled. Because the only words that have the same length as slots $vSlot_{j(i,k),1}^{i,k}$, $k \in [a_i]$, are the words of $\{d_{i,k} : k \in [a_i]\}$, we know that in the end of slots $hSlot_{j(i,k),1}^{i,k}$, $k \in [a_i]$, each letter of $\{s_1, \ldots, s_{a_i}\}$ appears exactly once. It is not hard to see that no combination of words except $\{d_{i,k,T} : k \in [a_i]\}$ or $\{d_{i,k,F} : k \in [a_i]\}$, gives the same letters in the shared positions. □

Now we are ready to present the proof of Theorem 8.5.

Proof. We show a reduction from Restricted Exactly 1 (3,2)-SAT. We claim that

ϕ is a yes instance of RESTRICTED EXACTLY 1 (3,2)-SAT iff we can fill all the slots of the grid.

Suppose $f : X \to \{T, F\}$ is a truth assignment so that each clause of ϕ has exactly one true literal that satisfies ϕ.

We are going to show a way to fill all the slots of the grid. Each variable x_i appears in a_i literals; let $l(i,k)$, $k \in [a_i]$, be these literals and $j(i,k) \in [m]$, $k \in [a_i]$, be the indices of the clauses $c_{j(i,k)}$ that contain the corresponding literals.

For each variable x_i, fill the $3a_i$ slots $hSlot^{i,k}_{j(i,k),1}$, $hSlot^{i,k}_{j(i,k),2}$ and $vSlot^{i,k}_{j(i,k),1}$ for all $k \in [a_i]$ as follows. If $f(x_i) = T$, then:

- assign $d_{i,k,T}$ to $hSlot^{i,k}_{j(i,k),1}$ for all $k \in [a_i]$ and
- assign $d_{i,k,F}$ to $hSlot^{i,k}_{j(i,k),2}$ for all $k \in [a_i]$.

Otherwise ($f(x_i) = F$):

- assign $d_{i,k,F}$ to $hSlot^{i,k}_{j(i,k),1}$ for all $k \in [a_i]$ and
- assign $d_{i,k,T}$ to $hSlot^{i,k}_{j(i,k),2}$ for all $k \in [a_i]$.

Finally, in both cases, we assign the words of $\{d_{i,k} : k \in [a_i]\}$ to the slots $vSlot^{i,k}_{j(i,k),1}$ for $k \in [a_i]$ in any way they fit.

In order to fill the grid completely, for each $j \in [m]$, we assign to the nl_j slots, $vSlot^{i,k}_{j,2}$, the words $d^{k'}_j$ for $k' \in [nl_j]$ in any way they fit.

It is not hard to see that we have assigned words to slots of the same length. It remains to prove that the words we have assigned have the same letters in the shared positions.

First observe that for a variable x_i and the slots $hSlot^{i,k}_{j(i,k),1}$, $k \in [a_i]$, we have put either $\{d_{i,k,T} : k \in [a_i]\}$ or $\{d_{i,k,F} : k \in [a_i]\}$. Therefore, we know by Property A.10 that we can use the words of $\{d_{i,k} : k \in [a_i]\}$ in the slots $vSlot^{i,k}_{j(i,k),1}$, $k \in [a_i]$.

In the nl_j slots, $vSlot^{i,k}_{j,2}$, related to clause c_j, we have put the words $d^{k'}_j$, $k' \in [nl_j]$. One of these words starts with s_2 and the $nl_j - 1$ others start with s_1. We will show that the same holds for the words we have assigned in the nl_j slots $hSlot^{i,k}_{j,2}$.

Observe that each literal $l \in c_j$ can be described by a unique triplet (j, i, k) where $j \in [m]$ is the index of the clause, $i \in [n]$ is the index of the variable x_i on which l is built, and $k \in [a_i]$ is the number of times that x_i has appeared in ϕ until now. We claim that if the literal l described by (j, i, k) satisfies c_j, then the word assigned to $hSlot^{i,k}_{j,2}$ starts with s_2, otherwise it starts with s_1.

If l satisfies c_j, then either $l = x_i$ and $f(x_i) = T$ or $l = \neg x_i$ and $f(x_i) = F$. If $l = x_i$ (resp., $l = \neg x_i$), then we have assigned $d_{i,k,F}$ (resp., $d_{i,k,T}$) to $hSlot^{i,k}_{j,2}$ which starts with s_2 because $f(x_i) = T$ (resp., $f(x_i) = F$). If l does not satisfy c_j, then we used $d_{i,k,T}$ (resp., $d_{i,k,F}$) which starts with s_1.

Finally, because we assumed that each clause is satisfied by exactly one literal, we know that one of the clause words starts with s_2 and the other $nl_j - 1$ clause words start with s_1.

Conversely, we claim that if we can fill the whole grid, then we can construct a truth assignment $f : X \to \{T, F\}$ such that each clause of ϕ has exactly one true literal. Furthermore, one such assignment is the following:

$$f(x_i) = \begin{cases} T, \text{ if } d_{i,1,T} \text{ is assigned to } hSlot^{i,1}_{j(i,1),1}, \\ F, \text{otherwise.} \end{cases} \tag{A.9}$$

We first prove the following claim.

Claim A.11. *Let l be the literal of a clause c_j corresponding to the k-th appearance of some variable x_i. l is true under the truth assignment (A.9) iff the word in $hSlots^{i,k}_{j,2}$ starts with s_2.*

Proof. Due to its length, $hSlots^{i,k}_{j,2}$ receives either $d_{i,k,T}$ or $d_{i,k,F}$, and one of these words starts with s_2 whereas the other starts with s_1. Therefore, we have two cases. In the first case $d_{i,k,F}$ starts with s_2, then $d_{i,k,T}$ starts with s_1 and $l = x_i$. In the second case, $d_{i,k,T}$ starts with s_2, $d_{i,k,F}$ starts with s_1 and $l = \neg x_i$.

Assume that $d_{i,k,F}$ (resp., $d_{i,k,T}$) starts with s_2. By construction, we have that $l = x_i$ (resp., $l = \neg x_i$).

If $d_{i,k,F}$ (resp., $d_{i,k,T}$) is assigned to $hSlots^{i,k}_{j,2}$, then $d_{i,k,T}$ (resp., $d_{i,k,F}$) is assigned to $hSlots^{i,k}_{j,1}$. By Property A.10 we know that $hSlots^{i,1}_{j,1}$ must contain $d_{i,1,T}$ (resp., $d_{i,1,F}$) so $f(x_i) = T$ (resp., $f(x_i) = F$). So, if $d_{i,k,F}$ (resp., $d_{i,k,T}$) is assigned to $hSlots^{i,k}_{j,2}$, then we know that $f(x_i) = T$ (resp., $f(x_i) = F$) and $l = x_i$ (resp., $l = \neg x_i$) which means that l must be true under the truth assignment (A.9).

In reverse direction, if we have assigned $d_{i,k,T}$ (resp., $d_{i,k,F}$) to $hSlots^{i,k}_{j,2}$, then we know that $f(x_i) = F$ (resp., $f(x_i) = T$) and $l = x_i$ (resp., $l = \neg x_i$) thus, l is false under the truth assignment (A.9). □

Based on the previous claim, we will show that each clause has exactly one true literal under the truth assignment f given in (A.9).

For any $j \in [m]$ there are exactly nl_j pairs (i, k) where $i \in [n]$ and $k \in [a_i]$ such that the k-th appearance of x_i is in c_j. Let C_j be the set that contains contains all these pairs (i, k).

Observe that for each pair $(i, k) \in C_j$ there exists a pair of slots $hSlots^{i,k}_{j,2}$, $vSlots^{i,k}_{j,2}$ which share their first cells. Because the grid is full, the nl_j vertical slots, $vSlots^{i,k}_{j,2}$, where $(i, k) \in C_j$, must contain the words d^t_j, $t \in [nl_j]$. One of these words starts with s_2 and $nl_j - 1$ others start with s_1. Therefore, the same must hold for the words that have been assigned in the slots $hSlots^{i,k}_{j,2}$ for $(i, k) \in C_j$.

Using the previous claim, we know that one of the literals in c_j is true and the other $nl_j - 1$ are false under the truth assignment A.9. Therefore, if we can fill the whole grid,

then there exists a truth assignment such that exactly one literal of each clause of ϕ is true. $\square$

RÉSUMÉ

L'étude des problèmes combinatoires est une tâche importante car ils sont l'abstraction de très nombreux problèmes auxquels nous sommes confrontés dans notre vie depuis très longtemps.

Dans cette thèse, nous étudions des variantes de problèmes d'optimisation combinatoire classiques tels que : Subset Sum, Feedback Vertex Set et Coloring. Pour ces problèmes, nous présentons d'abord des résultats de NP-difficulté pour de nombreux cas particuliers. Ensuite, nous considérons des solutions d'approximation; nous donnons quelques algorithmes d'approximation et des bornes inférieures sur le rapport d'approximation sous des hypothèses standards. Enfin, nous présentons quelques algorithmes paramétrés qui construisent des solutions exactes et nous prouvons leur optimalité sous l'hypothèse ETH.

MOTS CLÉS

Optimisation combinatoire, Complexité, Approximation, Complexité paramétrée, Problèmes de graphes.

ABSTRACT

The study of combinatorial problems is an important task as they are abstraction of many problems that we face in our life for a very long time.

In this thesis we study variants of classical combinatorial optimization problems such as, Subset Sum, Feedback Vertex Set and Coloring. For these problems we first present NP-hardness results for many special cases. Then, we consider approximation solutions; we give some approximation algorithms and lower bounds on approximation ratio under standard assumptions. thesis Finally, we present some parameterized algorithms that compute exact solutions and we prove their optimality under the Exponential Time Hypothesis.

KEYWORDS

Combinatorial optimization, Complexity, Approximation, Parameterized complexity, Graph problems