Towards efficient and fault-tolerant optical networks : complexity and algorithms

Fatima Zahra Moataz

UNIVERSITÉ NICE SOPHIA ANTIPOLIS
ÉCOLE DOCTORALE DES SCIENCES ET TECHNOLOGIES DE
L'INFORMATION ET DE LA COMMUNICATION

THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'Université Nice Sophia Antipolis

Mention : INFORMATIQUE

Présentée et soutenue par
Fatima Zahra MOATAZ

Vers des Réseaux Optiques Efficaces et Tolérants aux Pannes : Complexité et Algorithmes

---

Thèse dirigée par : **David COUDERT**
préparée dans le projet **COATI (Inria, I3S (CNRS/UNS))**
financée par la Région Provence Alpes Côte d'Azur

Région
Provence-Alpes-Côte d'Azur

Soutenue le 30 Octobre 2015

Jury:

Rapporteurs:	Arie M.C.A. Koster	-	Université de RWTH Aachen (Allemagne)
	Yann Vaxès	-	Université de Aix-Marseille (France)
Examinateurs:	Alain Jean-Marie	-	Inria (Montpellier, France)
	Christian Laforest	-	ISIMA (Clermont-Ferrand, France)
	Hervé Rivano	-	Inria (Lyon, France)
	Jean-Claude Bermond	-	I3S et Inria (Sophia Antipolis, France)

To Mimina and my sisters ...

Acknowledgments

First of all, I would like to express my gratitude to my advisor, David Coudert, for welcoming me into the COATI team, directing this thesis, and for the faith he has shown in me.

I would also like to thank all of my co-authors and especially Jean-Claude Bermond and Nicolas Nisse. Collaborating with them was a real pleasure and their insights, feedback and guidance were an enormous help.

Many thanks to Pr Arie Koster and Pr Yann Vaxès for reviewing this manuscript and for providing me with remarks to improve it.

My warm thanks go to all the COATIs and the former COATIs. Thank you all for creating a delightful work environment.

I would also like to express my appreciation to all the friends and people I met during my thesis and who made my experience pleasant and enjoyable: Olfa, Bi, Sawsan, Alvinice, Ameni, Karol, Julio, Zineb, Guillaume, Patricia, Ewa ...

Special thanks to my dear friend Khouloud who has shared with me, day by day, my whole french experience. It would not have been the same without her.

Finally, all my gratitude to my family, my parents and my sisters Nabila, Hind and Sara. Thank you for all of your support, prayers and encouragements. You were, are and will always be my source of inspiration.

Sophia Antipolis
October 22, 2015

Towards Efficient and Fault-Tolerant Optical Networks: Complexity and Algorithms

Abstract:

We study in this thesis optimization problems with application in optical networks. The problems we consider are related to fault-tolerance and efficient resource allocation and the results we obtain are mainly related to the computational complexity of these problems.

The first part of this thesis is devoted to finding paths and disjoint paths. Finding a path is crucial in all types of networks in order to set up connections and finding disjoint paths is a common approach used to provide some degree of protection against failures in networks. We study these problems under different settings. We first focus on finding paths and node or link-disjoint paths in networks with asymmetric nodes, which are nodes with restrictions on their internal connectivity. Afterwards, we consider networks with star Shared Risk Link Groups (SRLGs) which are groups of links that might fail simultaneously due to a localized event. In these networks, we investigate the problem of finding SRLG-disjoint paths.

The second part of this thesis focuses on the problem of Routing and Spectrum Assignment (RSA) in Elastic Optical Networks (EONs). EONs are proposed as the new generation of optical networks and they aim at an efficient and flexible use of the optical resources. RSA is the key problem in EONs and it deals with allocating resources to requests under multiple constraints. We first study the static version of RSA in tree networks. Afterwards, we examine a dynamic version of RSA in which a non-disruptive spectrum defragmentation technique is used.

Finally, we present in the appendix another problem that has been studied during this thesis. It is a graph-theoretic problem referred to as minimum size tree-decomposition and it deals with the decomposition of graphs in a tree-like manner with the objective of minimizing the size of the tree.

Keywords: Asymmetric nodes, forbidden transitions, shared risk link group, routing and spectrum assignment, tree-decomposition, complexity.

Vers des Réseaux Optiques Efficaces et Tolérants aux Pannes: Complexité et Algorithmes

Résumé :

Nous étudions dans cette thèse des problèmes d'optimisation avec applications dans les réseaux optiques. Les problèmes étudiés sont liés à la tolérance aux pannes et à l'utilisation efficace des ressources. Les résultats obtenus portent principalement sur la complexité de calcul de ces problèmes.

La première partie de cette thèse est consacrée aux problèmes de trouver des chemins et des chemins disjoints. La recherche d'un chemin est essentielle dans tout type de réseaux afin d'y établir des connexions et la recherche de chemins disjoints est souvent utilisée pour garantir un certain niveau de protection contre les pannes dans les réseaux. Nous étudions ces problèmes dans des contextes différents. Nous traitons d'abord les problèmes de trouver un chemin et des chemins lien ou nœud-disjoints dans des réseaux avec nœuds asymétriques, c'est-à-dire des nœuds avec restrictions sur leur connectivité interne. Ensuite, nous considérons les réseaux avec des groupes de liens partageant un risque (SRLG) en étoile : ensembles de liens qui peuvent tomber en panne en même temps suite à un événement local. Dans ce type de réseaux, nous examinons le problème de recherche des chemins SRLG-disjoints.

La deuxième partie de cette thèse est consacrée au problème de routage et d'allocation de spectre (RSA) dans les réseaux optiques élastiques (EONs). Les EONs sont proposés comme la nouvelle génération des réseaux optiques et ils visent une utilisation plus efficace et flexible des ressources optiques. Le problème RSA est central dans les EONs. Il concerne l'allocation de ressources aux requêtes sous plusieurs contraintes. Nous étudions d'abord la version statique de RSA dans des réseaux sous forme d'arbres. Ensuite, nous examinons une version dynamique de RSA en utilisant une technique de défragmentation de spectre non-perturbatrice.

Enfin, nous présentons en annexe un autre problème traité durant cette thèse. Il s'agit d'un problème de théorie de graphes appelé décomposition arborescente de taille minimum, où l'objectif est de décomposer un graphe sous la forme d'un arbre de taille minimum.

Mots clés : Nœuds asymétriques, transitions interdites, groupe de liens partageant un risque, routage et allocation de spectre, décomposition arborescente, complexité.

Contents

Chapter 1

Introduction

Contents

We study in this thesis problems motivated by applications in optical networks. In this introduction, we briefly present the context and motivations behind the problems we consider. Then, we present our contributions and the thesis organization.

1.1 Context and motivation

Optical networks are at the heart of long-distance communication networks [Sen92, Muk97, RS02]. In optical networks, the data is encoded into pulses of light and carried on a very thin strand of glass or plastic called the optical fiber. This fiber is a very efficient transmission medium thanks to its large bandwidth and its low dispersion and attenuation properties [Sen92, GT98]. It is usually composed of a core of glass or plastic surrounded by a cladding of a different glass or plastic, with a lower index of refraction. The light signal travels in the core of the fiber through a series of reflections as illustrated in Fig. 1.1.

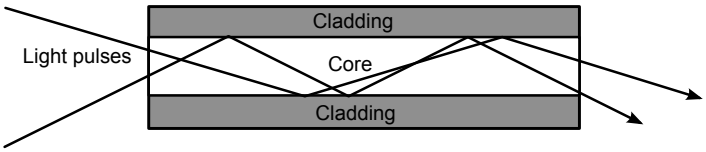


Figure 1.1: Light guided in an optical fiber [Wik]

The optical fiber enables the optical networks to provision very high data rates on very long distances and to constitute hence the backbone carrying the global data traffic. This traffic is however, growing on a daily basis at exponential rates and is not expected to slow down in the near future. According to a Cisco report [CS15], the global IP traffic has increased more than fivefold in the past 5 years, and will increase nearly threefold over the next 5 years. This makes it necessary to think of ways to use the optical resources more efficiently, in order to satisfy the growing traffic, and also to provide more fault-tolerance in order to protect this traffic on which the

world has become very dependent. In this thesis, we tackle mainly optimization problems related to these two desired properties of optical networks: fault-tolerance and efficiency.

Fault-tolerant optical networks. Fault-tolerance is an important requirement in optical networks [RS02]. These networks carry significant amounts of data and any failure can result in the loss of considerable traffic and the disruption of numerous services. Failures in optical networks can be full or partial, single or multiple, software or hardware caused. They can be triggered by a fiber cut or a power outage, by a human error or a natural disaster. Service providers have to account for all these types of failures while designing the schemes which ensure the resiliency of their networks. These schemes are generally of two types: protection schemes and restoration schemes [RM99]. In the protection schemes, redundant capacity is reserved in the network and used to reroute the traffic in case of failures. These schemes ensure a fast failure recovery (within 50 ms in general). In the restoration schemes, the resources ensuring the recovery are computed after the detection of a failure and the traffic is restored on a slower time scale [RS02]. We will be concerned in the first part of this thesis with protection schemes and more precisely with the *dedicated path protection* scheme (DPP).

The DPP consists in computing for each demand two paths. One path, called working or primary path, is used to transmit data under normal operation and the other one, the protection or alternate path, is reserved as a backup to carry the traffic in case of failures. A general requirement is that these paths have to be disjoint, so that at least one of them can survive a failure in the network. In the first part of this thesis, we investigate the DPP problem under two different settings.

Firstly, we consider networks with asymmetric nodes. These are nodes with restrictions or constraints on their internal connectivity; nodes in which the signal entering from a given ingress port can only reach a subset of the egress ports. In these networks, we consider single link or node failures and formulate the DPP as the problem of finding link or node-disjoint paths with some restrictions.

Secondly, we consider a different type of failures; namely multiple simultaneous localized failures captured by the notion of star Shared Risk Link Groups (SRLG). An SRLG is a set of links that are likely to fail at the same time due to a single event and star SRLGs are SRLGs localized around nodes. In networks with star SRLGs, we formulate the DPP as the problem of finding SRLG-disjoint paths, i.e. paths that share no risk of failing at the same time with respect to the multiple failures we consider.

Efficient use of the optical spectrum. In order to meet the growing demand, the optical networks need to evolve. New research has been conducted in this direction and a new generation of optical networks has emerged; one that is more efficient and more scalable: the *Elastic Optical Networks* (EONs) [GJLY12, JTK+09]. EONs offer the possibility to use the optical spectrum efficiently so as to allocate to de-

mands exactly the resources they need. The technologies enabling EONs, e.g. new transceivers and switches, are being actively developed and experiments are being conducted both in the laboratory and in the field. In parallel, optimization problems related to the resource allocation in EONs are being formulated and studied. In the second part of this thesis, we are concerned with one of these problems, namely, the problem of Routing and Spectrum Assignment (RSA).

In the RSA problem, we are given a network and a set of requests or connection demands and the aim is to provision the requests or set up the connections with the objective of minimizing the number of utilized optical resources and under constraints dictated by the new EON technologies. To provision a request, a path has to be found from its source to its destination (routing), and a spectrum interval has to be allocated to the request on all the links of this path (spectrum assignment).

We study different versions of the problem of RSA. Firstly, we study a static version of RSA in the particular case of optical networks in form of trees. In the static RSA, all of the demands are known in advance and the objective is to provision them all with minimum spectrum. Since in tree networks, there is only one path between each pair of nodes, the routing problem is already solved. RSA reduces then to the problem of Spectrum Assignment (SA). Secondly, we study a dynamic version of RSA. In the dynamic RSA, requests arrive and leave the network dynamically and the objective is to provision each request as it arrives to the network. The decisions made to allocate optical resources to an arriving request have to be wise in the sense that they should not decrease the chances of provisioning future requests nor cause more spectrum fragmentation.

1.2 Contributions and organization of the thesis

This thesis is organized in two parts and an appendix. In the first part, entitled *On paths and disjoint paths*, the focus is on the problems related to the fault-tolerance property in optical networks. In the second part, entitled *On routing and spectrum assignment*, the focus is on the problems related to the efficiency. The appendix is devoted to another approach followed in the study of the problem of routing and spectrum assignment as well as to another graph-theoretic problem tackled during this thesis. We use throughout the thesis graph models and various tools of algorithmic graph theory (see [BM08] for definitions). Our contributions for the problems we consider consist mainly of establishing computational complexity results; we prove the NP-completeness of many of these problems and then we design approximation algorithms or polynomial-time algorithms for special cases. We provide, in what follows, short descriptions of the chapters of this thesis including our contributions.

Part I: On paths and disjoint paths

Chapter 2: On paths in Networks with Asymmetric Nodes. We tackle in this chapter the problems of finding a path and disjoint paths in optical networks

with asymmetric nodes. The graph theoretic framework we use to study these problems is the graph with forbidden transitions where a transition is a pair of adjacent edges. In this model, the network is a graph $G = (V, E)$ and the asymmetric nodes are captured by a set of forbidden transitions $\mathcal{F} \subseteq E \times E$ associated to the graph. If a signal entering via link e (ingress port e) cannot leave via link e' (egress port e'), then the pair of edges $\{e, e'\}$ forms a forbidden transition.

Given a graph $G = (V, E)$, a set of forbidden transitions $\mathcal{F} \subseteq E \times E$ and two vertices $s, t \in V$, we first study the problem of finding a path avoiding forbidden transitions. This problem consists in finding a path, from s to t, which uses none of the forbidden transitions of $\mathcal{F}$; said otherwise it is forbidden for the path to consecutively use two edges forming a pair in $\mathcal{F}$. It has been proved in the literature that finding a non-elementary path (i.e. a path which can repeat vertices) avoiding forbidden transitions is polynomial while finding an elementary path is NP-hard in general graphs. We strengthen the hardness result and prove that finding an elementary path avoiding forbidden transitions is NP-hard even in well-structured graphs such as grids. We also prove that the problem is polynomial in graphs with bounded treewidth.

Afterwards, we show that finding $k \geq 2$ arc-disjoint non-elementary paths avoiding forbidden transitions can be done in polynomial time in directed graphs while it is NP-complete to find $k \geq 2$ vertex-disjoint non-elementary paths in directed as well as undirected graphs. We recently discovered that this last result follows also from results of [GLMM12] and we highlight this in section 2.5. On the positive side, we also prove that finding $k \geq 2$ vertex-disjoint non-elementary paths is polynomial in directed acyclic graphs.

Some of the results presented in this chapter were presented in *Algotel'2015* [KMMN15b] and *WG'2015* [KMMN15a].

Chapter 3: On Disjoint Paths in Networks with Star SRLGs. This chapter deals with the problem of k-diverse routing in networks with star Shared Risk Link Groups (SRLGs). An SRLG is a set of network links that fail simultaneously when a given event (risk) occurs and a star SRLG is an SRLG in which all links share an endpoint. In networks with SRLGs, the k-diverse routing problem consists in finding k pairwise SRLG-disjoint paths between a pair of nodes. The graph theoretic framework we use for studying problems in networks with SRLGs is the *colored graph* model. The network topology is modeled by a graph $G = (V, E)$ and the set of SRLGs by a set of colors $\mathcal{C}$. Each SRLG is modeled by a distinct color, and that color is assigned to all the edges corresponding to the network links subject to this SRLG. A colored graph is therefore defined by the triple $(V, E, \mathscr{C})$, where $\mathscr{C}$ is a *coloring* function, $\mathscr{C} : E \to 2^{\mathcal{C}}$, that assigns a subset of colors to each edge.

Using this model, we study the problem of finding SRLG-disjoint paths in networks with star SRLGs as the problem of finding k color-disjoint paths. We first prove that finding k SRLG-disjoint paths is NP-complete even for only two paths (this implies that the "polynomial" algorithm proposed in [LW05] to find two SRLG-

disjoint paths in a network with star SRLGs is not correct unless P=NP). Afterwards, we show that the problem can be solved in polynomial time in particular sub-cases which are relevant in practice. Namely, we solve the problem in polynomial time when the maximum degree is at most 4 or when the input network is a directed acyclic graph. Moreover, we show that the problem is fixed-parameter tractable when parameterized by the number of SRLGs. Finally we consider the problem of finding the maximum number of SRLG-disjoint paths in networks with SRLGs satisfying the star property. We prove that this problem is hard to approximate within $O(|V|^{1-\varepsilon})$ for any $0 < \varepsilon < 1$, where V is the set of nodes in the network. Then, we provide exact and approximation algorithms for relevant sub-cases.

The results presented in this chapter have been published in *TCS* [BCDM15] and have been presented in the *Student Workshop ACM CoNEXT'2012* [BCDM12], *Algotel'2013* [BCDM13b], and *DRCN'2013* [BCDM13a].

Part II: On Routing and Spectrum Assignment

We start this part by giving more details on Elastic Optical Networks (EONs) and by defining the problem of Routing and Spectrum Allocation (RSA) and its different versions.

Chapter 4: On Spectrum Assignment in Tree-Networks. In this chapter, we focus on the problem of static routing and spectrum assignment in elastic optical tree networks. In trees, since the routing is fixed, the RSA problem reduces to the problem of Spectrum Allocation (SA). In the SA problem, we are given a graph $\mathcal{N} = (N, L)$ modeling the optical network and a set of requests $\mathcal{R}$ such that each request $r \in \mathcal{R}$ has a path $P(r)$ and a demand $d(r)$. A spectrum assignment of $(\mathcal{N}, \mathcal{R})$ is a mapping f from $\mathcal{R}$ to $\mathbb{N}^*$ such that for every pair of requests $r, r' \in \mathcal{R}$, if $P(r)$ and $P(r')$ share a link, then $\{f(r), \ldots, f(r)+d(r)-1\} \cap \{f(r'), \ldots, f(r')+d(r')-1\} = \emptyset$. The objective of the SA problem is to find a spectrum assignment f which minimizes the number of used spectrum slots, i.e. the smallest integer $s(f)$ such that for each request $r \in \mathcal{R}$, $f(r) + d(r) - 1 \leq s(f)$.

In trees, even though the routing is fixed, the spectrum allocation (SA) is NP-hard. We survey the complexity and approximability results that have been established for the SA in trees and prove new results for stars and binary trees. Namely, we prove that SA is NP-hard in undirected stars of 3 links and in directed stars of 4 links, and show that it can be approximated within a factor of 4 in general stars. Afterwards, we use the equivalence of SA with a graph coloring problem (interval coloring) to find constant-factor approximation algorithms for SA on binary trees with special demand profiles. Namely, we examine the cases where the demands are in a set $\{k, kX\}$ $(k, X \in \mathbb{N}^*)$, in a set $\{kX, k(X+1)\}$ $(k, X \in \mathbb{N}^*)$, or bounded by D. For the latter case, we give a general approximation when the demands are bounded by $D \in \mathbb{N}$ and then give better approximations for the cases where the demands are bounded by $D \in \{3, 4, 5, 6\}$.

Results presented in this chapter were presented in *Algotel'2015* [Moa15].

Chapter 5: On Dynamic RSA with Push-Pull. In this chapter, we tackle the problem of dynamic routing and spectrum assignment with the use of a non-disruptive defragmentation technique called Push-Pull. In dynamic traffic scenarios, as requests arrive and leave, the optical spectrum becomes fragmented. This means that small and unusable fragments of spectrum are accumulated. This fragmentation can cause the blocking of a request even though the overall free resources in the network can provision it. To deal with this problem, many defragmentation techniques are proposed to consolidate the existing free spectrum. In this chapter, we consider one of these techniques, namely the Push-Pull technique [CSS+12]. This technique uses the characteristics of the EONs and of the flexible transponders to shift the spectrum allocated to a request without disrupting the traffic. This is done as follows. The frequency at the transmitter is pushed at a frequency sweep rate, and since the difference between the frequencies of the transmitter and the receiver should not exceed a given offset, the frequency at the receiver is pulled towards the frequency of the transmitter. This technique can only perform spectrum re-allocation and not re-routing. Furthermore, the order of the requests in the spectrum interval does not change. We propose algorithms to solve the dynamic RSA with the use of Push-Pull while optimizing the length of the routing path and the delay of the defragmentation.

Results presented in this chapter were presented in *Algotel'2014* [CJM14]

Appendix

In the appendix, we present another direction we followed in the study of the static routing and spectrum assignment problem in elastic optical networks. We also present another problem tackled during this thesis; it is a problem not related to optical networks and it concerns tree-decompositions with minimum size.

Appendix A: On static Routing and Spectrum Assignment. We present in this chapter column generation formulations for the problem of routing and spectrum assignment in a static setting. The formulations we present are inspired from a column generation model for the problem of routing and wavelength assignment; the counterpart of RSA in the classical optical networks.

Appendix B: Minimum Size Tree-Decompositions. We study in this chapter, a problem of graph theory concerning graph decomposition. Namely we study the problem of minimum size tree-decomposition. A tree-decomposition of a graph is a way to represent it by a family of subsets (bags) of its vertex-set organized in a tree-like manner and satisfying some connectivity property. Tree-decompositions have been widely studied for their algorithmic applications especially that they are the corner-stone of many dynamic programming algorithms for solving graph problems. We consider the problem of computing a tree-decomposition of a graph with width (i.e number of vertices per bag) at most k and minimum size (i.e. number of

bags). We prove that the problem is NP-complete for any fixed $k \geq 4$ and polynomial for $k \leq 2$; for $k = 3$, we show that it is polynomial in the class of trees and 2-connected outerplanar graphs.

The results of this chapter were presented in *ICGT'2014* [LMN14] and *LAGOS'2015* [LMNS15].

1.3 List of publications

We list in this section the publications associated to the research presented in this thesis. In all papers the authors are alphabetically ordered.

International journals

[BCDM15] J.-C. Bermond, D. Coudert, G. D'Angelo, and F. Z. Moataz. Finding disjoint paths in networks with star shared risk link groups. *Theoretical Computer Science*, 579(0):74 – 87, 2015

International conferences

[BCDM12] J.-C. Bermond, D. Coudert, G. D'Angelo, and F. Z. Moataz. Diverse routing in networks with star SRLGs. In *Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop*, CoNEXT Student '12, pages 23–24, New York, NY, USA, 2012. ACM.

[BCDM13a] J.-C. Bermond, D. Coudert, G. D'Angelo, and F. Moataz. SRLG-diverse routing with the star property. In *Design of Reliable Communication Networks (DRCN), 2013 9th International Conference on the*, pages 163–170, March 2013.

[LMN14] B. Li, F. Z. Moataz, and N. Nisse. Minimum Size Tree-Decompositions. In *9th International colloquium on graph theory and combinatorics (ICGT)*, Grenoble, France, 2014.

[LMNS15] B. Li, F. Z. Moataz, N. Nisse, and K. Suchan. Minimum Size Tree-decompositions. In *LAGOS 2015 – VIII Latin-American Algorithms, Graphs and Optimization Symposium*, Beberibe, Ceará, Brazil, May 2015.

[KMMN15a] M. M. Kanté, F. Z. Moataz, B. Momège, and N. Nisse. Finding Paths in Grids with Forbidden Transitions. In *WG 2015, 41st International Workshop on Graph-Theoretic Concepts in Computer Science*, Munich, Germany, June 2015.

National conferences

[BCDM13b] J.-C. Bermond, D. Coudert, G. D'Angelo, and F. Z. Moataz. Diverse Routing with Star SRLGs. In *15èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (AlgoTel)*, pages 1–4, Pornic, France, May 2013.

[CJM14] D. Coudert, B. Jaumard, and F. Z. Moataz. Dynamic Routing and Spectrum Assignment with Non-Disruptive Defragmentation. In *ALGOTEL 2014 – 16èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, pages 1–4, Le Bois-Plage-en-Ré, France, June 2014.

[KMMN15b] M. M. Kanté, F. Z. Moataz, B. Momège, and N. Nisse. On paths in grids with forbidden transitions. In *ALGOTEL 2015 - 17èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, Beaune, France, June 2015.

[Moa15] F. Z. Moataz. On Spectrum Assignment in Elastic Optical Tree-Networks. In *ALGOTEL 2015 - 17èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, Beaune, France, June 2015. *Best student paper award.*

Part I

Paths and Disjoint Paths

The aim in networks in general and optical networks in particular is to ensure the communication between different nodes. This relies primarily on finding a path from a source to a destination. This problem can be solved efficiently if there are no constraints on the network using well-known algorithms such as the Dijkstra's algorithm and the Bellman-Ford's algorithm. However, finding a path becomes more challenging when some constraints are introduced. In the first chapter of this part (Chapter 2) we focus on the problem of finding a path in networks with constraints on the internal connectivity of their nodes. We refer to these networks as networks with asymmetric nodes.

Another aim in optical networks is to provide fault-tolerance services and protect the connections against failures (fiber cuts, failures of transceivers, power outages, etc). Towards this purpose, many protection and restoration schemes have been proposed [RM99, RS02]. In the protection schemes, backup resources are reserved and dedicated to recover the traffic in case of failures. In the restoration schemes, the backup resources are found upon the occurrence of a failure. Protection schemes provide better guarantees on the recovery of the network and they perform on a fast time scale. They can be broadly classified under four categories defined by whether they are dedicated or shared, and whether they provide link or path protection [RM99]. In dedicated protection, each connection has dedicated resources to ensure its protection. In the shared protection, on the other hand, the reserved resources are shared between many connections. In link protection, resources are reserved such that the traffic on a failed link can be rerouted around that link. In path protection, resources are reserved to restore the traffic between the endpoints of a path upon its failure. We focus in this part on the *dedicated path protection* (DPP) scheme. The DPP consists in computing for each demand two paths. One path, called working or primary path, is used to carry the traffic under normal operation and the other one, the protection or alternative path, is reserved to reroute the traffic when failures occur. A general requirement is that these paths have to be disjoint, so that at least one of them can survive a single failure in the network. In this part, we investigate the DPP problem in two different contexts. First, in Chapter 2, we consider single link or node failures in networks with asymmetric nodes. In this setting, the DPP problem consists in finding link or node-disjoint paths. Second, in Chapter 3, we consider multiple simultaneous localized failures captured by the notion of star Shared Risk Link Groups (SRLG). In this setting, the DPP problem consists in finding SRLG-disjoint paths.

Chapter 2

On Paths in Networks with Asymmetric Nodes

Contents

In this chapter, we study the problem of finding paths in networks with constraints on the internal connectivity of their nodes. This is particularly the case of optical networks with asymmetric nodes and road networks with prohibited turns. The results in Section 2.4 are joint work with N. Nisse, B. Momège and M. M. Kanté and they were presented in *WG'2015* [KMMN15a] and *Algotel'2015* [KMMN15b]. The results in Section 2.5 are joint work with N. Nisse.

2.1 Introduction

In optical networks, nodes can be highly asymmetric with respect to their switching capabilities as pointed out in [BLGM09]. This means that an optical node might not be fully connected internally and that, consequently, signal on a certain ingress port cannot reach all of the egress ports. As explained in [BLGM09, CHW+13, HTTN11], a node can be asymmetrically configured for many reasons such as the limitation on the number of physical ports of the optical switch components and the low cost of asymmetric nodes compared to symmetric ones. Figure 2.1 from [CHW+13] shows a

diagram of a 4-degree Reconfigurable Optical Add-Drop Multiplexer (ROADM) with four directions named east, west, north and south. In Figure 2.1-a, the ROADM is symmetrically configured and an ingress port can reach anyone of the egress ports. In Figure 2.1-b, an asymmetric architecture of the ROADM is presented. In this architecture, cheaper equipment is used at the ports and the optical signal coming from any of the four directions can only reach two of the other three directions.

Asymmetric nodes are not exclusive to optical networks but can be found in road networks as well. In road networks, it is possible that some roads are closed due to traffic jams, construction, etc. It is also frequent to encounter no-left, no-right and no U-turn signs at intersections. With these prohibited roads and turns, it might be not possible to go from a given road to any other one at an intersection.

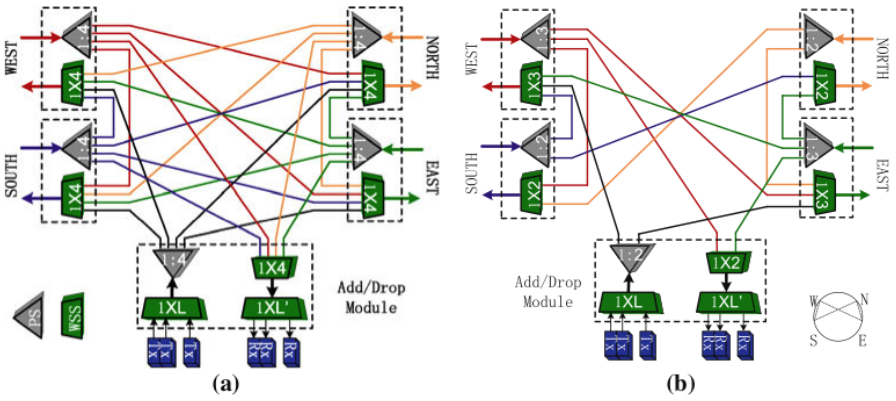


Figure 2.1: Diagram of a 4-degree ROADM [CHW+13]: **a.** symmetric case; **b.** asymmetric case

The existence of asymmetric nodes in a network needs to be accounted for while solving key problems such as finding one or many disjoint paths. In road networks for instance, a path should not use a forbidden turn, and in optical networks a path should cross an asymmetric node according to the way it is internally connected. The graph theoretic framework we use to study problems in networks with asymmetric nodes is the graph with forbidden transitions where a transition is a pair of adjacent edges. In this model, the network is a graph $G = (V, E)$ and the asymmetric nodes are captured by a set of forbidden transitions $\mathcal{F} \subseteq E \times E$ associated to the graph. In this setting, finding a path or k disjoint paths consists in finding a path or k disjoint paths avoiding forbidden transitions. A path $P = (v_0, \ldots, v_q)$ is avoiding forbidden transitions if it contains none of the transitions of $\mathcal{F}$, i.e., $\{\{v_{i-1}, v_i\}, \{v_i, v_{i+1}\}\} \notin \mathcal{F}$ for $i \in \{1, \ldots, q-1\}$.

Related work

Finding a path. When the problem of finding a path avoiding forbidden transitions (PAFT) is studied, a distinction has to be made according to whether the path to find is elementary (cannot repeat vertices) or non-elementary. Indeed, PAFT can be solved in polynomial time [GM08, ABP96, SJK03, Win02] for the non-elementary case while finding an elementary path avoiding forbidden transitions has been proved NP-complete in [Sze03]. We will discuss in more details results related to each problem in Section 2.3.

PAFT is a special case of the problem of finding a path avoiding forbidden paths (PFP) introduced in [VD05]. Given a graph G, two vertices s and t, and a set $\mathcal{S}$ of forbidden paths, PFP aims at finding an s-t-path which contains no path of $\mathcal{S}$ as a subpath. When the forbidden paths are composed of exactly two edges, PFP is equivalent to PAFT. Many papers address the non-elementary version of PFP, proposing exact polynomial-time solutions [VD05, HCD09, AL13]. The elementary counterpart has been recently studied in [PG13] where a mathematical formulation is given and two solution approaches are developed and tested. The computational complexity of the elementary PFP can be deduced from the complexity of PAFT which has been established in [Sze03]. Szeider proved in [Sze03] that finding an elementary path avoiding forbidden transitions is NP-complete and gave a complexity classification of the problem according to the types of the forbidden transitions. In more details, in [Sze03], Szeider defines for each vertex $v \in G$; a transition graph $T(v)$. The vertices of $T(v)$ are the edges incident to v and there is an edge between two vertices of $T(v)$ if and only if the two corresponding edges form a forbidden transition. Afterwards, he shows that with respect to the class $\mathcal{A}$ including $T = \{T(v) \mid v \in V\}$, PAFT is either NP-complete or it can be solved in linear time.

Another problem that is a generalization of PAFT is the problem of finding a path avoiding forbidden pairs (PAFP). In this problem, we are given a graph G, two vertices s and t and a set $\mathcal{K}$ of forbidden pairs of edges (or vertices) and the objective is to find a path from s to t which contains at most one edge (or vertex) of each pair of the set $\mathcal{K}$. When each pair of $\mathcal{K}$ consists of adjacent edges, then PAFP is equivalent to PAFT. This problem has been proven NP-complete in [GMO76] and in [KP09], other complexity results with respect to the structure of the pairs have been established.

PAFT is a generalization of the problem of finding a properly colored path in an edge-colored graph (PEC). Given an edge-colored graph G and two vertices s and t, the PEC problem aims at finding an s-t-path such that any consecutive two edges have different colors. It is easy to see that PEC is equivalent to PAFT when the set of forbidden transitions consists of all pairs of adjacent edges that have the same color. The PEC problem is polynomial-time solvable in undirected graphs [Sze03] and it has been proved to be NP-complete in directed graphs [GLMM13] which directly implies that the PAFT problem is NP-complete in directed graphs[1].

[1] Note that, in [GLMM13], the authors state that their result can be extended to planar directed

In this chapter, we study the elementary version of the PAFT problem in planar graphs and more particularly in grids. Our interest for planar graphs is motivated by the fact that they are closely related to road networks. They are also an interesting special case to study while trying to capture the difficulty of the problem. Furthermore, to the best of our knowledge, this case has not been addressed before in the literature. We also study the elementary PAFT in general graphs with bounded treewidth.

Finding disjoint paths. The problem of finding disjoint paths avoiding forbidden transitions has not been addressed much before. In the context of dedicated path protection in optical networks with asymmetric nodes, [HTTN11] study the problem of finding two vertex-disjoint paths. In the paper, the authors present a heuristic to solve the problem. The heuristic uses graph transformations in order to adapt the classical Bhandari's algorithm [Bha97] used to find 2 vertex-disjoint paths with minimum cost. These transformations are designed to avoid some trap scenarios caused by the asymmetric nodes. It is not specified in [HTTN11] whether the paths to find are elementary or not and the algorithm used to find one path is not detailed. In the context of road networks, the problem of alternate routing has been studied in many papers such as [BDGS11, ADGW13, LS12]. In this problem, the objective is not to find one route but several different ones. The papers discuss the different criteria to choose good alternate routes and to quantify their quality and develop algorithms to find them. In [DGSB10], disjointness is considered among the criteria to choose alternate routes and a classical heuristic is proposed to compute disjoint routes by iteratively computing the shortest path, adding it to the solution, and deleting all its edges.

As discussed earlier, finding an elementary path avoiding forbidden transitions is an NP-complete problem. It is straightforward then that the problem of finding k disjoint paths (vertex-disjoint or edge-disjoint) is NP-complete as well if the paths are required to be elementary. For this reason, we focus in this chapter on the problem of finding k disjoint non-elementary paths avoiding forbidden transitions. This problem still captures the protection and the alternate routing and at the same time it is a relaxation of the elementary version and may be hence easier to solve.

Contribution

In this chapter we study two problems; the problem of finding an elementary path avoiding forbidden transitions and the problem of finding k disjoint non-elementary paths avoiding forbidden transitions. We establish the following results:

- We demonstrate that when the path is required to be elementary, the PAFT problem is NP-complete in grids. This NP-completeness result strengthens

graphs. However, there is a mistake in the proof of the corresponding Corollary 7: to make their directed graph planar, vertices are added when edges intersect. Unfortunately, this transformation does not preserve the fact that the path is elementary.

the one of Szeider [Sze03] established in 2003 and extends to the problems of PFP and PAFP.

- We prove that the problem of finding an elementary path avoiding forbidden transitions can be solved in time $O(k\Delta^2(3k\Delta)^{2k}n)$ in n-node graphs with treewidth at most k and maximum degree Δ. In other words, we prove that the PAFT problem is FPT in $k+\Delta$.

- We show that finding k arc-disjoint non-elementary paths avoiding forbidden transitions can be done in polynomial time in directed graphs while it is NP-complete to find k vertex-disjoint non-elementary paths in directed as well as undirected graphs[2]. We also prove that finding k vertex-disjoint paths is polynomial in directed acyclic graphs.

The chapter is organized as follows. In Section 2.2, we present some notations and definitions. In Section 2.3, we discuss in details the difference between finding an elementary and non-elementary path avoiding forbidden transitions. Afterwards, in Section 2.4, we study the problem of finding an elementary path avoiding forbidden transitions. We present the results obtained for the problem of finding k disjoint paths avoiding forbidden transitions in Section 2.5. Finally, we give perspectives and open questions in Section 2.6.

2.2 Notations and definitions

We model the network as an undirected graph $G = (V, E)$, where the vertices represent the nodes and the edges represent the links. Given a subgraph H of G, a *transition* in H is a (not ordered) set of two distinct edges of H incident to a same vertex. Namely, $\{e, f\}$ is a transition if $e, f \in E(H)$ and $e \cap f \neq \emptyset$. Let $\mathcal{T}$ denote the set of all transitions in G. Let $\mathcal{F} \subseteq \mathcal{T}$ be a set of *forbidden transitions*. A transition in $\mathcal{A} = \mathcal{T} \setminus \mathcal{F}$ is said *allowed*.

A *walk* in G is any sequence $(v_0, v_1, \cdots, v_r)$ of vertices such that $e_i = \{v_i, v_{i+1}\} \in E$ for any $0 \leq i < r$. Given two vertices s and t in G, a walk $P = (v_0, v_1, \cdots, v_r)$ is called an *s-t-walk* if $v_0 = s$ and $v_r = t$. If the walk uses distinct edges, i.e., for any $0 \leq i, j < r$, $e_i \neq e_j$, then it is called a *trail* (or a non-elementary path). If the walk uses distinct vertices, i.e., for any $0 \leq i, j \leq r$, $v_i \neq v_j$, then it is called a *path* (or an elementary path). Finally, a walk, a trail or a path $P = (v_0, v_1, \cdots, v_r)$ is *$\mathcal{F}$-valid* if any transition in P is allowed, i.e., $\{e_i, e_{i+1}\} \notin \mathcal{F}$ for any $0 \leq i < r$. We say that k walks, trails or paths are vertex-disjoint if they share no vertex and edge-disjoint if they share no edge.

Since the model of a directed graph is also relevant in the networks we consider, and because the problems we consider and particularly the problems of disjoint trails seem to be different according to the class of graphs, we also prove and state

[2] After proving these results, we have found out recently that even stronger NP-completeness results, for the problem of finding vertex-disjoint non-elementary paths, can be deduced from some related problems. We discuss this in more details in the beginning of Section 2.5.

results in terms of directed graphs (especially in Section 2.5). For a directed graph $D = (V, A)$, a transition in a subgraph K of D, is a pair of arcs $(u, v), (u', v')$ such that $v = u'$. All of the definitions above apply to directed graphs when considering arcs instead of edges.

2.3 Walks, trails and paths avoiding forbidden transitions

Finding a walk, a trail or a path are equivalent problems in directed and undirected graphs. However, when the graph has a set of forbidden transitions $\mathcal{F}$ and the walk, the trail or the path is required to be $\mathcal{F}$-valid, the equivalence between the three problems does not hold. In this section, we discuss the relation between the following three problems in both directed and undirected graphs.

Problem 1 (Finding a Walk Avoiding Forbidden Transitions, WAFT)**.** *Given a graph $G = (V, E)$, a set $\mathcal{F}$ of forbidden transitions and two vertices $s, t \in V$. Is there an $\mathcal{F}$-valid s-t-walk in G?*

Problem 2 (Finding a Trail Avoiding Forbidden Transitions, TAFT)**.** *Given a graph $G = (V, E)$, a set $\mathcal{F}$ of forbidden transitions and two vertices $s, t \in V$. Is there an $\mathcal{F}$-valid s-t-trail in G?*

Problem 3 (Finding a Path Avoiding Forbidden Transitions, PAFT)**.** *Given a graph $G = (V, E)$, a set $\mathcal{F}$ of forbidden transitions and two vertices $s, t \in V$. Is there an $\mathcal{F}$-valid s-t-path in G?*

Table 2.1 summarizes the complexities of these three problems.

	WAFT	**TAFT**	**PAFT**
Directed Graphs	WAFT $\iff$ TAFT Polynomial [Win02, ABP96]		NP-complete [Sze03]
Undirected Graphs	Polynomial	Polynomial (Theorem 1)	NP-complete [Sze03]

Table 2.1: Complexity of WAFT, TAFT, and PAFT.

2.3.1 Directed graphs

In directed graphs, any walk avoiding forbidden transitions contains a trail avoiding forbidden transitions, and since any trail is a walk, the problems of WAFT and TAFT are equivalent. The problems TAFT and PAFT are however different, because the existence of an $\mathcal{F}$-valid trail does not imply the existence of an $\mathcal{F}$-valid path as illustrated in the example of Figure 2.2. In the example, there exists an $\mathcal{F}$-valid trail from s to t, namely $(s, v_5, v_6, v_7, v_4, v_2, v_1, v_4, t)$ while there is no $\mathcal{F}$-valid path from s to t.

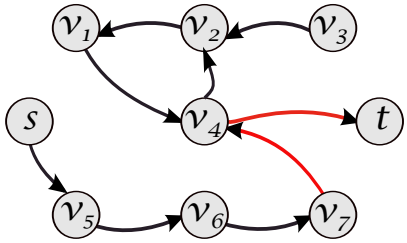


Figure 2.2: A directed graph G with a set of forbidden transitions $\mathcal{F} = \{(v_7, v_4), (v_4, t)\}$

In fact, as pointed out in the introduction, TAFT and PAFT have different complexities. The PAFT problem is NP-complete [Sze03] while TAFT can be solved in polynomial time [Win02, ABP96]. The TAFT problem has been addressed in the context of routing in road networks modeled as directed graphs. Two main solution approaches can be distinguished:

Dual graph technique. This technique has been used in [Win02, ABP96]. Given a directed graph $D = (V, A)$, two vertices s and t in V and a set of forbidden transitions $\mathcal{F}$, a dual graph $D' = (V', A')$ is constructed (which is a subgraph of the line graph [BLS99]). Vertices of D' correspond to arcs of D and arcs of D' correspond to allowed transitions. In more details, D' is built as follows.

- A node $v_a \in V'$ is associated to each arc $a \in A$.
- An arc (v_a, v_b) is added to A' if (a, b) is an allowed transition in D.
- Two additional vertices s' and t' are added to V'. For each outgoing arc a of s, an arc (s', v_a) is added to A' and for each ingoing arc b of t', an arc (v_b, t') is added to A'.

Finding a trail avoiding forbidden transitions from s to t in D is equivalent to finding a path from s' to t' in D'. In fact, a trail $(s, u_1, \ldots, u_q, t)$ in D corresponds to a path $(s', v_{(u_1,u_2)}, \ldots, v_{(u_{q-1},u_q)}, t')$ and vice-versa.

Labeling algorithm technique. This technique is used in [GM08] to find a trail avoiding forbidden turns in a directed road network. The authors propose an algorithm which is an extension of the Dijkstra's algorithm. The key idea is to maintain labels and distances at arcs instead of vertices. At the beginning of the algorithm, temporary labels are assigned to the outgoing arcs of the source. Afterwards, at every iteration, the temporarily labeled arc at minimum distance from the source is permanently labeled and the distances to the arcs which form allowed transitions with it are updated. The algorithm stops when the destination is reached or all arcs are permanently labeled.

Remark for DAGs. Note that in DAGs, since there are no cycles, every trail is a path. This implies that the PAFT problem is equivalent to TAFT and can be solved in polynomial time in DAGs.

2.3.2 Undirected graphs

Unlike in directed graphs, the existence of an $\mathcal{F}$-valid walk does not imply the existence of an $\mathcal{F}$-valid trail. This is mainly due to the fact that an edge can be crossed in two different directions. In the example presented in Figure 2.3, there exists an $\mathcal{F}$-valid walk from s to t, namely $(s, v_5, v_6, v_7, v_4, v_2, v_1, v_3, v_2, v_4, t)$ while there is no $\mathcal{F}$-valid trail from s to t. The TAFT problem is then different from WAFT in undirected graphs, and it is also different from PAFT. In fact, the undirected version of the example of Figure 2.2 illustrates the fact that the existence of an $\mathcal{F}$-valid trail does not imply the existence of an $\mathcal{F}$-valid path.

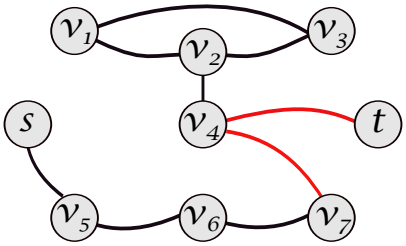


Figure 2.3: An undirected graph G with a set of forbidden transitions $\mathcal{F} = \{\{v_7, v_4\}, \{v_4, t\}\}$

In undirected graphs, the PAFT problem is NP-complete [Sze03] and WAFT can be solved in polynomial time as follows. Given a graph G, two vertices s and t and a set of forbidden transitions $\mathcal{F}$, we first transform G to the corresponding symmetric digraph D where each edge $\{u, v\}$ is replaced by two arcs (u, v) and (v, u). Afterwards we associate to D, the set of forbidden transitions $\mathcal{F}'$ defined as follows. If $\{\{u, v\}, \{v, w\}\} \in \mathcal{F}$, then $\{(u, v), (v, w)\}, \{(w, v), (v, u)\} \in \mathcal{F}'$ and if $\{\{u, v\}, \{v, u\}\} \in \mathcal{F}$, then $\{(u, v), (v, u)\} \in \mathcal{F}'$. It is easy to check that an $\mathcal{F}$-valid walk from s to t in G is equivalent to finding an $\mathcal{F}'$-valid walk from s to t in D. The latter problem can be solved using the techniques described in the previous subsection.

As for the TAFT problem, we prove the following theorem.

Theorem 1. *The TAFT problem can be solved in polynomial time in undirected graphs with forbidden transitions.*

Proof. Let $G = (V, E)$ be a graph, s and t two vertices of V and $\mathcal{F}$ a set of forbidden transitions. We build the dual graph $G' = (V', E')$ of G as we explained for the directed graphs: for each edge $e \in E$, we associate a vertex v_e in V' and we connect two vertices v_e and v'_e of V' if and only if $\{e, e'\}$ is an allowed transition. We also add a vertex s' (resp. t') connected to all vertices $v_e \in V$ such that e is incident to

s (resp. t') in G. Furthermore, we color the edges of G' with colors $\{c_v \mid v \in V\}$ as follows. To each edge $\{v_e, v_f\}$ such that $e \cap f = \{u\}$ of G', we associate the color c_u, and to each edge $\{s', v_e\}$ (resp. $\{v_e, t'\}$), we associate the color c_s (resp. c_t). We are going to prove that finding an $\mathcal{F}$-valid trail from s to t in G is equivalent to finding a properly colored path (PEC) from s' to t' in the edge-colored graph G'. As we have stated in the introduction, in the problem of finding a PEC, we are given an edge-colored graph G_c and two vertices s and t, the objective is to find an s-t-path such that any consecutive two edges have different colors. It has been proved in [Sze03] that finding a PEC can be done in polynomial time in undirected graphs.

Let $T = (s, u_1, \ldots, u_k, t)$ be an $\mathcal{F}$-valid trail from s to t in G. Let $e_0 = \{s, u_1\}$, $e_k = \{u_k, t\}$ and $e_i = \{u_i, u_{i+1}\}$ for $i \in \{1, \ldots, k-1\}$. The path $(s', v_{e_0}, \ldots, v_{e_k}, t')$ is a properly colored path in G'. In fact, for each $i \in \{0, \ldots, k-1\}$, $f_i = \{v_{e_i}, v_{e_{i+1}}\}$ is an edge in G' since $\{e_i, e_{i+1}\}$ is an allowed transition in G. Furthermore, if we suppose that two consecutive edges f_i and f_{i+1} have the same color, then the edges e_i, e_{i+1}, and e_{i+2} share a vertex v which implies that the trail T uses twice the edge e_{i+1} (in two directions) which is not possible in a trail.

Now let us suppose that there is a properly colored path $P = (s', v_{e_0}, \ldots, v_{e_k}, t')$ from s' to t' in G'. Let us prove that $T = (s, e_0, \ldots, e_k, t)$ is an $\mathcal{F}$-valid trail in G. Let us first suppose that T is not a trail, this implies that either there exist $0 \leq i < j \leq k$ such that $e_i = e_j$ or that T uses consecutively three edges sharing a vertex (and hence that one of these edges is used twice in two different directions). The first case would imply that there exist $0 \leq i < j \leq k$ such that $v_{e_i} = v_{e_j}$, and this is not possible since P is a path. The second case would imply that P is not properly colored, which is not the case. The sequence of edges T is hence a trail from s to t. Furthermore, since $f_i = \{v_{e_i}, v_{e_{i+1}}\}$, for $i \in \{0, \ldots, k-1\}$, is an edge in G', then $\{e_i, e_{i+1}\} \notin \mathcal{F}$. This implies that T is an $\mathcal{F}$-valid trail.

□

2.4 A path avoiding forbidden transitions

We focus in this section on the PAFT problem. We prove first that the PAFT problem is NP-complete in grids and then we present an FPT algorithm when the parameter is the sum of the treewidth and the maximum degree. All proofs are done for the undirected graph model but they also work for the directed model as we will point out at the end of the first subsection.

2.4.1 NP-completeness in grids

We start by proving that the PAFT problem is NP-complete in grids. For this purpose, we first prove that it is NP-complete in planar graphs with maximum degree at most 8 by a reduction from 3-SAT. Then, we propose simple transformations to reduce the degree of the vertices and prove that the PAFT problem is NP-complete in planar graphs with degree at most 4. Finally, we prove it is NP-complete in grids.

Lemma 1. *The PAFT problem is NP-complete in planar graphs with maximum degree* 8*.*

Proof. The problem is clearly in NP. We prove the hardness using a reduction from the 3-SAT problem. Let Φ be an instance of 3-SAT, i.e., Φ is a boolean formula with variables $\{v_1, \cdots, v_n\}$ and clauses $\{C_1, \cdots, C_m\}$. We build a grid-like planar graph G where rows correspond to clauses and columns correspond to variables. We define a set of forbidden transitions $\mathcal{F}$ and two vertices s and t in G such that an $\mathcal{F}$-valid path from s to t passes through every row and for each row passes through every column. The way the path crosses the columns of a row defines a truth assignment which satisfies the corresponding clause. Moreover, the way the graph G is built forces the same truth assignment to be defined at all rows. In what follows, we present the gadgets used to build a row, the way we build a clause-graph corresponding to a row and then the way we combine the rows to build the main graph G.

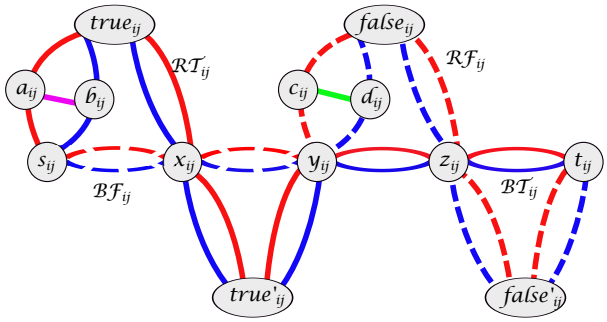
Please note that the colors are only used to make the presentation easier. Note also that we use a multigraph in the reduction for the sake of simplicity. This multigraph can easily be transformed into a simple graph by subdividing each edge once and without changing the maximum degree. We present in what follows the gadgets needed to build the graph G.

Gadget G_{ij}. For any $i \leq n$ and $j \leq m$, we define the gadget G_{ij} depicted in Figure 2.4a and that consists of 4 edge-disjoint paths from s_{ij} to t_{ij}: two "blue" paths $\mathcal{B}T_{ij}$ and $\mathcal{B}F_{ij}$, and two "red" paths $\mathcal{R}T_{ij}$ and $\mathcal{R}F_{ij}$ defined as follows.

- $\mathcal{R}T_{ij} = (s_{ij}, a_{ij}, true_{ij}, x_{ij}, true'_{ij}, y_{ij}, z_{ij}, t_{ij})$;
- $\mathcal{B}T_{ij} = (s_{ij}, b_{ij}, true_{ij}, x_{ij}, true'_{ij}, y_{ij}, z_{ij}, t_{ij})$;
- $\mathcal{R}F_{ij} = (s_{ij}, x_{ij}, y_{ij}, c_{ij}, false_{ij}, z_{ij}, false'_{ij}, t_{ij})$;
- $\mathcal{B}F_{ij} = (s_{ij}, x_{ij}, y_{ij}, d_{ij}, false_{ij}, z_{ij}, false'_{ij}, t_{ij})$.

The forbidden transitions $\mathcal{F}_{ij}$ (equivalently the set of allowed transitions $\mathcal{A}_{ij}$) of the gadget G_{ij} are defined in such a way that the only way to go from s_{ij} to t_{ij} is by following one of the paths in $\{\mathcal{B}T_{ij}, \mathcal{B}F_{ij}, \mathcal{R}T_{ij}, \mathcal{R}F_{ij}\}$. It is forbidden to use any transition consisting of two edges from two different paths of the set $\{\mathcal{B}T_{ij}, \mathcal{B}F_{ij}, \mathcal{R}T_{ij}, \mathcal{R}F_{ij}\}$. The allowed transitions are explicitly defined in Table 2.4b. Note that in case there are multi-edges between two vertices x and y, then there are exactly two parallel edges, a blue one and a red one. We denote the blue edge by $\{x, y\}_b$ and the red edge by $\{x, y\}_r$. We use this notation in the whole chapter.

Intuitively, assigning the variable v_i to $True$ will be equivalent to choosing one of the paths $\mathcal{B}T_{ij}$ or $\mathcal{R}T_{ij}$ (called *positive* paths) depicted with full lines in Figure 2.4a. Respectively, assigning v_i to $False$ will correspond to choosing one of the paths $\mathcal{B}F_{ij}$ or $\mathcal{R}F_{ij}$ (called *negative* paths) depicted with dotted lines in Figure 2.4a.



(a) Example of the Gadget-graph G_{ij} for variable v_i, and $1 \leq j \leq m$. The pink (resp., green) edge is added if v_i appears positively (resp., negatively) in C_j. If $v_i \notin C_j$, none of the green nor the pink edges appear.

vertex v	**allowed transitions around v**
a_{ij}	$\{\{s_{ij}, a_{ij}\}, \{a_{ij}, true_{ij}\}\}$ and, if v_i appears positively in C_j, $\{\{s_{ij}, a_{ij}\}, \{a_{ij}, b_{ij}\}\}$ and $\{\{b_{ij}, a_{ij}\}, \{a_{ij}, true_{ij}\}\}$
b_{ij}	$\{\{s_{ij}, b_{ij}\}, \{b_{ij}, true_{ij}\}\}$ and, if v_i appears positively in C_j, $\{\{s_{ij}, b_{ij}\}, \{b_{ij}, a_{ij}\}\}$ and $\{\{a_{ij}, b_{ij}\}, \{b_{ij}, true_{ij}\}\}$
$true_{ij}$	$\{\{a_{ij}, true_{ij}\}, \{true_{ij}, x_{ij}\}_r\}$ and $\{\{b_{ij}, true_{ij}\}, \{true_{ij}, x_{ij}\}_b\}$
x_{ij}	$\{\{true_{ij}, x_{ij}\}_r, \{x_{ij}, true'_{ij}\}_r\}$, $\{\{true_{ij}, x_{ij}\}_b, \{x_{ij}, true'_{ij}\}_b\}$, $\{\{s_{ij}, x_{ij}\}_r, \{x_{ij}, y_{ij}\}_r\}$, and $\{\{s_{ij}, x_{ij}\}_b, \{x_{ij}, y_{ij}\}_b\}$
$true'_{ij}$	$\{\{x_{ij}, true'_{ij}\}_r, \{true'_{ij}, y_{ij}\}_r\}$ and $\{\{x_{ij}, true'_{ij}\}_b, \{true_{ij}, y_{ij}\}_b\}$
y_{ij}	$\{\{x_{ij}, y_{ij}\}_r, \{y_{ij}, c_{ij}\}\}$, $\{\{x_{ij}, y_{ij}\}_b, \{y_{ij}, d_{ij}\}\}$, $\{\{true'_{ij}, y_{ij}\}_r, \{y_{ij}, z_{ij}\}_r\}$, and $\{\{true'_{ij}, y_{ij}\}_b, \{y_{ij}, z_{ij}\}_b\}$
c_{ij}	$\{\{y_{ij}, c_{ij}\}, \{c_{ij}, false_{ij}\}\}$ and, if v_i appears negatively in C_j, $\{\{y_{ij}, c_{ij}\}, \{c_{ij}, d_{ij}\}\}$ and $\{\{d_{ij}, c_{ij}\}, \{c_{ij}, false_{ij}\}\}$
d_{ij}	$\{\{y_{ij}, d_{ij}\}, \{d_{ij}, fasle_{ij}\}\}$ and, if v_i appears negatively in C_j, $\{\{y_{ij}, d_{ij}\}, \{d_{ij}, c_{ij}\}\}$ and $\{\{c_{ij}, d_{ij}\}, \{d_{ij}, false_{ij}\}\}$
$false_{ij}$	$\{\{c_{ij}, false_{ij}\}, \{false_{ij}, z_{ij}\}_r\}$, and $\{\{d_{ij}, false_{ij}\}, \{false_{ij}, z_{ij}\}_b\}$
z_{ij}	$\{\{false_{ij}, z_{ij}\}_r, \{z_{ij}, false'_{ij}\}_r\}$, $\{\{false_{ij}, z_{ij}\}_b, \{z_{ij}, false'_{ij}\}_b\}$, $\{\{y_{ij}, z_{ij}\}_r, \{z_{ij}, t_{ij}\}_r\}$, and $\{\{y_{ij}, z_{ij}\}_b, \{z_{ij}, t_{ij}\}_b\}$

(b) Allowed transitions $\mathcal{A}_{ij}$

Figure 2.4: Gadget G_{ij} and the corresponding set of allowed transitions $\mathcal{A}_{ij}$

So far, it is *a priori* not possible to start from s_{ij} by one path and arrive in t_{ij} by another path. In particular, the color by which s_{ij} is left must be the same by which t_{ij} is reached. If Variable v_i appears in Clause C_j, we add one edge to G_{ij} as follows. If v_i appears positively in C_j, we add the *pink* edge $\{a_{ij}, b_{ij}\}$ that creates a "bridge" between $\mathcal{BT}_{ij}$ and $\mathcal{RT}_{ij}$. Similarly, if v_i appears negatively in C_j, we add the *green* edge $\{c_{ij}, d_{ij}\}$ that creates a "bridge" between $\mathcal{BF}_{ij}$ and $\mathcal{RF}_{ij}$. When the gadget G_{ij} contains a pink (resp. green) edge, all the transitions containing the pink (resp. green) edge are allowed; this makes it possible to switch between the

positive (resp. negative) paths $\mathcal{BT}_{ij}$ and $\mathcal{RT}_{ij}$ (resp. $\mathcal{BF}_{ij}$ and $\mathcal{RF}_{ij}$) when going from s_{ij} to t_{ij}. Hence, if v_i appears in C_j, it will be possible to start from s_{ij} with one color and arrive to t_{ij} with a different one. Note that, the type of path (positive or negative) cannot be modified between s_{ij} and t_{ij}.

We characterize the $\mathcal{F}_{ij}$-valid s_{ij}-t_{ij}-paths in G_{ij} with the following straightforward claims.

Claim 1. *The $\mathcal{F}_{ij}$-valid s_{ij}-t_{ij}-paths in G_{ij} are $\mathcal{RT}_{ij}, \mathcal{BT}_{ij}, \mathcal{RF}_{ij}, \mathcal{BF}_{ij}$ and*

- *if variable v_i appears positively in Clause C_j:*
  - *the path $\mathcal{RBT}_{ij}$ that starts with the first edge $\{s_{ij}, a_{ij}\}$ of $\mathcal{RT}_{ij}$, then uses pink edge $\{a_{ij}, b_{ij}\}$ and ends with all edges of $\mathcal{BT}_{ij}$ but the first one;*
  - *the path $\mathcal{BRT}_{ij}$ that starts with the first edge $\{s_{ij}, b_{ij}\}$ of $\mathcal{BT}_{ij}$, then uses pink edge $\{a_{ij}, b_{ij}\}$ and ends with all edges of $\mathcal{RT}_{ij}$ but the first one;*
- *if variable v_i appears negatively in Clause C_j:*
  - *the path $\mathcal{RBF}_{ij}$ that starts with the subpath $(s_{ij}, x_{ij}, y_{ij}, c_{ij})$ of $\mathcal{RF}_{ij}$, then uses green edge $\{c_{ij}, d_{ij}\}$ and ends with the subpath of $\mathcal{BF}_{ij}$ that starts at d_{ij} and ends at t_{ij};*
  - *the path $\mathcal{BRF}_{ij}$ that starts with the subpath $(s_{ij}, x_{ij}, y_{ij}, d_{ij})$ of $\mathcal{BF}_{ij}$, then uses green edge $\{d_{ij}, c_{ij}\}$ and ends with the subpath of $\mathcal{RF}_{ij}$ that starts at c_{ij} and ends at t_{ij};*

Claim 2. *Let P be an $\mathcal{F}_{ij}$-valid $s_{ij}t_{ij}$-paths in G_{ij}. Then, either*

- *P passes through $true_{ij}$ and $true'_{ij}$ and does not pass through $false_{ij}$ nor $false'_{ij}$, or*
- *P passes through $false_{ij}$ and $false'_{ij}$ and does not pass through $true_{ij}$ nor $true'_{ij}$.*

Claim 3. *Let P be an $\mathcal{F}_{ij}$-valid $s_{ij}t_{ij}$-paths in G_{ij}. Then the first and last edges of P have different colors if and only if P uses a green or a pink edge, i.e., if $P \in \{\mathcal{RBT}_{ij}, \mathcal{BRT}_{ij}, \mathcal{RBF}_{ij}, \mathcal{BRF}_{ij}\}$.*

Clause-graph G_j. For any $j \leq m$, the Clause-gadget G_j is built by combining the graphs G_{ij}, $i \leq n$, in a "line" (see Figure 2.5). The subgraphs G_{ij} are combined from "left to right" (for $i = 1$ to n) if j is odd and from "right to left" (for $i = n$ to 1) otherwise. In more details, for any $j \leq m$, G_j is obtained from a copy of each gadget G_{ij}, $1 \leq i \leq n$, and two additional vertices s_j and t_j as follows:

- If j is odd, the subgraph G_j starts with a red edge $\{s_j, s_{1j}\}$ and then, for $1 < i \leq n$, the vertices s_{ij} and $t_{i-1,j}$ are identified. Finally, there is a blue edge from t_{nj} to vertex t_j.

- If j is even, the subgraph G_j starts with a blue edge $\{s_j, s_{nj}\}$ and then, for $1 < i \leq n$, the vertices t_{ij} and $s_{i-1,j}$ are identified. Finally, there is a red edge from t_{1j} to vertex t_j.

The forbidden transitions $\mathcal{F}_j$ include, besides all transitions in $\mathcal{F}_{ij}$, $i = 1, \ldots, n$, new transitions which are defined such that, when passing from a gadget G_{ij} to the next one, the same color must be used. This means that if we enter a vertex $t_{ij} = s_{i,j+1}$ by an edge with a given color, the same color must be used to leave this vertex. However, in such vertices, we can change the type (positive or negative) of path. These new transitions are explicitly given in Table 2.5c.

Note that if we enter a Clause-graph with a red (resp. blue) edge, we can only leave it with a blue (resp. red) edge. This means that a path must change its color inside the Clause-graph, and must hence use a pink or green edge in some gadget-graph. The use of a pink (resp. green) edge forces a variable that appears positively (resp. negatively) in the clause to be set to true (resp. false) and validates the Clause.

The key property of G_j relates to the structure of $\mathcal{F}_j$-valid paths from s_j to t_j, which we summarize in Claims 4 and 5.

Claim 4. *Any $\mathcal{F}_j$-valid path P from s_j to t_j in G_j consists of the concatenation of:*

Case j odd. *the red edge $\{s_j, s_{1j}\}$, then the concatenation of $\mathcal{F}_{ij}$-valid paths from s_{ij} to t_{ij} in G_{ij}, for $1 \leq i \leq n$ in this order (from $i = 1$ to n), and finally the blue edge $\{t_{nj}, t_j\}$;*

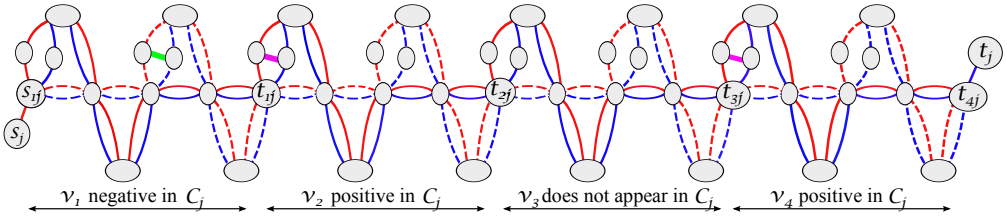
Case j even. *the blue edge $\{s_j, s_{nj}\}$, then the concatenation of $\mathcal{F}_{ij}$-valid paths from s_{ij} to t_{ij} in G_{ij}, for $1 \leq i \leq n$ in the reverse order (from $i = n$ to 1), and finally the red edge $\{t_{1j}, t_j\}$.*

By the previous claim, for any $\mathcal{F}_j$-valid path P from s_j to t_j, the colors of the first and last edges differ. Hence, by Claim 3 and the definition of the allowed transitions between two gadgets, we have the following claim.

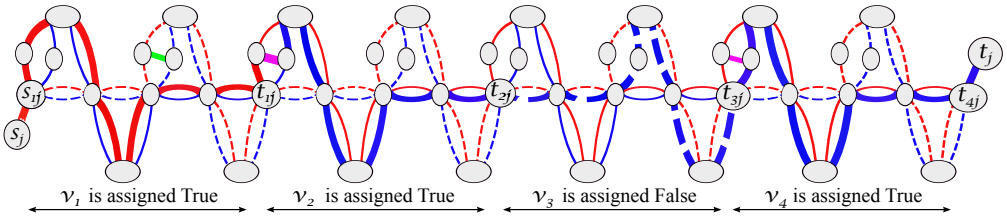
Claim 5. *Any $\mathcal{F}_j$-valid path P from s_j to t_j must use a green or a pink edge in a gadget G_{ij} for some $1 \leq i \leq n$.*

Main graph. To conclude, we have to ensure that the assignment of the variables is coherent between the clauses. For this purpose, let us combine the subgraphs G_j, $j \leq m$, as follows (see Figure 2.6). First, for any $1 \leq j < m$, let us identify t_j and s_{j+1}. Then, some vertices (depicted in dark grey in Fig 2.6) of G_{ij} are identified with vertices of $G_{i,j+1}$ in such a way that using a positive (resp., negative) path in G_{ij} forces the use of the same type of path in $G_{i,j+1}$. That is, the choice of the path used in G_{ij} is transferred to $G_{i,j+1}$ and therefore the same truth assignment for Variable v_i is defined at lines j and $j + 1$.

Namely, for each $1 \leq j < m$ and for each $1 \leq i \leq n$, we identify the vertices $true_{i,j+1}$ and $false'_{ij}$ on the one hand, and the vertices $true'_{ij}$ and $false_{i,j+1}$ on



(a) Clause-graph G_j for a Clause $C_j = \bar{v}_1 \vee v_2 \vee v_4$ in a formula with 4 Variables.



(b) The bold path corresponds to an assignment of v_1, v_2 and v_4 to $True$, and of v_3 to $False$.

vertex v	**allowed transitions around v**
$\boldsymbol{s_{1j}}$	$\{\{s_j, s_{1j}\}, \{s_{1j}, a_{ij}\}\}$ and $\{\{s_j, s_{1j}\}, \{s_{1j}, x_{1j}\}_r\}$
$\boldsymbol{t_{i,j} = s_{i+1,j}}$ $\boldsymbol{1 \leq i < n}$	$\{\{z_{ij}, t_{ij}\}_r, \{t_{ij}, a_{i+1,j}\}\}$, $\{\{z_{ij}, t_{ij}\}_r, \{t_{ij}, x_{i+1,j}\}_r\}$, $\{\{z_{ij}, t_{ij}\}_b, \{t_{ij}, b_{i+1,j}\}\}$, $\{\{z_{ij}, t_{ij}\}_b, \{t_{ij}, x_{i+1j}\}_b\}$, $\{\{false'_{ij}, t_{ij}\}_r, \{t_{ij}, a_{i+1,j}\}\}$, $\{\{false'_{ij}, t_{ij}\}_r, \{t_{ij}, x_{i+1,j}\}_r\}$, $\{\{false'_{ij}, t_{ij}\}_b, \{t_{ij}, b_{i+1,j}\}\}$, and $\{\{false'_{ij}, t_{ij}\}_b, \{t_{ij}, x_{i+1,j}\}_b\}$
$\boldsymbol{t_{nj}}$	$\{\{z_{nj}, t_{nj}\}_b, \{t_{nj}, t_j\}\}$ and $\{\{false'_{nj}, t_{nj}\}_b, \{t_{nj}, t_j\}\}$

(c) Allowed transitions around vertices s_{ij}, t_{ij}

Figure 2.5: Gadget G_j (j is odd) and allowed transitions around the vertices s_{ij} and t_{ij}

the other hand to obtain the "grey" vertices. Finally, forbidden transitions $\mathcal{F}$ of G, include, besides all transitions in $\mathcal{F}_j$ for $j = 1, \ldots, m$, new transitions which are defined in order to forbid "crossing" a grey vertex, i.e., it is not possible to go from $G_{i,j}$ to $G_{i,j+1}$ via a grey vertex (see Figure 2.6). The following claims present the key properties of an $\mathcal{F}$-valid path in G. Claim 6 is straightforward and its proof is omitted.

Claim 6. *Any $\mathcal{F}$-valid path P from s_1 to t_m in G consists of the concatenation of $\mathcal{F}_j$-valid paths from s_j to t_j in G_j from $j = 1$ to m.*

Claim 7. *Let P be an $\mathcal{F}$-valid s_1-t_m-path in G. Then, for any $1 \leq i \leq n$, either*

- *for any $1 \leq j \leq m$, the subpath of P between s_{ij} and t_{ij} passes through $true_{ij}$ and $true'_{ij}$ and does not pass through $false_{ij}$ nor $false'_{ij}$, or*

- *for any* $1 \leq j \leq m$*, the subpath of* P *between* s_{ij} *and* t_{ij} *passes through* $false_{ij}$ *and* $false'_{ij}$ *and does not pass through* $true_{ij}$ *nor* $true'_{ij}$*.*

Proof of claim. By Claims 4 and 6, for any $1 \leq i \leq n$ and any $1 \leq j \leq m$, there is a subpath P_{ij} of P that goes from s_{ij} to t_{ij}. Moreover, the paths P_{ij} are pairwise vertex-disjoint (since the path P does not use a vertex more than once).

For $1 \leq i \leq n$, by Claim 2, P_{i1} either passes through $true_{i1}$ and $true'_{i1}$, or through $false_{i1}$ and $false'_{i1}$. Let us assume that we are in the first case (the second case can be handled symmetrically). We prove by induction on $j \leq m$ that P_{ij} passes through $true_{ij}$ and $true'_{ij}$ and does not pass through $false_{ij}$ nor $false'_{ij}$. Indeed, if P passes through $true_{ij} = false'_{i,j+1}$ and $true'_{ij} = false_{i,j+1}$, then $P_{i,j+1}$ cannot use $false_{i,j+1}$ nor $false'_{i,j+1}$ since P_{ij} and $P_{i,j+1}$ are vertex-disjoint. By Claim 2, $P_{i,j+1}$ passes through $true_{i,j+1}$ and $true'_{i,j+1}$.

◇

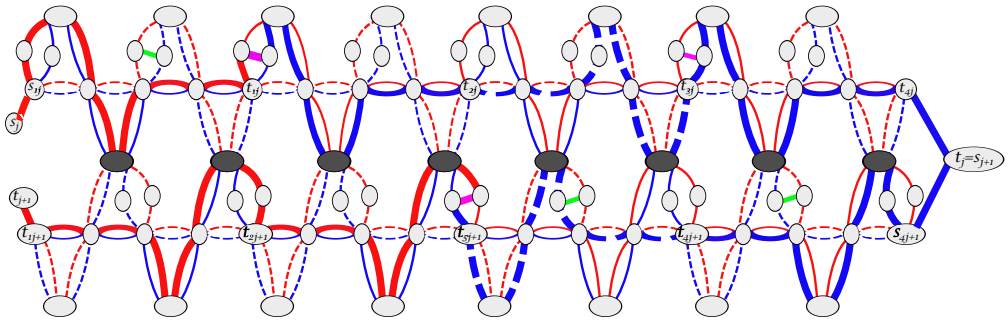


Figure 2.6: Combining $C_j = \bar{v}_1 \vee v_2 \vee v_4$ and $C_{j+1} = v_2 \vee \bar{v}_3 \vee \bar{v}_4$ (Case j odd). The bold path corresponds to an assignment of v_1, v_2 and v_4 to $True$, and of v_3 to $False$ and it satisfies both clauses.

Note that $(G, \mathcal{F})$ can be constructed in polynomial-time. Moreover, G is clearly planar with maximum degree 8. Hence, the next claim allows to prove Lemma 1.

Claim 8. Φ *is satisfiable if and only if there is an* $\mathcal{F}$*-valid* s_1*-*t_m*-path in* G*.*

Proof of claim. Let φ be a truth assignment which satisfies Φ. We can build an $\mathcal{F}$-valid s_1-t_m-path in G as follows. For each row $1 \leq j \leq m$, we build a path P_j from s_i to t_j by concatenating the paths P_{ij}, $1 \leq i \leq n$, which are built as follows. Among the variables that appear in C_j, let v_q be the variable with the smallest index, which satisfies the clause. Note that this choice is arbitrary and that we can choose any variable satisfying the clause.

- For $1 \leq i < q$, if $\varphi(v_i) = true$, then $P_{ij} = \mathcal{R}T_{ij}$ if j is odd and $P_{ij} = \mathcal{B}T_{ij}$ if j is even. If $\varphi(v_i) = false$, then $P_{ij} = \mathcal{R}F_{ij}$ if j is odd, and $P_{ij} = \mathcal{B}F_{ij}$ if j is even.

- For $i = q$, if $\varphi(v_i) = true$, then $P_{ij} = \mathcal{RBT}_{ij}$ if j is odd, and $P_{ij} = \mathcal{BRT}_{ij}$ if j is even. If $\varphi(v_i s) = false$, then $P_{ij} = \mathcal{RBF}_{ij}$ if j is odd, and $P_{ij} = \mathcal{BRF}_{ij}$ if j is even.

- For $q < i \leq n$, if $\varphi(v_i) = true$, then $P_{ij} = \mathcal{BT}_{ij}$ if j is odd, and $P_{ij} = \mathcal{RT}_{ij}$ otherwise. If $\varphi(v_i) = false$, then $P_{ij} = \mathcal{BF}_{ij}$ if j is odd, and $P_{ij} = \mathcal{RF}_{ij}$ otherwise.

The path P obtained from the concatenation of paths P_j for $1 \leq j \leq m$ is an $\mathcal{F}$-valid path from s_1 to t_m.

Now let us suppose that there is an $\mathcal{F}$-valid path P from s_1 to t_m. According to Claim 7, for any $1 \leq i \leq n$, for any $1 \leq j \leq m$, P passes through $true_{ij}$ and $true'_{ij}$ and does not pass through $false_{ij}$ and $false'_{ij}$ or for any $1 \leq j \leq m$, P passes through $false_{ij}$ and $false'_{ij}$ and does not pass through $true_{ij}$ and $true'_{ij}$. Let us then consider the truth assignment φ of Φ such that for each $1 \leq i \leq n$:

- If P uses $true_{ij}$ and $true'_{ij}$ in all rows $1 \leq j \leq m$, then $\varphi(v_i) = true$.

- If P uses $false_{ij}$ and $false'_{ij}$ in all rows $1 \leq j \leq m$, then $\varphi(v_i) = false$.

By Claim 7, φ is a valid truth assignment which means that each variable is assigned either the value $true$ or the value $false$ and not both. We need to prove that φ satisfies Φ. According to Claim 6, for each row $1 \leq j \leq m$, P contains an $\mathcal{F}_j$-valid path P_j from s_j to t_j. Each path P_j uses a green or a pink edge as stated by Claim 3. With respect to the possible ways to use a green or a pink edge which are stated in Claim 2, the use of a pink edge in P_j forces P_j (and hence P) to use the vertices $true_{ij}$ and $true'_{ij}$ for a variable v_i that appears positively in C_j. Similarly, the use of a green edge in P_j forces P_j (and hence P) to use the vertices $false_{ij}$ and $false'_{ij}$ for a variable v_i that appears negatively in C_j. This means that for each clause C_j, for one of the variables that appear in C_j which we denote v_q, $\varphi(v_q) = true$ (resp. $\varphi(v_q) = false$) if v_q appears positively (resp. negatively) in C_j. Thus, the truth assignment φ satisfies Φ.

◇

With this we have proved that the PAFT problem is NP-complete in planar graphs with maximum degree 8. □

Lemma 2. *The PAFT problem is NP-complete in planar graphs with maximum degree* 4.

Proof. The graph G used in the reduction of the proof of Lemma 1 is planar and each vertex of G has either degree 8, degree 5 or degree at most 4. Let $\mathcal{E}$ be the planar embedding of G that is obtained by embedding the smaller gadgets as in Figures 2.4a, 2.5, and 2.6. We transform G into a planar graph G' with maximum degree 4 and an associated set of forbidden transitions $\mathcal{F}'$ such that finding an $\mathcal{F}$-valid path in G is equivalent to finding an $\mathcal{F}'$-valid path in G'. The transformation goes as follows.

Vertices of degree 5. The vertices of degree 5 in G are the vertices s_{1j} and t_{nj} for j odd , and s_{nj} and t_{1j} for j even. We transform these to vertices of degree 3 as follows. For j odd, we remove the two blue edges incident to s_{1j} and the two red edges incident to t_{nj}. For j is even, we remove the two red edges incident to s_{nj} and the two blue edges incident to t_{1j}. The removed edges do not belong to any allowed transitions around the vertices s_{1j}, t_{nj}, s_{nj} and t_{1j}. Therefore, their removal does not affect the elementary $\mathcal{F}$-valid paths from s_1 to t_m.

Vertices of degree 8. We replace each vertex v of degree 8 by a gadget g_v of maximum degree 4. Gadget g_v is designed such that it can be crossed at most once by a path of G' and only if the edges used to enter and leave g_v correspond to an allowed transition around v. According to its corresponding transitions and to the planar embedding of its adjacent vertices, a vertex v of degree 8 of G is of one of 3 different types. We present in what follows these types as well as the corresponding gadget g_v for each type.

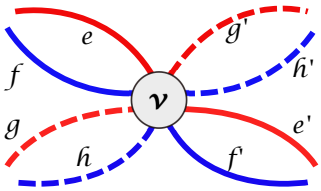
Type 1: The edges incident to v are $E(v) = \{e, e', f, f', g, g', h, h'\}$ and the allowed transitions around v are $A(v) = \{\{e, e'\}, \{f, f'\}, \{g, g'\}, \{h, h'\}\}$. The edges incident to v appear in the planar embedding $\mathcal{E}$ as presented in Figure 2.7a. In the graph G, v is one of the vertices x_{ij} or z_{ij}, $1 \leq i \leq n$, $1 \leq j \leq m$.

In this case, g_v is built as follows. For each $\alpha \in E(v)$, a vertex v_α is created. For each $\{\alpha, \beta\} \in A(v)$, vertices v_α and v_β are linked with a path $P_{\alpha\beta}$ of length four. The four paths $P_{\alpha\beta}$, $\{\alpha, \beta\} \in A(v)$ are pairwise intersecting in distinct vertices as illustrated in Figure 2.7b. The allowed transitions in g_v are the transitions of the paths $P_{\alpha\beta}$, $\{\alpha, \beta\} \in A(v)$. Now to replace v with g_v in G, each edge $\alpha \in E(v)$ of G is linked to vertex v_α of g_v. The gadget g_v is planar, and edges $\alpha \in E(v)$ are connected to it in the same "order" they are connected to v in the planar embedding $\mathcal{E}$ of G as illustrated in Figure 2.7.

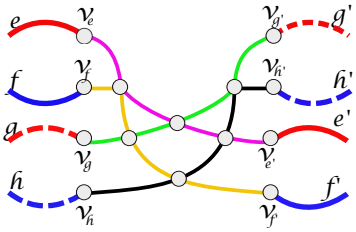
Note the gadget g_v cannot be crossed twice with the same path (i.e. no path has two subpaths in g_v), otherwise the path is not simple. Moreover, g_v can be crossed if and only if the edges used to enter and leave form an allowed transition around v.

Type 2: The edges incident to v are $E(v) = \{e, e', f, f', g, g', h, h'\}$ and the allowed transitions around v are $A(v) = \{\{e, e'\}, \{f, f'\}, \{g, g'\}, \{h, h'\}\}$. The edges incident to v appear in the planar embedding $\mathcal{E}$ as presented in Figure 2.8a. In the graph G, v is one of the vertices $true_{ij}$, $true'_{ij}$, $false_{ij}$, $false'_{ij}$, or y_{ij}, $1 \leq i \leq n$, $1 \leq j \leq m$.

In this case, g_v is built as follows. For each $\alpha \in E(v)$, a vertex v_α is created. For each $\{\alpha, \beta\} \in A(v)$, vertices v_α and v_β are linked with a path $P_{\alpha\beta}$ of length 7. Each two of the four paths $P_{\alpha\beta}$, $\{\alpha, \beta\} \in A(v)$ intersect in two different vertices as illustrated in Figure 2.8b. The allowed transitions in g_v are the transitions of the paths $P_{\alpha\beta}$, $\{\alpha, \beta\} \in A(v)$. Now to replace v with g_v



(a) A vertex v of degree 8 and allowed transitions $A(v) = \{\{e, e'\}, \{f, f'\}, \{g, g'\}, \{h, h'\}\}$ (edges are ordered as in the planar embedding $\mathcal{E}$ of G)

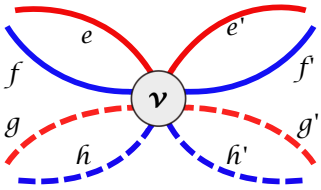


(b) Gadget g_v: the paths $P_{ee'}$, $P_{ff'}$,$P_{gg'}$, and $P_{hh'}$ are respectively the pink, yellow, green and black paths. Transitions around vertices v_α, $\alpha \in E(v)$ and transitions of paths $P_{ee'}$, $P_{ff'}$,$P_{gg'}$, and $P_{hh'}$ are allowed

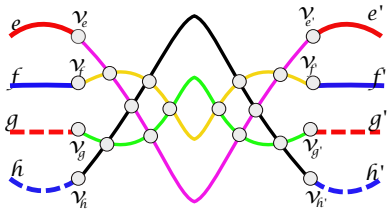
Figure 2.7: Type 1

in G, each edge $\alpha \in E(v)$ of G is linked to vertex v_α of g_v. The gadget g_v is planar, and edges $\alpha \in E(v)$ are connected to it in the same "order" they were connected to v in the planar embedding $\mathcal{E}$ of G as illustrated in Figure 2.8.

Note that the gadget g_v cannot be crossed twice with the same path (i.e. no path has two subpaths in g_v), otherwise the path is not simple. Moreover, g_v can be crossed if and only if the edges used to enter and leave form an allowed transition around v.



(a) A vertex v of degree 8 and allowed transitions $A(v) = \{\{e, e'\}, \{e, f'\}, \{f, f'\}, \{f, e'\}, \{g, g'\}, \{g, h'\}, \{h, h'\}, \{h, g'\}\}$ (edges are ordered as in the planar embedding $\mathcal{E}$ of G)

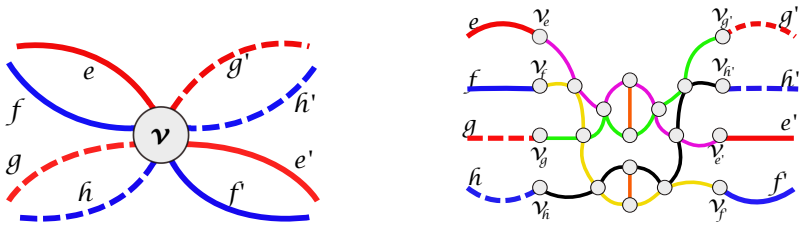


(b) Gadget g_v: the paths $P_{ee'}$, $P_{ff'}$,$P_{gg'}$, and $P_{hh'}$ are respectively the pink, yellow, green and black paths. Transitions around vertices v_α, $\alpha \in E(v)$ and transitions of paths $P_{ee'}$, $P_{ff'}$,$P_{gg'}$, and $P_{hh'}$ are allowed

Figure 2.8: Type 2

Type 3: The edges incident to v are $E(v) = \{e, e', f, f', g, g', h, h'\}$ and the allowed transitions around v are $A(v) = \{\{e, e'\}, \{e, g'\}, \{f, f'\}, \{f, h'\}, \{g, g'\}, \{g, e'\}, \{h, h'\}, \{h, f'\}\}$. The edges incident to v appear in the planar embedding $\mathcal{E}$ as depicted in Figure 2.9a. In the graph G, v is one of the vertices s_{ij}, $1 \leq i \leq n$, $1 \leq j \leq m$.

In this case, g_v is built as follows. Let $A_1(v) = \{\{e, e'\}, \{f, f'\}, \{g, g'\}, \{h, h'\}\}$. For each $\{\alpha, \beta\} \in A_1(v)$, vertices v_α and v_β are linked with a path $P_{\alpha\beta}$ of length



(a) A vertex v of degree 8 and allowed transitions $A(v) = \{\{e, e'\}, \{e, g'\}, \{f, f'\}, \{f, h'\}, \{g, g'\}, \{g, e'\}, \{h, h'\}, \{h, f'\}\}$ (edges are ordered as in the planar embedding $\mathcal{E}$ of G)

(b) Gadget g_v: the paths $P_{ee'}$, $P_{ff'}$,$P_{gg'}$, and $P_{hh'}$ are respectively the pink, yellow, green and black paths. transitions around vertices v_α, $\alpha \in E(v)$, transitions of the paths $P_{ee'}$, $P_{ff'}$,$P_{gg'}$, and $P_{hh'}$, and transitions containing the orange edge are allowed

Figure 2.9: Type 3

6. Each two of the paths $P_{\alpha\beta}$ intersect in one or two vertices as illustrated in Figure 2.9b. Furthermore, we add two edges linking the paths $P_{ee'}$ and $P_{gg'}$, and $P_{ff'}$ and $P_{hh'}$, respectively, depicted by orange edges in Figure 2.9b. Now to replace v with g_v in G, each edge $\alpha \in E(v)$ of G is linked to vertex v_α of g_v. The gadget g_v is planar, and edges $i \in E(v)$ are connected to it in the same "order" they were connected to v in the planar embedding $\mathcal{E}$ of G as illustrated in Figure 2.9.

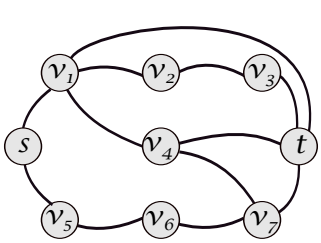
   For $\{\alpha, \beta\} \in A(v) \setminus A_1(v)$, let α' and β' be the vertices such that $\{\alpha, \alpha'\}, \{\beta, \beta'\} \in A_1(v)$. We define the path $P_{\alpha\beta}$, as the path which starts at v_α, uses a subpath of $P_{\alpha\alpha'}$, an orange edge and then a subpath of $P_{\beta\beta'}$ and then ends at v_r.

   Note that the gadget g_v cannot be crossed twice with the same path (i.e. no path has two subpaths in g_v), otherwise the path is not simple. Moreover, g_v can be crossed if and only if the edges used to enter and leave form an allowed transition around v.

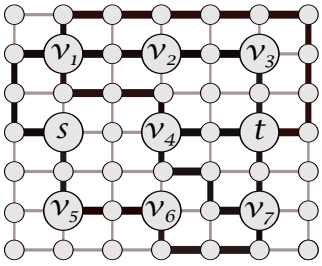
The graph G' obtained from G after applying the transformations described above is planar and has maximum degree 4. The set of forbidden transitions $\mathcal{F}'$ consists of the transitions of the set $\mathcal{F}$ and the forbidden transitions of the gadgets g_v as described above.

Let us now suppose that there is an $\mathcal{F}$-valid path P from s to t in G. Let P' be the s-t-path of G' constructed as follows: P' uses all edges used by P. Furthermore, if P uses a degree 8 vertex with a transition $\{e, e'\}$ then P' uses e, subpath $P_{ee'}$, and e'. The path P' is $\mathcal{F}'$-valid.

Now, let us suppose that there is an $\mathcal{F}'$-valid path P' from s to t in G'. If P' only uses edges from G, then it can be considered as an $\mathcal{F}$-valid path from s to t in G. If P' uses an edge that is not in G, then P' crosses one of the gadgets g_v. Gadgets g_v are designed such that they can be crossed at most once by a path



(a) A planar graph G of maximum degree 4



(b) A planar grid embedding of G (v is used to denote $Q(v)$)

Figure 2.10: Example of a planar grid embedding

because otherwise the path is not simple, furthermore, the edges used to enter and leave the gadget form an allowed transition in G. This implies that the intersection of each gadget g_v with the path P' is either the empty set or exactly one subpath of P'. If it is a subpath then the edges surrounding it in P' form an allowed transition in G. We can then remove the edges of P' that do not belong to G to obtain an $\mathcal{F}$-valid path P in G. □

Theorem 2. *The PAFT problem is NP-complete in grids.*

Proof. To prove the theorem we use the notion of planar grid embedding [Tam87]. A planar grid embedding of a graph G is a mapping Q of G into a grid such that:

- Q maps each vertex v of G into a distinct vertex $Q(v)$ of the grid, and
- Q maps each edge $e = (u, v)$ of G into a path of the grid $Q(e)$ whose endpoints are $Q(u)$ and $Q(v)$, the mappings of the vertices linked by e, and
- for every pair $\{e, e'\}$ of edges of G, the corresponding paths $Q(e)$ and $Q(e')$ have no points in common, except, possibly, the endpoints.

Figure 2.10 illustrates an example of a planar grid embedding of a planar graph G. In bold are the paths corresponding to the edges of G in the planar grid embedding.

It has been proved in [Val81], that if $G = (V, E)$ is a planar graph with n vertices and maximum degree $\Delta \leq 4$, then a planar grid embedding of G in a grid of size at most $9n^2$ can be found in polynomial-time. Let us consider an instance of the problem of finding a path avoiding forbidden transitions in a planar graph $G = (V, E)$ of maximum degree at most 4 with a set of allowed transitions $\mathcal{A}$. Let Q be a grid planar embedding of G into a grid K of size at most $O(|V|^2)$. Finding a PAFT between two nodes s and t in G with the set $\mathcal{A}$ is equivalent to finding a PAFT between the nodes $Q(s)$ and $Q(t)$ in K with the set of allowed transitions $\mathcal{A}'$ defined such that:

- For each $e \in E$, all the transitions in the path $Q(e)$ are allowed.

- For each $\{e, e'\} \in \mathcal{A}$, the pair of edges of $Q(e)$ and $Q(e')$, which share a vertex, is an allowed transition.
- All other transitions are forbidden.

Sometimes, vertices s and t might be mapped to adjacent vertices in K while they are not adjacent in G. This would make the problem of solving PAFT between $Q(s)$ and $Q(t)$ in K easier than solving PAFT between s and t in G. To avoid this, we always make sure that except for $Q((s,t))$, no other path between $Q(s)$ and $Q(t)$ consists of only one edge in K, if it is not already the case, we increase the size of K to endure that the distance between $Q(s)$ and $Q(t)$ is at least 2.

Since we have proved in Lemma 2 that PAFT is NP-complete in planar graph with maximum degree 4, Theorem 2 follows.

□

Remark for directed graphs. Please note that all of the constructions in the NP-completeness proofs can be done with directed graphs, which implies that PAFT is NP-complete in directed grids. In fact, the gadgets G_{ij} which are the elementary components of our construction, can be built with directed paths $\mathcal{RT}_{ij}, \mathcal{BT}_{ij}, \mathcal{RF}_{ij}$ and $\mathcal{BF}_{ij}$ instead of undirected ones. As for the edges $\{a_{ij}, b_{ij}\}$ and $\{c_{ij}, d_{ij}\}$, they can be replaced by arcs (a_{ij}, b_{ij}) and (c_{ij}, d_{ij}) if j is odd and arcs (b_{ij}, a_{ij}) and (d_{ij}, c_{ij}) otherwise. In the clause-graphs G_j, we can have arcs (s_j, s_{1j}) and (t_{nj}, t_n) replace of the edges $\{s_j, s_{1j}\}$ and $\{t_{nj}, t_n\}$. Now, to transform the vertices from a degree 8 to a degree 4, it is always possible to use the mini-gadgets g_v but with directed paths instead of undirected ones. Finally, a directed graph of degree at most 4 can be embedded in a directed grid by first embedding its undirected underlying graph and then directing the edges of the embedding.

2.4.2 Parameterized complexity

On the positive side, by using dynamic programming on a tree-decomposition of the input graph, we prove that the PAFT problem is FPT when the parameter is the sum of the treewidth and the maximum degree.

A *tree-decomposition* of a graph [RS86] is a way to represent G by a family of subsets of its vertex-set organized in a tree-like manner and satisfying some connectivity property. The *treewidth* of G measures the proximity of G to a tree. More formally, a tree-decomposition of $G = (V, E)$ is a pair $(T, \mathcal{X})$ where $\mathcal{X} = \{X_u | u \in V(T)\}$ is a family of subsets, called *bags*, of V, and T is a tree, such that:

- $\bigcup_{u \in V(T)} X_u = V$;
- for any edge $\{a, b\} \in E$, there is a bag X_u (for some node $u \in V(T)$) containing both a and b;
- for any vertex $a \in V$, the set $\{u \in V(T) | a \in X_u\}$ induces a subtree of T.

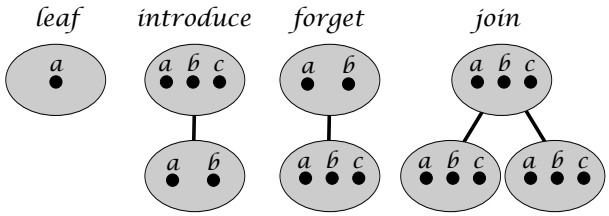


Figure 2.11: Types of nodes in a nice tree-decompositions

The *width* of a tree-decomposition $(T, \mathcal{X})$ is $max_{u \in V(T)} |X_u| - 1$. The treewidth of G, denoted by $tw(G)$, is the minimum width over all possible tree-decompositions of G.

Computing an optimal tree-decomposition - i.e., with width $tw(G)$ - is NP-complete in the class of general graphs G [ACP87]. For any fixed $k \geq 1$, Bodlaender designed an algorithm that computes, in time $O(k^{k^3} n)$, a tree-decomposition of width k of any n-node graph with treewidth at most k [Bod96]. Very recently, a single-exponential (in k) algorithm has been proposed that computes a tree-decomposition with width at most $5k$ in the class of graphs with treewidth at most k [BDD+13].

Many NP-hard problems can be solved in polynomial time in the class of graphs of bounded treewidth using dynamic programming algorithms. For instance, the Maximum Independent Set, the 3-Coloring, the Vertex Cover, and the Hamiltonian cycle are all FPT when parametrized by the treewidth of the graph. A notion that is usually used to prove such results and that we will also use in what follows is the nice tree-decomposition [Klo94, CNP+11].

Definition 1. *A rooted tree-decomposition $((T, \mathcal{X}), r)$ of G is nice if $r \in T$, T is rooted in r and for every node $u \in V(T)$:*

- *u has no children and $|X_u| = 1$ (u is called a leaf node), or*
- *u has one child v with $X_u \subset X_v$ and $|X_u| = |X_v| - 1$ (u is called a forget node),or*
- *u has one child v with $X_v \subset X_u$ and $|X_u| = |X_v| + 1$ (u is called an introduce node), or*
- *u has two children v and w with $X_u = X_v = X_w$ (u is called a join node.).*

The following lemma is a well-known result about tree-decomposition [Klo94].

Lemma 3. *[Klo94] When given a tree-decomposition of width k of G, in polynomial time we can construct a nice tree-decomposition $(T, \mathcal{X})$ of G of width at most k, with $|V(T)| \in O(kn)$, where $n = |V(G)|$.*

In this section, we use the nice tree-decomposition and dynamic programming techniques described in a general context in [Bod97, TP97], to prove Theorem 3 .

Our approach is similar to the one used to find Hamiltonian cycles in graphs with bounded treewidth and which is nicely presented in [Mar10].

Theorem 3. *The PAFT problem is FPT when parameterized by $k + \Delta$ where k is the treewidth and Δ is the maximum degree. In particular, given a graph G with n vertices and maximum degree Δ, and a tree-decomposition of G with width k, PAFT can be solved in G in time $O(k\Delta^2(3k\Delta)^{2k}n)$.*

Proof. Let $G = (V, E)$ be a graph with bounded treewidth k, $\mathcal{F} \subseteq E \times E$ a set of forbidden transitions, and s and t two vertices of V. We would like to find the shortest $\mathcal{F}$-valid path P from s to t.

Let e and f be two edges incident to s and t, respectively. We define $G_{e,f}$ as the graph obtained from G, by deleting all edges incident to s and t except for e and f. Finding the shortest $\mathcal{F}$-valid path from s to t in G is equivalent to finding the shortest path among all shortest $\mathcal{F}$-valid paths from s to t in $G_{e,f}$, for each possible pair (e, f). In the following we will present how to find the shortest $\mathcal{F}$-valid path P from s to t in $G_{e,f}$. To obtain the solution in G, we will need to repeat the algorithm at most Δ^2 times.

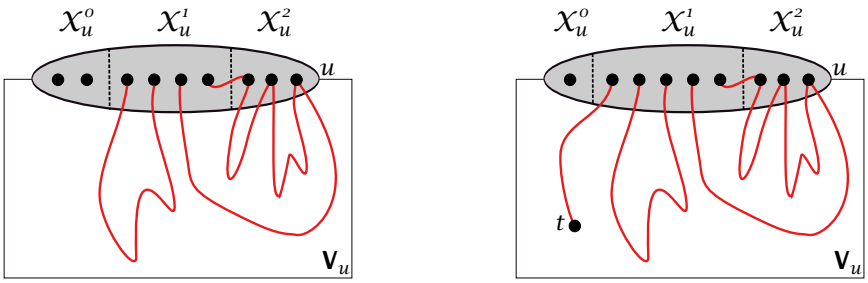
Let $(T, \mathcal{X})$ be a nice tree-decomposition of width k of $G_{e,f}$. To minimize the number of cases to handle in the proof, we assume that s appears in one introduce bag which is the root of T and t in two bags, a leaf and its introduce parent. This is easy to obtain since we can first find a tree decomposition of $G \setminus \{s, t\}$ and then add a node containing $e = \{s, x\}$ adjacent to a bag containing x and a node containing $\{t, y\}$ adjacent to a bag containing y. Finally, we root this tree-decomposition at the bag containing s and make the tree-decomposition nice.

Let $A \subseteq V$, $G[A]$ is the subgraph of $G_{e,f}$ induced by the vertices A. For each $u \in V(T)$ we denote by X_u,T_u and V_u the bag corresponding to u (i.e. the set X_u of vertices of G), the subtree of T rooted at u, and the union of the bags corresponding to the nodes of T_u, respectively. We also denote by δX_u, the set of vertices $X_u \cap X_v$, where v is the parent of u and u is not the root of the tree. If u is the root, we set $\delta X_u = \emptyset$.

If there exists an $\mathcal{F}$-valid path P from s to t, then the intersection of this path with $G[V_u]$ for a node $u \in T$, which is not the root, is a collection of $\mathcal{F}$-valid paths with endpoints exclusively in $\delta X_u \cup \{t\}$ as illustrated in Figure 2.12. We denote this collection by $\mathcal{P}_u = G[V_u] \cap P$. With respect to $\mathcal{P}_u$, vertices in X_u can be partitioned into three subsets (see Figure 2.12):

- X_u^0, which are the vertices of X_u that are not in $\mathcal{P}_u$.
- X_u^1, which are the vertices of X_u that have degree 1 in $\mathcal{P}_u$. Note that $X_u^1 \subset \delta X_u \cup \{t\}$.
- X_u^2, which are the vertices of X_u that have degree 2 in $\mathcal{P}_u$.

To encode the collection of paths $\mathcal{P}_u$ at a node u, we need besides the sets X_u^0, X_u^1, and X_u^2, two more sets which are:



(a) $t \notin V_u$ (b) $t \in V_u$

Figure 2.12: Examples of the intersection of P with G_u

- M_u, which is a matching of the vertices of X_u^1. This matching decides which vertices are endpoints of the same path in $\mathcal{P}_u$. According to whether t is in V_u or not, M is either a matching of the vertices of X_u^1, or a matching of the vertices of $X_u^1 \cup \{t\}$.

- S_u, which for each vertex v in X_u^1, contains an edge incident to v. This set decides which edges incident to X_u^1 are in $\mathcal{P}_u$ and it is needed to be able to extend $\mathcal{P}_u$ (using only allowed transitions) to a new collection of $\mathcal{F}$-valid paths in the dynamic programming approach.

For each node $u \in T$, we say that (X, Y, Z, M, S) is a realization of u if there exists a set of $\mathcal{F}$-valid paths $\mathcal{P}_u$ in V_u such that $(X, Y, Z, M, S) = (X_u^0, X_u^1, X_u^2, M_u, S_u)$. To find an $\mathcal{F}$-valid path from s to t, we need to check whether there exists a realization (X, Y, Z, M, S) of the root such that $Y = \{s\}$; the collection of paths corresponding to such a realization is an F-valid path. To do that, we compute at each node u of the tree-decomposition, from the leaves to the root, all of the realizations of u. Since several path collections may correspond to one realization, we only keep for a realization (X, Y, Z, M, S) the collection which minimizes the number of edges which we denote by $\mathcal{S}((X, Y, Z, M, S))$. This way the $\mathcal{F}$-valid path we obtain at the end (if there exists one) is a shortest path.

For each node $u \in T$, there are at most 3^{k+1} possible partitions of the vertices of X_u into the 3 different sets, k^k possible matchings for a set of k elements and Δ possible edges for each element of X_u^1. Hence, at node u there are at most $3^{k+1}k^k\Delta^k$ realizations. Let us see how to decide if (X, Y, Z, M, S) is a realization of a node u supposing that all the realizations of its children have been computed.

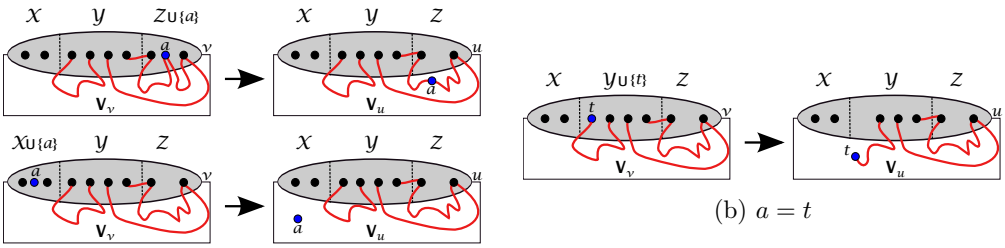
For two collection of paths $\mathcal{A}$ and $\mathcal{B}$, $\min(\mathcal{A}, \mathcal{B})$, is the collection with the minimum number of edges. We distinguish the following cases.

Node u is a leaf. We have $X_u = \{a\}$. The only realization of u is $(\{a\}, \emptyset, \emptyset, \emptyset, \emptyset)$ and the corresponding collection of subpaths is the empty set, i.e. $\mathcal{S}((\{a\}, \emptyset, \emptyset, \emptyset, \emptyset)) = \emptyset$.

Node u is a forget node. Let v be the child of u. We have $X_u = X_v \setminus \{a\}$ and we can distinguish two cases:

- If $a \neq t$, then $R_u = (X, Y, Z, M, S)$ is a realization of u if and only if $R_v^1 = (X \cup \{a\}, Y, Z, M, S)$ or $R_v^2 = (X, Y, Z \cup \{a\}, M, S)$ are realizations of v. In this case $\mathcal{S}(R_u) = \min(\mathcal{S}(R_v^1), \mathcal{S}(R_v^2))$ (see Figure 2.13a).
- If $a = t$, then $R_u = (X, Y, Z, M, S)$ is a realization of u if and only if $R_v = (X, Y \cup \{a\}, Z, M, S)$ is a realization of v. In this case, $\mathcal{S}(R_u) = \mathcal{S}(R_v)$ (see Figure 2.13b).

To compute all the realizations of a forget node, we need at most $O(3^k k^k \Delta^k)$ steps.



(a) $a \neq t$ (b) $a = t$

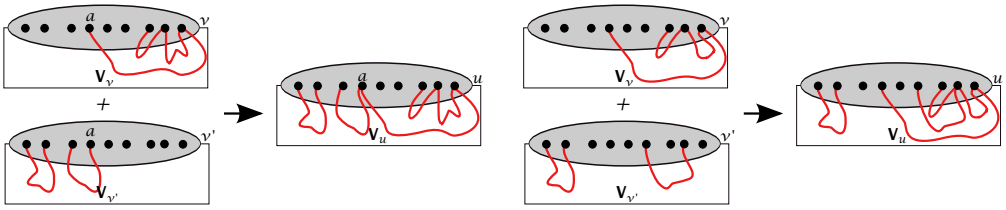
Figure 2.13: Finding solutions at a forget node: u is a forget node and v is its child and $X_u = X_v \setminus \{a\}$

Node u is a join node. Let v and v' be its children. Recall that $X_u = X_v = X_{v'}$. For any two realizations of v and v' we check if the union of the two solutions gives a realization of node u. The union $\mathcal{U}$ of two solutions gives a realization of node u if in $\mathcal{U}$, every vertex v of X_u has degree at most two and the transitions used around v are allowed (see Figure 2.14). For each realization of u, we keep the solution with the minimum number of edges.

Since there are at most $(3^{k+1}k^k\Delta^k)^2$ pairs of realizations at v and v' and for each pair of solutions we need to check the degree and the transitions around each vertex of X_u, we need at most $O((3^k k^k \Delta^k)^2 k)$ steps to find all of the realizations at a join node.

Node u is an introduce node. Let v be the child of u. We have $X_u = X_v \cup \{a\}$ and all neighbors of a in V_u are in X_u. Note that $a \neq t$ since t appears in a leaf node and its introduce parent. We can distinguish the following cases:

- If $a \in X$, then (X, Y, Z, M, S) is a realization of u if and only if $(X \setminus \{a\}, Y, Z, M, S)$ is a realization of v (see Figure 2.15a).
- if $a \in Y$, let $\{a, b\}$ be the edge incident to a in S. Since all neighbors of a in V_u are in X_u, then $b \in X_u \cap X_v$. If $b = t$, then $X_u = \{a, b\}$ (recall that t



(a) The union of the two solutions at v and v' is a solution at u if the transition used around a is allowed

(b) The union of the two solutions at v and v' is not a solution at u because it has a vertex of degree 3

Figure 2.14: Finding solutions at a join node: u is a join node and v and v' are its children

appears only in a leaf and its introduce parent) and $R_u = (\emptyset, \{a,t\}, \emptyset, \{(a,t)\}, \{\{a,t\}\})$ is the only realization of u and $\mathcal{S}(R_u)$ is the edge $\{a,t\}$. If $b \neq t$, we have the following cases:

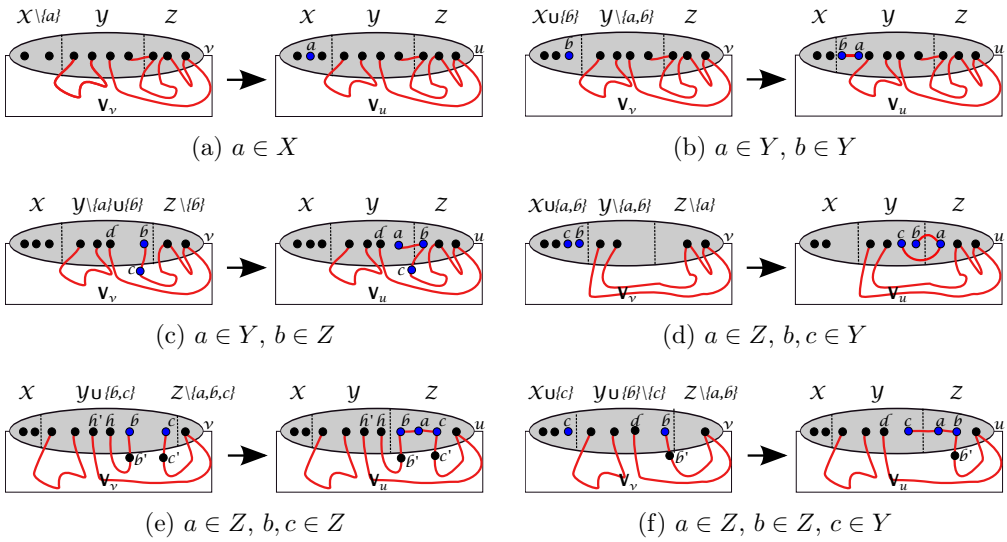
- If $b \in Y$, then $R_u = (X,Y,Z,M,S)$ is a realization if and only if $(a,b) \in M$, the edge incident to b in S is $\{a,b\}$ and $R_v = (X \cup \{b\}, Y \setminus \{a,b\}, Z, M', S')$ where $M' = M \setminus \{(a,b)\}$ and $S' = S \setminus \{\{a,b\}\}$ is a realization of v. In this case $\mathcal{S}(R_u) = \mathcal{S}(R_v) \cup \{\{a,b\}\}$ (see Figure 2.15b).
- If $b \in Z$, then $R_u = (X,Y,Z,M,S)$ is a realization of u if and only if v has a realization among the collections $R_v^c = (X, Y \setminus \{a\} \cup \{b\}, Z \setminus \{b\}, M', S')$ where $M' = (M \setminus (a,c)) \cup (b,c)$ and $S' = (S \setminus \{a,b\}) \cup \{b,c\}$ where c is a neighbor of b in V_v and $\{\{a,b\},\{b,c\}\} \notin \mathcal{F}$ (there are at most Δ such possible realizations). In this case $\mathcal{S}(R_u) = \min_c(\mathcal{S}(R_v^c)) \cup \{\{a,b\}\}$ (see Figure 2.15c).

- If $a \in Z$, then for every two neighbors b and c of a in X_u such that $\{\{b,a\},\{a,c\}\}$ is an allowed transition, we have the following cases (Note that the number of pairs of neighbors of a to consider is of the order of k^2.).
  - If $b \in Y$ and $c \in Y$, then $R_u = (X,Y,Z,M,S)$ is a realization of u if and only if $R_v = (X^0 \cup \{b,c\}, Y \setminus \{b,c\}, Z \setminus \{a\}, M', S')$, where $M' = M \setminus \{(b,c)\}$ and $S' = S \setminus \{\{a,b\},\{b,c\}\}$, is a realization of v. In this case $\mathcal{S}(R_u) = \mathcal{S}(R_v) \cup \{\{a,b\},\{a,c\}\}$ (see Figure 2.15d).
  - If $b \in Z$ and $c \in Z$, then $R_u = (X,Y,Z,M,S)$ is a realization of u if and only if v has a realization among the following collections $R_v^{h,h',b',c'} = R_v(X, Y \cup \{b,c\}, Z \setminus \{a,b,c\}, M', S')$, where $M' = M \cup \{(b,h),(c,h')\} \setminus \{(h,h')\}$, $(h,h') \in M$, and $S' = S \cup \{\{b,b'\},\{c,c'\}\}$ where b' and c' are neighbors of b and c in V_v, respectively, and $\{\{a,b\},\{b,b'\}\},\{\{a,c\},\{c,c'\}\} \notin \mathcal{F}$. There are at most Δ^2 possible choices for a pair (b',c'), and at most k possible choices for the pair

(h, h'). In this case $\mathcal{S}(R_u) = \min_{h,h',c',b'}(\mathcal{S}(R_v^{h,h',c',b'})) \cup \{\{a,b\},\{a,c\}\}$ (see Figure 2.15e).

– If $b \in Z$ and $c \in Y$, Let d be the vertex matched with c in M. $R_u = (X, Y, Z, M, S)$ is a realization of u if and only if v has a realization among the collections $R_v^{b'} = (X \cup \{c\}, Y \setminus \{c\} \cup \{b\}, Z \setminus \{a,b\}, M', S')$, where $M' = M \setminus \{c,d\} \cup \{b,d\}$ and $S' = S \setminus \{\{a,c\}\} \cup \{\{b,b'\}\}$ where b' is a neighbor of b in V_v such that $\{\{a,b\},\{b,b'\}\} \notin \mathcal{F}$. There are at most Δ possible choices for b'. In this case $\mathcal{S}(R_u) = \min_{b'}(\mathcal{S}(R_v^{b'})) \cup \{\{a,b\},\{a,c\}\}$ (see Figure 2.15f).

To solve all of the subproblems at an introduce node, we need at most $O((3^k k^k \Delta^k) k\Delta^2)$ steps.



(a) $a \in X$ (b) $a \in Y$, $b \in Y$

(c) $a \in Y$, $b \in Z$ (d) $a \in Z$, $b, c \in Y$

(e) $a \in Z$, $b, c \in Z$ (f) $a \in Z$, $b \in Z$, $c \in Y$

Figure 2.15: Finding solutions at an introduce node: u is an introduce and v is its child, $X_u = X_v \cup \{a\}$

Time complexity: The algorithm for $G_{e,f}$ runs in time $O(k(3k\Delta)^{2k}n)$ and knowing that we repeat it at most Δ^2 times to obtain the solution for G, the overall time complexity is $O(k\Delta^2(3k\Delta)^{2k}n)$.

□

Complexity for planar graphs The complexity of many dynamic programming algorithms on graphs with bounded treewidth can be improved for planar graphs using their planarity and structural properties. A speedup can usually be achieved, by using instead of the classical tree-decompositions, new decompositions that have been defined for planar graphs such as the *sphere cut branch decomposition* [DPBF05] and the *geometric tree-decomposition* [Dor10]. In particular, these

techniques have been used to improve the complexity of the algorithm which solves the Hamiltonian cycle problem in planar graphs from $O((3k)^k)$ to $O(6^k)$, where k is the treewidth. This is primarily due to the fact that the number of matchings to consider in such special tree-decompositions is bounded by 2^{k+1} as proved in [DPBF05]. This is roughly due to the fact that, in these special tree-decompositions, the vertices of a bag form a Jordan curve. This curve does not intersect with the curves corresponding to the edges of the planar embedding of the graph. Since in the matching we need to define, two matched vertices correspond to the endpoints of a subpath and the subpaths cannot cross each other, the number of valid matchings is bounded by a Catalan number. Taking this into consideration, our algorithm can run in time $O(k\Delta^2(6\Delta)^{2k}n))$ for planar graphs.

2.5 Disjoint trails avoiding forbidden transitions

We have studied in the previous section the problem of finding a path avoiding forbidden transitions and we have proved that it is NP-complete even in grids. This implies that finding disjoint paths is also NP-complete. In fact, taking an instance of PAFT from s to t in a graph G, we transform it into an instance of finding $k \geq 2$ disjoint paths avoiding forbidden transitions as follows. We construct a graph G' by taking G and adding $k-1$ valid disjoint paths from s to t. Finding k valid paths from s to t in G' is equivalent to finding a valid path from s to t in G.

In this section, since finding a trail avoiding forbidden transitions can be solved in polynomial time as we discussed in Section 2.3, we focus on the problem of finding disjoint trails avoiding forbidden transitions. We present results on the problems of finding k vertex-disjoint and k edge-disjoints trails avoiding forbidden transitions which are formulated as follows.

Problem 4 (Finding a k Vertex-Disjoint Trails avoiding Forbidden Transitions, k-VDT)**.** *Given a graph $G = (V, E)$, a set $\mathcal{F}$ of forbidden transitions and two vertices $s, t \in V$. are there k vertex-disjoint $\mathcal{F}$-valid s-t-trails in G?*

Problem 5 (Finding a k Edge-Disjoint Trails avoiding Forbidden Transitions, k-EDT)**.** *Given a graph $G = (V, E)$, a set $\mathcal{F}$ of forbidden transitions and two vertices $s, t \in V$. are there k edge-disjoint $\mathcal{F}$-valid s-t-trails in G?*

We have proved that k-EDT can be solved in polynomial time in directed graphs and then shown that k-VDT is NP-complete in directed and undirected graphs and polynomial in DAGs. After proving our results, we have found out that even though the complexity of the problem of finding disjoint $\mathcal{F}$-valid trails has not been established in the context of road networks nor optical networks, the related problems of finding vertex-disjoint monochromatic paths and edge-disjoint properly colored trails have been already studied and they imply stronger hardness results than some of the results we have proven. We briefly present these results before giving our proofs.

2.5.1 Related work

In an edge-colored graph, a properly-colored path (trail or walk) is a path which does not use consecutively two edges of the same color and a monochromatic path (trail or walk) is a path (trail or walk) which uses only edges of the same color. Both problems can be seen as problems of finding a path (trail or walk) avoiding forbidden transitions. In the first one, we forbid all of the transitions consisting of edges of the same color and in the second problem, we forbid all of the transitions consisting of edges of different colors. It has been proved in [GLMM12], that finding two vertex-disjoint monochromatic paths with different colors between two vertices is NP-complete in undirected edge-colored graphs with maximum degree 4 and directed graphs with maximum degree 8. Since the problems of monochromatic paths and trails are equivalent, it is NP-complete to find two vertex-disjoint monochromatic trails in undirected graphs. This implies that for $k \geq 2$, k-VDT is NP-complete in undirected graphs with maximum degree 4 and directed graphs of maximum degree 8. In [ADF+08], it has been proven that finding 2 edge-disjoint properly colored trails is NP-complete in undirected graphs. As a consequence, for $k \geq 2$, the problem k-EDT is NP-complete for undirected graphs.

2.5.2 Arc-disjoint trails avoiding forbidden transitions

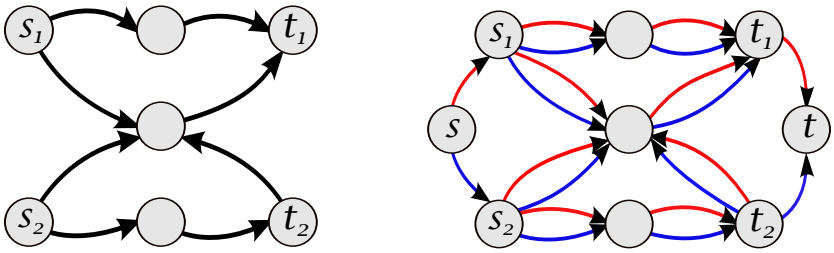
Theorem 4. *Finding $k \geq 2$ arc-disjoint trails avoiding forbidden transitions between two given vertices in a directed graph can be done in polynomial time.*

Proof. Let $D = (V, A)$ be a directed graph and s and t two vertices of V. Let $D' = (V', A')$ be the corresponding dual graph (as defined in section 2.3.1). Finding k arc-disjoint trails from s to t in $D = (V, A)$ is equivalent to finding k vertex-disjoint paths from s' to t' in $D' = (V', A')$. We do the proof for $k = 2$.

Let $P_1 = (s, v_1, \ldots, v_p, t)$ and $P_2 = (s, w_1, \ldots, w_q, t)$ be two arc-disjoint $\mathcal{F}$-valid trails in D. The two paths $P'_1 = (s', u_{(v_1,v_2)}, \ldots, u_{(v_{p-1},v_p)}, t')$ and $P'_2 = (s', u_{(w_1,w_2)}, \ldots, u_{(w_{q-1},w_q)}, t')$ are two vertex-disjoint paths in D'. In fact, since $\{\{v_i, v_{i+1}\}, \{v_{i+1}, v_{i+2}\}\} \notin \mathcal{F}$ and $\{\{w_j, w_{j+1}\}, \{w_{j+1}, w_{j+2}\}\} \notin \mathcal{F}$ the arcs $(u_{(v_i,v_{i+1})}, u_{(v_{i+1},v_{i+2})})$ and $(u_{(w_j,w_{j+1})}, u_{(w_{j+1},w_{j+2})})$ exist in D' and since every vertex in D' corresponds to an arc in D, the arc-disjointness implies the vertex-disjointness.

Now let us suppose that there are two vertex-disjoint paths in D', $P'_1 = (s', u_{a_1}, \ldots, u_{a_p}, t')$ and $P'_2 = (s', u_{b_1}, \ldots, u_{b_q}, t')$. Let P_1 and P_2 be the trails of D formed by the arcs $a_1, \ldots, a_p$ and the arcs $b_1, \ldots, b_q$, respectively. Since there is an arc between u_{a_i} and $u_{a_{i+1}}$ in D' for $1 \leq i \leq q$, then the arcs a_i and a_{i+1} form an allowed transition in D and hence P_1 is an $\mathcal{F}$-valid trail. Similarly, P_2 is an $\mathcal{F}$-valid trail. Finally, since every vertex in D' corresponds to an arc in D, the vertex-disjointness implies the arc-disjointness.

□



(a) A directed graph D with two pairs of vertices (s_1, t_1) and (s_2, t_2)

(b) The corresponding directed graph D' (The parallel directed paths of length 2 are represented by multi-arcs)

Figure 2.16: Example of the reduction

2.5.3 Vertex-disjoint trails avoiding forbidden transitions

We focus in this section on the k-VDT problem. We first prove that it is NP-hard in general directed graphs and polynomial in directed acyclic graphs. Afterwards, we show that it is NP-hard in undirected graphs.

2.5.3.1 Vertex-disjoint trails in directed graphs

As pointed out in Section 2.5.1, since finding two vertex-disjoint monochromatic paths with different colors between two vertices is NP-complete in arc-colored directed graphs with maximum degree 8 [GLMM12], then finding 2 vertex-disjoint trails is NP-complete in directed graphs with maximum degree 8. We present in what follows a weaker result with an easy reduction.

Theorem 5. *The k-VDT problem, for $k \geq 2$, is NP-complete in directed graphs.*

Proof. The problem is clearly in NP. To prove the hardness we use a reduction from the problem of finding two vertex-disjoint paths in a directed graph. In this problem, we are given a directed graph D and two pairs of vertices (s_1, t_1) and (s_2, t_2) and the objective is to find two vertex-disjoint paths P_1 and P_2 in D such that P_1 is from s_1 to t_1 and P_2 is from s_2 to t_2. This problem has been proven NP-complete in [FHW80].

Let $(D = (V, A), (s_1, t_1), (s_2, t_2))$ be an instance of the problem of finding 2 vertex-disjoint paths in a directed graph. We build a directed graph D' with a set of forbidden transitions $\mathcal{F}$ and two vertices s and t such that finding 2 vertex-disjoint paths in D is equivalent to finding 2 vertex-disjoint $\mathcal{F}$-valid trails from s to t in D'. To build D', we replace each arc of A with two parallel directed paths of length 2; one red path and one blue path. We add a vertex s connected to s_1 with a red arc (s, s_1) and connected to s_2 with a blue arc (s, s_2). We add a vertex t connected to t_1 with a red arc (t_1, t) and to t_2 with a blue arc (t_2, t). The set of forbidden transitions $\mathcal{F}$ contains all the transitions of D' which consist of a blue arc and a red arc. Figure 2.16 presents an example illustrating the reduction.

Let P_1 and P_2 be two vertex-disjoint paths in D, such that P_1 is from s_1 to t_1 and P_2 is from s_2 to t_2. Let P'_1 (resp. P'_2) be the path in D' obtained from P_1 (resp. P_2) by replacing each arc by the corresponding red (resp. blue) directed path of length 2 in D'. Let $P^{"}_1$ be the directed path of D' obtained from the concatenation of the arc (s, s_1), the directed path P'_1 and then the arc (t_1, t) and $P^{"}_2$ be the directed path of D' obtained from the concatenation of the arc (s, s_2), the directed path P'_2 and then the arc (t_2, t). The two paths $P^{"}_1$ and $P^{"}_2$ are $\mathcal{F}$-valid since they only use transitions of arcs of the same colors, they are also vertex-disjoint since the subpaths P'_1 and P'_2 are vertex-disjoint.

Now let P_1 and P_2 be two vertex-disjoint $\mathcal{F}$-valid trails from s to t in D' such that P_1 is the path using arc (s, s_1). Since P_1 is $\mathcal{F}$-valid and $\mathcal{F}$ contains all transitions consisting of a red and blue edge, P_1 only uses red arcs and it hence uses the arc (t_1, t). Similarly, P_2 uses only blue arcs and hence uses the arc (t_2, t). Let P'_1 be the subtrail of P_1 from s_1 to t_1 and P'_2 the subtrail of P_2 from s_2 to t_2. By removing all the subcycles of P'_1 and P'_2, we obtain two vertex-disjoint paths in D, one from s_1 to t_1 and the other from s_2 to t_2. □

2.5.3.2 Vertex-disjoint trails in DAGs

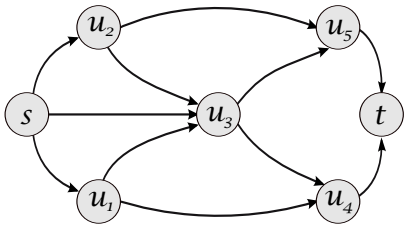
We prove in this section, through Theorem 6, that k-VDT is polynomial in DAGs. In the proof of the theorem, we use ideas of [Cha03], in particular that of layered directed graph and a construction similar to that used to find a polynomial time algorithm for disjoint paths with forbidden pairs in layered directed graphs (Theorem 6 of [Cha03]).

Definition 2 (Layered directed graph)**.** *A directed graph $G = (V, E)$ is layered if there is a layering function $l : V \to [0, 1, \ldots, (|V| - 1)]$ such that for every arc $(u, v) \in E$, $l(v) = l(u) + 1$. We say that vertex u is in layer $l(u)$ and arc (u, v) is in layer $l(u)$. Layered directed graphs are acyclic.*

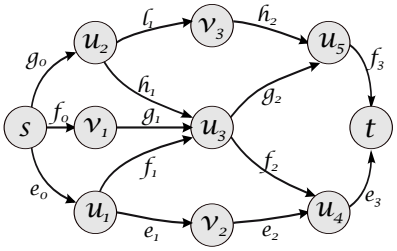
Theorem 6. *The k-VDT problem, for a fixed $k \geq 2$, can be solved in polynomial time in directed acyclic graphs.*

Proof. Let D be DAG with a set of forbidden transitions $\mathcal{F}$ and let s and t be two given vertices. As we want to find (in polynomial time) directed paths from s to t, we can delete the vertices not on a directed path from s to t, and so we suppose in what follows that, in D, s is the unique vertex with no in-neighbor and t the unique vertex with no out-neighbor. For the ease of presentation, we first give the transformation for $k = 2$. The algorithm that finds 2 vertex-disjoint paths from s to t in D uses two transformations:

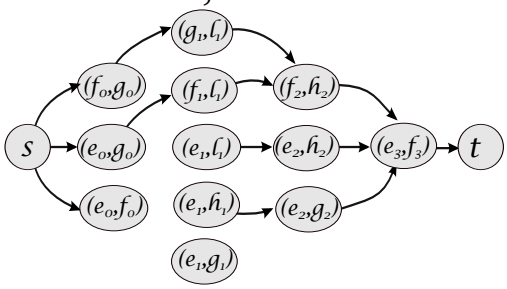
Transformation 1 We first associate with the DAG D and the set of forbidden transitions $\mathcal{F}$ a layered DAG LD and a set of forbidden transitions $\mathcal{F}'$ as follows. We denote by $\Gamma^-(v)$ the set of the in-neighbors of v, i.e; vertices u such that $(u, v) \in E$.



(a) A DAG D with a set of forbidden transitions $\mathcal{F} = \Big\{\{\{s,u_1\},\{u_1,u_4\}\}, \{\{s,u_2\},\{u_2,u_3\}\},\{\{s,u_3\},\{u_3,u_5\}\}\Big\}$



(b) The corresponding layered DAG LD with a set of forbidden transitions $\mathcal{F}' = \big\{\{e_0,e_1\},\{g_0,h_1\},\{g_1,g_2\}\big\}$



(c) The corresponding directed graph H

Figure 2.17: Example the transformations for a DAG

We compute the function $l : V \to \mathbb{N}$ defined as follows:

$$l(v) = \begin{cases} 0 & \text{when } v = s, \\ 1 + \max_{u \in \Gamma^-(v)} l(u) & \text{otherwise.} \end{cases}$$

In such a level function, $l(v)$ is the length of the longest path from s to v, and t has the maximum value as there is a directed path from any vertex to t in the reduced DAG. In the example of Figure 2.17a, we have $l(u_1) = l(u_2) = 1$, $l(u_3) = 2$, $l(u_4) = l(u_5) = 3$, and $l(t) = 4$.

Now we replace every arc (u, v), such that $l(v) > l(u) + 1$, with a directed path P_{uv} from u to v of length $l(v) - l(u)$ (thus possibly adding new vertices and arcs). The set of forbidden transitions $\mathcal{F}'$ in LD is defined as follows. For an arc (u, v) of D, all of the transitions of the path P_{uv} are allowed. Furthermore, if (w, u) and (u, v) form an allowed (resp. forbidden) transition in D, then the last arc of P_{wu} and the first arc of P_{uv} form an allowed (resp. forbidden) transition in LD. In Figure 2.17b, we give the layered DAG LD obtained from the DAG D of Figure 2.17a. We have given a name to each arc with a lower index indicating the level of the arc.

Therefore, in what follows we consider a layered DAG LD with its set of forbidden transitions $\mathcal{F}'$ and two specific vertices s and t.

Transformation 2 We use in this transformation ideas similar to that used in [Cha03] to solve the problem of finding a pair of vertex-disjoint paths with for-

bidden pairs of edges in a layered DAG such that the edges of a forbidden pair are in the same layer.

With LD we will associate a directed graph H with two specific vertices s and t, such that there exist 2 vertex-disjoint $\mathcal{F}'$-valid paths in LD from s to t if and only if there exists a directed path from s to t in H.

There is a vertex in H for every pair $\{e_i, f_i\}$ of arcs in LD at the same layer i, with $0 \leq i \leq l(t)-1$, such that $e_i \cap f_i \in \{s, t\}$. We also add to H two vertices s and t. Now we join, by an arc in H, s to all the vertices (pairs) $\{e_0, f_0\}$. Similarly we join every vertex $\{e_{l(t)-1}, f_{l(t)-1}\}$ in H by an arc to t. Finally, for $0 \leq i \leq l(t) - 2$, we join in H each vertex $\{e_i, f_i\}$ to a vertex $\{e_{i+1}, f_{i+1}\}$ if in LD we have the following properties:

1. the terminal vertex u_i of e_i is the initial vertex of e_{i+1}, and $\{e_i, e_{i+1}\}$ is an allowed transition (i.e. not forbidden); and

2. the terminal vertex u_v of f_i is the initial vertex of f_{i+1}, and $\{f_i, f_{i+1}\}$ is an allowed transition (i.e. not forbiden).

Figure 2.17c indicates the directed graph H obtained from the layered DAG LD of Figure 2.17b. For example, we have three vertices corresponding to pairs of arcs of layer 0 of LD: (e_0, f_0), (e_0, g_0) and (f_0, g_0) and 5 vertices corresponding to pairs of arcs of layer 1: (e_1, g_1), (e_1, h_1), (e_1, l_1), (f_1, l_1), and (g_1, l_1). Vertex (e_0, f_0) is not connected to vertex (e_1, g_1) since the transition (e_0, e_1) is forbidden.

The existence of two vertex-disjoint $\mathcal{F}'$-valid paths in LD named $P = (s, e_0, u_0, e_1, \ldots, e_{l(t)-2}, u_{l(t)-1}, e_{l(t)-1}, t)$ and $Q = (s, f_0, v_0, f_1, \ldots, f_{l(t)-2}, v_{l(t)-1}, f_{l(t)-1}, t)$ implies the existence of a directed path from s to t namely $PQ = (s, \{e_0, f_0\}, \{e_1, f_1\}, \ldots, \{e_{l(t)-1}, f_{l(t)-1}\}, t)$ in H.

Conversely, let W be a path in H written in the form $W = (s, w_0, w_1, \ldots, w_{l(t)-1}, t)$ where w_i corresponds to the pair $\{e_i, f_i\}$ and w_{i+1} to the the pair $\{e_{i+1}, f_{i+1}\}$ such that $\{e_i, e_{i+1}\}$ and $\{f_i, f_{i+1}\}$ are allowed transitions in LD. Then, the two directed paths $P = (s, e_0, u_0, e_1, \ldots, e_{l(t)-2}, u_{l(t)-1}, e_{l(t)-1}, t)$ and $Q = (s, f_0, v_0, f_1, \ldots, f_{l(t)-2}, v_{l(t)-1}, f_{l(t)-1}, t)$ are vertex-disjoint and $\mathcal{F}'$-valid. In the example of Figure 3.6c, H has two directed paths from s to t. For example with the directed path $P = (s, \{e_0, g_0\}, \{f_1, l_1\}, \{f_2, h_2\}, \{e_3, f_3\}, t)$, the vertex-disjoint $\mathcal{F}'$-valid paths $P_1 = (s, e_0, f_1, f_2, e_3, t)$ and $P_2 = (s, g_0, l_1, h_2, f_3, t)$ in LD and the two vertex-disjoint $\mathcal{F}$-valid paths (s, u_1, u_3, u_4, t) and (s, u_2, u_5, t) in D are associated.

The algorithm can be generalized to find k color-disjoint paths from s to t in a DAG D, for any $k \geq 2$. We first transform D to a layered graph LD as before. Then, in the second transformation, instead of having a vertex for every pair of arcs of the same layer, we create a vertex for every k-tuple of arcs $\{e_i^1, e_i^2, \ldots, e_i^k\}$ at the same layer i, such that the e_i^j , for $j = 1, \ldots, k$, are disjoint or only share s or t. Then an arc is added from node $\{e_i^1, e_i^2, \ldots, e_i^k\}$ to node $\{e_{i+1}^1, e_{i+1}^2, \ldots, e_{i+1}^k\}$ if there exists an ordering of the e_i^j and of the e_{i+1}^j such that the terminal vertex of e_i^j is the initial vertex of e_{i+1}^j and the transitions $\{e_i^j, e_{i+1}^j\}$ are allowed.

□

2.5.3.3 Vertex-disjoint trails in undirected graphs

As pointed out in Section 2.5.1, since finding two vertex-disjoint monochromatic paths with different colors between two vertices is NP-complete in undirected edge-colored graphs with maximum degree 4 [GLMM12], then finding 2 vertex-disjoint trails is NP-complete in undirected graphs with maximum degree 4. We present in what follows a weaker result with a different reduction in which the graph is 1-planar, i.e. a graph that can be embedded in the plane in such a way that each edge has at most one crossing point, where it crosses a single additional edge.

Theorem 7. *The k-VDT problem, for $k \geq 2$, is NP-complete in undirected graph with maximum degree 8.*

Proof. The problem is clearly in NP. We prove the hardness using a reduction from the 3-SAT problem. The reduction is similar to the one used in the proof of Lemma 1. Let Φ be an instance of 3-SAT, i.e., Φ is a boolean formula with variables $\{v_1, \cdots, v_n\}$ and clauses $\{C_1, \cdots, C_m\}$. We build a grid-like planar graph G similar to the one in the proof of Lemma 1. The difference between the graph G and the main graph in the proof of Lemma 1 is mainly in the way the clause-graphs are combined; a new gadget called the 2-path-gadget is used to combine the clause-graphs. In particular, this new way of combining the clause-graphs breaks the planarity our reduction. In what follows we present how to build G using the clause-gadgets and the 2-path-gadgets. Here again, multigraphs can be easily transformed to graphs by subdividing the edges and the colors and the multi-edges are only used to make the presentation easier.

Clause-graph G_j. For each $1 \leq j \leq m$, we build a Clause-gadget G_j with a set of forbidden transitions $\mathcal{F}_j$ similar to the one built in the proof of Lemma 1 (see Figure 2.18b).

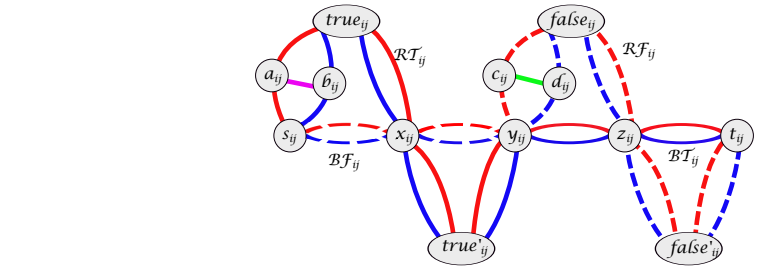
Any $\mathcal{F}_j$-valid trail from s_j to t_j starts with a red edge and ends with a blue edge (or the inverse). Since it is not possible to go from a red edge to a blue edge without using a pink or a green edge, Claim 9 follows.

Claim 9. *Any $\mathcal{F}_j$-valid trail P from s_j to t_j must use a green or a pink edge in a gadget G_{ij} for some $1 \leq i \leq n$.*

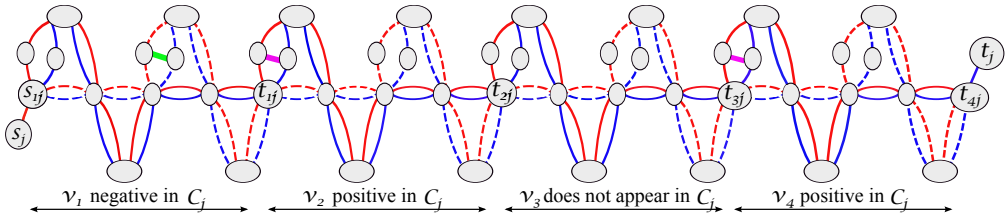
Another important property of $\mathcal{F}_j$-valid trails is the following.

Claim 10. *If P is an $\mathcal{F}_j$-valid trail from s_j to t_j in G_j, then for each $1 \leq i \leq n$:*

- *P passes through both vertices $true_{ij}$ and $true'_{ij}$, or*
- *P passes through both vertices $false_{ij}$ and $false'_{ij}$.*



(a) Gadget G_{ij}



(b) Clause-graph G_j (j odd) for a Clause $C_j = \bar{v}_1 \vee v_2 \vee v_4$ in a formula with 4 Variables.

Figure 2.18: The clause-gadget (reminder)

Proof of claim.

Let P be an $\mathcal{F}_j$-valid trail from s_j to t_j in G_j. For $1 \leq i \leq n$, P passes necessarily through $true_{ij}$ or $false_{ij}$. Let us suppose P uses vertex $true_{ij}$, for a given $1 \leq i \leq n$. If $true_{ij}$ appears in P after one of the vertices a_{ij} and b_{ij}, then with the way the set of forbidden transitions $\mathcal{F}_j$ is defined, the vertices after $true_{ij}$ should be x_{ij} and $true'_{ij}$. Namely, around $true_{ij}$ the allowed transitions containing a_{ij} (resp. b_{ij}) is $\{\{a_{ij}, true_{ij}\}, \{true_{ij}, x_{ij}\}\}$ (resp. $\{\{b_{ij}, true_{ij}\}, \{true_{ij}, x_{ij}\}\}$) and the allowed transition around x_{ij} is $\{\{true_{ij}, x_{ij}\}, \{x_{ij}, true'_{ij}\}\}$. Otherwise, if $true_{ij}$ appears in P after x_{ij}, then with the way the set of forbidden transitions $\mathcal{F}_j$ is defined, the vertex used before x_{ij} should be $true'_{ij}$. If P uses vertex $true'_{ij}$, for a given $1 \leq i \leq n$, then if $true'_{ij}$ appears in P after the vertex x_{ij}, then with the way the set of forbidden transitions $\mathcal{F}_j$ is defined, the vertex before x_{ij} should be $true_{ij}$. Otherwise, if $true'_{ij}$ appears in P after the vertex y_{ij}, then with the way the set of forbidden transitions $\mathcal{F}_j$ is defined, the vertices that appear after $true'_{ij}$ should be x_{ij} and then $true_{ij}$. The proof is similar for the case where P uses vertices $false_{ij}$ and $false'_{ij}$.

Note that unlike the paths in the proof of Lemma 1, an $\mathcal{F}_j$-valid trail from s_j to t_j in G_j can use $true_{ij}$ and $true'_{ij}$ and at the same time $false_{ij}$ and $false'_{ij}$. ◇

2-path-gadget Π. This gadget consists of two intersecting paths of length $4n$ each.

- $(x_1, true_1, y_1, true'_1, \ldots, x_n, true_n, y_n, true'_n, x_{n+1})$; this path is depicted in full black in the example of Figure 2.19.

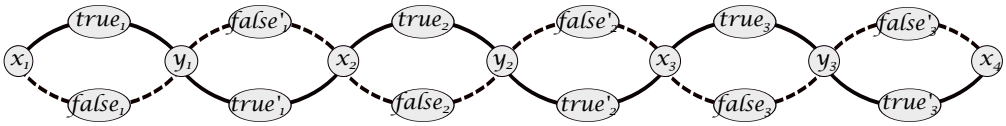


Figure 2.19: A 2-path-gadget for an instance with 3 variables $\{v_1, v_2, v_3\}$

- $(x_1, false_1, y_1, false'_1, \ldots, x_n, false_n, y_n, false'_n, x_{n+1})$; this path is depicted in dotted black in the example of Figure 2.19.

The set of forbidden transitions $\mathcal{F}_\pi$ in the 2-path-gadget Π contains all the transitions $\{\{true_i, y_i\}, \{y_i, false'_i\}\}$, and $\{\{false_i, y_i\}, \{y_i, true'_i\}\}$, $1 \leq i \leq n$. Claim 11 presents the properties of an $\mathcal{F}_\pi$-valid trail in Π. Note that any $\mathcal{F}_\pi$-valid trail between x_1 and x_{n+1} is a path.

Claim 11. *Let P be an $\mathcal{F}_\pi$-valid trail between x_1 and x_{n+1} in Π. Then, for $1 \leq i \leq n$ either*

- *P passes through $true_i$ and $true'_i$ and does not pass through $false_i$ nor $false'_i$, or*
- *P passes through $false_i$ and $false'_i$ and does not pass through $true_i$ nor $true'_i$.*

Main Graph G. The graph G is built by combining the clause-gadgets G_j, $1 \leq j \leq m$, as well as $m-1$ copies Π^j, $1 \leq j < m$, of the 2-path-gadget Π (we denote by v^j a vertex v of the copy Π^j). An illustrating example of the combination is presented in Figure 2.20 and the details are in what follows.

- For each $1 \leq j < m$, we identify the vertices t_j and s_{j+1}.
- For each $1 \leq j \leq m-2$, we add a black edge between x^j_{n+1} and x^{j+1}_1. Note that these are the only edges breaking the planarity.
- We add a black edge between s_1 and x^1_1 and another black edge between x^{m-1}_{n+1} and t_m.
- For each $1 \leq j < m$, and for each $1 \leq i \leq n$ we do the following:
  - We identify the vertices $true'_{ij}$ and $false'_{ij}$ of G_j with the vertices $true^j_i$ and $false'^j_i$ of Π^j, respectively.
  - We identify the vertices $true'^j_i$ and $false^j_i$ of Π^j with the vertices $true_{i,j+1}$ and $false_{i,j+1}$ of G_{j+1}, respectively.

The set of forbidden transitions $\mathcal{F}$ include besides all the forbidden transitions $\mathcal{F}_j$ of G_j, $1 \leq j \leq m$, and $\mathcal{F}^j_\pi$ of Π^j, $1 \leq j < m$, all transitions consisting of a black edge and a blue or red one. This way it is not possible to go from G_j to Π^j. Note that the maximum degree of the graph G is 8.

The following claims present the key properties of an $\mathcal{F}$-valid trail from s_1 to t_m in G and the properties of 2 vertex-disjoint $\mathcal{F}$-valid trails from s_1 to t_m in G.

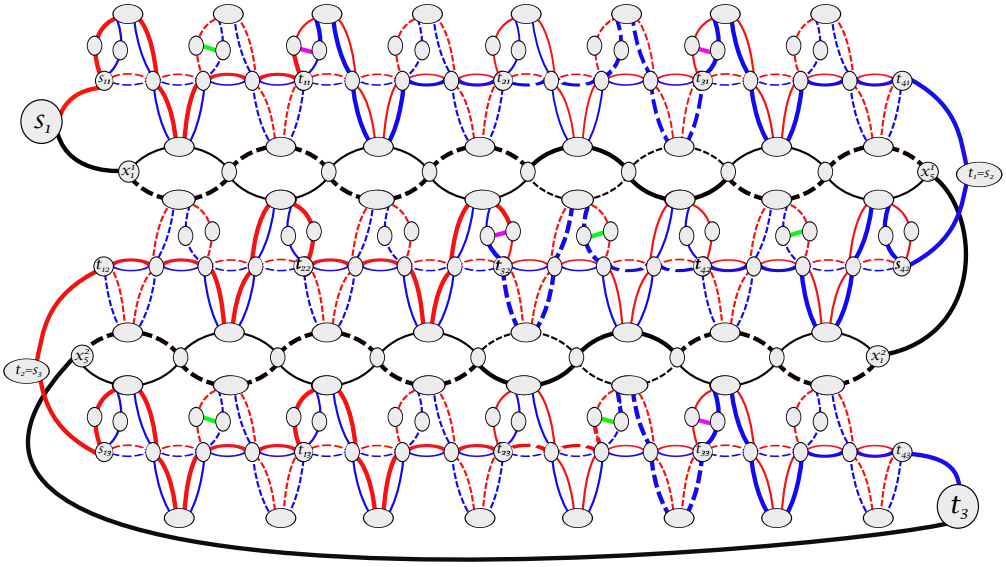


Figure 2.20: The graph G for an instance of 3-SAT with 4 variables v_1, v_2, v_3 and v_4 and 3 clauses $C_1 = \bar{v}_1 \vee v_2 \vee v_4$, $C_2 = v_2 \vee \bar{v}_3 \vee \bar{v}_4$ and $C_3 = \bar{v}_1 \vee \bar{v}_3 \vee v_4$. The two bold paths in black and red-blue from s_1 to t_3 are 2 vertex-disjoint paths corresponding to the assignment of v_1, v_2 and v_4 to True and v_3 to False. This assignment satisfies all clauses.

Claim 12. *If P is an $\mathcal{F}$-valid trail from s_1 to t_m in G, then:*

- *P consists of the concatenation of $\mathcal{F}_j$-valid trails from s_j to t_j in G_j, from $j = 1$ to m, P is called then a **colored** trail; or*
- *P consists of the concatenation of $\mathcal{F}^j_\pi$-valid trails from x^j_1 to x^j_{n+1} in Π^j, from $j = 1$ to $m-1$, and the edges $\{s_1, x^1_1\}$, $\{x^j_{n+1}, x^{j+1}_1\}$ for $1 \leq j \leq m-2$ and $\{x^{m-1}_{n+1}, t_m\}$, P is called then a **black** trail.*

Proof of claim. Let P be an $\mathcal{F}$-valid trail from s_1 to t_m in G. If P uses the black edge $\{s, x^1_1\}$, then P only uses black edges since all transitions containing a black edge and a blue or red edge are forbidden. In the subgraph of G induced by the black edges, an $\mathcal{F}$-valid trail from s_1 to t_m consists of the concatenation of $\mathcal{F}^j_\pi$-valid trails from x^j_1 to x^j_{n+1} in Π^j, from $j = 1$ to $m-1$, and the edges $\{s_1, x^1_1\}$, $\{x^j_{n+1}, x^{j+1}_1\}$ for $1 \leq j \leq m-2$ and $\{x^{m-1}_{n+1}, t_m\}$. If P uses the red edge $\{s, s_{11}\}$, then P cannot use black edges. In the subgraph of G induced by the red, blue, pink and green edges, an $\mathcal{F}$-valid trail from s_1 to t_m consists of the concatenation of $\mathcal{F}_j$-valid trails from s_j to t_j in G_j, from $j = 1$ to m. ◇

Note that for every 2 vertex-disjoint $\mathcal{F}$-valid trails from s_1 to t_m in G, one trail uses the black edge $\{s, x^1_1\}$ and is hence a *black* trail and the other uses the red edge $\{s, s_{11}\}$ and is hence a *colored* trail.

Claim 13. *If P and P' are 2 vertex-disjoint $\mathcal{F}$-valid trails from s_1 to t_m in G, then P is a colored trail and P' is a black trail or vice-versa.*

Claim 14. *If P and P' are 2 vertex-disjoint $\mathcal{F}$-valid trails from s_1 to t_m in G and P is the colored path, then, for any $1 \leq i \leq n$, either*

- *for any $1 \leq j \leq m$, the trail P passes through $true_{ij}$ and $true'_{ij}$ and does not pass through $false_{ij}$ nor $false'_{ij}$, or*
- *for any $1 \leq j \leq m$, the trail P passes through $false_{ij}$ and $false'_{ij}$ and does not pass through $true_{ij}$ nor $true'_{ij}$.*

Proof of claim. By Claims 12, for any $1 \leq j \leq m$, there is a subtrail P_j of P that goes from s_j to t_j. Let us prove first that for $1 \leq j \leq m$, for any $1 \leq i \leq n$ the subtrail P_j passes through $true_{ij}$ and $true'_{ij}$ and does not pass through $false_{ij}$ nor $false'_{ij}$ or passes through $false_{ij}$ nor $false'_{ij}$ and does not pass through $true_{ij}$ and $true'_{ij}$. By Claim 10, for $1 \leq i \leq n$, P_j either passes through $true_{ij}$ and $true'_{ij}$, or through $false_{ij}$ and $false'_{ij}$. Let us assume that we are in the first case (the second case can be handled symmetrically). Since P_j is using $true'_{ij} = true^j_i$ and P is vertex disjoint from P', then by Claims 12 and 11, P' is using $false'^j_i = false'_{ij}$. Again, since P and P' are vertex disjoint, P_j does not use $false'_{ij}$ nor $false_{ij}$.

Now, let us suppose that for a given $1 \leq i \leq n$, that P_1 passes through $true_{i1}$ and $true'_{i1}$, and does not pass through $false_{i1}$ and $false'_{i1}$ (the second case can be handled symmetrically). We prove by induction on $j \leq m$ that P_j passes through $true_{ij}$ and $true'_{ij}$ and does not pass through $false_{ij}$ nor $false'_{ij}$. Indeed, if P_j passes through $true_{ij} = true'^{j-1}_i$ and $true'_{ij} = true^j_i$, then P_{j+1} cannot use $false_{i,j+1}$ nor $false'_{i,j+1}$ since P' which is vertex disjoint from P uses by claim 11 $false^j_i = false_{i,j+1}$. ◇

Claim 15. *Φ is satisfiable if and only if there are 2 vertex-disjoint $\mathcal{F}$-valid trails from s_1 to t_m in G.*

Proof of claim. Let φ be a truth assignment which satisfies Φ. We can build two vertex-disjoint $\mathcal{F}$-valid trails P and P' from s_1 to t_m in G.

The trail P is a colored trail and can be built as was built the path in the proof of Claim 8. In more details, For each row $1 \leq j \leq m$, we build a path P_j from s_i to t_j by concatenating the paths P_{ij}, $1 \leq j \leq m$, which are built as follows. Among the variables that appear in C_j, let v_q be the variable with the smallest index, which satisfies the clause.

- For $1 \leq i < q$, if $\varphi(v_i) = true$, then $P_{ij} = \mathcal{R}T_{ij}$ if j is odd and $P_{ij} = \mathcal{B}T_{ij}$ if j is even. If $\varphi(v_i) = false$, then $P_{ij} = \mathcal{R}F_{ij}$ if j is odd, and $P_{ij} = \mathcal{B}F_{ij}$ if j is even.
- For $i = q$, if $\varphi(v_i) = true$, then $P_{ij} = \mathcal{R}\mathcal{B}T_{ij}$ if j is odd, and $P_{ij} = \mathcal{B}\mathcal{R}T_{ij}$ if j is even. If $\varphi(v_i s) = false$, then $P_{ij} = \mathcal{R}\mathcal{B}F_{ij}$ if j is odd, and $P_{ij} = \mathcal{B}\mathcal{R}F_{ij}$ if j is even.

- For $q < i \leq n$, if $\varphi(v_i) = true$, then $P_{ij} = \mathcal{B}T_{ij}$ if j is odd, and $P_{ij} = \mathcal{R}T_{ij}$ otherwise. If $\varphi(v_i) = false$, then $P_{ij} = \mathcal{B}F_{ij}$ if j is odd, and $P_{ij} = \mathcal{R}F_{ij}$ otherwise.

The path P obtained from the concatenation of paths P_j for $1 \leq j \leq m$ is an $\mathcal{F}$-valid path from s_1 to t_m.

The trail P' is a black trail and is built from the concatenation of the edges $\{s_1, x_1^1\}$, $\{x_{n+1}^j, x_1^{j+1}\}$ for $1 \leq j \leq m-2$ and $\{x_{n+1}^{m-1}, t_m\}$ and the subtrails P^j, $1 \leq j < m$, built as follows. From $i = 1$ to m, P^j uses the vertex x_i^j, then:

- If P_{ij} uses $true_{ij}$, P^j uses vetices $false_{i,j+1}$, y_i^j and then $false'_{ij}$.
- If P_{ij} uses $false_{ij}$, P^j uses vetices $true_{i,j+1}$, y_i^j and then $true'_{ij}$.

At last, P^j uses vertex x_n^j. The trail P' is $\mathcal{F}$-valid and by definition vertex-disjoint from P_1.

Now let us suppose that there are two vertex-disjoint $\mathcal{F}$-valid paths P and P' from s_1 to t_m. Let P be the colored one. According to Claim 14, for any $1 \leq i \leq n$, for any $1 \leq j \leq m$, P passes through $true_{ij}$ and $true'_{ij}$ and does not pass through $false_{ij}$ and $false'_{ij}$, or P passes through $false_{ij}$ and $false'_{ij}$ and does not pass through $true_{ij}$ and $true'_{ij}$. Let us then consider the truth assignment φ of Φ such that for each $1 \leq i \leq n$:

- If P uses $true_{ij}$ and $true'_{ij}$ in all rows $1 \leq j \leq m$, then $\varphi(v_i) = true$.
- If P uses $false_{ij}$ and $false'_{ij}$ in all rows $1 \leq j \leq m$, then $\varphi(v_i) = false$.

Thanks to Claim 14, φ is a valid truth assignment. We need to prove that φ satisfies Φ. According to Claim 12, for each $1 \leq j \leq m$, P contains an $\mathcal{F}_j$-valid path P_j from s_j to t_j and according to Claim 9, P_j uses a green or a pink edge. We prove in the following that for $1 \leq i \leq n$ and $1 \leq j \leq m$, if P_j uses a pink edge $\{a_{ij}, b_{ij}\}$, then P uses the vertex $true_{ij}$ and if P_j uses a pink edge $\{c_{ij}, d_{ij}\}$, then P uses the vertex $false_{ij}$.

- Let us suppose that P_j uses a pink edge $\{a_{ij}, b_{ij}\}$ and does not use the vertex $true_{ij}$. This implies that P_1 contains the cycle $\{t_{i-1,j}, a_{ij}, b_{ij}\}$ and hence $P_{1,j}$ uses at least 3 edges incident to $t_{i-1,j}$ from the side of s (two edges to leave and enter the cycle and an additional edge to cross again the cut-vertex $t_{i-1,j}$ towards t_j). Trail P_j uses then at least one edge $\{z_{i,j-1}, t_{i,j-1}\}$ and at least one edge $\{false'_{i,j-1}, t_{i,j-1}\}$. This is not possible since it implies that P_j uses vertices $true_{i-1,j}$ and $false_{i-1,j}$ which contradicts Claim 14.
- Let us suppose that P_j uses a green edge $\{c_{ij}, d_{ij}\}$ and does not use the vertex $false_{ij}$. This implies that P_j contains the cycle $\{y_{ij}, c_{ij}, d_{ij}\}$ and hence P_j uses at least 3 edges incident to $y_{i,j}$ from the side of s (two edges to enter and leave the cycle and an additional one to cross again the cut-vertex $y_{i,j}$ towards t_j). This implies that P uses one edge $\{z_{i-1,j}, t_{i-1,j}\}$ and one edge

$\{false'_{i-1,j}, t_{i-1,j}\}$. This is not possible since it implies that P_j uses vertices $true_{i-1,j}$ and $false_{i-1,j}$ which contradicts Claim 14.

Thus, the use of a pink edge in P_j forces P_j (and hence P) to use, for a variable v_i that appears positively in C_j, the vertices $true_{ij}$ and $true'_{ij}$. Similarly, the use of a green edge in P_j forces P_j (and hence P) to use, for a variable v_i that appears negatively in C_j, the vertices $false_{ij}$ and $falce'_{ij}$. This means that for each clause C_j, for one of the variables that appear in C_j which we denote v_i, $\varphi(v_i) = true$ ($\varphi(v_i) = false$) if v_i appears positively (negatively) in C_j, respectively. Thus, the truth assignment φ satisfies Φ. ◇

□

Remark. Please note that all of the constructions in the NP-completeness proof of Theorem 7 can be done with directed graphs, which implies that k-VDT is NP-complete in directed graphs with maximum degree 8. In fact, the gadgets G_{ij} can be built with directed paths $\mathcal{RT}_{ij}, \mathcal{BT}_{ij}, \mathcal{RF}_{ij}$ and $\mathcal{BF}_{ij}$ instead of undirected ones. As for the edges $\{a_{ij}, b_{ij}\}$ and $\{c_{ij}, d_{ij}\}$, they can be replaced by arcs (a_{ij}, b_{ij}) and (c_{ij}, d_{ij}) if j is odd and arcs (b_{ij}, a_{ij}) and (d_{ij}, c_{ij}) otherwise. In the clause-graphs G_j, we can have arcs (s_j, s_{1j}) and (t_{nj}, t_n) replace of the edges $\{s_j, s_{1j}\}$ and $\{t_{nj}, t_n\}$. Moreover, the 2-path-gadget can be built with two directed paths $(x_1, true_1, y_1, true'_1, \ldots, x_n, true_n, y_n, true'_n, x_{n+1})$ and $(x_1, false_1, y_1, false'_1, \ldots, x_n, false_n, y_n, false'_n, x_{n+1})$ and the edges $\{x^j_{n+1}, x^{j+1}_1\}$ can be replaced by the arcs (x^j_{n+1}, x^{j+1}_1).

2.6 Conclusion

Table 2.2 summarizes the complexity results on the problems we have investigated in this chapter: finding a path, a trail and disjoint trails avoiding forbidden transitions.

We have proved that the problem of finding a path avoiding forbidden transitions is NP-complete even in well-structured graphs as grids. We have also proved that PAFT can be solved in polynomial time when the treewidth is bounded. In fact, we believe that the PAFT problem is actually $W[1]$-hard when parameterized by the treewidth. This would imply that unless $FPT = W[1]$, PAFT is not FPT when parameterized by the treewidth only, that is there is no algorithm which finds a path avoiding forbidden transitions in $O(f(k)poly(n))$, where k is the treewidth and n is the size of the graph. Future work might focus on proving this conjecture as well as on the study of the parameterized complexity with other parameters such as the number of asymmetric nodes or forbidden transitions. Another interesting direction in the study of PAFT could be to consider the optimization problem where the objective is to find a path with minimum number of forbidden transitions and to investigate possible approximation solutions.

As for the problem of finding disjoint trails, we have proved that when the trails are required to be vertex-disjoint the problem is NP-complete both in directed

	Directed graphs	Undirected graphs
Finding a path (PAFT)	• NP-complete in general [Sze03] • NP-complete in directed grids • Polynomial in DAGs	• NP-complete in general [Sze03] • NP-complete in grids • Polynomial in graphs with bounded treewidth
Finding a trail (TAFT)	Polynomial [Win02, ABP96]	Polynomial
Finding k vertex-disjoint trails (k-VDT)	• NP-complete in directed graphs with maximum degree 8 (also implied by [GLMM12]) • Polynomial in DAGs	• NP-complete in undirected graphs with maximum degree 8 • NP-complete in undirected graphs with degree ≤ 4 [GLMM12]
Finding k edge-disjoint trails (k-EDT)	Polynomial	NP-complete [ADF+08]

Table 2.2: Summary of complexity results for problems of finding paths avoiding forbidden transitions.

and undirected graphs. On the positive side, if the paths are only required to be edge-disjoint, the problem can be solved in polynomial time in directed graphs. The reduction graph in the proof of the NP-completeness of finding 2 vertex-disjoint trails avoiding forbidden transitions is 1-planar, which means that it can be embedded in the plane in such a way that each edge has at most one crossing point, where it crosses a single additional edge. This makes us wonder if the k-VDT problem is NP-complete in planar graphs. Future work might focus on answering this question. Moreover, we have proved that for a fixed k, k-VDT is polynomial in DAGs. Since, the algorithm we use has time complexity $O(n^{poly(k)})$, we would like to see if this complexity can be improved and if k-VDT is FPT in DAGs when parameterized by k the number of disjoint paths. It would be also interesting to study the generalization of the k-EDT problem in which we look for k edge-disjoint paths avoiding forbidden subpaths. Since k-EDT is polynomial in directed graphs, it would be nice to see if the generalized problem is polynomial as well in directed graphs. We would also like to investigate possible approximations for the optimization version of the k-VDT problem. That is the problem of finding k minimum vertex-overlapping trails, i.e. trails sharing the minimum number of vertices.

Chapter 3

On Disjoint Paths in Networks with Star SRLGs

Contents

We study in this chapter, the problem of finding risk disjoint paths in networks where links are subject to risks localized around the nodes. Results presented in this chapter are joint work with J.-C. Bermond, D. Coudert, and G. D'Angelo and they have been published in *Theoretical Computer Science* [BCDM15] and presented in the *Student Workshop ACM CoNEXT'2012* [BCDM12], *Algotel'2013* [BCDM13b], and *DRCN'2013* [BCDM13a].

3.1 Introduction

The dedicated path protection scheme, which we have defined in the beginning of Part I, is used to ensure reliable communications in networks in general and optical networks in particular. It consists in finding a pair of disjoint paths: one working path to ensure the communication and one protection path to reroute the traffic in case of a failure. This method works well in a single link failure scenario, as it consists in finding two edge-disjoint paths between a pair of nodes. This is a well-known problem in graph theory for which there exist efficient polynomial time algorithms [Suu74, ST84]. However, the problem of finding two disjoint paths

between a pair of nodes becomes much more difficult, in terms of computational complexity, in case of multiple correlated link failures that can be captured by the notion of *Shared Risk Link Group* (or *SRLG*, for short). In fact, a SRLG is a set of network links that fail simultaneously when a given event (risk) occurs. The scope of this concept is very broad. It can correspond, for instance, to a set of fiber links of an optical backbone network that are physically buried at the same location and therefore could be cut simultaneously (i.e. backhoe or JCB fade). It can also represent links that are located in the same seismic area, or radio links in access and backhaul networks subject to localized environmental conditions affecting signal transmission, or traffic jam propagation in road networks. Note that a link can be affected by more than one risk. In practice, the failures are often localized and common SRLGs satisfy the *star property* [LW05] (coincident SRLGs in [DG05]). Under this property, all links of a given SRLG share an endpoint. Such failure scenarios can correspond to risks arising in router nodes like card failures or to the cut of a conduit containing links issued from a node (see Figure 3.1).

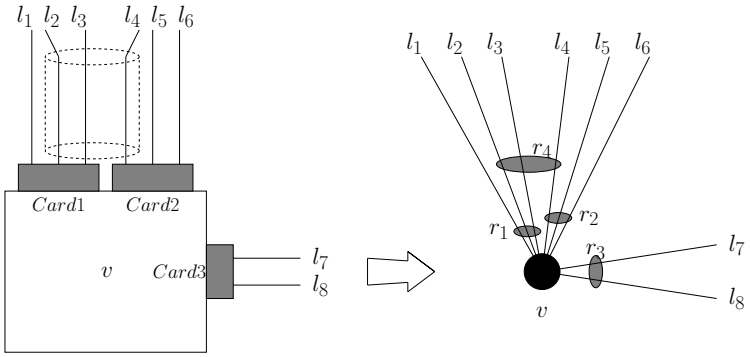


Figure 3.1: Example of localized risks: link l_4 shares risk r_2, corresponding to Card 2 failure, with links l_5 and l_6, and shares risk r_4, corresponding to a conduit cut, with links l_2 and l_3.

The graph theoretic framework for studying optimization problems in networks with SRLGs is the *colored graph* model [DS08, YVJ05, Far06, DG05, LW05, CDP+07]. In this model, the network topology is modeled by a graph $G = (V, E)$ and the set of SRLGs by a set of colors $\mathcal{C}$. Each SRLG is modeled by a distinct color, and that color is assigned to all the edges corresponding to the network links subject to this SRLG. Also, an edge modeling a network link subject to several SRLGs will be assigned as many colors as SRLGs. A colored graph is therefore defined by the triple $(V, E, \mathscr{C})$, where $\mathscr{C}$ is a *coloring* function, $\mathscr{C} : E \to 2^{\mathcal{C}}$, that assigns a subset of colors to each edge. The colored graph model is also known as the *labeled graph* model [FGK10]. Furthermore, some studies assumed that an edge is assigned at most one color [CDP+07, FGK10, JRN04]. Notice that the computational complexity of some optimization problems may be different in the model in which an edge is assigned at most one color than in the model in which it can be assigned multiple colors, and the impact of the transformation from one model

to the other on problems complexities has been investigated in [CPRV14]. In the setting of colored graphs, the star property means that all the edges with a given color share a common vertex.

3.1.1 Related work

In the context of colored graphs, basic graph connectivity problems have been restated in terms of colors and proven much more difficult to address than their basic counterparts. For instance, the minimum color *st*-path problem is to find a path from vertex s to vertex t in the graph that minimizes the cardinality of the union of the colors of the edges along that path. This problem has been proven NP-hard [SYR05, BLWZ05, CDKM00] and hard to approximate [CDP+07, HMS07] in general, $W[2]$-hard when parameterized by the number of used colors and $W[1]$-hard when parameterized by the number of edges of the path [FGK10]. However, it has been proven in [CPRV14] that the minimum color *st*-path problem can be solved in polynomial time in colored graphs with the star property. Other optimization problems on graphs have been studied in the context of colored graphs such as the minimum color cut [Far06, CDP+07], the minimum color *st*-cut [CDP+07], the minimum color maximum matching [FGK10] .

The *k-diverse routing* problem in presence of SRLGs consists in finding a set of k SRLG-disjoint paths between a pair of vertices (i.e. paths having no risk in common). Note that many authors use, in the case $k = 2$, *diverse routing* instead of 2-diverse routing. With no restriction on the graph structure and on the assignment of SRLGs to edges, even finding two SRLG-disjoint paths is NP-complete [Hu03], and therefore many heuristics have been proposed [SYR05, YVJ05, GL07, TR04b, TR04a, ZSXT07]. The problem is polynomial in some specific cases of localized failures: when SRLGs have span 1 (i.e. an edge can be affected by only one SRLG, and the set of edges belonging to the same SRLG forms a connected component, see [CDP+07]), and in a specific case of SRLGs having the star property [DS08] in which a link can be affected by at most two risks and two risks affecting the same link form stars at different nodes (this result also follows from [CDP+07]).

3.1.2 Our results

We study the k-diverse routing problem when SRLGs have the star property and there are no restrictions on the number of risks per link nor on the number of links per risk. This case has been studied in [LW05] in which the authors claim that finding two SRLG-disjoint paths under the star property can be solved in polynomial time. In this chapter, we establish the following results:

- We demonstrate that the algorithm proposed in [LW05] is not correct; indeed we exhibit, in Section 3.3, counterexamples for which their algorithm concludes to the non existence of two SRLG-disjoint paths although two such paths exist.
- We prove, in Section 3.4, that finding k SRLG-disjoint paths is in fact NP-complete even for only two paths.

- On the positive side, we show in Section 3.5, that the k-diverse routing problem can be solved in polynomial time in particular subcases which are relevant in practice. Namely, we solve the problem in polynomial time when the maximum degree is at most 4 or when the input network is a directed acyclic graph. Moreover, we show that the problem is fixed-parameter tractable when parameterized by the number of colors in $\mathcal{C}$.
- Finally, we consider the problem of finding the maximum number of SRLG-disjoint paths. We prove that, under the star property, the problem is hard to approximate within $O(|V|^{1-\varepsilon})$ for any $0 < \varepsilon < 1$, where V is the set of nodes in the network, and we give polynomial time algorithms for some of the above relevant subcases.

We give the notation used in this chapter in Section 3.2.

3.2 Notations and problem statement

We model the network as an undirected connected graph $G = (V, E)$, where the vertices in V represent the nodes and the edges in E represent the links. We associate a color with each SRLG. Let us denote by $\mathcal{C}$ the set of all the colors. Then a network with SRLGs is modeled by a *colored graph* that is a triple $G_c = (V, E, \mathscr{C})$, where (V, E) is a graph and $\mathscr{C}$ is a *coloring* function, $\mathscr{C} : E \to 2^{\mathcal{C}}$, that assigns a subset of colors to each edge of E.

We denote by $E(c)$ the set of edges having color $c \in \mathcal{C}$, by $\mathscr{C}(e)$ the set of colors associated with edge $e \in E$, by $\text{CPE} = \max_{e \in E} |\mathscr{C}(e)|$ the *maximum number of colors per edge*, and by $\text{EPC} = \max_{c \in \mathcal{C}} |E(c)|$ the *maximum number of edges having the same color*. We assume that $\mathscr{C}(e) \neq \emptyset$ for each $e \in E$. Given a vertex v, $\Gamma(v)$ denotes the set of neighbors of v and $d(v) = |\Gamma(v)|$ its *degree*. A color is *incident* to v if it is assigned to an edge incident to v. The *colored degree* of v, denoted by $d_{\mathcal{C}}(v)$, is the number of colors incident to v. The *maximum degree* and the *maximum colored degree* of a graph are denoted by Δ and $\Delta_{\mathcal{C}}$, respectively.

We can now model the star property defined in the introduction as follows. A color $c \in \mathcal{C}$ is called a *star color* if all edges of $E(c)$ are incident to the same vertex. A colored graph has the *star property* if it has only star colors.

Given a colored graph G_c and two vertices s and t, an st-path is an alternating sequence of vertices and edges, beginning with s and ending with t, in which each edge is incident to the vertex immediately preceding it and to the vertex immediately following it. A path is denoted by the sequence of vertices and edges. We say that two paths P_1 and P_2 are *color-disjoint* if $(\cup_{e \in P_1} \mathscr{C}(e)) \cap (\cup_{e \in P_2} \mathscr{C}(e)) = \emptyset$, i.e. the edges of one path do not have any color in common with the edges of the other path.

The k-diverse routing problem defined in the introduction consists then in finding k color-disjoint paths and for every k can be formally formulated as follows:

Problem 6 (k-Diverse Colored st-Paths, k-DCP)**.** *Given a colored graph G_c and two vertices s and t, are there k color-disjoint paths from s to t?*

In this chapter we study the k-DCP problem where the colored graphs have the star property.

3.3 Counterexamples to the algorithm of Luo and Wang

Luo and Wang [LW05] proposed an algorithm to find a pair of color-disjoint paths with minimum total cost from a source s to a destination t in graphs with colors (SRLGs) satisfying the star property. The algorithm is an adaptation of the Bhandari's edge-disjoint shortest-pair of paths algorithm [Bha98, Chapter 3.3, pages 46-68] (which itself is a variation of the Suurballe-Tarjan's algorithm [Suu74, ST84]) and is based on augmenting a shortest path P_a between s and t.

In what follows, we argue that the algorithm is incorrect, as there are at least two problems with it.

Counterexample 1

The first problem comes from the fact that the algorithm implies that the first and last edges of the shortest s-t path P_a should be contained necessarily in the pair of paths returned by the algorithm. However, if no edge incident to s (to t) is color-disjoint with the first (the last) edge of P_a, respectively, the algorithm will ignore the existence of 2 color-disjoint paths even if they exist.

The counterexample in Figure 3.2 illustrates this first problem: color c is shared between edges $\{s, v_0\}$ and $\{s, v_1\}$ and color $c' \neq c$ is shared between edges $\{s, v_0\}$ and $\{s, v_2\}$. All other unmarked edges have distinct colors different from c and c'. The shortest path from s to t is $P_a = (s, v_0, t)$. Applied on the graph of Figure 3.2, the algorithm described in [LW05, page 451, lines 9–10] does not find any edge to start and hence terminates concluding that there are no two color-disjoint paths. However two color-disjoint paths clearly exist, namely they are $P_1 = (s, v_1, w_1, t)$ and $P_2 = (s, v_2, w_2, t)$.

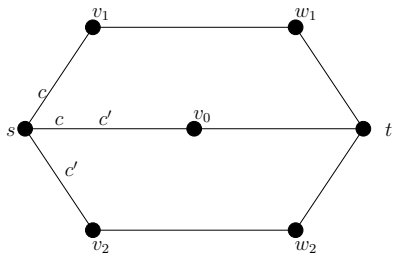


Figure 3.2: Example 1.

Counterexample 2

The second problem is that the algorithm only checks color-disjointness around nodes of P_a and never checks other nodes. It assumes implicitly that the only nodes

that can be shared by the two color-disjoint paths are nodes belonging to P_a and ignores the existence of any other possibilities.

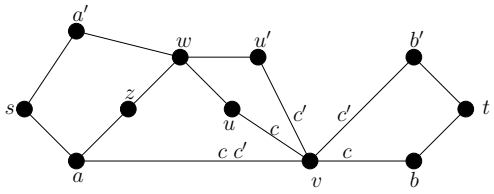


Figure 3.3: Example 2

Figure 3.3, illustrating the second problem, shows a counterexample to the algorithm in [LW05] that furthermore can give to the reader a flavor of the difficulty of the problem. In this figure, we have 2 specific colors c and $c' \neq c$ forming a star in v. All other unmarked edges have distinct colors different from c and c'. As vertex v is a cut-vertex any s-t path should contain v. Moreover $\{a, v\}$ cannot be used as it shares a color both with $\{v, b\}$ and $\{v, b'\}$. Therefore, to ensure the color-disjointness, one path should use the subpath u, v, b and the other one should use the subpath (u', v, b'). We have two color-disjoint paths $P_1 = (s, a, z, w, u, v, b, t)$ and $P_2 = (s, a', w, u', v, b', t)$. However, the algorithm of [LW05] uses the shortest path $P_a = (s, a, v, b, t)$ and then performs a backwards phase which never finds w again. Then the algorithm terminates, missing the fact that there exist two color-disjoint paths. Note that the disjointness is not ensured, if there exists in w some common color for example on the edges $\{w, u\}$ and $\{w, u'\}$, showing that a local consideration around the shortest path is not sufficient. In fact, in the next section we will prove that the problem is NP-complete.

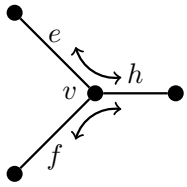
3.4 NP-completeness

In this section we will prove that, even with the star property, the k-DCP problem is NP-complete for every constant $k \geq 2$. We use a reduction from the problem of finding a path avoiding forbidden transitions which we have studied in Chapter 2 and which was proven NP-complete in [Sze03]. This again contradicts the supposed correctness of the polynomial algorithm of [LW05], unless $P = NP$.

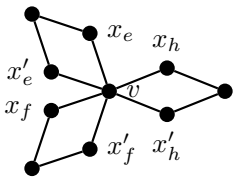
We recall that in the PAFT problem we are given a graph $G = (V, E)$, a set $\mathcal{F}$ of forbidden transitions and two vertices $s, t \in V$ and we need to decide whether an $\mathcal{F}$-valid s-t-path (i.e. a path which uses none of the transitions in $\mathcal{F}$) exists in G or not.

It has been proven in [Sze03] that PAFT is NP-complete and it remains NP-complete for the family $\mathcal{G}_4$ of simple graphs where vertices s and t have degree 3 and any other vertex has degree 3 or 4, and the set of allowed transitions $\mathcal{A}(v)$ around a vertex v is such that:

- If $d(v) = 3$, $\mathcal{A}(v)$ consists of two pairs of edges $\{e, h\}$ and $\{f, h\}$ where e, f and h are the 3 edges incident to v;



(a) v in G, $\mathcal{A}(v) = \{\{e, h\}, \{f, h\}\}$

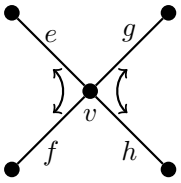


(b) v in H

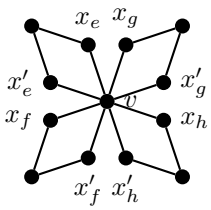
Edge	Colors
$\{v, x_e\}$	C_{ef}, $C_{ef'}$
$\{v, x'_e\}$	$C_{e'f}$, $C_{e'f'}$
$\{v, x_f\}$	C_{ef}, $C_{e'f}$
$\{v, x'_f\}$	$C_{ef'}$, $C_{e'f'}$
$\{v, x_h\}$	C_h,
$\{v, x'_h\}$	C'_h,

(c) Color assignment

Figure 3.4: Color assignment for vertices with degree 3.



(a) v in G, $\mathcal{A}(v) = \{\{e, f\}, \{g, h\}\}$



(b) v in H

Edge	Colors
$\{v, x_e\}$	C_{eg}, $C_{eg'}$, C_{eh}, $C_{eh'}$
$\{v, x'_e\}$	$C_{e'g}$, $C_{e'g'}$, $C_{e'h}$, $C_{e'h'}$
$\{v, x_f\}$	C_{fg}, $C_{fg'}$, C_{fh}, $C_{fh'}$
$\{v, x'_f\}$	$C_{f'g}$, $C_{f'g'}$, $C_{f'h}$, $C_{f'h'}$
$\{v, x_g\}$	C_{eg}, $C_{e'g}$, C_{fg}, $C_{f'g}$
$\{v, x'_g\}$	$C_{eg'}$, $C_{e'g'}$, $C_{fg'}$, $C_{f'g'}$
$\{v, x_h\}$	C_{eh}, $C_{e'h}$, C_{fh}, $C_{f'h}$
$\{v, x'_h\}$	$C_{eh'}$, $C_{e'h'}$, $C_{fh'}$, $C_{f'h'}$

(c) Color assignment

Figure 3.5: Color assignment for vertices with degree 4.

- If $d(v) = 4$, $\mathcal{A}(v)$ consists of two pairs of distinct edges $\{e, f\}$ and $\{g, h\}$ where e, f, g and h are the 4 edges incident to v.

Theorem 8. *The k-DCP problem is NP-complete for any fixed constant $k \geq 2$, even if all the following properties hold:*

- *the star property;*
- *the maximum degree Δ is fixed with $\Delta \geq \max\{8, k\}$;*
- CPE, EPC *and $\Delta_{\mathcal{C}}$ are fixed with either $\lceil$CPE ≥ 4, EPC ≥ 2, and $\Delta_{\mathcal{C}} \geq \max\{16, k\}\rfloor$ or $\lceil$CPE ≥ 2, EPC ≥ 4 and $\Delta_{\mathcal{C}} \geq \max\{4, k\}\rfloor$.*

Proof. We first prove the statement for $k = 2$ and then extend it for any fixed $k \geq 3$.

It is easy to see that the problem is in NP since, given two paths, we just have to check whether they are color-disjoint.

Given an instance $(G, s, t, \mathcal{F})$ of the PAFT problem with G in the family $\mathcal{G}_4$, we define an instance of 2-DCP as follows. We associate with G a colored graph $H = (V_H, E_H, \mathscr{C})$ where:

- For each node in G, we associate a node in H;

- For each edge $e = \{u, v\}$ in G, we associate in H two nodes x_e and x'_e and two paths of length 2: $\{u, x_e, v\}$ and $\{u, x'_e, v\}$. H has then $|V(G)| + 2|E(G)|$ vertices and $4|E(G)|$ edges.
- We assign the colors to edges incident to a vertex v in H as follows:
  - Distinct new colors are assigned to the edges incident to t.
  - For each pair of edges e and f incident to s in G, such that $e \neq f$, we will use 4 colors $C_{ef}, C_{ef'}, C_{e'f}$ and $C_{e'f'}$. We assign colors C_{ef} and $C_{ef'}$ to the edge $\{s, x_e\}$; colors $C_{e'f}$ and $C_{e'f'}$ to the edge $\{s, x'_e\}$, colors C_{ef} and $C_{e'f}$ to the edge $\{s, x_f\}$ and colors $C_{ef'}$ and $C_{e'f'}$ to the edge $\{s, x'_f\}$.
  - For each $v \neq s, t$, and for each pair of edges e and f incident to v in G such that $e \neq f$ and $\{e, f\}$ is a forbidden transition (i.e. $\{e, f\} \in \mathcal{F}$), we assign colors C_{ef} and $C_{ef'}$ ($C_{e'f}$ and $C_{e'f'}$) to the edge $\{v, x_e\}$ ($\{v, x'_e\}$), and colors C_{ef} and $C_{e'f}$ ($C_{ef'}$ and $C_{e'f'}$) to the edge $\{v, x_f\}$ ($\{v, x'_f\}$), respectively. As each vertex has either degree 3 or 4, two cases can occur:
    (i) If $d(v) = 3$, let e, f and h be the 3 edges incident to v and let $\mathcal{A}(v) = \{\{e, h\}, \{f, h\}\}$, then the colors are assigned as described in Figure 3.4.
    (ii) If $d(v) = 4$, let e, f, g and h be the 4 edges incident to v and $\mathcal{A}(v) = \{\{e, f\}, \{g, h\}\}$, then the colors are assigned as described in Figure 3.5.

The transformation is polynomial time computable and the star property holds. Moreover, note that each edge has at most 4 colors, each color is associated with two edges, the degree of each vertex is at most 8 and the color degree is at most 16. It follows that CPE ≤ 4, EPC ≤ 2, $\Delta \leq 8$, and $\Delta_{\mathcal{C}} \leq 16$.

To prove the theorem, we will use the following properties.

Property 1. *Given an edge e incident to s in G, the edge $\{s, x_e\}$ in H shares a color with all the other edges incident to s, except $\{s, x'_e\}$. In other words, the only pair of edges incident to s having no color in common are of the form $\{\{s, x_e\}, \{s, x'_e\}\}$ for some e.*

Property 2. *If $v \neq s, t$, $d(v) = 3$ and $\mathcal{A}(v) = \{\{e, h\}, \{f, h\}\}$, then two edges incident to v share a color if and only if one is $\{v, x_e\}$ or $\{v, x'_e\}$ and the other is $\{v, x_f\}$ or $\{v, x'_f\}$.*

Property 3. *If $v \neq s, t$, $d(v) = 4$ and $\mathcal{A}(v) = \{\{e, f\}, \{g, h\}\}$, then two edges incident to v share a color if and only if one is $\{v, x_e\}$, $\{v, x'_e\}$, $\{v, x_f\}$ or $\{v, x'_f\}$ and the other is $\{v, x_g\}$, $\{v, x'_g\}$, $\{v, x_h\}$ or $\{v, x'_h\}$.*

In other words, two edges incident to a node v are color-disjoint if and only if they correspond to an allowed transition of v or are of the form $\{v, x_e\}$ and $\{v, x'_e\}$.

We first show that if there exists an $\mathcal{F}$-valid path in G, then there exist two color-disjoint paths in H. Let $P = (s, e_1, v_1, \ldots, e_p, v_p, e_{p+1}, t)$ be an $\mathcal{F}$-valid path

from s to t in G. Then $Q = (s \equiv v_0, x_{e_1}, v_1, \ldots, x_{e_p}, v_p, x_{e_{p+1}}, t)$ and $Q' = (s \equiv v_0, x'_{e_1}, v_1, \ldots, x'_{e_p}, v_p, x'_{e_{p+1}}, t)$ are two color-disjoint paths in H. In particular, by Properties 1, 2, and 3, any edge $\{v_i, x_{e_{i+1}}\}$ ($\{x_{e_i}, v_i\}$) has no color in common with $\{v_i, x'_{e_{i+1}}\}$ ($\{x'_{e_i}, v_i\}$), respectively, for each $i = 0, 1, \ldots, p$.

Conversely, we now show that if there exist two color-disjoint paths in H, then there exists an $\mathcal{F}$-valid path in G. Let the two color-disjoint paths in H be $Q = (s, x_1, v_1, \ldots, x_p, v_p, x_{p+1}, t)$ and $Q' = (s, y_1, u_1, \ldots, y_{p'}, u_{p'}, y_{p'+1}, t)$. We prove by induction on $i \in \{1, \ldots, p+1\}$, that $\{x_i, y_i\} = \{x_{e_i}, x'_{e_i}\}$, $v_i = u_i$ and $p = p'$.

For $i = 1$, by Property 1, $\{s, x_1\}$ and $\{s, y_1\}$ have no color in common only if $\{x_1, y_1\} = \{x_e, x'_e\}$ for an edge e incident to s and then $v_1 = u_1$.

Let us suppose that the statement is true until $i = l$; we will prove it for $i = l + 1$. Let the two edges entering $u_l = v_l$ used by Q and Q' be $\{x_{e_l}, v_l\}$ and $\{x'_{e_l}, v_l\}$.

If $d(v_l) = 3$, we distinguish two cases:

- e_l belongs to only one allowed transition of $\mathcal{A}(v_l)$ say $\{e_l, h_l\}$ and the paths Q and Q', being color-disjoint, can only use the edges $\{v_l, x_{h_l}\}$ and $\{v_l, x'_{h_l}\}$.

- e_l belongs to two allowed transitions of $\mathcal{A}(v_l)$, $\{e_l, f_l\}$ and $\{e_l, h_l\}$. If one path uses the edge $\{v_l, x_{h_l}\}$ ($\{v_l, x'_{h_l}\}$) the other path cannot use the edge $\{v_l, x_{f_l}\}$ or $\{v_l, x'_{f_l}\}$ by Property 2, it has then to use edge $\{v_l, x'_{h_l}\}$ ($\{v_l, x_{h_l}\}$), respectively.

Therefore, in both cases $\{x_{l+1}, y_{l+1}\} = \{x_{h_l}, x'_{h_l}\}$ and $v_{l+1} = u_{l+1}$.

If $d(v_l) = 4$, by Property 3, the only possibility as Q and Q' are color-disjoint is that they use the edges $\{v_l, x_{e_{l+1}}\}$ and $\{v_l, x'_{e_{l+1}}\}$, respectively, where $\{e_l, e_{l+1}\} \in \mathcal{A}(v_l)$ and so the statement is true for $i = l + 1$.

It follows that the path $P = (s, e_1, v_1, \ldots, e_p, v_p, e_{p+1}, t)$ satisfies $\{e_i, e_{i+1}\} \in \mathcal{A}(v_i)$ for every $i \in \{1, \ldots, p\}$ and then it is $\mathcal{F}$-valid.

To show that the problem remains NP-complete even for fixed CPE ≥ 2, EPC ≥ 4 and $\Delta_{\mathcal{C}} \geq 4$, it is enough to modify the above transformation by using a different color assignment. In detail, the color assignment differs from the one given above as follows:

- Edges incident to vertex s (which has degree 3) have the color assignment reported in Table 3.1a;

- Edges incident to vertices with degree 4 in G have the color assignment reported in Table 3.1b;

- The other vertices (i.e. t and those with degree 3 in G) keep the same color assignment as before.

The above proof works with this color assignment with slight changes. Indeed in Property 3 edge $\{v, x_e\}$ shares colors with edge $\{v, x_f\}$. But the proof is still valid as when $d(v_l) = 4$, if Q (Q') uses the edge $\{v_l, x_{e_l}\}$ ($\{v_l, x'_{e_l}\}$), then Q (Q') uses the edge $\{v_l, x_{e_{l+1}}\}$ ($\{v_l, x'_{e_{l+1}}\}$), respectively, where $\{e_l, e_{l+1}\} \in \mathcal{A}(v_l)$. It follows that 2-DCP is NP-hard even for fixed CPE ≥ 2, EPC ≥ 4 and $\Delta_{\mathcal{C}} \geq 4$.

Edge	Colors
$\{s, x_e\}$	C_1, C_2
$\{s, x'_e\}$	C_3, C_4
$\{s, x_f\}$	C_1, C_3
$\{s, x'_f\}$	C_2, C_4
$\{s, x_h\}$	C_1, C_4
$\{s, x'_h\}$	C_2, C_3

(a) Color assignment for vertex s.

Edge	Colors
$\{v, x_e\}$	C_1, C_2
$\{v, x'_e\}$	C_3, C_4
$\{v, x_f\}$	C_1, C_2
$\{v, x'_f\}$	C_3, C_4
$\{v, x_g\}$	C_1, C_3
$\{v, x'_g\}$	C_2, C_4
$\{v, x_h\}$	C_1, C_3
$\{v, x'_h\}$	C_2, C_4

(b) Color assignments for vertices with degree 4.

Table 3.1: Color assignments for vertex s (Table 3.1a) and for vertices with degree 4 (Table 3.1b) when CPE ≥ 2 and EPC ≥ 4.

We can extend the proof to the case where $k \geq 3$ in various ways. In a first version we added $k-2$ paths of length 2 from s to t, $P_i = (s, w_i, t)$ for $i = 3, 4, \ldots k$, with a new color assigned to each edge $\{s, w_i\}$. The following construction which gives better results was suggested by one referee; we modify $H = (V_H, E_H, \mathscr{C})$ as follows:

- We introduce two additional vertices s' and t'.
- We add $k-2$ paths of length 2 from s' to t', $P_i = (s', w_i, t')$ for $i = 3, 4, \ldots k$, with a new color assigned to each edge $\{s', w_i\}$ and $\{t', w_i\}$. These paths are pairwise color-disjoint.
- We add two paths of length 2 from s to s', $A_i = (s', a_i, s)$ for $i = 1, 2$, with a new color assigned to each edge $\{s, a_i\}$ and $\{s', a_i\}$.
- We add two paths of length 2 from t to t', $B_i = (t, b_i, t')$ for $i = 1, 2$, with a new color assigned to each edge $\{t, b_i\}$ and $\{t', b_i\}$.

Finding k color-disjoint paths from s' to t' in this new graph is equivalent to finding 2 color-disjoint paths from s to t . Moreover, this assignment does not change CPE and EPC,and Δ and $\Delta_{\mathcal{C}}$ are either the same or equal to k. □

3.5 Polynomial cases

In this section we give polynomial time algorithms for k-DCP for some important special cases. In detail, we solve k-DCP for the cases where the number of colors is bounded by a constant (i.e. $|\mathcal{C}| = O(1)$), for some cases where the maximum degree of the graph is strictly smaller than 5, and for the cases where the input graph is a Directed Acyclic Graph (DAG). All the given algorithms work only when the star property holds, but the one for the case when $|\mathcal{C}| = O(1)$ which works for any possible color assignment.

3.5.1 Bounded number of colors

In this section, we give an algorithm to find k color-disjoint paths in the special case where the number $|\mathcal{C}|$ of colors in the network is bounded by a constant, i.e. $|\mathcal{C}| = O(1)$. We observe that such an algorithm works for every graph topology and even if the star property does not hold.

We will reduce our problem to the Set Packing problem.

Problem 7 (Set Packing)**.** *Given a set X, a collection $\mathcal{S}$ of subsets of X and an integer k, is there a collection of disjoint sets $\mathcal{S}' \subseteq \mathcal{S}$ such that $|\mathcal{S}'| = k$?*

The Set Packing problem is known to be NP-hard [GJ79, Problem SP3, page 221] but is polynomial-time solvable when the size of X is bounded [BHK09].

A subset $A \subseteq \mathcal{C}$ of colors will be called *realizable*, if the subgraph G_A induced by the edges whose colors are *all* in A (i.e. edges e such that $\mathscr{C}(e) \subseteq A$) contains at least one path from s to t. Note that such a path uses only colors of A.

The idea of the algorithm is to enumerate all the realizable subsets of $\mathcal{C}$ and then find k disjoint realizable subsets by using an exact algorithm for the Set Packing. As the size of $\mathcal{C}$ is constant, the computational time required by such algorithm is polynomial.

The details of the algorithm along with its correctness and complexity are given in the next theorem.

Theorem 9. *The k-DCP problem is FPT when parameterized by the number of colors $|\mathcal{C}|$. In particular, there exists an algorithm for solving the k-DCP problem in time $O(f(|\mathcal{C}|)(|V| + |E|))$, where f is a function depending solely on $|\mathcal{C}|$.*

Proof. Let $X = \mathcal{C}$ and let $\mathcal{S}$ be the family of realizable subsets of colors. Then there exists a collection of k disjoint sets $\mathcal{S}' \subseteq \mathcal{S}$ if and only if there exist k color-disjoint paths from s to t. Indeed, to each subset A' of $\mathcal{S}'$ is associated a path using uniquely colors of A' (as A' is realizable) and two disjoint subsets correspond to two color-disjoint paths.

Determining if a subset of colors is realizable requires $O(|V|+|E|)$ computational time. Furthermore, it is known that there exist polynomial time algorithms to solve the Set Packing problem when the size of X is bounded. For instance, the exact algorithm proposed in [BHK09] has time complexity $O(|\mathcal{S}|2^{|X|}|X|^{O(1)})$. As

$|X| = |\mathcal{C}|$ and $|\mathcal{S}| \leq 2^{|\mathcal{C}|}$, we deduce that finding k color-disjoint paths requires $O(2^{2|\mathcal{C}|}|\mathcal{C}|^{O(1)}(|V|+|E|))$ overall time, and so that the k-DCP problem is FPT when parameterized by the number of colors $|\mathcal{C}|$. □

3.5.2 Bounded degree

In this section, we assume that the star property holds and that $|\mathcal{C}|$ is unbounded and we give algorithms for finding k color-disjoint paths when $\Delta < 4$ and for finding 2 color-disjoint paths when $\Delta = 4$. First, note that the maximum number of color-disjoint paths in a graph is upper bounded by Δ.

If $\Delta \leq 2$ the problem is trivial as the graph is either a path or a cycle. In the first case, there always exists only one path from s to t. In the second case, the only vertices where the two possible paths can share colors are s and t and hence it is enough to check if the two edges incident to s (and t) are color-disjoint.

If $\Delta \leq 3$, observe that if two paths share an internal vertex of degree 3, they necessarily share also an edge and hence all the colors of that edge. Consequently, they cannot be color-disjoint. Furthermore, if two paths are color-disjoint the colors of their first edges should be disjoint and also the colors of their last edges should be disjoint.

If $\Delta = 3$ and $k = 3$, there are 3 color-disjoint paths if and only if G has 3 vertex-disjoint paths between s and t and the 3 first edges of these paths have disjoint colors and also the 3 last edges. That can be checked in $O(|V|+|E|)$ time: constant time for checking the color disjointness of the 3 first (last) edges, and $O(|V|+|E|)$ time for checking the existence of 3 vertex-disjoint paths between s and t (see [FF57]).

If $k = 2$ and $\Delta = 3$ or 4, we give an algorithm in the proof of the following theorem:

Theorem 10. *Algorithm 1 solves 2-DCP in graphs with the star property and $\Delta = 3$ or 4 in time $O(|V|+|E|)$.*

Proof. We say that a pair $\{s_i, s_j\}$ of neighbors of s in G is admissible, if the edges joining s to them have disjoint colors, i.e. $\mathscr{C}(\{s, s_i\}) \cap \mathscr{C}(\{s, s_j\}) = \emptyset$ and similarly we say that $\{t_{i'}, t_{j'}\}$ is an admissible pair of neighbors of t if the edges joining them to t have disjoint colors. Then, with each admissible pair $\{s_i, s_j\}$ and each admissible pair $\{t_{i'}, t_{j'}\}$, we associate the admissible graph $G(s_i, s_j, t_{i'}, t_{j'})$ obtained from G by deleting the edges $\{s, s_\ell\}$ with $\ell \neq i, j$ and the edges $\{t_{\ell'}, t\}$ with $\ell' \neq i', j'$.

We solve 2-DCP when $\Delta = 3, 4$ by using Algorithm 1 whose correctness is given in what follows.

Note first that since all colors satisfy the star property, they are localized around vertices and the color-disjointness of two paths can be ensured by the color-disjointness around their shared vertices.

By definition, if there exist two vertex-disjoint paths from s to t in $G(s_i, s_j, t_{i'}, t_{j'})$ the first edges and last edges of such paths have disjoint colors and so we conclude that there are 2 color-disjoint paths (lines 2, 3).

Algorithm 1: Solving 2-DCP when $\Delta = 3, 4$.

foreach *admissible graph* $G(s_i, s_j, t_{i'}, t_{j'})$ **do**
  if *there exist 2 vertex-disjoint paths from* s *to* t *in* $G(s_i, s_j, t_{i'}, t_{j'})$ **then**
    There exist two color-disjoint paths from s to t in G;
  else
    if *all the cut-vertices that separate* s *from* t *in* $G(s_i, s_j, t_{i'}, t_{j'})$ *have degree 4 and are not incident to bridges* **then**
      foreach *cut vertex* v *that separates* s *from* t *in* $G(s_i, s_j, t_{i'}, t_{j'})$ **do**
        Let e and f be the edges incident to v in the connected component containing s, and let e' and f' be the incident edges in the connected component containing t;
        if ***not***

$$\left(\begin{array}{ll} & \mathscr{C}(e) \cap \mathscr{C}(f) = \emptyset \text{ and } \mathscr{C}(e') \cap \mathscr{C}(f') = \emptyset \\ \text{and} & \left[\begin{array}{ll} & [\mathscr{C}(e) \cap \mathscr{C}(e') = \emptyset \text{ and } \mathscr{C}(f) \cap \mathscr{C}(f') = \emptyset] \\ \text{or} & [\mathscr{C}(e) \cap \mathscr{C}(f') = \emptyset \text{ and } \mathscr{C}(f) \cap \mathscr{C}(e') = \emptyset] \end{array} \right] \end{array} \right)$$

        then
          No 2 color-disjoint paths from s to t exist in G;
      There exist two color-disjoint paths from s to t in G;
  No 2 color-disjoint paths from s to t exist in G.

Otherwise, if there exists a cut vertex (i.e., a vertex which removal disconnects s from t and hence should be included in any path from s to t) v of degree 3, we cannot have color-disjoint paths containing this vertex. That is in particular the case when $\Delta = 3$. If there exists a cut-vertex v of degree 4 that is incident to a bridge, then v belongs to at most one 2-connected component which is not a bridge. In this case, there are no two color-disjoint paths from s to t. So, let us now assume that $\Delta = 4$, all the cut vertices are of degree 4 and, each cut vertex is incident to two 2-connected components which are not bridges. Between every two cut vertices, the two paths use vertex-disjoint subpaths. If at the cut vertex v one path uses edges e and e' (e and f'), the other path uses necessarily f and f' (f and e'), respectively, and the conditions on colors are necessary and sufficient for the color disjointness of the paths at v (the center of the colors used in v).

Since we have at most 6 admissible pairs of neighbors of s (of t), respectively, we have at most 36 graphs to consider. For each graph we have to check if it is 2-connected (that can be done in time $O(|V| + |E|)$ [HT73]) and if it is not 2-connected to satisfy coloring conditions at each cut vertex (all of the cut-vertices can be determined in linear time using the algorithm for finding the biconnected components of a graph in [HT73]), which can be done in constant time for a given vertex and so overall in time $O(|V|)$. □

Note that Algorithm 1 cannot be extended in a straightforward manner neither to find 3 or 4 color-disjoint paths on a graphs with $\Delta = 4$, nor to the case of $\Delta = 5, 6, 7$ in polynomial time. In fact, for these cases, while the number of admissible graphs stays bounded and finding vertex-disjoint path segments stays polynomial, the number of possible ways to cross the cuts explodes exponentially.

3.5.3 Directed acyclic graphs

In this section, we propose an algorithm for finding k color-disjoint paths in a colored directed acyclic graph (DAG) with the star property. The definitions given for undirected colored graphs can be easily extended to colored DAGs by assigning an acyclic orientation to the edges of the graph. As each color is a star color we can associate with each color c its center v defined as the common vertex to all arcs with color c. If the color has only one occurrence we choose arbitrarily as associated center one of the end vertices of the arc containing this color. We will say that the color c is centered in v.

The algorithm given in the proof of the next theorem is similar to the one proposed in Section 2.5.3.2 to find k vertex-disjoint paths avoiding forbidden transitions in DAGs. We again use ideas from [Cha03] and especially the notion of layered directed graph that we have defined in Section 2.5.3.2. Although the transformations are a bit similar, we include them in detail for the sake of completeness.

In Theorem 11, we present an algorithm for solving the k-DCP problem in a DAG with the star property in time $O(\text{CPE}^2|V||E|^{2k})$. This algorithm is therefore polynomial only when k is a fixed constant.

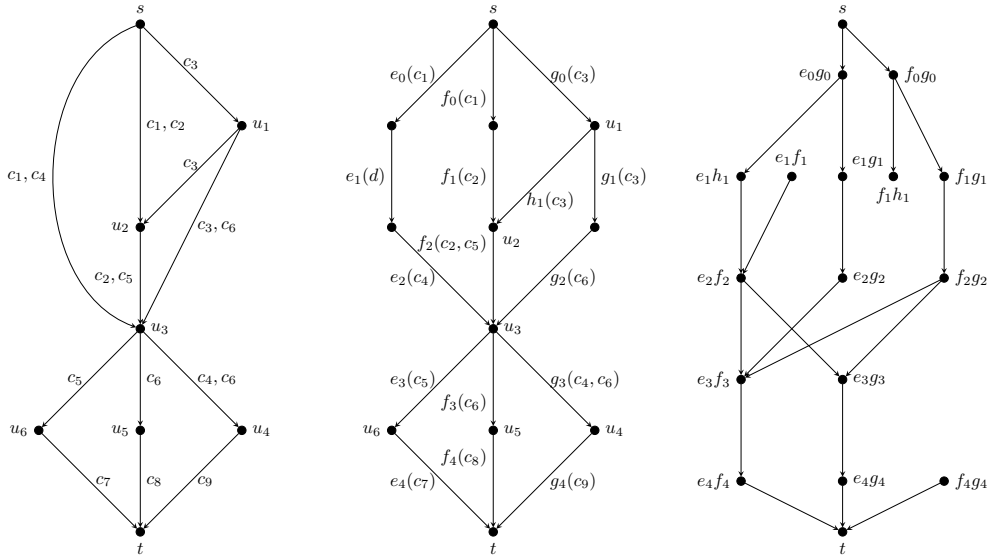
We will then show in Section 3.6 that the problem is $W[1]$-hard and therefore it is not possible to find an FPT algorithm (i.e. having time complexity $O(f(k) \cdot poly(|V| + |E|))$, for any function f), unless $FPT = W[1]$.

Theorem 11. *There exists an algorithm that solves k-DCP in a DAG with the star property in time $O(\text{CPE}^2|V||E|^{2k})$.*

Proof. Let D be a multicolored DAG and let s and t be two given vertices. As we want to find (in polynomial time) directed paths from s to t, we can delete the vertices not on a directed path from s to t, and so we suppose in what follows that D is this reduced DAG. Now s is the unique vertex with no in-neighbor and t the unique vertex with no out-neighbor. The algorithm that finds k color-disjoint paths from s to t in D uses two transformations:

Transformation 1 We first associate with a multicolored DAG D a multicolored layered DAG LD as follows. We denote by $\Gamma^-(v)$ the set of vertices preceding v, i.e; vertices u such that $(u, v) \in E$. We compute the function $l : V \to \mathbb{N}$ defined as follows:

$$l(v) = \begin{cases} 0 & \text{when } v = s, \\ 1 + \max_{u \in \Gamma^-(v)} l(u) & \text{otherwise.} \end{cases}$$



(a) Example of muticolored DAG D

(b) LD obtained after the layering of D

(c) The graph H associated with LD

Figure 3.6: Transformations for the DAG.

In such a level function, t has the maximum value as there is a directed path from any vertex to t in the reduced DAG. In the example of Figure 3.6a, we have $l(u_1) = 1$, $l(u_2) = 2$, $l(u_3) = 3$, $l(u_4) = l(u_5) = l(u_6) = 4$, and $l(t) = 5$.

Now we replace every arc (u, v), such that $l(v) > l(u) + 1$, with a directed path P_{uv} from u to v of length $l(v) - l(u)$ (thus possibly adding new vertices and arcs). We assign to the first arc of the directed path P_{uv} the colors of the arc (u, v) centered in u (or a new color if there are no colors centered in u) and to the last arc of the directed path P_{uv} the colors of the arc (u, v) centered in v (or a new color if there are no colors centered in v) and to the intermediate arcs, if any, new distinct colors. The resulting layered DAG LD is such that there exist k color-disjoint paths in the DAG D from s to t if and only if there exist k color-disjoint paths in LD from s to t. In Figure 3.6b, we indicate the layered DAG LD obtained from the DAG D of Figure 3.6a. We have given a name to each arc with a lower index indicating the level of the arc; we also indicate inside parentheses the colors attributed to each arc. For example the arc (s, u_3) which had colors c_1, c_4 has been replaced by a path with 3 arcs: e_0 at level 0 which gets the color c_1 (centered at s), e_1 at level 1 which gets a new color d and e_2 at level 2 which gets the color c_4 (centered at u_3).

Therefore, in what follows we consider a layered DAG LD with two specific vertices s and t.

Transformation 2 We use in this transformation ideas similar to that used in [Cha03] to solve the problem of finding a pair of vertex-disjoint paths with forbidden pairs of edges. Here instead of vertex-disjointness we seek edge-disjointness

and the forbidden pairs of edges are forbidden pairs of subpaths sharing a color.

With LD we will associate a directed graph H with two specific vertices s and t, such that there exist k color-disjoint directed paths in LD from s to t if and only if there exists a directed path from s to t in H. For the ease of presentation, we first give the transformation for $k = 2$.

There is a vertex in H for every pair $\{e_i, f_i\}$ of arcs in LD at the same layer i, with $0 \leq i \leq l(t) - 1$, such that $\mathscr{C}(e_i) \cap \mathscr{C}(f_i) = \emptyset$. We also add to H two vertices s and t. Now we join, by an arc in H, s to all the vertices (pairs) $\{e_0, f_0\}$. Similarly we join every vertex $\{e_{l(t)-1}, f_{l(t)-1}\}$ in H by an arc to t. Finally, for $0 \leq i \leq l(t)-2$, we join in H each vertex $\{e_i, f_i\}$ to a vertex $\{e_{i+1}, f_{i+1}\}$ if in LD we have the following properties:

1. the terminal vertex u_i (v_i) of e_i (f_i) is the initial vertex of e_{i+1} (f_{i+1}), respectively, and
2. either $u_i \neq v_i$
3. or $u_i = v_i$ and the set of colors of $\mathscr{C}(e_i) \cup \mathscr{C}(e_{i+1})$ is disjoint from the set of colors of $\mathscr{C}(f_i) \cup \mathscr{C}(f_{i+1})$
4. or $u_i = v_i$ and the set of colors of $\mathscr{C}(e_i) \cup \mathscr{C}(f_{i+1})$ is disjoint from the set of colors of $\mathscr{C}(f_i) \cup \mathscr{C}(e_{i+1})$.

Figure 3.6c indicates the graph H obtained from the layered DAG LD of Figure 3.6b. For example, we have three vertices corresponding to pairs of arcs of layer 2 of LD: e_2f_2, e_2g_2 and f_2g_2 but only two vertices corresponding to pairs of arcs of layer 3: e_3f_3 and e_3g_3. Vertex e_2g_2 is connected to vertex e_3g_3 as condition 3 is fulfilled but it is not connected to f_3g_3 as none of the conditions 3 and 4 is fulfilled.

The existence of two disjoint colored directed paths in LD named $P = (s, e_0, u_0, e_1, \ldots, e_{l(t)-2}, u_{l(t)-1}, e_{l(t)-1}, t)$ and $Q = (s, f_0, v_0, f_1, \ldots, f_{l(t)-2}, v_{l(t)-1}, f_{l(t)-1}, t)$ implies the existence of a directed path from s to t namely $PQ = (s, \{e_0, f_0\}, \{e_1, f_1\}, \ldots, \{e_{l(t)-1}, f_{l(t)-1}\}, t)$ in H.

Conversely, let W be a path in H written in the form $W = (s, w_0, w_1, \ldots, w_{l(t)-1}, t)$ where w_i corresponds to the pair $\{e_i, f_i\}$ and w_{i+1} to the pair $\{e_{i+1}, f_{i+1}\}$ such that the set of colors of $\mathscr{C}(e_i) \cup \mathscr{C}(e_{i+1})$ is disjoint from the set of colors of $\mathscr{C}(f_i) \cup \mathscr{C}(f_{i+1})$; such ordering is possible since one of the color conditions is fulfilled. Then, the two directed paths $P = (s, e_0, u_0, e_1, \ldots, e_{l(t)-2}, u_{l(t)-1}, e_{l(t)-1}, t)$ and $Q = (s, f_0, v_0, f_1, \ldots, f_{l(t)-2}, v_{l(t)-1}, f_{l(t)-1}, t)$ are color-disjoint. In the example of Figure 3.6c, H has many directed paths from s to t. For example with the directed path $P = (s, \{e_0, g_0\}, \{e_1, h_1\}, \{e_2, f_2\}, \{e_3, g_3\}, \{e_4, g_4\}, t)$, the two color-disjoint directed paths $P_1 = (s, e_0, e_1, e_2, g_3, g_4, t)$ and $P_2 = (s, g_0, h_1, f_2, e_3, e_4, t)$ in LD and the two color-disjoint directed paths (s, u_3, u_4, t) and $(s, u_1, u_2, u_3, u_6, t)$ in D are associated.

The algorithm can be generalized to find k color-disjoint paths from s to t in a DAG D, for any $k \geq 2$. We first transform D to a layered graph LD as before.

Then, in the second transformation, instead of having a vertex for every pair of arcs of the same layer, we create a vertex for every k-tuple of arcs $\{e_i^1, e_i^2, \ldots, e_i^k\}$ at the same layer i, such that the $\mathscr{C}(e_i^j)$, for $j = 1, \ldots, k$, are disjoint. Then an arc is added from node $\{e_i^1, e_i^2, \ldots, e_i^k\}$ to node $\{e_{i+1}^1, e_{i+1}^2, \ldots, e_{i+1}^k\}$ if there exists an ordering of the e_i^j and of the e_{i+1}^j such that the terminal vertex of e_i^j is the initial vertex of e_{i+1}^j and the sets $\mathscr{C}(e_i^j) \cup \mathscr{C}(e_{i+1}^j)$ are pairwise disjoint.

To decide if a k-tuple $\{e_i^1, e_i^2, \ldots, e_i^k\}$ is a vertex of H, we need to check the color-disjointness of $\mathscr{C}(e_i^j)$, for $j = 1, \ldots, k$. This can be done in at most $\frac{k(k-1)}{2}\text{CPE}^2$ steps. Deciding on the existence of an edge between two vertices of H can be done in $k!k(k-1)\text{CPE}^2$ ($O(\text{CPE}^2)$) time; indeed we can choose an ordering of $\{e_i^1, e_i^2, \ldots, e_i^k\}$ and for each of the $k!$ possible orderings of $\{e_{i+1}^1, e_{i+1}^2, \ldots, e_{i+1}^k\}$, we check whether the sets $\mathscr{C}(e_i^j) \cup \mathscr{C}(e_{i+1}^j)$ are pairwise disjoint in at most $k(k-1)\text{CPE}^2$ steps. Finally, as each arc in D is replaced in LD by a path containing at most one arc of each layer, the number of arcs at a given layer in LD is at most $|E|$. So, the graph H has at most $l(t)|E|^k$ vertices and $l(t)|E|^{2k}$ edges. Therefore, we get the complexity of the theorem as $l(t) \leq |V|$.

□

Remark 1. *The algorithm presented in the proof of Theorem 11 can be adapted to find a minimum cost pair of color-disjoint paths in an arc-weighted DAG by applying the modifications presented in what follows.*

Let us consider a weight function on the arcs of a DAG D. We assign the original weight of the arc (u, v) to the first arc of the path replacing it in LD. Then, in H we assign to the edge joining s to $\{e_0, f_0\}$ the sum of the weights of e_0 and f_0, and to the edge joining $\{e_i, f_i\}$ to $\{e_{i+1}, f_{i+1}\}$ the sum of the weights of $\{e_{i+1}$ and $f_{i+1}\}$. With these modifications, the shortest path in H corresponds to the optimal pair of color-disjoint paths in D.

Remark 2. *We can also use the algorithm presented in the proof of Theorem 11 to find a pair of color-disjoint paths with the minimum total number of colors by applying the modifications presented in what follows.*

Let $\mathscr{C}^-((u, v))$ ($\mathscr{C}^+((u, v))$) be the set of colors of arc (u, v) centered at u (v), respectively. In H, we assign to the arc from $\{e_i, f_i\}$ to $\{e_{i+1}, f_{i+1}\}$ a weight equal to $|\mathscr{C}^+(e_i) \cup \mathscr{C}^-(e_{i+1}) \cup \mathscr{C}^+(f_i) \cup \mathscr{C}^-(f_{i+1})|$, to the arc from s to $\{e_0, f_0\}$ a weight equal to $|\mathscr{C}^-(e_0) \cup \mathscr{C}^-(f_0)|$ and to the arc from $\{e_{l(t)-1}, f_{l(t)-1}\}$ to t a weight equal to $|\mathscr{C}^+(e_{l(t)-1}) \cup \mathscr{C}^+(f_{l(t)-1})|$. We have proven above that every directed path P from s to t in H corresponds to two color-disjoint directed paths P_1 and P_2 from s to t in the layered graph LD (and equivalently to two color-disjoint paths from s to t in D) and with the way we have defined the weights in H, the weight of P is equal to the number of colors used by P_1 and P_2. The shortest path in the weighted graph H will then correspond to the pair of color-disjoint paths with the minimum number of colors.

3.6 Maximum number of color-disjoint paths

In this section we reformulate the problem of finding SRLG-disjoint paths as an optimization problem where we aim at finding the maximum number of color-disjoint paths:

Problem 8 (Max Diverse Colored *st*-Paths, MDCP)**.** *Given a colored graph G_c and two vertices s and t, find the maximum number of color-disjoint st-paths.*

In the next theorem we give complexity results for MDCP by using an approximation factor preserving reduction from Maximum Set Packing (MSP).

Definition 3 (Maximum Set Packing, MSP)**.** *Given a set X and a collection $\mathcal{S}$ of subsets of X, find the maximum cardinality set packing, i.e., a collection of disjoint sets $\mathcal{S}' \subseteq \mathcal{S}$ such that $|\mathcal{S}'|$ is maximized.*

It has been proven that problem MSP is equivalent to the problem of finding a maximum clique in a graph under a PTAS reduction where the number n of vertices in the graph corresponds to $|\mathcal{S}|$ [ADP80]. In detail, approximation algorithms and inapproximability results (in terms of the number of vertices in the graph) carry over to the MSP problem. It is NP-hard to approximate the problem of finding a maximum clique within $O(n^{1-\varepsilon})$, for any $0 < \varepsilon < 1$ [Hås99] and then, unless $P = NP$, MSP is not approximable within $O(|\mathcal{S}|^{1-\varepsilon})$, for any $0 < \varepsilon < 1$. Moreover, if the cardinality of all sets in $\mathcal{S}$ is upper bounded by a constant $c \geq 3$, then the problem is APX-complete [Kan91]. The next theorem gives inapproximability results for the MDCP problem. The proof uses an approximation-preserving reduction from MSP to MDCP where the vertices of V correspond to the elements of $\mathcal{S}$.

Theorem 12. *Unless $P = NP$,* MDCP *cannot be approximated within $O(|V|^{1-\varepsilon})$, for any $0 < \varepsilon < 1$, even if* EPC *is fixed,* EPC ≥ 2*. Moreover, it is APX-hard if* CPE *is fixed,* CPE ≥ 3*. These inapproximability results hold even in DAGs with the star property.*

Proof. Given an instance $I_{\texttt{MSP}}$ of MSP over a set X and a collection $\mathcal{S}$, we define an instance $I_{\texttt{MDCP}}$ of MDCP on a graph G_C as follows.

- for each element S_i of $\mathcal{S}$, we associate a vertex v_{S_i};
- we add two vertices s and t and the edges $\{s, v_{S_i}\}$ and $\{v_{S_i}, t\}$, for each $S_i \in \mathcal{S}$;

First Coloring

- for each $S_i, S_j \in \mathcal{S}$, such that $i \neq j$ and $S_i \cap S_j \neq \emptyset$, we add a new color c_{ij} and associate it with $\{s, v_{S_i}\}$ and $\{s, v_{S_j}\}$;
- for each edge not yet colored (in particular for each edge $\{v_{S_i}, t\}$) we put a new color.

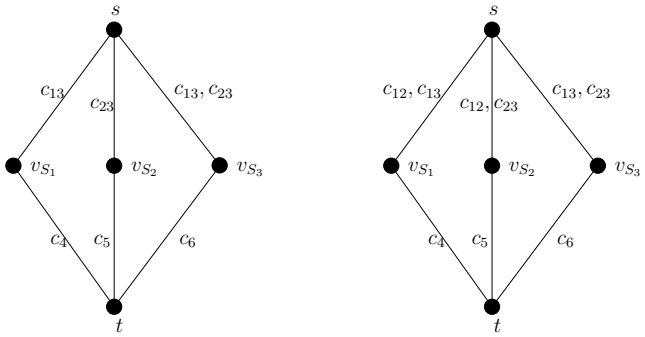


Figure 3.7: Examples where $\mathcal{S} = \{S_1 = \{a, b, c\}, S_2 = \{d, e, f\}, S_3 = \{a, b, d\}\}$ (left) and $\mathcal{S} = \{S_1 = \{a, b, d\}, S_2 = \{b, c, d\}, S_3 = \{b, e, f\}\}$ (right).

See Figure 3.7 for two examples. In the left figure as $S_1 \cap S_3 \neq \emptyset$ and $S_2 \cap S_3 \neq \emptyset$ we have a color c_{13} on $\{s, v_{S_1}\}$ and $\{s, v_{S_3}\}$ and a color c_{23} on $\{s, v_{S_2}\}$ and $\{s, v_{S_3}\}$. On the right figure we have furthermore a color c_{12} on $\{s, v_{S_1}\}$ and $\{s, v_{S_2}\}$ as $S_1 \cap S_2 \neq \emptyset$. By definition each color is associated with at most two edges and hence EPC ≤ 2.

Second Coloring

We define a color c_x for each element $x \in X$ and, for each $S_i = \{x_1, \ldots, x_h\} \in C$, we associate the $|S_i|$ colors $c_{x_1}, \ldots, c_{x_h}$ with the edge $\{s, v_{S_i}\}$. For each edge not yet colored (in particular for each edge $\{v_{S_i}, t\}$ we put a new color. In this way, if the cardinality of all sets in $\mathcal{S}$ is upper bounded by a constant $c \geq 3$, then CPE $\leq c$.

In both cases the transformation is polynomial-time computable and the star property holds. Furthermore, the graph we obtain is the bipartite complete graph $K_{2,|\mathcal{S}|}$. This graph can be oriented into a DAG in a straightforward way.

Now we can associate to a family $\mathcal{S}' = \{S_1, S_2, \ldots, S_q\}$ of sets in $\mathcal{S}$, the set $\mathcal{P}$ of paths $P_j = (s, v_{S_j}, t)$, $j = 1, 2, \ldots, |\mathcal{S}'|$ of G_C and vice versa. Note that by construction two sets S_i and S_j are disjoint if and only if the corresponding paths P_i and P_j are color-disjoint.

Consider an optimal solution $\mathcal{S}'_{OPT}$ for MSP, with $\mathcal{S}'_{OPT} = \{S_1, S_2, \ldots, S_{|\mathcal{S}'_{OPT}|}\}$; the associated set $\mathcal{P}$ of paths $P_j = (s, v_{S_j}, t)$, for each $j = 1, 2, \ldots, |\mathcal{S}'_{OPT}|$ is a feasible solution for $I_{\texttt{MDCP}}$ with $|\mathcal{P}| = |\mathcal{S}'_{OPT}|$, and so,

$$OPT(I_{\texttt{MSP}}) \leq OPT(I_{\texttt{MDCP}}). \tag{3.1}$$

Now suppose that there exists an α-approximation algorithm A for MDCP, the output of this algorithm for the instance $I_{\texttt{MDCP}}$ is a set $\mathcal{P}$ of disjoint paths $P_j = (s, v_{S_j}, t)$, $j = 1, 2, \ldots, |\mathcal{P}|$, whose cardinality satisfies $|\mathcal{P}| = val_A(I_{\texttt{MDCP}}) \geq \frac{1}{\alpha} OPT(I_{\texttt{MDCP}})$. Consider the algorithm A' applied to $I_{\texttt{MSP}}$ which gives as output the family $\mathcal{S}' = \{S_1, S_2, \ldots, S_{|\mathcal{P}|}\}$ associated with $\mathcal{P}$. The family $\mathcal{S}'$ is a feasible solution for $I_{\texttt{MSP}}$, whose value is $val_{A'}(I_{\texttt{MSP}}) = |\mathcal{P}| \geq \frac{1}{\alpha} OPT(I_{\texttt{MDCP}})$ and by inequality (3.1)

$val_{A'}(I_{\texttt{MSP}}) \geq \frac{1}{\alpha} OPT(I_{\texttt{MSP}})$ and so A' is an α-approximation algorithm for MSP.

Finally the first statement of the theorem follows from the $O(|S|^{1-\varepsilon})$ inapproximability of MSP, for any $0 < \varepsilon < 1$, and from the fact that $|V| = |\mathcal{S}| + 2$. Note that in the first coloring each color is associated with at most two edges which implies that EPC ≤ 2. The second statement follows from the fact that MSP is APX-hard if the cardinality of all sets in $\mathcal{S}$ is upper bounded by a constant $c \geq 3$ and that using the second coloring CPE $\leq c$. The results for DAGs come from the fact that the graph $K_{2,|\mathcal{S}|}$ obtained in the transformation can be oriented into a DAG in a straightforward way. □

Parameterized complexity

The next results are expressed in terms of *parameterized complexity* [DF13]. Recall that a problem with input size n is *fixed parameter tractable* with respect to some parameter k (and so is *in FPT*) if it can be solved in time $O(f(k) \cdot n^{O(1)})$ where the function f depends only on k. A problem is $W[1]$-hard if a $W[1]$-complete problem (e.g., deciding if the graph contains a clique of size k) can be reduced to it in FPT-time.

Theorem 13. MDCP *is $W[1]$-hard when parameterized by the number k of color-disjoint paths, even in DAGs and when the star property holds.*

Proof. It is enough to observe that the reduction used in Theorem 12 is a parameterized-preserving reduction where the parameter is the number of color-disjoint paths which corresponds to the number of disjoint subsets in a set packing. In [DF95] (see also [Nie06, Chapter 11.4.2, pages 193–195]) it has been shown that MSP is $W[1]$-hard if the parameter is the number of disjoint subsets. □

The above theorem implies that, unless $FPT = W[1]$, MDCP is not in FPT when parameterized by the number of color-disjoint paths, that is there is no algorithm which finds k color-disjoint paths in $O(f(k) \cdot poly(|V| + |E|))$ time in DAGs, unless $FPT = W[1]$. Moreover, Theorem 8 shows that even finding a fixed number $k \geq 2$ of color-disjoint paths is NP-complete in general undirected graphs. This implies that MDCP is ParaNP-hard in undirected graphs, that is, it is impossible to devise an algorithm which finds k color-disjoint paths in $O((|V| + |E|)^{f(k)})$ time, unless $P = NP$.

The algorithm in Section 3.5.1 can be used to find an exact polynomial-time algorithm for MDCP when $|\mathcal{C}| = O(1)$. In fact, it is enough to search for the maximum k for which such algorithm returns k color-disjoint paths. As the maximum number of color-disjoint paths is upper bounded by Δ, we can use a binary search approach to solve the problem, applying at most $\log \Delta$ times the algorithm of Section 3.5.1. The next corollary follows.

Corollary 1. *The* MDCP *problem is FPT when parameterized by the number of colors $|\mathcal{C}|$. Moreover, there exists an algorithm for solving the* MDCP *problem in time*

$O(f(|\mathcal{C}|)(|V|+|E|)\log\Delta)$, *where* f *is a function depending solely on* $|\mathcal{C}|$, *and* Δ *is the maximum degree of the graph.*

We can use the algorithms for bounded degree presented in Section 3.5.2 for solving MDCP when $\Delta \leq 4$. We get a polynomial-time exact algorithm for $\Delta \leq 3$ and a 2-approximation algorithm for $\Delta = 4$ as the maximum number of color-disjoint paths is upper bounded by $\Delta = 4$. Both algorithms require $O(|V|+|E|)$ time.

	Δ	EPC	CPE	*k*-DCP	MDCP
Undirected graphs	unbounded	1	unbounded	Solvable in $O(\|V\|+\|E\|)$	Solvable in $O(\Delta\|E\|)$
		unbounded	1	Solvable in $O(\|V\|+\|E\|)$ [CDP+07]	Solvable in $O(\Delta_{\mathcal{C}}\|E\|)$ [CDP+07]
	≥ 8	≥ 2	≥ 4	NP-hard for $\Delta \geq \max\{8,k\}$	Not approximable within $O(\|V\|^{1-\varepsilon})$, for any $0<\varepsilon<1$
		≥ 4	≥ 2		
	≤ 3	unbounded	unbounded	Solvable in $O(\|V\|+\|E\|)$	Optimum in $O(\|V\|+\|E\|)$
	$= 4$	unbounded	unbounded	Solvable in $O(\|V\|+\|E\|)$ for $k=2$	2-approximation in $O(\|V\|+\|E\|)$
$\|\mathcal{C}\| = O(1)$, even without star	unbounded	unbounded	unbounded	Solvable in $O(f(\|\mathcal{C}\|)(\|V\|+\|E\|))$, in FPT when parameterized by $\|\mathcal{C}\|$	Optimum in $O(f(\|\mathcal{C}\|)(\|V\|+\|E\|)\log\Delta)$, in FPT when parameterized by $\|\mathcal{C}\|$
DAG	unbounded	≥ 3	≥ 3	Solvable in $O(\text{CPE}^2\|V\|\|E\|^{2k})$	NP-hard
		≥ 2	≥ 6		
		≥ 2	unbounded		Not approximable within $O(\|V\|^{1-\varepsilon})$, for any $0<\varepsilon<1$
		unbounded	$= 3$		*APX*-hard
		unbounded	unbounded		$W[1]$-hard when parameterized by the number of paths

Table 3.2: Summary of complexity results for k-diverse routing.

3.7 Conclusion

Our results, presented in this chapter and summarized in Table 3.2, give an almost complete characterization of the problem of finding SRLG-disjoint paths in networks with SRLGs satisfying the star property. For the case EPC $= 1$, the problem of finding k color-disjoint paths is equivalent to finding k edge-disjoint paths, thus a flow algorithm such as the Ford-Fulkerson's [FF57] can be used to solve k-DCP in time $O(|V|+|E|)$ and MDCP in time $O(\Delta|E|)$. As for the case CPE $= 1$, every edge has one color and since the star property is satisfied, all colors have span 1 (i.e. an edge has only one color and the set of edges having the same color forms a connected component). The problem of finding color disjoint-paths in graphs with span 1 has been proven polynomial in [CDP+07]. To conclude, we point out some open questions for further research:

- The complexity of the problem is still open for the cases where the maximum degree of the network is equal to 5, 6 or 7 and for the cases where EPC $\in \{2,3\}$ and CPE $\in \{2,3\}$. Solving these cases will give a complete complexity

characterization of the problem with respect to the maximum degree of the network and the parameters EPC and CPE.

- In the problem definition, we assumed that $\mathscr{C}(e) \neq \emptyset$, for each $e \in E$, which means that each edge of the network must have at least one color associated with. If we allow edges with no colors, then color-disjoint paths are not necessarily edge-disjoint, since an edge that is not affected by any SRLG can be shared by any number of paths. Therefore, an interesting direction would be to constrain the paths to be edge-disjoint as well; so, the problem would be to find k SRLG-disjoint edge-disjoint paths in the case where the number of SRLGs is bounded by a constant and the number of edges with no SRLG is linear in the size of the network. This does not affect the hardness results that hold for the (restricted) case where each edge has a color and hence also for the (more general) case where colorless edges are allowed. However, the polynomial-time algorithms proposed in Section 3.5 do not work in this case.

- It would be interesting to consider the unsolved cases for directed graphs since we only have results for the specific case of DAGs so far.

Part II

Routing and Spectrum Assignment

We have tackled in the first part of this thesis problems of finding paths and disjoint paths. These problems are related to the dedicated path protection scheme used to make networks more fault-tolerant. In this part, we focus on problems of resource allocation in optical networks with improved spectrum efficiency.

The internet traffic has been growing exponentially due to the emergence of high-rate applications, such as high-definition videos and cloud-based services. To face this rapid growth, a new more efficient generation of optical networks is being developed; the Elastic Optical Networks (EONs), also known as Flex-grid or SLICE optical networks [GJLY12, JTK+09]. EONs are emerging as a potential candidate to replace the currently deployed Wavelength Division Multiplexing (WDM) networks.

In WDM networks [Muk97, MG02], which have been in use for more than a decade, the optical spectrum is subdivided into channels of fixed width (e.g. 50GHz) called wavelengths. This results in what is referred to as the ITU fixed-grid. Each wavelength of this fixed-grid can provision a connection with a bitrate up to 100Gbps. When a connection requires less than 100Gbps, it is still assigned a full wavelength which leads to an inefficient use of the spectrum. On the other hand, when a connection requires more than 100Gbps, it cannot be supported by one wavelength. High bitrate connections can still be provisioned on a fixed grid network by demultiplexing the demand to smaller ones, this however results in the use of more spectrum than if the connection was carried as a single entity.

In EONs [GJLY12, WLV13, JTK+09], new technologies such as optical Orthogonal Frequency Division Multiplexing (OFDM), modulation techniques (such as the Quadrature phase shift keying (QPSK) and the 16-ary quadrature-amplitude (16-QAM)), bandwidth variable transponders (BV-Ts), and flexible spectrum selective switches are used to ensure an efficient utilization of the optical resources and to enable a flexible grid as opposed to the WDM fixed-grid. In this flexible grid, the available optical spectrum is divided into a set of slots of a fixed and finer spectral width, e.g. 25GHz, 12.5GHz or even 6.25GHz. With these fine granularity slots, connections of small bitrates are not over-provisioned and big bitrates can be satisfied as single entities by using multiple contiguous slots. In other words, instead of forcing all connections to use the same spectrum width, EONs allow connections to use a number of slots according to their needs.

Figure 3.8, taken from [GJLY12], shows an existing ITU grid (top) vs. a flexible grid (bottom) and spectral widths of different bitrates (side). In the fixed grid, we can see that for small bitrates a channel is not fully utilized and that bitrates of 400Gbps and 1Tbps cannot be supported as such since they overlap with at least one 50 GHz grid boundary. On the other side, for the flexible grid, these big bitrates can be provisioned and the spectrum is used more efficiently for small bitrates.

EONs offer then an efficient and flexible use of the spectrum as presented in more details in [GJLY12, WLV13, JTK+09]. However, the advantages they propose make the problems of resource allocation more difficult than their counterparts in WDM networks. The key resource allocation problem in WDM networks is the Routing and Wavelength Assignment (RWA) [RS95, ZJ00]. In this problem, we are given an optical network and a set of requests where each request has a source

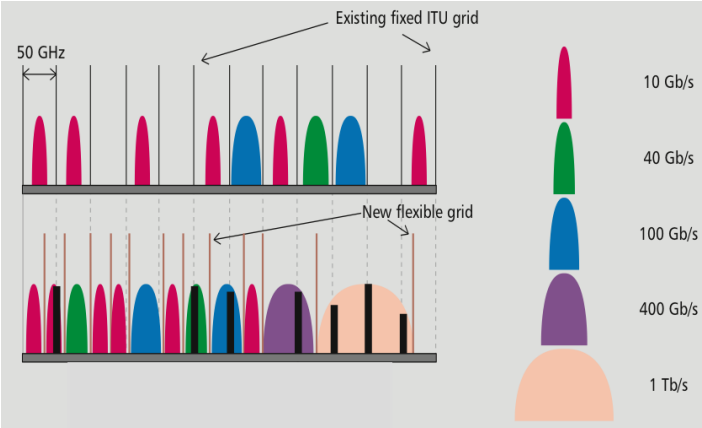


Figure 3.8: Fixed-grid vs Flexgrid [GJLY12]

and a destination and the objective is to find for each request a path from the source to the destination and a wavelength over this path. Each request must use the same wavelength over all the links of its path (continuity constraint) and on a given link, a wavelength can be used by at most one request (non-overlapping constraint). In EONs, since we allow heterogeneous demands and we accord to each request contiguous slots, RWA is replaced by the problem of Routing and Spectrum Assignment (RSA). In RSA, we are still given an optical network and a set of requests and each request has, besides its source and destination, a demand in terms of the number of slots it needs. The objective in RSA is to allocate to each request a path in the optical network and an interval of spectrum slots along that path. The spectrum allocated to a demand has to be contiguous (contiguity constraint), it has to be the same over all links of the routing path (continuity constraint), and a spectrum slot on a link can be used by at most one request (non-overlapping constraint). RWA can be seen as the special case of RSA where all requests have unit demands.

With respect to the pattern of the considered traffic, we can distinguish two versions of RSA: the static RSA and the dynamic RSA.

- *Static Routing and Spectrum Assignment*, also referred to as offline RSA. In this case, the traffic is static and the entire set of requests is known in advance. Static RSA allocates paths and spectrum to these known requests with the objective of minimizing the utilized spectrum or alternatively, maximizing the number of provisioned requests for a fixed spectrum interval. Since static RWA is NP-complete [CGK92], static RSA is NP-complete as well and to make it more tractable, it is usually partitioned into two subproblems that are solved separately: the routing subproblem where the aim is to find paths for the requests and the Spectrum Assignment (SA) subproblem. In SA, we suppose that each request has a fixed path and the objective is to assign spectrum to the requests under the continuity, contiguity and non-overlapping constraints.

- *Dynamic Routing and Spectrum Assignment*, also referred to as online RSA. In this case, the traffic is dynamic, i.e. requests arrive and leave the network dynamically. The objective of dynamic RSA is to allocate paths and spectrum to the arriving requests so as to minimize the spectrum blocking probability. This is not an easy problem to solve, since without a prior knowledge of the requests that will arrive or leave, it is difficult to make routing and spectrum assignment decisions that are guaranteed to cause less blocking. Furthermore, the dynamicity of the traffic usually causes spectrum fragmentation which is the accumulation of small unusable fragments of spectrum. This makes it necessary to use some defragmentation techniques together with the dynamic RSA algorithms.

This part of the thesis is devoted to the problem of Routing and Spectrum Assignment. In Chapter 4, we study static RSA in tree networks. Since the routing is fixed in trees, the problem reduces to Spectrum Assignment. In Chapter 5, we investigate the problem of dynamic RSA with the use of a non-disruptive defragmentation technique called Push-Pull.

Chapter 4

On Spectrum Assignment in Tree-Networks

Contents

In this chapter, we study the problem of static routing and spectrum allocation in elastic optical tree-networks. In trees, even though the routing is fixed, the Spectrum Allocation (SA) is NP-hard. We look at the complexity and approximability results that have been established for the SA in paths and prove new results for stars and binary trees. The results in this chapter are joint work with J.-C. Bermond and they have been presented in *Algotel'2015* [Moa15].

4.1 Introduction

In the problem of Spectrum Assignment (SA) in elastic optical networks, we are given a network and a set of requests where each request has a demand and a predefined path. The objective is to assign to each request an interval of spectrum of width equal to its demand such that no two requests share the same spectrum slot on a given link. This problem is a generalization of the well studied problem of Wavelength Assignment (WA) in fixed-grid networks. In fact, WA can be seen as the special case of SA where all requests have unit demands. Since WA has been proved

NP-complete in [CGK92], SA is also NP-complete. In fact, SA remains NP-hard even in networks where WA is tractable, particularly in path networks. Indeed, SA has been proved to be equivalent to other problems studied in the literature which we describe in details in Section 4.2. Using the results obtained for the equivalent problems, SA is NP-complete in paths even if the requests' demands do not exceed 2 slots [BEJ+06]. Furthermore, SA is NP-complete in paths with 4 links and unidirectional rings with 3 links [TBL+14]. On the positive side, SA can be approximated within a factor of $2+\varepsilon$ in paths, a factor of $4+\varepsilon$ in rings (for all $\varepsilon > 0$), and a factor of $O(log(k))$ in binary trees where k is the number of requests [SZDS13].

Contribution In this chapter, we study the spectrum assignment problem in trees. We focus on special cases where the tree is a star or where the demands of the requests are bounded by a constant and the tree is binary. By studying these special cases, we hope to gain more insight into the general problem in trees and design a constant-factor approximation algorithm or prove that such algorithm does not exist. We establish the following results.

- We prove that SA is NP-hard in undirected stars of 3 links and in directed stars of 4 links, and show that it can be approximated within a factor of 4 in stars in general.
- Afterwards, we use the equivalence of SA with a graph coloring problem (interval coloring) to find constant-factor approximation algorithms for SA on binary trees with special demand profiles. Namely, we examine the cases where the demands are in a set $\{k, kX\}$ $(k, X \in \mathbb{N}^*)$, in a set $\{kX, k(X+1)\}$ $(k, X \in \mathbb{N}^*)$, or bounded by D. For the latter case, we give a general approximation when the demands are bounded by $D \in \mathbb{N}$ and then give better approximations for the cases where the demands are bounded by $D \in \{3, 4, 5, 6\}$.

This chapter is organized as follows. In Section 4.2, we formally define the problem of spectrum assignment and survey its relation to other problems and its complexity in path networks in particular. Afterwards, we present our results in stars and binary trees in Sections 4.3 and 4.4, respectively.

4.2 Problem statement and related problems

In this section, we first define the problem of Spectrum Allocation (SA) and then present some related problems and highlight their equivalence to SA. In the last subsection, we list implications of these equivalences on the complexity of SA in paths.

4.2.1 Spectrum Assignment

Let $\mathcal{N} = (N, L)$ be a graph modeling an optical network such that N is the set of nodes and L is the set of links. Let $\mathcal{R}$ be a set of requests where each $r \in \mathcal{R}$

has a path $P(r)$ in $\mathcal{N}$ and a spectrum demand $d(r) \in \mathbb{N}$ (number of slots). We say that two requests $r, r' \in \mathcal{R}$ are *conflicting* if their paths $P(r)$ and $P(r')$ share a link. A spectrum assignment of $(\mathcal{N}, \mathcal{R})$ is a mapping f from $\mathcal{R}$ to $\mathbb{N}^*$ such that for every pair of conflicting requests $r, r' \in \mathcal{R}$, we have $\{f(r), \ldots, f(r) + d(r) - 1\} \cap \{f(r'), \ldots, f(r') + d(r') - 1\} = \emptyset$. We say that all the slots in $\{f(r), \ldots, f(r) + d(r) - 1\}$ are *occupied* by r. In fact, the set of slots $\{f(r), \ldots, f(r) + d(r) - 1\}$ corresponds to the spectrum interval $]f(r) - 1, f(r) + d(r) - 1]$ and we sometimes use them interchangeably.

The *span* of a spectrum assignment f, denoted $s(f)$, is the smallest integer s such that for each request $r \in \mathcal{R}$, $f(r) + d(r) - 1 \leq s$. The span of an instance $(\mathcal{N}, \mathcal{R})$, denoted by $s(\mathcal{N}, \mathcal{R})$ is the minimum of spans over all possible spectrum assignments. We formulate the spectrum assignment problem as follows:

Problem 9 (Spectrum Assignment (SA))**.** *Given an instance* $(\mathcal{N}, \mathcal{R})$*, compute* $s(\mathcal{N}, \mathcal{R})$.

For an instance $(\mathcal{N}, \mathcal{R})$ of SA, the *load of a link* ℓ, denoted by $\pi(\ell)$, is the sum of the demands of the requests using ℓ and the *load of an instance*, denoted by $\Pi(\mathcal{N}, \mathcal{R})$, is the maximum load over all its links. It is straightforward that $\Pi(\mathcal{N}, \mathcal{R}) \leq s(\mathcal{N}, \mathcal{R})$. In the approximations we obtain for SA in this chapter, the span is usually upper bounded by a function of the maximum load.

The *greedy algorithm* for SA is an algorithm which assigns spectrum to requests ordered in a given order $r_1, \ldots, r_n$, such that a request r_i is assigned the smallest positive integer $g(r_i)$ such that $\{g(r_i), \ldots, g(r_i) + d(i) - 1\} \cap \{g(r_j), \ldots, g(r_j) + d_j - 1\} = \emptyset$ for each r_j in $\{r_1, \ldots, r_{i-1}\}$ conflicting with r_i. We will use this algorithm many times in the remainder of this chapter.

4.2.2 Related problems

We list in this section problems which are related to SA.

4.2.2.1 Scheduling Tasks on Multiprocessor Systems

It has been proved in [TBL+14] that SA in a network of k links can be reduced to the problem of Scheduling Tasks on Multiprocessor Systems (STMS) with k processors. In the STMS we are given a set of n tasks and a set of identical processors, a processing time d_j and a prespecified set P_j of processors for each task j, $j \in \{1, \ldots, n\}$. The objective is to schedule the tasks so as to minimize the makespan $C_{max} = \max_j C_j$, where C_j denotes the completion time of task j, under the following constraints: (1) preemptions (interruptions of a task) are not allowed, (2) each task must be processed simultaneously by all processors in P_j, and (3) each processor can work on at most one task at a time.

Given an instance $(\mathcal{N}, \mathcal{R})$ of SA, an instance of STMS is constructed as follows. For each link ℓ of $\mathcal{N}$, we associate a processor w_ℓ, and for each request r in $\mathcal{R}$ with path $P(r)$ and demand $d(r)$, we associate a task t_r with processing time $d(r)$ and a set of processors $\{w_\ell \mid \ell \in P(r)\}$.

Complexity of STMS. It has been shown in [HvdVV94] that the problem of STMS is strongly NP-complete even if the number of used processors is at most 3. On the positive side, it has been proved in [Goe95] and [HCC00] that STMS can be approximated within $\frac{7}{6}$ and 1.5 when the number of processors is 3 and 4, respectively. Theorem 14 follows from these approximations.

Theorem 14. *There are approximation algorithms with ratios $\frac{7}{6}$ and 1.5 for SA in networks with 3 and 4 links, respectively.*

4.2.2.2 Dynamic Storage Allocation

When the network is a path, the SA problem is equivalent to the problem of Dynamic Storage Allocation (DSA). In the DSA problem, we are given a set A of items to be stored, each $a \in A$ having size $d(a)$, an arrival time $\alpha(a)$, and a departure time $\beta(a)$ (with $\beta(a) > \alpha(a)$). A storage allocation for A is a function $f : A \to \mathbb{N}^*$ which associates to each item $a \in A$ a storage interval $I(a) = \{f(a), \ldots, f(a) + d(a) - 1\}$ such that for all $a, a' \in A$ with $a \neq a'$, if $I(a) \cap I(a')$ is non empty then $]\alpha(a), \beta(a)] \cap]\alpha(a'), \beta(a')]$ is empty. The storage size used by a storage allocation f denoted by $s(f)$ is the smallest integer s such that for each item $a \in A$, $f(a) + d(a) - 1 \leq s$. The objective in DSA is to find a storage allocation which minimizes the used storage size.

If we consider the time interval as a path network and each of the items to be stored as requests we can see the equivalence between the problem of SA in paths and the DSA problem. In more details, given an instance of SA on a path $(v_1, \ldots, v_k)$, we associate to each request r with demand $d(r)$, an item a_r of size $d(r)$. We also associate to each vertex v_i of the path network time i. Let v_i and v_j be the endvertices of the path $P(r)$ of the request r $(i < j)$, then we choose for the associated item a_r the arrival time $\alpha(a_r) = i$ and the departure time $\beta(a_r) = j$. The fact that two requests r and r' are conflicting corresponds to the fact that the time intervals $]\alpha(a_r), \beta(a_r)]$ and $]\alpha(a_{r'}), \beta(a_{r'})]$ intersect. Then a spectrum assignment with value γ corresponds to a storage allocation using a storage size γ. Conversely, using the opposite transformation, we can associate to an instance of DSA an instance of SA on a path.

Complexity of DSA. The problem of DSA has been extensively studied. It has been proved that DSA is strongly NP-complete, even when restricted to instances where the storage size of all items is in $\{1, 2\}$. The proof of NP-completeness is by reduction from the 3-PARTITION problem and can be found in the appendix of [BEJ+06]. On the positive side, many approximation algorithms have been proposed to solve DSA. The first proposed algorithms are based on a greedy algorithm called First Fit (FF) and its performance for online coloring of interval graphs. The relation between online coloring of interval graphs and dynamic storage allocation can be found in [CS88]. Using FF, a linear approximation of DSA has been proved in [Kie88] and a 6-approximation was given in [Kie91]. Gergov has adopted another approach not using FF, yielding an approximation of 5 and 3 sequentially in [Ger96]

and [Ger99]. In his approach, Gergov defines and uses a 2-allocation which is a storage allocation where two items but not three are allowed to overlap. A better approximation has been achieved in [BKK+04] where the authors use the idea of boxing items to design a $2+\varepsilon$-approximation algorithm. Better approximations were achieved for DSA with restricted item sizes. In [LLQ04], the authors present a $\frac{4}{3}$-approximation algorithm when the maximum size is 2, and a $1,7$-approximation algorithm when the maximum size is 3. In [MB99], it is proved that for instances with sizes of 1 and X, an approximation of ratio $2-\frac{1}{X}$ can be guaranteed. All these results established for DSA apply, by equivalence, to SA in paths.

4.2.2.3 Interval Coloring

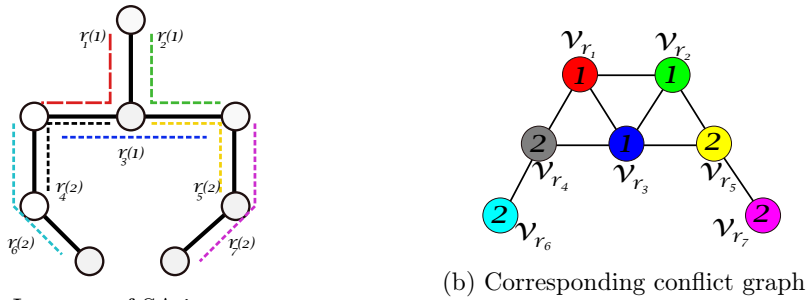
As pointed out in [SZDS13], the problem of SA is also equivalent to a graph coloring problem called Interval Coloring (IC). An interval coloring or a contiguous coloring [Gol04] of a vertex-weighted graph $(G=(V,E),w)$ is a mapping $f: V \to \mathbb{N}^*$ such that for every $v,v' \in V$, if $(v,v') \in E$ then $\{f(v),\ldots,f(v)+w(v)-1\} \cap \{f(v'),\ldots,f(v')+w(v')-1\} = \emptyset$. The number of colors used by an interval coloring f, denoted by $\chi_f(G,w)$ is the smallest integer α such that for each vertex $v \in V$, $f(v)+w(v)-1 \leq \alpha$. The interval chromatic number of a weighted graph (G,w), denoted by $\chi(G,w)$, is the least number of colors needed to color the vertices with intervals, i.e. it is the minimum $\chi_f(G,w)$ among all possible interval colorings f of (G,w). The interval coloring problem is defined as follows.

Problem 10 (Interval Coloring (IC))**.** *Given a vertex-weighted graph (G,w), compute $\chi(G,w)$.*

To see the equivalence between SA and IC, we do the following. For an instance $(\mathcal{N},\mathcal{R})$ of SA, we create a weighted graph $(G=(V,E),w)$ modeling the dependency between the different requests and call it *the conflict graph*. We associate to every request $r \in \mathcal{R}$ a vertex v_r in V. We add an edge between two vertices v_r and $v_{r'}$ if and only if the corresponding requests r and r' are conflicting. The weight $w(v_r)$ of each vertex v_r is equal to the bandwidth demand of the corresponding request r (i.e. $w(v_r)=d(r)$). Figure 4.1 illustrates an instance of SA on a tree ($r_i(j)$ is request r_i with demand j) and the corresponding conflict graph (weights of the vertices are inside the vertices).

If $(\mathcal{N},\mathcal{R})$ is an instance of SA and (G,w) is its conflict graph, then finding a spectrum assignment of $(\mathcal{N},\mathcal{R})$ is equivalent to finding an interval coloring of (G,w) and $s(\mathcal{N},\mathcal{R})=\chi(G,w)$.

Complexity of IC. The problem of IC has been introduced in [Gol04] where its relation to other problems such as DSA has been highlighted. It has also been proved in [Gol04] that IC is equivalent to the problem of finding, for a vertex-weighted graph, an acyclic orientation which minimizes the length of the longest path, where the length of a path is the sum of the weights of its vertices. The complexity of DSA implies that IC is strongly NP-complete in interval graphs. IC is



(a) Instance of SA in a tree

(b) Corresponding conflict graph

Figure 4.1: Example of the construction of a conflict graph

also strongly NP-complete in proper interval graphs [Sha15]. On the positive side, IC is polynomial in comparability graphs [Gol04] and can be approximated within a factor of $2+\varepsilon$ in interval graphs [BKK+04], a factor of 2 for proper interval graphs [Sha15] and claw-free chordal graphs [CDG02], and a factor of $O(log(n))$ in chordal graphs where n is the number of vertices.

Since the conflict graph associated to an instance of SA in a path is an interval graph, all the results established for IC in interval graphs apply to SA in paths. In section 4.4, we will use the fact that the conflict graph associated to a binary tree is a chordal graph to obtain results for SA in binary trees using interval coloring of chordal graphs.

4.2.3 Spectrum Assignment in Paths

The problem of spectrum assignment in paths has been studied in [TBL+14] and [SZDS13]. The complexity results established in these two papers as well as the ones deduced from the equivalence to the problems defined above can be summarized as follows.

- With respect to the number of links, SA is NP-complete in path networks with 4 links and polynomial in paths with at most 3 links [TBL+14], and it can be approximated within a factor of 1.5 in paths with 4 or 5 links [TBL+14].

- With respect to the demands, SA is strongly NP-complete even if the requests have demands in the set $\{1,2\}$ [BEJ+06]. It can be approximated within a factor of $\frac{4}{3}$ and a factor of $1,7$ when the maximum demand is 2 and 3, respectively [LLQ04]. It also can be approximated within a factor of $2-\frac{1}{X}$ when the demands are in the set $\{1,X\}$ [MB99].

- In general, SA in paths can be approximated in paths within a factor of $2+\varepsilon$ [BKK+04] and it can be approximated within a factor of 2 when the paths of the requests are such that no path is strictly included in another [Sha15].

4.3 Spectrum Assignment in Stars

A star is a tree-network with at most one node of degree at least 2. The problem of wavelength assignment (WA) is NP-complete in undirected stars but polynomial in directed stars [Bea00]. The polynomiality of WA in directed stars was useful because optical networks are usually symmetrically directed and because it helped in the design of constant-factor approximation algorithms for WA in directed trees [Bea00]. We prove in this section that SA is not only NP-complete in undirected stars but also in directed stars with 4 links. On the positive side, we prove the existence of a 4-approximation algorithm for the general case.

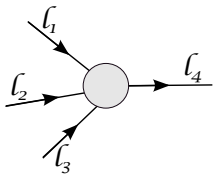
Theorem 15. *The problem of Spectrum Assignment is strongly NP-complete in undirected stars with 3 links.*

Proof. It was shown in [TBL+14] that the SA problem is strongly NP-complete in a 3-link unidirectional ring. Let us consider an instance of SA in a 3-link ring $C = (l_1, l_2, l_3)$ with a request set $\mathcal{R}$. Let us build a star S with three links l'_1, l'_2 and l'_3, and a set of requests $\mathcal{R}'$ defined as follows. For each request $r \in \mathcal{R}$ using at most 2 links, we create a request r' in $\mathcal{R}'$ such that if the path of r in C is $P(r) = l_i$, $i \in \{1, 2, 3\}$, then the path of r' in S is $P(r') = l'_i$, and if $P(r) = l_i l_j$, then $P(r') = l'_i l'_j$. Solving SA in $(C, \mathcal{R})$ is equivalent to solving SA in $(S, \mathcal{R}')$. □

Theorem 16. *The problem of Spectrum Assignment is weakly NP-complete in directed stars with 3 ingoing links and one outgoing link or 3 outgoing links and one ingoing link.*

Proof. The proof is by reduction from the 2-PARTITION problem. In the 2-PARTITION problem, we are given a set A of k integers $a_1, a_2, \ldots, a_k$ such that $B = \sum_{j=1}^{k} a_j$ and the objective is to decide whether A can be partitioned into two disjoint sets A_1 and A_2 such that $\sum_{a_j \in A_1} a_j = \sum_{a_j \in A_2} a_j$.

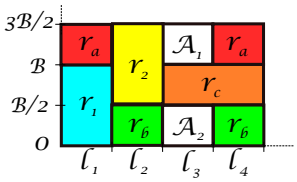
Given an instance of the 2-PARTITION problem with a set of k integers $A = \{a_1, a_2, \ldots, a_k\}$ such that $B = \sum_{j=1}^{k} a_j$, we create an instance of spectrum assignment in a 4-link directed star network S (Figure 4.2a) and a set of requests $\mathcal{R}$. The star S has 3 ingoing links l_1,l_2, and l_3 and one outgoing link l_4. The set of requests $\mathcal{R}$ consists of the requests presented in Figure 4.2b: requests r_a, r_b, r_c, r_1, and r_2 and for every integer a_i in the set A, a request r_3^i with demand a_i and using link l_3. We prove that finding a spectrum assignment for $(S, \mathcal{R})$ with span $\frac{3}{2}B$ is equivalent to finding a partition of A into two sets A_1 and A_2 such that $\sum_{a_j \in A_1} a_j = \sum_{a_j \in A_2} a_j = \frac{B}{2}$. In fact, if there is a partition of A into A_1 and A_2 such that $\sum_{a_j \in A_1} a_j = \sum_{a_j \in A_2} a_j = \frac{B}{2}$, then we can assign spectrum as shown in Figure 4.2c. Now let us suppose there is a spectrum assignment for $(S, \mathcal{R})$ with span $\frac{3}{2}B$. There are two possible symmetric assignments to the requests on links l_1 and l_2. We suppose we assign to r_1, r_a, r_2 and r_b spectrum intervals $]0, B]$, $]B, \frac{3}{2}B]$, $]\frac{B}{2}, \frac{3}{2}B]$, and $]0, \frac{B}{2}]$, respectively (the analysis is similar for the other assignment). This assignment forces request r_c to use the interval $]\frac{B}{2}, B]$ and the other requests on link l_3 will have to be partitioned into two sets of the same size $\frac{B}{2}$.



(a) Star S

request	demand	path
r_a	$B/2$	$l_1 l_4$
r_b	$B/2$	$l_2 l_4$
r_c	$B/2$	$l_3 l_4$
r_1	B	l_1
r_2	B	l_2
r_3^i, $1 \leq i \leq k$	a_i	l_3

(b) Requests $\mathcal{R}$



(c) Spectrum assignment

Figure 4.2: Reduction from 2-PARTITION to SA in a directed star

The proof is similar if we consider a reduction graph which is a star with 3 outgoing links and 1 ingoing link. □

Theorem 17. *The problem of Spectrum Assignment in directed stars with at most 3 links or exactly 2 ingoing links and 2 outgoing links can be solved in polynomial time.*

Proof. In all of these cases, the span is equal to the maximum load and the greedy algorithm with specific orders can achieve the optimal span.

- When the star has only ingoing or outgoing links, the problem is trivial since any conflicting requests use the same link and the greedy algorithm with any order can achieve the optimal span.

- For the case where the star is a directed path of length 2, an optimal spectrum assignment consists in using the greedy algorithm with an order where the requests using two links come first. This way, the spectrum span will be defined by the link with the maximum load.

- For the case where the star has two ingoing links l_1 and l_2 and one outgoing link l_3 (or the opposite), let A_{i3} be the sum of the demands of the requests using l_i and l_3 for $i \in \{1, 2\}$ and let A_i be the sum of the demands of the requests using only link l_i for $i \in \{1, 2, 3\}$. First, the requests using l_1 and l_3 are assigned with the greedy algorithm; the span of the spectrum used on links l_1 and l_3 is equal to A_{13}. Afterwards all requests using only one link are assigned with the greedy algorithm; the span of the spectrum used on links l_1, l_2, and l_3 is $A_{13} + A_1$, A_2, and $A_{13} + A_3$, respectively. Finally the requests using links l_2 and l_3 are assigned; the span of the spectrum of links l_1, l_2, and l_3 is $A_{13} + A_1$, $\max(A_2, A_{13} + A_3) + A_{23}$, and $\max(A_2, A_{13} + A_3) + A_{23}$, respectively. The span of this spectrum assignment is equal to $\max(A_{13} + A_1, \max(A_2, A_{13} + A_3) + A_{23})$ which is exactly the maximum load of the instance.

- When the star has 2 ingoing links l_1 and l_2 and 2 outgoing links l_3 and l_4, let A_{ij} be the sum of the demands of the requests using l_i and l_j for $i \in \{1, 2\}$ and

$j \in \{3, 4\}$ and let A_i be the sum of the demands of the requests using only link l_i for $i \in \{1, 2, 3, 4\}$. First, the requests using l_1 and l_3 and the requests using l_2 and l_4 are assigned with the greedy algorithm; the span of the spectrum used on links l_1 and l_3 is equal to A_{13}, and the span of the spectrum used on links l_2 and l_4 is equal to A_{24}. Afterwards, the requests using only one link are assigned; the span of the spectrum used on links l_1, l_2, l_3, and l_4 is $A_{13} + A_1$, $A_{24} + A_2$, $A_{13} + A_3$, and $A_{24} + A_4$, respectively. Finally, the requests using l_1 and l_4 and the requests using l_2 and l_3 are assigned with the greedy algorithm; the span of the spectrum used on links l_1 and l_4 is equal to $\max(\pi(l_1), \pi(l_4))$, and the span of the spectrum used on links l_2 and l_3 is equal to $\max(\pi(l_2), \pi(l_3))$. This means that the span of this spectrum assignment is equal to the maximum load of the instance.

□

Theorem 18. *Let $(\mathcal{N}, \mathcal{R})$ be an instance of SA. If the length of the paths associated to the requests in $\mathcal{R}$ is at most α, then the greedy algorithm gives a 2α-approximation for the SA problem. In particular there is a 4-approximation polynomial-time algorithm for the SA problem in stars.*

Proof. Let $(\mathcal{N}, \mathcal{R})$ be an instance of SA. Let the requests of $\mathcal{R}$ be ordered in the non-increasing order of demands $r_1, r_2, \ldots, r_q$ (i.e., $d(r_1) \geq d(r_2) \geq \cdots \geq d(r_q)$). Let Π be the maximum load. We will use at most $2\alpha\Pi$ slots to allocate spectrum to the requests of $\mathcal{R}$. Suppose that we have already assigned spectrum to the first requests r_j, $j < i$ with the span $2\alpha\Pi$ and consider the request r_i with demand $d(r_i) = d$. For each link l of the path $P(r_i)$, let $R_i(l)$ be the set of already assigned requests conflicting with r_i on the link l. As the load of the link l is at most Π, the sum of the demands of the requests of $R_i(l)$ is at most $\Pi - d$. Since each of these requests has demand at least d, we have at most $\frac{\Pi - d}{d}$ requests in $R_i(l)$. This implies that the path $P(r_i)$ has at most $\frac{\alpha(\Pi - d)}{d}$ requests conflicting with r_i which have been already assigned spectrum. Consider the slots not occupied by these requests (available slots). If there exists an interval of d or more available slots below these requests or between two requests, we can assign to request r_i the first such interval.

Otherwise, between slot 1 and the first slot occupied by the conflicting requests and between the last slot occupied by a request and the first slot of the next request there are at most $d - 1$ available slots. As there are at most $\frac{\alpha(\Pi - d)}{d}$ requests conflicting with r_i, we have at most $\frac{\alpha(\Pi - d)}{d}$ such intervals. As the requests in $R_i(l)$ occupy at most $(\Pi - d)$ slots, we have at most $\alpha(\Pi - d)$ slots occupied by the conflicting requests and at most $\frac{\alpha(\Pi - d)}{d}(d - 1)$ slots available where we cannot provision r_i. Altogether, we have a number of non usable slots equal to $\alpha(\Pi - d) + \frac{\alpha(\Pi - d)}{d}(d - 1) = 2\alpha\Pi - 2\alpha d - \frac{\alpha(\Pi - d)}{d}$. So, there is an interval of $2\alpha d + \frac{\alpha(\Pi - d)}{d}$ available contiguous slots above all of the requests conflicting with r_i. We can allocate to r_i d contiguous slots in this interval. Therefore, if $\alpha = 2$ as is the case in stars, we obtain a 4-approximation. □

This approximation algorithm for stars together with the $2+\varepsilon$-approximation algorithm for paths presented in [BKK+04], imply constant factor approximations for tree networks which are spiders. A *spider* is a tree with one vertex of degree at least 3 and all others with degree at most 2. The following theorem states the approximation result on SA in spiders.

Theorem 19. *There is a $6+\varepsilon$-approximation for SA in spiders, for all $\varepsilon > 0$.*

Proof. Let $(S, \mathcal{R})$ be an instance of SA in a spider tree. Let v be the vertex of S of degree 3 and let S' be the star induced by v and all the vertices of S which are at distance 1 from v. The idea is to use the 4-approximation presented in Theorem 18, to allocate spectrum to the requests $\mathcal{R}'$ using the star S' and then for each path P of the paths of $S \setminus S'$, use the $2+\varepsilon$-approximation algorithm to allocate spectrum to the requests using P and which are not in $\mathcal{R}'$. This would yield the $6+\varepsilon$-approximation algorithm.

□

4.4 Spectrum Assignment in Binary Trees

The problem of Spectrum Assignment in binary trees has been studied in [SZDS13]. It has been proved that SA can be approximated within a ratio of $O(log(k))$ where k is the number of requests. The proof is based on the equivalence between SA and the problem of Interval Coloring (IC). In fact, the conflict graph of an instance of SA in a binary tree corresponds to an edge intersection graph of paths in a binary tree. These graphs have been proved to be chordal graphs in [GLS08, GJ85]. Using the problem of interval coloring in chordal graphs, we give in this section some constant-factor approximation algorithms for the problem of spectrum assignment in binary trees with special demand profiles. Namely, we examine the cases where the demands are in a set $\{k, kX\}$ $(k, X \in \mathbb{N}^*)$, in a set $\{kX, k(X+1)\}$ $(k, X \in \mathbb{N}^*)$, or bounded by D. For the latter case, we give a general approximation when the demands are bounded by $D \in \mathbb{N}$ and then give better approximations for the cases where the demands are bounded by $D \in \{3, 4, 5, 6\}$. It is important to recall here that even if the network is a path and the demands are bounded by 2, SA is strongly NP-complete. We first start by giving some definitions and then we state our results.

4.4.1 Definitions

A chord of a cycle C in a graph is an edge of the graph connecting two vertices that are not adjacent in C. A graph G is chordal if every cycle of G with at least 4 vertices has a chord. One important property of chordal graphs is their perfect elimination order. The *perfect elimination order (PEO)* of a graph is an ordering $x_1, x_2, \ldots, x_n$ of the vertices of the graph such that for $i = 1, \ldots, n-1$, the neighbors of x_i in $G[\{x_{i+1}, \ldots, x_n\}]$[1] form a clique. It is well known that a graph is chordal if and only

[1] For $S \subseteq V$, we define $G[S]$ as the subgraph of G induced by the vertices of S, i.e. the subgraph of G containing the vertices of S and all the edges of G which have both endpoints in S.

if it has a perfect elimination order. Paper [TY84] describes a linear time algorithm called maximum cardinality search that can be used to determine if a given graph has a perfect elimination order and construct such an ordering if it exists. Throughout the remainder of this chapter, we use *the reverse perfect elimination order (RPEO)* in the design of some algorithms. Note that if $v_1, v_2, \ldots, v_n$ is a RPEO of the vertices of a chordal graph, then for $i = 2, \ldots, n$, the neighbors of v_i in $G[\{v_1, \ldots, v_{i-1}\}]$ form a clique. Another tool we will be using is the *greedy algorithm* for IC. Defined similarly to the greedy algorithm for SA, the greedy algorithm for IC, also called the *First Fit algorithm* (FF) is an algorithm which assigns colors to vertices in a given order $v_1, \ldots, v_n$ such that a vertex v_i is assigned the smallest positive integer $g(v_i)$ such that $\{g(v_i), g(v_i) + w(v_i) - 1\} \cap \{g(v_j), g(v_j) + w(v_j) - 1\} = \emptyset$ for each v_j in $\{v_1, \ldots, v_{i-1}\}$ which is adjacent to v_i.

In a weighted graph $(G = (V, E), w)$, we define the weight of a subset $S \subseteq V$ to be the quantity $w(S) = \sum_{v \in S} w(v)$. The maximum weighted clique is a clique with the biggest weight. The density of $(G = (V, E), w)$ is the weight of the maximum weighted clique and is denoted by $\Delta(G, w)$. It is straightforward that $\Delta(G, w) \leq \chi(G, w)$, where $\chi(G, w)$ is the interval chromatic number of (G, w).

In the remainder of this section, we present our results for SA in binary trees with bounded demands as corollaries after proving theorems for IC in weighted chordal graphs with bounded weights. We note here that even though every conflict graph of an instance of SA in a binary tree is a chordal graph, the opposite is not true [GJ85]. In what follows, we provide approximation algorithms for IC in weighted chordal graphs with bounded weights. These algorithms give an upper bound on the interval chromatic number in terms of the density. The results we prove for IC in weighted chordal graphs with bounded weights with respect to upper bounds on $\chi(G, w)$ in terms of $\Delta(G, w)$ are summarized in Table 4.1.

4.4.2 Demands k and kX

In this section, we present an approximation algorithm for the SA problem when the demand of each request is either k or kX, $k, X \in \mathbb{N}^*$. We start by proving the following theorem for interval coloring in chordal graphs.

Theorem 20. *Let (G, w) be a weighted chordal graph with weights in the set $\{1, X\}$. There exists a polynomial-time algorithm that finds an interval coloring of (G, w) with $2\Delta(G, w) - \lfloor \frac{\Delta(G,w)}{X} \rfloor$ colors.*

Proof. It has been proved in [MB99] that there is an algorithm to find a $(2 - \frac{1}{X})$-approximation for the problem of interval coloring in interval graphs whenever we have only two weights 1 and X. We generalize this algorithm for chordal graphs as follows.

Let (G, w) be a weighted chordal graph with weights in $\{1, X\}$ and let $\Delta = \Delta(G, w)$ be its density. We will use $2\Delta - \lfloor \frac{\Delta}{X} \rfloor$ colors to color (G, w) as follows. We partition the colors into two sets. The first set S_1 contains colors from 1 to Δ and the second set S_2 contains colors from $\Delta + 1$ to $2\Delta - \lfloor \frac{\Delta}{X} \rfloor$.

Weights	**Upper bounds**
$\boldsymbol{\{k, kX\}}$ $\boldsymbol{k, X \in \mathbb{N}^*}$	$(2 - \frac{1}{X})\Delta(G, w) + k$
$\boldsymbol{\{kX, k(X+1)\}}$ $\boldsymbol{k, X \in \mathbb{N}^*}$	$(1 + \frac{1}{X})\Delta(G, w)$
$\boldsymbol{\leq 3}$	$\frac{19}{10}\Delta(G, w) + \frac{8}{5}$
$\boldsymbol{\leq 4}$	$\frac{59}{27}\Delta(G, w) + \frac{67}{27}$
$\boldsymbol{\leq 5}$	$\frac{859}{336}\Delta(G, w) + \frac{229}{56}$
$\boldsymbol{\leq 6}$	$\frac{287}{100}\Delta(G, w) + \frac{885}{200}$
$\boldsymbol{\leq W}$	$2\lceil \log_2(W) \rceil \Delta(G, w)$

Table 4.1: Summary of the upper bounds obtained on $\chi(G, w)$ for bounded weights

We order the vertices of G in the reverse perfect elimination order. Let $v_1, \ldots, v_n$ be the obtained ordering. Recall that the neighbors of v_i in $\{v_1, \ldots, v_{i-1}\}$ form a clique in the graph induced by $\{v_1, \ldots, v_{i-1}\}$. We use the greedy algorithm to assign colors to the vertices in this order with the additional property that colors assigned to a vertex are either included in S_1 or S_2 (We cannot use colors from both sets). We prove that with this algorithm, all vertices will be assigned colors in S_1 or S_2.

- All vertices of weight 1 will have a color in S_1. In fact, if a vertex v_i of weight 1 cannot be assigned a color in S_1, then its neighbors in $\{v_1, \ldots, v_{i-1}\}$ occupy all colors of S_1. This implies that the density of the graph is at least $\Delta + 1$ which is not possible.

- For vertices of weight X, suppose that there is a vertex v_j of weight X to which we cannot assign colors in S_1 nor in S_2. The minimum number of colors used in S_1 that can make it not possible to color v_j with colors from S_1 is $\lfloor \frac{\Delta}{X} \rfloor$ ($X - 1$ free colors then 1 occupied color, then $X - 1$ free colors and 1 occupied color ...). The weight of the neighbors of v_j in $\{v_1, \ldots, v_{j-1}\}$ which use colors in S_1 is at least $\lfloor \frac{\Delta}{X} \rfloor$. Since we cannot assign colors from S_2 to v_j and knowing that only vertices of the same weight X use colors from S_2 with the greedy algorithm, we deduce that the sum of the weights of the neighbors of v_j in $\{v_1, \ldots, v_{j-1}\}$ which use colors in S_2 is at least $|S_2| - (X - 1)$. So v_j and its neighbors form a clique of size $X + \lfloor \frac{\Delta}{X} \rfloor + |S_2| - (X - 1) = \Delta + 1$ as $|S_2| = \Delta - \lfloor \frac{\Delta}{X} \rfloor$. This implies that the density of G is at least $\Delta + 1$, which is not possible.

We have proved that when the weights are in the set $\{1, X\}$, we can color the

graph with at most $2\Delta - \lfloor \frac{\Delta}{X} \rfloor$ colors. □

Corollary 2. *Let (G, w) be a weighted chordal graph with weights in the set $\{k, kX\}$. There exists a polynomial time algorithm that finds an interval coloring of (G, w) with $2\Delta(G, w) - k\lfloor \frac{\Delta(G,w)}{kX} \rfloor$ colors.*

Proof. Note that to color a graph (G, w) with weights in $\{k, kX\}$, we can transform it to a graph (G, w') with weights in $\{1, X\}$, color (G, w') and then transform the colors we found into intervals of colors of size k. The number of colors used for G will be then at most k times the number of colors used for G'. Note also that the density of (G, w) is k times the density of (G, w'). This means that if we can color (G, w') with $a\Delta((G, w')) + b$ colors, then we can color (G, w) with $a\Delta((G, w)) + kb$ colors.

This implies that when the weights are in the set $\{k, kX\}$, we can color the graph with at most $2\Delta(G, w) - k\lfloor \frac{\Delta(G,w)}{kX} \rfloor$ colors. □

Thanks to Corollary 2, we can deduce the following corollary.

Corollary 3. *Let $\mathcal{I}$ be an instance of SA in a binary tree such that the demands of requests are in the set $\{k, kX\}$ and the span of $\mathcal{I}$ is $\mathcal{OPT}$. There is a polynomial-time algorithm that finds a spectrum assignment for $\mathcal{I}$ with span less than $(2-\frac{1}{X})\mathcal{OPT}+k$.*

Now, in what follows of this subsection, we find a lower bound on the number of colors that can be used to find an interval coloring of a weighted chordal graph with weights in $\{k, kX\}$.

Theorem 21. *There exists a family of weighted chordal graphs $(G_m)_{m\in\mathbb{N}^*}$, with weights in the set $\{k, kX\}$, for which the ratio between the interval chromatic number and the density tends to $2 - \frac{1}{X}$ when m tends to infinity.*

Proof. For $m > 0$, we build the weighted graph G_m of density $k(mX^2+1)$ as follows.

- $mX^2 + 1$ vertices of weight k each forming a "big" clique.
- For each subset S of $mX + 1$ vertices of the big clique, we add $m(X - 1)$ new vertices of weight kX each. These vertices form a clique with the vertices of S.

In any contiguous coloring of G_m, there exists an integer λ in $\{0, \ldots, kX-1\}$ such that the "big" clique uses $mX + 1$ colors congruent to λ modulo kX. Suppose that this is not true and that the big clique uses for each integer i in $\{0, \ldots, kX - 1\}$ at most mX colors which are congruent to i modulo kX. This means that the number of colors used is at most kmX^2. This is not possible since this maximum clique has weight $k(mX^2+1)$. Let S be the subset of vertices of the big clique using colors that are congruent to λ modulo kX. Vertices of S form a clique with $m(X - 1)$ vertices of weight kX. Each of these vertices uses a color congruent to λ modulo kX. In total, $m(2X - 1) + 1$ colors which are congruent to λ are used. This means that the number of colors used is at least $kmX(2X-1)+1$. The ratio between the chromatic

number and the density is then at least $\frac{kmX(2X-1)+1}{k(mX^2+1)} = 2 - \frac{1}{X} - \frac{(2k-1)X-k}{kX(mX^2+1)}$. When m goes to infinity, this ratio goes to $2 - \frac{1}{X}$.

□

4.4.3 Demands kX and $k(X+1)$

In this section, we present an approximation algorithm for the SA problem when the demand of each request is either kX or $k(X+1)$. We start by proving the following theorem for interval coloring in chordal graphs.

Theorem 22. *Let (G, w) be a weighted chordal graph with weights in $\{kX, k(X+1)\}$. There is a polynomial time algorithm to color G with at most $\frac{X+1}{X}\Delta(G, w)$ colors.*

Proof. Let (G, w) be a weighted chordal graph with weights in $\{kX, k(X+1)\}$. Let $m = \lfloor \frac{\Delta(G,w)}{kX} \rfloor$, we prove that we can color (G, w) with $k(X+1)m$ colors. We partition the set of colors $\{1, \ldots, k(X+1)m\}$ into m contiguous intervals I_i, $1 \leq i \leq m$ of size $k(X+1)$ each. Let us order the vertices of (G, w) in the RPEO order. We use the greedy algorithm to color the vertices in this order using for each vertex colors from exactly one interval I_i, $1 \leq i \leq m$. Suppose that we cannot color some vertex v_j, this means that each interval I_i, $1 \leq i \leq m$, contains a neighbor of v_j with weight at least kX (recall that the weights are either kX or $k(X+1)$). Since the neighbors of v_j which appear first in the RPEO form a clique with v_j, we have a clique of weight at least $mkX + kX > \Delta(G, w)$ which is not possible. Therefore we can color all the vertices.

□

Theorem 22 implies the following corollary.

Corollary 4. *There is a $\frac{X+1}{X}$-approximation algorithm for SA in binary trees when the demands of the requests are in the set $\{kX, k(X+1)\}$.*

4.4.4 Maximum demand D

In this section, we present an approximation algorithm for the SA problem when the maximum demand is D. The approximation algorithm proposed for Interval Coloring in chordal graphs in [PPR05] and which gives $2log_2(n)$ approximation where n is the number of vertices, allows to prove the following theorem.

Theorem 23. *Let (G, w) be a weighted chordal graph with maximum weight W. There is a polynomial time algorithm that finds an interval coloring of (G, w) which uses at most $2\lceil \log_2(W) \rceil \Delta(G, w)$ colors.*

Proof. The proof is due to [PPR05] which proposes a $O(\log_2(n))$-approximation where n is the number of vertices. We only replace n by the maximum weight. We include it for completeness.

We prove that there is a $2\lceil \log_2(W) \rceil$-approximation for interval coloring of chordal graphs where W is the maximum weight. Let (G, w) be a vertex-weighted

chordal graph with maximum weight W. We set $k = \lceil \log_2(W) \rceil$. Let us partition the set of vertices V into k subsets S_i, $i \in \{1, \ldots, k\}$ such that for each vertex $v \in S_i$, $w(v) \in [\frac{W}{2^i}, \frac{W}{2^{i-1}}]$. We first ignore the weights and optimally color each graph G_i induced by the subset S_i. As the graphs are chordal, we can color the vertices of G_i with $\omega(G_i)$ colors where $\omega(G_i)$ is the clique number of G_i. Afterwards, we replace the color of each vertex $v \in G_i$ by an interval of $w(v)$ colors. This way, we obtain an interval coloring of G_i with at most $\frac{W}{2^{i-1}}\omega(G_i)$ colors. Therefore, the vertices of G can be colored with c colors where $c = \sum_{i=1}^{k} \frac{W}{2^{i-1}}\omega(G_i)$. Note that $\frac{W}{2^i}\omega(G_i) \leq \Delta(G, w)$ which implies that $c \leq \sum_{i=1}^{k} 2\Delta(G, w) = 2k\Delta(G, w)$. The $2\lceil \log_2(W) \rceil$-approximation follows. □

Theorem 23 implies the following corollary.

Corollary 5. *There is a $2\lceil \log_2(D) \rceil$-approximation for SA in binary trees where D is the maximum demand.*

4.4.5 Maximum demand at most 6

In the previous subsection, a $2\lceil \log_2(D) \rceil$-approximation algorithm for the SA problem in binary trees where the maximum demand is at most D has been presented. This approximation is achieved by partitioning the requests into subsets of close demands. This technique is used not only in binary trees but also in general graphs as a heuristic [WM14]. In what follows, we use different techniques to find better approximations for SA in binary trees for some given values of the maximum demand D. The techniques we use were introduced in [LLQ04] to approximate DSA. Results in [LLQ04] can extend directly to SA in path networks giving approximation algorithms with factors $\frac{4}{3}$ and 1.7 when the spectrum demands are bounded by 2 and 3, respectively. In what follows we use the same techniques to design constant-factor approximations for SA in binary trees when the spectrum demand is bounded by 6.

We first prove the following theorem for interval coloring.

Theorem 24. *Let (G, w) be a weighted chordal graph. There are polynomial-time algorithms which find an interval coloring of (G, w) with at most $\frac{19}{10}\Delta(G, w) + \frac{8}{5}$, $\frac{59}{27}\Delta(G, w) + \frac{67}{27}$, $\frac{859}{336}\Delta(G, w) + \frac{229}{56}$ and $\frac{287}{100}\Delta(G, w) + \frac{885}{200}$ colors when the maximum weight is bounded by 3,4,5 and 6, respectively.*

Proof. As in the previous sections, $\Delta(G, w)$ refers to the density of the weighted graph (G, w) and will be abbreviated in this proof to Δ.

Let $\mathcal{C}(d, S)$ denote the set of instances of IC in which the graph is chordal, the density is at most d and the weights are in the set S. Let $c(d, S)$ denote the smallest integer α such that for each instance of $\mathcal{C}(d, S)$, there is an interval coloring with at most α colors (if such α exists).

We present first the general approach to solve the problem for any maximum weight W before presenting the cases $W \in \{3, 4, 5, 6\}$ in details.

General Approach Let (G, w) be a weighted chordal graph with maximum weight W. To color the graph G, we proceed in two phases as follows.

- *Partitioning the vertices into multi-level blocks:* in this phase, the vertices are partitioned into blocks. We will have for each $i \in \{1, \ldots, W\}$, a set $\mathcal{B}_i$ of n_i level-i blocks $B_i^1, \ldots, B_i^{n_i}$ each of density d_i. We order the blocks in the lexicographic order: block B_i^j is before block $B_{i'}^{j'}$ if $i < i'$ or $i = i'$ and $j < j'$.

  Our algorithm consists in considering successively the vertices in the RPEO order and assigning a new vertex v to the first available block (in the block's order). In more details, we assign a vertex v to a block B if the weight of the clique induced by v and its neighbors in B does not exceed the density of the block. The vertex v and its neighbors in B indeed form a clique since the graph is chordal and we consider the vertices in the RPEO order.

  We will choose the parameters d_i and n_i (see details after) in such a way that the following property is satisfied:

  Property *: Each vertex of weight i is assigned to some block in the set $\mathcal{B}_l$ such that $l \leq i$.

  In particular, this means that at the end of the algorithm each vertex is assigned to some block.

- *Solving the problem of interval coloring for each block:* in this second phase, the vertices of each block B_i^j are colored using an algorithm to solve instances with density d_i and weights in $S_i = \{i, \ldots, W\}$ (the possible weights of the vertices in B_i^j). Note that the vertices of a block B_i^j induce a graph which belongs to $\mathcal{C}(d_i, S_i)$. The algorithm we use is designed to use no more than $c(d_i, S_i)$ colors.

Therefore, the total number of colors used to color the whole graph is at most

$$\sum_{i=1}^{W} n_i c(d_i, \{i, \ldots, W\})$$

The total number of colors depends on n_i and d_i. In fact, we will proceed as follows. For a chosen set of values of the densities d_i, we will choose the smallest possible n_i such that Property* is satisfied. Afterwards, we will compute $c(d_i, \{i, \ldots, W\})$ and therefore the total number of colors for the chosen values of d_i. We will do this for many values of the densities d_i and keep the set of values which minimize the total number of colors.

Choice of the n_i Note that, if for some i, $d_i < i$, then $n_i = 0$ as a block of $\mathcal{B}_i$ cannot be used to assign a vertex of weight $\geq i$ (recall that the vertices of weight $< i$ are by Property * all assigned to blocks of $\mathcal{B}_l$ with $l < i$). So, in the following claims, we suppose $d_i \geq i$ for all i.

Claim 16. $n_1 = \left\lceil \frac{\Delta}{d_1} \right\rceil$

Proof. Suppose that a vertex v of weight 1 cannot be assigned to any block of $\mathcal{B}_1$. This means that, for each block B of $\mathcal{B}_1$, vertex v and its neighbors in B form a clique of size $> d_1$ and so the weight of the neighbors of v in B is at least d_1. This implies that the weight of the neighbors of v in all of the blocks in $\mathcal{B}_1$ is at least $n_1 d_1$. Since we are considering the vertices in the RPEO, this implies that the clique induced by v and its neighbors in $\mathcal{B}_1$ is of weight $n_1 d_1 + 1$ which exceeds Δ for $n_1 = \left\lceil \frac{\Delta}{d_1} \right\rceil$. This is not possible. □

Claim 17. $n_2 = \left\lceil \frac{\Delta - 1 - n_1(d_1 - 1)}{\Delta_2^2} \right\rceil$ *where* $\Delta_2^2 = \max\{2, d_2 - 1\}$.

Proof. Suppose that a vertex v of weight 2 cannot be assigned to any block of $\mathcal{B}_1$ or $\mathcal{B}_2$. This means that, for each block B of $\mathcal{B}_1$ (resp. $\mathcal{B}_2$), vertex v and its neighbors in B form a clique of size $> d_1$ (resp. $> d_2$) and so the weight of the neighbors of v in B is at least $d_1 - 1$ (resp. $d_2 - 1$). However, if $d_2 = 2$, as all the vertices of weight 1 are assigned to blocks of $\mathcal{B}_1$, v has necessarily one neighbor of weight 2 in each block of $\mathcal{B}_2$. Therefore, if we let $\Delta_2^2 = \max\{2, d_2 - 1\}$, the clique induced by v and its neighbors in the RPEO has a weight at least $n_1(d_1 - 1) + n_2\Delta_2^2 + 2$ which exceeds Δ for $n_2 = \left\lceil \frac{\Delta - 1 - n_1(d_1 - 1)}{\Delta_2^2} \right\rceil$. □

Example: Consider the case $W = 2$, and let $d_1 = 2$ and $d_2 = 2$. Applying the formula we get $n_1 = \left\lceil \frac{\Delta}{2} \right\rceil$ and $n_2 = \left\lceil \frac{\Delta - 1 - n_1}{2} \right\rceil$. Using the fact that $c(2, \{1, 2\}) = c(2, \{2\}) = 2$, the number of colors used by the algorithm to color a chordal graph with weight bounded by 2 is at most $2n_1 + 2n_2 \leq \Delta + \left\lceil \frac{\Delta}{2} \right\rceil$.

Claim 18. $n_i = \left\lceil \frac{\Delta + 1 - i - \sum_{l=1}^{i-1} n_l \Delta_i^l}{\Delta_i^i} \right\rceil$ *where* $\Delta_i^l = \max\{l, d_l + 1 - i\}$.

Proof. Suppose that a vertex v of weight i cannot be assigned to any block of $\mathcal{B}_l$ with $l \leq i$. This means that, for each block B of $\mathcal{B}_l$, vertex v and its neighbors in B form a clique of size $> d_l$ and so the weight of the neighbors of v in B is at least $d_l + 1 - i$. Furthermore, as all the vertices of weight $< l$ are assigned to blocks of $\mathcal{B}_j$ for $j < l$, v has necessarily one neighbor of weight at least l in any block of $\mathcal{B}_l$. Therefore, if we let $\Delta_i^l = \max\{l, d_l + 1 - i\}$, the clique induced by v and its neighbors in the RPEO has a weight at least $\sum_{l=1}^{i} n_l \Delta_i^l + i$ which exceeds Δ for $n_i = \left\lceil \frac{\Delta + 1 - i - \sum_{l=1}^{i-1} n_l \Delta_i^l}{\Delta_i^i} \right\rceil$. □

Maximum weight 3 Let $W = 3$. We choose some values for d_i and using the claims above, we obtain the following values of n_i:

- $d_1 = d_2 = d_3 = 3$. $n_1 = \lceil \frac{\Delta}{3} \rceil$ and $n_2 = \lceil \frac{\Delta - 1 - 2n_1}{2} \rceil$ and $n_3 = \lceil \frac{\Delta - 2 - n_1 - 2n_2}{3} \rceil$.
- $d_1 = 5, d_2 = d_3 = 3$. $n_1 = \lceil \frac{\Delta}{5} \rceil$ and $n_2 = \lceil \frac{\Delta - 1 - 4n_1}{2} \rceil$ and $n_3 = \lceil \frac{\Delta - 2 - 3n_1 - 2n_2}{3} \rceil$.
- $d_1 = d_2 = 5, d_3 = 3$. $n_1 = \lceil \frac{\Delta}{5} \rceil$ and $n_2 = \lceil \frac{\Delta - 1 - 4n_1}{4} \rceil$ and $n_3 = \lceil \frac{\Delta - 2 - 3n_1 - 2n_2}{3} \rceil$.

To compare the values of the total number of colors we need to compute $c(3, S)$ for some basic sets S. We recall that $c(d, S)$ is the minimum number of colors which can be used in an interval coloring of any chordal graph with density d and weights in S.

- $\boldsymbol{c(3, \{1, 2, 3\}) = 4.}$

  We first prove that $c(3, \{1, 2\}) \geq 4$. Let us consider the example presented in Figure 4.3 in which the density is 3 and the maximum weight is 2. The graph in the example consists of a clique of 3 vertices of weight one, such that each vertex of weight one is joined to a vertex of weight 2. This graph cannot be colored using only 3 colors. If we suppose that it can be colored with 3 colors $\{1, 2, 3\}$, then one of the vertices of weight one will have to be assigned color 2. For this vertex, the neighbor of weight 2 cannot be colored since the only available colors are 1 and 3 which are not contiguous.

  To prove that $c(3, \{1, 2, 3\}) \leq 4$, we use the First Fit algorithm in the RPEO which needs at most 4 colors .

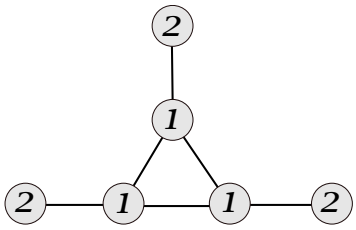


Figure 4.3: An example showing that $c(3, \{1, 2\}) \neq 3$

- $\boldsymbol{c(3, \{2, 3\}) = c(3, \{3\}) = 3.}$

  In fact in an instance of $\mathcal{C}(3, \{2, 3\})$, all vertices are isolated and we can hence easily color them with at most 3 colors.

- $\boldsymbol{c(4, \{1, 2, 3\}) = 6.}$

  We first prove that $c(4, \{1, 2, 3\}) \geq 6$. Let us consider the example presented in Figure 4.4 which consists of a clique of four vertices of weight one. Each vertex of weight one is joined to a vertex of weight 3 and each pair of vertices of weight one is joined to a vertex of weight 2. Suppose that we only use 5

colors $\{1, 2, 3, 4, 5\}$ to color this graph. If one of the vertices of weight 1 uses color 3, then its neighbor which has weight 3 cannot be colored. Otherwise, if the vertices of weight 1 use colors $\{1, 2, 4, 5\}$, then the vertex of weight 2 which is adjacent to the vertices of weight 1 which have colors 2 and 4 cannot be colored.

To color any instance in $\mathcal{C}(4, \{1, 2, 3\})$ with at most 6 colors, we use the first fit algorithm in the RPEO.

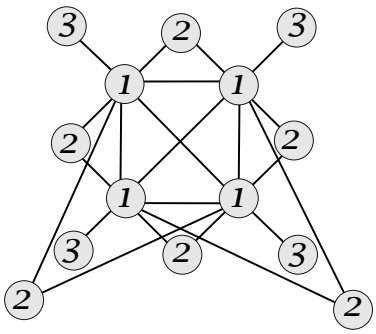


Figure 4.4: An example showing that $c(4, \{1, 2, 3\}) \neq 5$

- $\boldsymbol{c(4, \{2, 3\}) = 4.}$

  The First Fit algorithm in the RPEO, colors any instance in $\mathcal{C}(4, \{2, 3\})$ with at most 4 colors.

- $\boldsymbol{c(5, \{1, 2, 3\}) = 7.}$

  We first prove that $c(5, \{1, 3\}) \geq 7$. Let us consider the graph G consisting of a clique of 5 vertices each of weight 1 and such that each pair of vertices of the clique is connected to a vertex of weight 3. The graph G is chordal with density 5 and weights in $\{1, 3\}$. Let us suppose that we can color G with only six colors. There are either two vertices of weight one colored with colors 2 and 4 or two vertices of weight one colored with colors 3 and 5. In both cases the vertex of weight 3 adjacent to these two vertices cannot be colored.

  Now let us describe an algorithm that takes an instance of $\mathcal{C}(5, \{1, 2, 3\})$ and colors it with at most 7 colors. The algorithm is a First Fit algorithm in the RPEO of the vertices with the additional feature that colors 5 and 6 are forbidden for vertices of weight 1.

  - If a vertex v of weight 3 is considered, then if v has a neighbor of weight 2 colored with $\{\alpha, \alpha + 1\}$, we color v with $\{1, 2, 3\}$ if $\alpha \geq 4$ or $\{5, 6, 7\}$ if $\alpha \leq 3$. If v has two neighbors of weight 1; if color 7 is not used we color v with $\{5, 6, 7\}$. If color 7 is used, but not color 4 we color v with $\{4, 5, 6\}$. If both colors 4 and 7 are used, we color v with $\{1, 2, 3\}$.

– If a vertex v of weight 2 is considered, then if v has 3 neighbors of weight 1, we color v with $\{5,6\}$. If it has one neighbor of weight 2 colored $\{\alpha, \alpha+1\}$ and one of weight 1 colored β, then we color v with $\{5,6\}$ if $\alpha \leq 3$; with $\{1,2\}$ if $\alpha \geq 4$ and $\beta \geq 3$; with color $\{6,7\}$ if $\alpha = 4$ and $\beta \leq 2$ or $\{3,4\}$ if $\alpha \geq 5$ and $\beta \leq 2$.

- **$c(5,\{2,3\}) = 5$.**

  The First Fit algorithm in the RPEO in which we forbid color 3 to vertices of weight 2 uses at most 5 colors.

Now, we can compute the number of colors for the 3 cases considered above.

- If we set $d_1 = d_2 = d_3 = 3$, the number of colors used is $n_1 c(3,\{1,2,3\}) + n_2 c(3,\{2,3\}) + n_3 c(3,\{3\}) = 4n_1 + 3n_2 + 3n_3$. As $n_3 \leq \frac{\Delta - n_1 - 2n_2}{3}$ the number of colors is at most $\Delta + 3n_1 + n_2$ and as $n_2 \leq \frac{\Delta - 2n_1}{2}$ it is at most $\frac{3\Delta}{2} + 2n_1$. Finally, as $n_1 \leq \frac{\Delta}{3} + \frac{2}{3}$, the number of colors used is at most $\frac{13}{6}\Delta + \frac{4}{3}$.

- If we set $d_1 = 5, d_2 = d_3 = 3$, the number of colors used is $n_1 c(5,\{1,2,3\}) + n_2 c(3,\{2,3\}) + n_3 c(3,\{3\}) = 7n_1 + 3n_2 + 3n_3$. As $n_3 \leq \frac{\Delta - 3n_1 - 2n_2}{3}$ the number of colors is at most $\Delta + 4n_1 + n_2$ and as $n_2 \leq \frac{\Delta - 4n_1}{2}$ it is at most $\frac{3\Delta}{2} + 2n_1$. Finally, as $n_1 \leq \frac{\Delta}{5} + \frac{4}{5}$, the number of colors used is at most $\frac{19}{10}\Delta + \frac{8}{5}$.

- If we set $d_1 = d_2 = 5$, and $d_3 = 3$, the number of colors used is $n_1 c(5,\{1,2,3\}) + n_2 c(5,\{2,3\}) + n_3 c(3,\{3\}) = 7n_1 + 5n_2 + 3n_3$. As $n_3 \leq \frac{\Delta - 3n_1 - 3n_2}{3}$ the number of colors is at most $\Delta + 4n_1 + 2n_2$ and as $n_2 \leq \frac{\Delta - 4n_1 + 2}{4}$ it is at most $\frac{3\Delta}{2} + 2n_1 + 1$. Finally, as $n_1 \leq \frac{\Delta}{5} + \frac{4}{5}$, the number of colors used is at most $\frac{19}{10}\Delta + \frac{13}{5}$.

We have tried other values of d_i and n_i but we obtained bigger numbers of colors. For example:

- If we set $d_1 = 4$, $d_2 = d_3 = 3$, using $c(4,\{1,2,3\}) = 6$, the number of colors is at most $\frac{17}{8}\Delta + O(1)$.

- If we set $d_1 = d_2 = 4$, and $d_3 = 3$, using $c(4,\{2,3\}) = 4$ the number of colors is at most $\frac{13}{6}\Delta + O(1)$.

- If we set $d_1 = 6$, and $d_2 = d_3 = 3$, then using $c(6,\{1,2,3\}) = 9$ (to be computed after), the number of colors is at most $\frac{23}{12}\Delta + O(1)$.

- If we set $d_1 = 6$, $d_2 = 5$ and $d_3 = 3$, then using $c(5,\{2,3\}) = 5$, the number of colors is at most $\frac{23}{12}\Delta + O(1)$.

In summary, the best approximation obtained for maximum weight 3 is with a multiplicative ratio of $\frac{19}{10}$ and an additive constant of $\frac{8}{5}$.

Maximum weight 4 Let us first compute $c(4, S)$ for some basic sets S.

- $\boldsymbol{c(4, \{2, 3, 4\}) = 4.}$

  A First Fit algorithm in the RPEO uses at most 4 colors. In fact, in an instance of $\mathcal{C}(4, \{2, 3, 4\})$, vertices of weights 3 or 4 are isolated and it suffices to color vertices of weight 2 with $\{1, 2\}$ or $\{3, 4\}$.

- $\boldsymbol{c(4, \{3, 4\}) = c(4, \{4\}) = 4.}$

  In an instance of $\mathcal{C}(4, \{3, 4\})$, all vertices are isolated and can be colored independently.

- $\boldsymbol{c(6, \{1, 2, 3, 4\}) = c(6, \{1, 2, 3\}) = 9.}$

  We first prove that $c(6, \{1, 3\}) \geq 9$. Let us consider the graph G consisting of a clique of 6 vertices of weight 1 each and such that the vertices of each triple of the clique are connected to a vertex of weight 3. The graph G is chordal with density 6 and weights in $\{1, 3\}$. Let us suppose that we can color G with only 8 colors. There are three vertices of weight 1 using colors $\{1, 4, 7\}$ or three vertices of weight 1 using colors $\{2, 5, 8\}$ or two vertices of weight 1 using colors $\{3, 6\}$. In any of these three cases, a vertex of weight 3 cannot be colored.

  Now let us describe an algorithm that takes an instance of $\mathcal{C}(6, \{1, 2, 3, 4\})$ and colors it with at most 9 colors. The algorithm is a First Fit algorithm in the RPEO of the vertices with two additional features: colors 6,7 and 8 are forbidden to vertices of weight 1, and each vertex of weight 2 is assigned colors $\{1, 2\}$, $\{3, 4\}$, $\{5, 6\}$, or $\{7, 8\}$ and not any other contiguous combination of two colors. This algorithm uses at most 9 colors.

  - If a vertex v of weight 4 is considered, then if v has a neighbor of weight 2 which has been already colored, the possible sets of color used by this neighbor are $\{1, 2\}$, $\{3, 4\}$, $\{5, 6\}$, or $\{7, 8\}$. In any case, v can be colored with 4 contiguous colors. If v has two neighbors of weight 1 each that have been already colored, then either one of the colors 9 or 5 is not used by this neighbor, and in this case v can use it along with the colors $\{6, 7, 8\}$ (which are forbidden for vertices of weight 1), or both colors 9 and 5 are used and v can use colors $\{1, 2, 3, 4\}$.
  - If a vertex v of weight 3 is considered, then if v has a neighbor of weight 3 colored with $\{\alpha, \alpha + 1, \alpha + 2\}$, we color v with $\{1, 2, 3\}$ if $\alpha \geq 4$ or with $\{7, 8, 9\}$ if $\alpha \leq 3$. If v has two neighbors one of weight 2 colored with $\{\alpha, \alpha + 1\}$ and one of weight 1 colored with β, then we color v with $\{6, 7, 8\}$ if $\alpha = 1$ or 3; $\{1, 2, 3\}$ if $\alpha = 5$ or 7 and $\beta \geq 4$; $\{7, 8, 9\}$ if $\alpha = 5$ and $\beta \leq 3$; $\{4, 5, 6\}$ if $\alpha = 7$ and $\beta \leq 3$.
  - If a vertex v of weight 2 is considered. If all its neighbors that have been already colored are of weight 1, then v can be assigned colors $\{7, 8\}$. If

v has two colored neighbors of weight 2 each, or one colored neighbor of weight 2 and two other colored neighbors with weight 1, or one colored neighbor of weight 3 and another of weight 1, or one vertex of weight 4, then one of the channels $\{1,2\}$, $\{3,4\}$, $\{5,6\}$, or $\{7,8\}$ is necessarily free to be used.

- **$c(6,\{2,3,4\}) = 8$.**

  We first prove that $c(6,\{2,4\}) \geq 8$. Let us consider the graph G which consists of a clique of 3 vertices of weight 2 each, and such that each vertex of weight 2 is connected to a vertex of weight 4. The graph G is chordal and has density 6 and weights in $\{2,4\}$. Let us suppose that we can color G with only 7 colors. In any possible coloring of the vertices of weight 2, a vertex v of weight 2 has to use either colors $\{3,4\}$ or $\{4,5\}$. In both cases, the neighbor of v which has weight 4 cannot be colored.

  Now let us describe the algorithm that colors an instance of $\mathcal{C}(6,\{2,3,4\})$. The algorithm uses First Fit in the RPEO with the additional feature that the possible combinations of colors for vertices of weight 2 are the following: $\{1,2\}$, $\{3,4\}$, $\{5,6\}$, and $\{7,8\}$.

Now, we can compute the number of colors for $d_1 = 6$ and $d_2 = d_3 = d_4 = 4$.

The number of colors used is $n_1 c(6,\{1,2,3,4\}) + n_2 c(4,\{2,3,4\}) + n_3 c(4,\{3,4\}) + n_4 c(4,\{4\}) = 9n_1 + 4n_2 + 4n_3 + 4n_4$.

We have $n_1 = \lceil \frac{\Delta}{6} \rceil$; $n_2 = \lceil \frac{\Delta - 1 - 5n_1}{3} \rceil$; $n_3 = \lceil \frac{\Delta - 2 - 4n_1 - 2n_2}{3} \rceil$, and $n_4 = \lceil \frac{\Delta - 3 - 3n_1 - 2n_2 - 3n_3}{4} \rceil$.

As $n_4 \leq \frac{\Delta - 3n_1 - 2n_2 - 3n_3}{4}$ the number of colors is at most $\Delta + 6n_1 + 2n_2 + n_3$. As $n_3 \leq \frac{\Delta - 4n_1 - 2n_2}{3}$ the number of colors is at most $\frac{4\Delta}{3} + \frac{14}{3}n_1 + \frac{4}{3}n_2$, and as $n_2 \leq \frac{\Delta - 5n_1 + 1}{3}$ it is at most $\frac{16\Delta}{9} + \frac{22}{9}n_1 + \frac{4}{9}$. Finally, as $n_1 \leq \frac{\Delta + 5}{6}$, the number of colors is at most $\frac{59}{27}\Delta + \frac{67}{27}$.

We have computed the number of colors for other choices of the d_i but the values are bigger.

- For $d_1 = d_2 = d_3 = d_4 = 4$, we obtain $\frac{91}{36}\Delta + O(1)$ colors.
- For $d_1 = d_2 = 5$ and $d_3 = d_4 = 4$, we obtain $\frac{73}{30}\Delta + O(1)$ colors.
- For $d_1 = 5$ and $d_2 = d_3 = d_4 = 4$, we obtain $\frac{109}{45}\Delta + O(1)$ colors.
- For $d_1 = d_2 = d_3 = 6$, $d_4 = 4$, we obtain $\frac{139}{60}\Delta + O(1)$ colors.
- For $d_1 = d_2 = 6$ and $d_3 = d_4 = 4$, we obtain $\frac{67}{30}\Delta + O(1)$ colors.

The best approximation obtained for maximum weight 4 is with a multiplicative ratio of $\frac{59}{27}$ and an additive constant of $\frac{67}{27}$.

Maximum weight 5 Let us first compute $c(5, S)$ for some basic sets S.

- $\boldsymbol{c(5, \{1, 2, 3, 4, 5\}) = 8.}$

  In fact, we know that $c(5, \{1, \ldots, 4\}) = 8$ and for any chordal graph with density 5 and maximum weight 5, vertices of weight 5 are isolated and can be colored independently from the others.

- $\boldsymbol{c(5, \{2, 3, 4, 5\}) = 5.}$

  In a chordal graph of density 5 and weights in $\{2, \ldots, 5\}$, vertices of weight 4 or 5 are isolated and can be easily colored. For other vertices, we know that $c(5, \{2, 3\}) = 5$.

- $\boldsymbol{c(5, \{3, 4, 5\}) = c(5, \{4, 5\}) = 5.}$

  All vertices are isolated and can be easily colored.

- $\boldsymbol{c(6, \{1, 2, 3, 4, 5\}) = 10.}$

  We first prove that $c(6, \{1, \ldots, 5\}) = 10$. Let us consider the graph G which consists of a clique C of 6 vertices of weight 1 each such that each of the vertices of C is connected to a vertex of weight 5, and each pair of vertices of C is connected to a vertex of weight 4. Let us suppose that we can color G with 9 colors. No vertex of weight 1 can use color 5 because otherwise its neighbor of weight 5 cannot be colored. In any possible coloring of the vertices of weight 1 without using color 5, there are two vertices whose neighbor of weight 4 cannot be colored.

  The First Fit in the RPEO colors any instance of $\mathcal{C}(6, \{1, \ldots, 5\})$ with at most 10 colors.

- $\boldsymbol{c(6, \{2, 3, 4, 5\}) = 8.}$

  Vertices of weight 5 are isolated and can be easily colored. As for other vertices we have already proved that $c(6, \{2, 3, 4\}) = 8$.

- $\boldsymbol{c(6, \{3, 4, 5\}) = 6.}$

  Vertices of weight 4 and 5 are isolated and vertices of weight 3 can be colored using First Fit in the RPEO with either the colors $\{1, 2, 3\}$ or $\{4, 5, 6\}$.

- $\boldsymbol{c(7, \{1, 2, 3, 4, 5\}) \leq 12.}$

  To obtain a coloring with 12 colors, we use the First Fit algorithm in the RPEO with the additional feature of forbidding colors $\{8, 9, 10, 11, 12\}$ to vertices of weight 1 and colors $\{8, 9\}$ and $\{9, 10\}$ for vertices of weight 2. The proof that the algorithm works is done by considering the various possibilities when a new vertex is added.

Now we can compute the number of colors for $d_1 = 7$ and $d_2 = d_3 = d_4 = d_5 = 5$. The number of colors used is $n_1 c(7, \{1, 2, 3, 4, 5\}) + n_2 c(5, \{2, 3, 4, 5\}) + n_3 c(5, \{3, 4, 5\}) + n_4 c(5, \{4, 5\}) + n_5 c(5, \{5\}) = 12n_1 + 5n_2 + 5n_3 + 5n_4 + 5n_5$.

We have $n_1 = \lceil \frac{\Delta}{7} \rceil$, $n_2 = \lceil \frac{\Delta-1-6n_1}{4} \rceil$, $n_3 = \lceil \frac{\Delta-2-5n_1-3n_2}{3} \rceil$, $n_4 = \lceil \frac{\Delta-3-4n_1-2n_2-3n_3}{4} \rceil$, and $n_5 = \lceil \frac{\Delta-4-3n_1-2n_2-3n_3-4n_4}{5} \rceil$.

As in the preceding cases, we, successively, use upper bounds for the n_i. The number of colors is at most $\Delta + 9n_1 + 3n_2 + 2n_3 + n_4$, then $\frac{5\Delta}{4} + 8n_1 + \frac{5}{2}n_2 + \frac{5}{4}n_3$, then $\frac{5\Delta}{3} + \frac{71}{12}n_1 + \frac{5}{4}n_2$, then $\frac{95\Delta}{48} + \frac{97}{24}n_1 + \frac{5}{8} \leq \frac{859}{336}\Delta + \frac{229}{56}$.

We have computed the number of colors for other choices of the d_i but we obtained bigger values as we present in what follows.

- If we set $d_1 = d_2 = d_3 = d_4 = d_5 = 5$, then the number of colors used is $8n_1 + 5n_2 + 5n_3 + 5n_4 + 5n_5$ which is at most $\frac{679}{240}\Delta + O(1)$.
- If we set $d_1 = d_2 = d_3 = 6$,$d_4 = d_5 = 5$, then the number of colors used is $10n_1 + 8n_2 + 6n_3 + 5n_4 + 5n_5$ which is at most $\frac{659}{240}\Delta + O(1)$.
- If we set $d_1 = 6, d_2 = d_3 = d_4 = d_5 = 5$, then the number of colors used is $10n_1 + 5n_2 + 5n_3 + 5n_4 + 5n_5$ which is at most $\frac{763}{288}\Delta(G, w) + O(1)$.

The best approximation obtained for maximum weight 5 is with a multiplicative ratio of $\frac{859}{336}$ and an additive constant of $\frac{229}{56}$.

Maximum weight 6 Let us first compute $c(6, S)$ for some basic sets S. Note that with a density at most 6, any vertices of weight 6 are isolated. We can then deduce the following from what we have computed for a maximum weight of 5.

- $\boldsymbol{c(6, \{1, 2, 3, 4, 5, 6\}) = 10.}$
- $\boldsymbol{c(6, \{2, 3, 4, 5, 6\}) = 8.}$
- $\boldsymbol{c(6, \{3, 4, 5, 6\}) = c(6, \{4, 5, 6\}) = c(6, \{5, 6\}) = c(6, \{6\}) = 6.}$
- $\boldsymbol{c(7, \{1, 2, 3, 4, 5, 6\}) = 12.}$

  We use the algorithm which gives $c(7, \{1, 2, 3, 4, 5, \}) \leq 12$. If a vertex v of weight 6 is added it is joined to at most one vertex of weight 1 of color β. If $\beta \leq 6$, we color v with colors $\{7, 8, 9, 10, 11, 12\}$) and if $\beta = 7$ with colors $\{1, 2, 3, 4, 5, 6\}$) and so $c(7, \{1, 2, 3, 4, 5, 6\}) \leq 12$.

  To show that $c(7, \{1, 2, 3, 4, 5, 6\}) \geq 12$, we consider the chordal graph consisting of a clique of 7 vertices of weight one such that each vertex of this clique is joined to a vertex of weight 6. Furthermore we join each pair of vertices of weight 1 to a vertex of weight 5. Suppose that we can color the graph with 11 colors. Color 6 cannot be used for any vertex of weight 1, otherwise its neighbor of weight 6 cannot be colored. Furthermore we can use for vertices of weight 1 at most one of the pair of colors $\{2, 7\}$, $\{3, 8\}$, $\{4, 9\}$, $\{5, 10\}$. So we have altogether 5 colors forbidden for vertices of weight 1 and so only 6 available colors which is impossible.

Now we can compute the number of colors for $d_1 = 7$ and $d_2 = d_3 = d_4 = d_5 = d_6 = 6$.

The number of colors used is $n_1 c(7, \{1,2,3,4,5,6\}) + n_2 c(6, \{2,3,4,5,6\}) + n_3 c(6, \{3,4,5,6\}) + n_4 c(6, \{4,5,6\}) + n_5 c(6, \{5,6\}) + n_6 c(6, \{6\}) = 12n_1 + 8n_2 + 6n_3 + 6n_4 + 6n_5 + 6n_6$.

We have $n_1 = \lceil \frac{\Delta}{7} \rceil$, $n_2 = \lceil \frac{\Delta - 1 - 6n_1}{5} \rceil$, $n_3 = \lceil \frac{\Delta - 2 - 5n_1 - 4n_2}{4} \rceil$, $n_4 = \lceil \frac{\Delta - 3 - 4n_1 - 3n_2 - 3n_3}{4} \rceil$, $n_5 = \lceil \frac{\Delta - 4 - 3n_1 - 2n_2 - 3n_3 - 4n_4}{5} \rceil$, and $n_6 = \lceil \frac{\Delta - 5 - 2n_1 - 2n_2 - 3n_3 - 4n_4 - 5n_5}{6} \rceil$.

Like in the preceding cases, we obtain upperbounds on n_i. The number of colors is at most $\Delta + 10n_1 + 6n_2 + 3n_3 + 2n_4 + n_5$, then $\frac{6\Delta}{5} + \frac{47}{5}n_1 + \frac{28}{5}n_2 + \frac{12}{5}n_3 + \frac{6}{5}n_4$, then $\frac{3\Delta}{2} + \frac{41}{5}n_1 + \frac{47}{10}n_2 + \frac{3}{2}n_3$, then $\frac{15\Delta}{8} + \frac{253}{40}n_1 + \frac{16}{5}n_2 + \frac{3}{8}$, then $\frac{503\Delta}{200} + \frac{497}{200}n_1 + \frac{459}{200} \leq \frac{287}{100}\Delta + \frac{885}{200}$.

If we set $d_1 = d_2 = d_3 = 6 = d_4 = d_5 = d_6 = 6$, the number of colors used is $10n_1 + 8n_2 + 6n_3 + 6n_4 + 6n_5 + 6n_6$ which is at most $\frac{603}{200}\Delta(G, w) + O(1)$.

We could improve the value if we could prove that $c(8, \{1,2,3,4,5,6\}) \leq 14$ but that seems not possible. We can only prove $c(8, \{1,2,3,4,5,6\}) \leq 15$ which gives a bigger number of colors.

For maximum weight 6, we obtain an approximation is with a multiplicative ratio of $\frac{287}{100}$ and an additive constant of $\frac{885}{200}$. □

Theorem 24 implies the following corollary.

Corollary 6. *Let $\mathcal{I}$ be an instance of SA in a binary tree. Let $\mathcal{OPT}$ be the span of $\mathcal{I}$. There are polynomial-time algorithms which find a spectrum assignment for $\mathcal{I}$ with a span less than $\frac{19}{10}\mathcal{OPT} + \frac{8}{5}$, $\frac{59}{27}\mathcal{OPT} + \frac{67}{27}$, $\frac{859}{336}\mathcal{OPT} + \frac{229}{56}$ and $\frac{287}{100}\mathcal{OPT} + \frac{885}{200}$ when the maximum request demand is bounded by 3,4,5 and 6, respectively.*

4.5 Conclusion

We have studied in this chapter the problem of spectrum assignment in tree networks. We have proved that SA is NP-complete in undirected stars with 3 links and directed stars with 4 links. We have also shown that there is a 4-approximation algorithm to solve the problem in general stars. Afterwards, we have focused on SA in binary trees with special demand profiles and we have designed constant approximation algorithms for several cases. As future work, we would like to find approximation algorithms for interval coloring in chordal graphs in general and to SA in binary trees in particular. Towards this objective, we believe the following directions might be useful.

- It would be interesting to try to use the clique graph of the chordal graph [GHP95] to find an acyclic orientation where the number of maximal cliques to which a path belongs is bounded. In fact, finding a k-approximation for

interval coloring is equivalent to finding an acyclic orientation in which the longest directed path has vertices in at most k maximal cliques [Gol04]. This approach has been used to find a 2-approximation for interval coloring in claw-free chordal graphs [CDG02].

- It would be also helpful to try to use ideas from the approximation algorithms used for interval coloring in interval graphs. These algorithms were developed for the problem of Dynamic Storage Allocation (DSA) as we mentioned in Section 4.2.2.2 and they use mainly three techniques.
  - 2-coloring (2-allocation) [Ger99]: in this technique, which yields a 3-approximation for Interval Coloring (IC) in interval graphs, first, a 2-coloring is found where 2 adjacent vertices but not three might use the same color. This 2-coloring is transformed afterwards to a normal coloring. Is it possible to find a 2-coloring for chordal graphs in polynomial time? Knowing that if we directly apply the technique used by Gergov [Ger99] to find a 2-coloring for an interval graph to chordal graphs, we will have instances where a color is shared by an unbounded number of adjacent vertices.
  - Boxing vertices [BKK+04]: in this technique, which yields a $2+\varepsilon$ approximation for IC in interval graphs, vertices are modeled as rectangles (the dimensions of a rectangle corresponding to a vertex v are the weight of v and the interval corresponding to v in the interval representation of the graph). These rectangles are cleverly boxed or gathered in larger rectangles. Afterwards an exact algorithm is used to color these large rectangles. Is it possible to adapt such technique to chordal graphs and find a clever way to box the vertices?
  - Buddy-decreasing-size algorithm [CS88]: in this algorithm, which yields a 6-approximation for IC in interval graphs, vertices are colored in the decreasing order of their weights. Some of the challenges in this direction is that using it as it is for chordal graphs cannot give better than a $log(n)$-approximation; there is a tight example in [PPR05]. In the tight example however all the vertices have the same weight which means that there is an exponential number of possible orders. Is there a clever order (something similar to lexicographic order?) which can give a better approximation ratio?

Chapter 5

On Dynamic Routing and Spectrum Assignment with Push-Pull

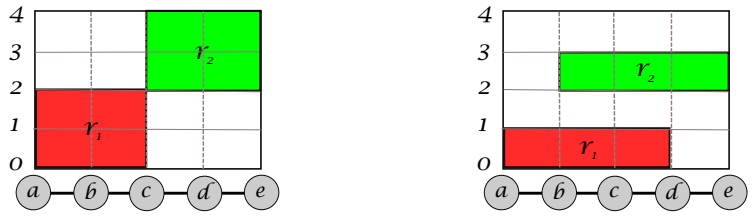
Contents

In this chapter, we study the problem of dynamic Routing and Spectrum Assignment with the use of a non-disruptive technique called Push-Pull to deal with spectrum fragmentation. Results in this chapter are joint work with D. Coudert and B. Jaumard and have been presented in *Algotel'2014* [CJM14].

5.1 Introduction

Spectrum fragmentation [WLV13] in Elastic Optical Networks is one of the issues that are subject to significant research in the optical community. As connections are set up and torn down in the network, the optical spectrum becomes fragmented as the hard disk of a computer does. Small free fragments of spectrum are accumulated and might become unusable either because they are not well aligned on the links of a given path or because they are not contiguous. The spectrum fragmentation caused

by the misalignment of free spectrum fragments is due to the continuity constraint and it was a problem even in the traditional optical networks; it is referred to as horizontal fragmentation. On the other hand, the fragmentation caused by the non-contiguity was introduced in elastic optical networks and is mainly due to the contiguity constraint; it is referred to as vertical fragmentation. To illustrate these types of fragmentation, let us consider a simple path network of 4 nodes, and an optical spectrum of 4 slots. In the first example in Figure 5.1a, request r_1 from a to c is using the spectrum interval $]0, 2]$ and request r_2 from c to e is using the spectrum interval $]2, 4]$. If a new request r requiring two spectrum slots from a to e arrives, it cannot be provisioned even though 2 contiguous spectrum slots are available on each link of the path from a to e. This is because the spectrum is horizontally fragmented and the continuity constraint cannot be satisfied. In the second example in Figure 5.1b, request r_1 from a to d is using the spectrum interval $]0, 1]$ and request r_2 from b to e is using the spectrum interval $]2, 3]$. If a new request r requiring two spectrum slots from a to e arrives, it cannot be provisioned even though the same 2 spectrum slots are available on each link of the path from a to e. This is because the spectrum is vertically fragmented and the contiguity constraint cannot be satisfied.



(a) Horizontal fragmentation (b) Vertical fragmentation

Figure 5.1: Fragmentation in EONs

To address the issue of fragmentation, many techniques have been proposed. These techniques are either preventive or remedial. The preventive techniques are used to avoid or limit the fragmentation and they consist in general in using fragmentation-aware routing and spectrum allocation algorithms. These are algorithms which allocate routes and spectrum to requests in a way to give new requests more chances to be accepted. For instance, in a preventive technique proposed in [WM14], the spectrum is partitioned and requests of almost the same demand are provisioned on the same partition, and we have seen in Chapter 4 that, for spectrum assignment in an offline setting, this partition technique achieves good approximation ratios in some types of networks.

The remedial techniques, on the other hand, offer a cure to fragmentation after it happens and they perform a defragmentation of the spectrum, which is a consolidation of the small fragments of free spectrum. To defragment the spectrum, rerouting and spectrum re-allocation of the already established requests are usually performed. This can cause some disruption to the traffic or can be done seamlessly

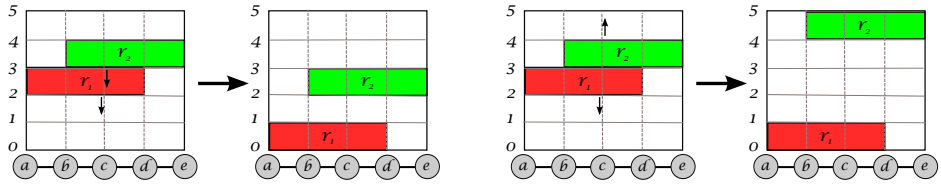
without causing any disruption in the network. The Make-before-Break technique [ABG+01], is one of the non-disruptive defragmentation techniques. In the Make-before-Break, to reroute or re-allocate spectrum to a request, a new connection is established from the source to the destination, the traffic is switched to this new connection and then the first connection is torn down. To establish the second connection, additional costly transponders are needed. Lately, a new non-disruptive technique, which does not require additional transponders, has been proposed. This technique uses the characteristics of the EONs and of the flexible transponders to shift the spectrum allocated to a request without disrupting the traffic. It is referred to as Push-Pull (PP) [CSS+12] and it performs as follows. The frequency at the transmitter is pushed at a certain frequency sweep rate and since at the receiver, the difference between the frequencies of the transmitter and the receiver should not exceed a given offset, the frequency at the receiver is pulled. This technique can only perform spectrum re-allocation and not re-routing. Furthermore, the order of the requests in the spectrum interval does not change. This means that if requests r and r' occupy on a link ℓ slots λ and λ', respectively, such that $\lambda < \lambda'$, then under any shifting with Push-Pull the new slots β and β' occupied by the requests r and r', respectively, are such that $\beta < \beta'$. All in all, with Push-Pull, requests are shifted in the spectrum, a request does not change its path and does not transgress other requests. We study in this part, spectrum defragmentation in EONs using Push-Pull.

Related work. The Push-Pull technique has been first introduced in [CSS+12]. It is considered to be cheap and non-disruptive because it does not require additional transponders and does not disrupt the optical layer by requiring channel re-equalizations [CSS+12]. To prove its feasibility, experimentations have been conducted with Push-Pull in [CPM+13] and with what seems to be a similar technique in [SAX14]. Moreover, algorithms have been proposed in [WM13] to use Push-Pull in the defragmentation of networks that are not heavily loaded where it performs quite well.

In [WM13], proactive and reactive algorithms are designed to defragment networks using non-disruptive defragmentation. The proactive algorithms are run periodically to defragment the spectrum and consist in general in shifting all requests to one end of the spectrum. In the reactive algorithms, the defragmentation is performed only when a request is blocked and already provisioned requests are shifted in order to create enough free contiguous spectrum for the blocked request.

Simulations in [WM13] showed that reactive strategies perform better than the proactive ones in terms of the blocking propability. Particularly, the proposed reactive optimal algorithm has the best performance. The algorithm routes and allocates spectrum to a new request in an EON, using the non-disruptive defragmentation Push-Pull. It first pre-computes a set of paths for the request. Then, it finds on each path a position in the spectrum which minimizes the delay. The delay of insertion of a new request using Push-Pull indicates the duration of the shifting done

to free the needed space. In [WM13], the authors take the number of slots through which the shifting is done over as an indicator of the delay and consider two types of parallelism to compute it as illustrated in Figure 5.2.



(a) Requests r_1 and r_2 are both shifted in the same direction by two slots and one slot, respectively. The delay of shifting is $\delta = \max(2, 1) = 2$

(b) Request r_1 is shifted by two slots and r_2 is shifted in the opposite direction by one slot. The delay of shifting is $\delta = \max(2, 1) = 2$

Figure 5.2: Shifting requests with Push-Pull

Contribution. In this chapter, we study the problem of dynamic and spectrum assignment in an online setting where requests arrive and leave dynamically and the Push-Pull defragmentation can be used. Given an optical network and already established requests, a new request arrives, we aim at deciding whether we can route and allocate spectrum to the new request knowing that the Push-Pull defragmentation can be used and respecting the continuity, contiguity and non-overlapping constraints. We first suppose that the routing is fixed and study the spectrum allocation and then we focus on the problem where we need to route and allocate spectrum. Our contribution is as follows.

- We introduce a graph-theoretic framework to manage requests and Push-Pull shiftings in the network (Section 5.2).
- We build on the algorithm of Wang and Mukherjee [WM13] to solve SA with PP with minimum delay (Section 5.3).
- We solve exactly the routing and spectrum assignment with two different objectives; shortest path and minimum delay. Both algorithms are pseudo-polynomial, i.e. polynomial in the size of the input and the numerical value of the available spectrum (Section 5.4).

5.2 Preliminaries

In this section, we give some definitions and notations used throughout the chapter. Afterwards, we give the graph model we use to manage the shiftings of the requests. Finally, we state the problems we tackle in the chapter.

5.2.1 Definitions and notations

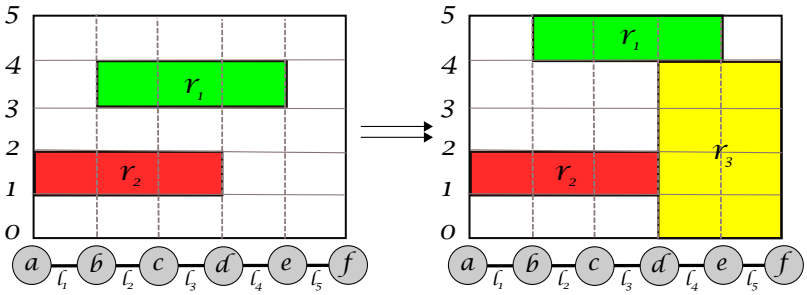
Let $\mathcal{N} = (N, L)$ be a graph modeling an optical network, N is the set of nodes and L is the set of links. The available spectrum on each link is S; the spectrum is slotted into S slots. Slot i, for $i \in \{1, \ldots, S\}$, represents the spectrum interval $]i-1, i]$. Let $\mathcal{R}$ be a set of established requests where each request $r \in \mathcal{R}$ has a path P_r in $\mathcal{N}$, a spectrum demand d_r (number of slots) and an interval $]b(r), e(r)]$ in the spectrum such that $e(r) = b(r) + d_r$. We refer to $b(r)$ as the beginning slot of r and $e(r)$ as the ending slot of r. Note that the non-overlapping, continuity and contiguity constraints are all satisfied. With respect to the possible shiftings of the requests, two network states have been defined in [WM13], the delta state and the del state.

- The delta state ($\triangle$ state) is the state of the network after shifting all the requests down (towards 0) until they are blocked.
- The del state ($\triangledown$ state) is the state of the network after shifting all the requests up (towards S) until they are blocked.

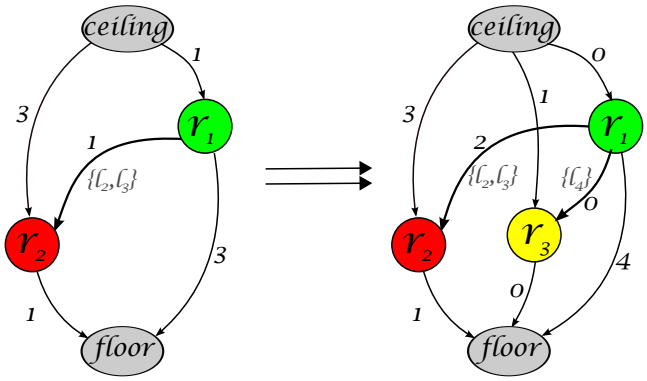
In these states we define $b_{\triangle}(r)/e_{\triangle}(r)$ and $b_{\triangledown}(r)/e_{\triangledown}(r)$ which are the beginning/ending spectrum slots of request r in the $\triangle$ state (i.e. when all existing requests are shifted to their lowest spectrum) and the $\triangledown$ state (i.e. when all existing requests are shifted to their highest spectrum), respectively.

5.2.2 Dependency DAG

We introduce in this section the model used to capture the dependency between the requests and the possible shiftings in the spectrum. This model is *the dependency DAG* and it is similar to the conflict graph we used in Chapter 4: vertices correspond to requests and two vertices are linked by an arc if they are routed through paths which share at least one link. In more details, the spectrum dependency graph $D = (V, E)$, is a directed acyclic graph with two special vertices $floor$ and $ceiling$ and where each vertex r of $V \setminus \{floor, ceiling\}$ is associated with a request $r \in \mathcal{R}$. For each vertex r, there exists an arc from $ceiling$ to r with weight $S - e(r)$ and an arc from r to the floor with weight $b(r)$. There exists an arc from r to r' if r and r' occupy consecutive spectrum intervals on some link $\ell \in L$ and $b(r) \geq e(r')$; this means that both r and r' use link ℓ, $b(r) \geq e(r')$, and no slot between $b(r)$ and $e(r')$ is used by another request on ℓ. The arc (r, r') has a weight equal to $b(r) - e(r')$ and a set of colors (labels) equal to the set of links in L used by both the paths of r and r' and where the requests are consecutive on the spectrum band. The example in Figure 5.3 illustrates how to build the dependency DAG. Note that we keep labels on arcs in order to keep track of the origin of the dependency. This is useful when we delete nodes since it helps establishing the dependencies between the in-neighbors and out-neighbors of the deleted node. We can restrain from using labels, but in that case we will need to have much more arcs in the DAG.



(a) Example



(b) Corresponding dependency DAG

Figure 5.3: Dependency DAG

The dependency DAG is used to capture the dependency between the requests and to compute the possible shiftings. It can be updated in case of shiftings by changing the weights of the arcs, and in case of arrival or departure of requests by adding or deleting nodes and updating the weights and labels of the arcs (see Figure 5.3 for a simple example). Moreover, the dependency DAG is used to compute for every request r the values of $e_{\triangledown}(r)$ and $b_{\triangle}(r)$ which are related respectively to the length of the shortest path from the *ceiling* to r and the length of the shortest path from r to the *floor*. Expressly, if α_1 and α_2 are the length of the shortest path from the *ceiling* to r and the length of the shortest path from r to the *floor*, respectively, then $e_{\triangledown}(r) = e(r) + \alpha_1$ and $b_{\triangle}(r) = b(r) - \alpha_2$.

The dependency DAG also helps keeping track of the dependency between the requests. In particular, we say that a request r_i is constrained to be below another request r_j if on the dependency graph, r_i is in a path from r_j to the *floor* (i.e. r_i is a successor of r_j). In the spectrum, the position of r_i cannot be bigger than the position of r_j, under any shifting, particularly, $e_{\triangle}(r_i) \leq b_{\triangle}(r_j)$ and $e_{\triangledown}(r_i) \leq b_{\triangledown}(r_j)$. A request r_i is constrained to be above another request r_j if on the dependency graph, r_i is in a path from the *ceiling* to r_j (i.e. r_i is a predecessor of r_j). In the spectrum, the position of r_i cannot be smaller than the position of r_j, under any shifting, particularly, $e_{\triangle}(r_j) \leq b_{\triangle}(r_i)$ and $e_{\triangledown}(r_j) \leq b_{\triangledown}(r_i)$.

5.2.3 Problems Statement

We have introduced the definitions and the notations we need and the model we use to keep track of the dependency between the requests. Now, in this section, we formally state the problems we consider using Push-Pull. Given an optical network and a set of provisioned requests. We consider two problems. The first problem is spectrum assignment using Push-Pull. In this problem, a new request arrives with a predefined path and the objective is to allocate spectrum to this request respecting all of the constraints and knowing that the spectrum can be defragmented using Push-Pull. The second problem we consider is more general. Given a new request with no predefined path, we would like to route and allocate spectrum to this new request. We recall that with Push-Pull, requests can be shifted in the spectrum but their order stays the same. We formally state the problems as follows.

Problem 11 (SA-PP)**.** *Given a network, a set of provisioned requests $\mathcal{R}$, and a new request q with demand d_q and path P_q. Is it possible to assign spectrum to q, knowing that only Push-Pull defragmentation can be used?*

Problem 12 (RSA-PP)**.** *Given a network, a set of provisioned requests $\mathcal{R}$, and a new request q with demand d_q, source s and destination t. Is it possible to route and assign spectrum to q, knowing that only Push-Pull defragmentation can be used?*

5.3 Spectrum Assignment with Push-Pull

It has been proved in [WM13], that SA-PP can be solved in polynomial time. We prove here that SA-PP with minimum delay can be solved in polynomial time. We will first explain the polynomial algorithm presented in [WM13], show that it does not necessarily allocate spectrum with minimum delay, and then state and prove our result.

5.3.1 Algorithm of Wang and Mukherjee [WM13]

Let us first present some definitions and notations borrowed from [WM13]. Let q be a new request and P_q its path. We call the conflict set, $CS(P_q)$, the set of provisioned requests that use paths sharing some links with P_q. We set $k = |CS(P_q)|$. If provisioned, the new request q will partition the set $CS(P_q)$ into two subsets, requests above q and requests below q, i.e., every position in the spectrum of q corresponds to a partition $A \cup \bar{A}$ of $CS(P_q)$.

A partition $A \cup \bar{A}$ gives the largest free interval when we shift the requests that are above (i.e. A) to the $\triangledown$ state and those below (i.e., $\bar{A}$) to $\triangle$ state. We define the floors of a position $\alpha = (A, \bar{A})$ before and after defragmentation as $f(\alpha) = max\{e(r) : r \in A\}$ and $f^*(\alpha) = max\{e_{\triangle}(r) : r \in A\}$, respectively. Similarly, we define the ceilings before and after defragmentation of the position α as $c(\alpha) = min\{b(r) : r \in \bar{A}\}$ and $c^*(\alpha) = min\{b_{\triangledown}(r) : r \in \bar{A}\}$, respectively. The widths of a position $\alpha = (A, \bar{A})$ before and after defragmentation are given by $w(\alpha) = c(\alpha) - f(\alpha)$ and $w^*(\alpha) =$

$c^*(\alpha) - f^*(\alpha)$, respectively. To check if we can provision request q on a position $\alpha = (A, \bar{A})$, it is enough to check if $w^*(\alpha) \geq d_r$.

If $w^*(\alpha) \geq d_r$, then if $w(\alpha) \geq d_r$, the minimum delay of insertion in position α is null. Otherwise, if $w(\alpha) < d_r$, this delay is given by the following formula (taking into account the types of parallelism presented in Figure 5.2):

$$\boldsymbol{\delta(\alpha) = d_r - w(\alpha) - \min\left\{f(\alpha) - f^*(\alpha), c^*(\alpha) - c(\alpha), \frac{d_r - w(\alpha)}{2}\right\}} \quad (1)$$

Let $CS(P_q)$ be sorted as $< r_1, r_2, \ldots, r_k >$ in the ascending order of the requests ending slots in the $\triangle$ state, i.e., $e_\triangle(r_1) \leq e_\triangle(r_2) \leq \cdots \leq e_\triangle(r_k)$. We define the *decision-positions* of q over path P_q as the $k+1$ positions α_i such that $\alpha_0 = (\emptyset, \{r_1, \ldots, r_k\})$ and $\alpha_i = (\{r_1, \ldots, r_i\}, \{r_{i+1}, \ldots, r_k\})$, for $i \in \{1, \ldots, k\}$. Wang and Mukherjee have proved that to decide if it is possible to allocate spectrum to q on path P_q, it is sufficient to check the $k+1$ decision-positions α_i, $i \in \{0, \ldots, k\}$, and that the following theorem holds.

Theorem 25. *[WM13] Let q be a request of demand d_q and path P_q, $|CS(P_q)| = k$ and α_i, $i \in \{0, \ldots, k\}$ the corresponding decision-positions. Request q is provisionable over P_q using Push-Pull, if and only if there exists some $i \in \{0, \ldots, k\}$, such that $w^*(\alpha_i) \geq d_q$.*

Thanks to Theorem 25, to solve the SA-PP problem, Wang and Mukherjee have designed an algorithm which checks all of the decision-positions then chooses the one over which the new request can be provisioned with minimum delay. However, we show with the following example that the returned position does not minimize the delay over all possible positions.

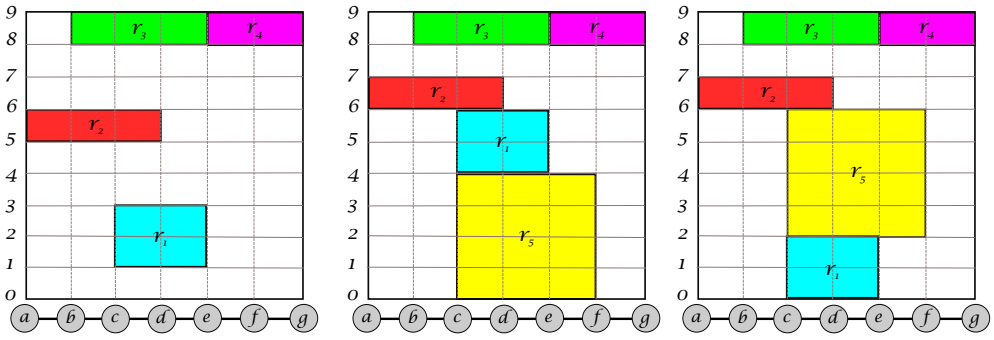
5.3.2 Example

Let us consider the example presented in Figure 5.4a. We have a path network of length 6 with an available spectrum of 9 slots and 4 requests: r_1 from a to d occupying the spectrum interval $]5, 6]$, r_2 from c to e occupying the spectrum interval $]1, 3]$, r_3 from b to e occupying the spectrum interval $]8, 9]$, and r_4 from e to g occupying the spectrum interval $]8, 9]$. Suppose that a new request r_5 from c to f requiring 4 spectrum slots arrives and needs to be provisioned. This request is conflicting with r_1, r_2, r_3 and r_4. To allocate spectrum to r_5, the algorithm of Mukherjee and Wang will first sort the existing requests in the increasing order of their spectrum occupancy $e_\triangle(r)$. The obtained order is $< r_4, r_1, r_2, r_3 >$ and the five possible decision-positions according to this order are:

- $(\emptyset, \{r_4, r_1, r_2, r_3\})$; below $\{r_4, r_1, r_2, r_3\}$, with delay 3
- $(\{r_4\}, \{r_1, r_2, r_3\})$; above r_4 and below $\{r_1, r_2, r_3\}$ with delay 8
- $(\{r_4, r_1\}, \{r_2, r_3\})$; above $\{r_4, r_1\}$ and below $\{r_2, r_3\}$ with delay 6

- $(\{r_4, r_1, r_2\}, \{r_3\})$; above $\{r_4, r_1, r_2\}$ and below r_3 with delay 5
- $(\{r_4, r_1, r_2, r_3\}, \emptyset)$; above $\{r_4, r_1, r_2, r_3\}$ with delay 4

Among these positions, the one with minimum delay is $(\emptyset, \{r_4, r_1, r_2, r_3\})$ which is illustrated in Figure 5.4b. The position with minimum delay over all possible positions is, however, above r_1 and below $\{r_2, r_3, r_4\}$ with delay 1 as illustrated in Figure 5.4c.



(a) (b) (c)

Figure 5.4: Spectrum assignment to a request with Push-Pull

5.3.3 SA-PP with minimum delay

In this section, we prove that by checking more positions, we can find a position which minimizes the delay. We prove the following theorem.

Theorem 26. *The SA-PP problem with minimum delay can be solved in polynomial time.*

Let q be the new request, P_q its path, and $|CS(P_q)| = k$. Instead of checking only the $k+1$ decision-positions α_i, $i \in \{0, \ldots, k\}$, defined earlier, we check $O(k^2)$ relative positions α_i^j, $i \in \{0, \ldots, k\}$, $j \in \{0, \ldots, i-1\}$, defined as follows. For $i \in \{1, \ldots, k\}$, let $\alpha_i = (A_i, \bar{A}_i)$ and let $< x_1, \ldots, x_i >$ be the set A_i sorted in the descending order of their ending slots, i.e. $e(x_1) \geq e(x_2) \geq \cdots \geq e(x_i)$. For $j = 0$, we set $\alpha_i^j = \alpha_i$ and for $j \in \{1, \ldots, i-1\}$, we set $B_j = \{x_1, x_2, \ldots, x_j\}$ and we define the position $\alpha_i^j = (A_i \setminus B_j, \bar{A}_i \cup B_j)$. Algorithm 2 (page 119) is the algorithm solving SA with minimum delay. It checks if it is possible to empty spectrum for the new request q with the use of one of the positions α_i^j, and then chooses the position with the minimum delay. Its correctness is drawn from Lemma 4.

Lemma 4. *Request q is provisionable over P_q with delay δ using Push-Pull, if and only if there exists some $i \in \{0, \ldots, n\}$, and $j \in \{0, \ldots, i-1\}$ such that $w^*(\alpha_i^j) \geq d_q$ and $\delta(\alpha_i^j) \leq \delta$.*

Proof. Let us suppose that a request q can be provisioned on path P_q with position $\beta = (B, \bar{B})$ (i.e. in the spectrum interval, q is above the requests in B and below the requests of $\bar{B}$) and delay $\delta(\beta) = \delta$.

In the proof of Theorem 25, β can be transformed into a decision-position $\alpha_i = (A, \bar{A})$, $i \in \{0, \ldots, n\}$ such that q can be provisioned on α_i. The position α_i is found as follows. Suppose r is the request that yields the floor of $\beta = (B, \bar{B})$ after defragmentation, i.e. $e_{\triangle}(r) = f^*(\beta)$. Let $< r_1, r_2, \ldots, r_n >$ be the requests in $CS(P_q)$ sorted in the ascending order of $\triangle$ state, i.e., $e_{\triangle}(r_1) \leq e_{\triangle}(r_2) \leq \cdots \leq e_{\triangle}(r_n)$ and let i be the index such that $e_{\triangle}(r_i) = e_{\triangle}(r)$ and $e_{\triangle}(r_{i+1}) > e_{\triangle}(r)$. It is proved in Theorem 25 that if q is provisionable on path P_q with position $\beta = (B, \bar{B})$ then q is provisionable on path P_q with position $\alpha_i = (A, \bar{A})$ where $A = \{r_1, \ldots, r_i\}$.

Note that the position α_i is obtained by shifting down some requests of $\bar{B}$, and that $B \subseteq A$ and $\bar{A} \subseteq \bar{B}$. These shiftings may affect the delay δ (see formula of delay in equation (1) page 116). The idea is to shift back up some of the requests that might change the delay which are the requests x such that $e(x) > e(r')$ where r' is the request that yields the floor of β, i.e., $f(\beta) = e(r')$.

Let $< x_1, x_2, \ldots, x_i >$ be the list of requests of A sorted in the descending order of their ending slots, i.e., $e(x_1) \geq e(x_2) \geq \cdots \geq e(x_i)$. Let x_j be the request such that $e(x_{j+1}) = e(r')$ and $e(x_j) > e(r')$. We prove that $\alpha_i^j = (A \setminus \{x_1, x_2, \ldots, x_j\}, \bar{A} \cup \{x_1, x_2, \ldots, x_j\})$ is a position which can provision q with delay $\delta(\alpha_i^j) \leq \delta$.

We first set $\alpha = \alpha_i^j$ to simplify the notation. For positions α and β, we have the following.

- $c^*(\alpha) \geq c^*(\beta)$; this is due to the fact that $\bar{A} \cup \{x_1, x_2, \ldots, x_j\} \subseteq \bar{B}$.
- $f^*(\alpha) \leq f^*(\beta)$; this is due to the fact that $f^*(\alpha) \leq f^*(\alpha_i)$ (since $A \setminus \{x_1, x_2, \ldots, x_j\} \subseteq A$) and α_i and β have the same floor after defragmentation (by construction of α_i).
- $f(\alpha) = f(\beta)$; this is due to the fact that $f(\beta) = e(r')$, $f(\alpha) = e(x_{j+1})$ and $e(r') = e(x_{j+1})$.
- $c(\alpha) \geq c(\beta)$; this is due to the fact that $\bar{A} \cup \{x_1, x_2, \ldots, x_j\} \subseteq \bar{B}$.

Since $w^*(\alpha) = c^*(\alpha) - f^*(\alpha)$ and with the equations above, we deduce that $w^*(\alpha) \geq c^*(\beta) - f^*(\beta)$, and hence $w^*(\alpha) \geq w^*(\beta) \geq d_q$. This implies that α can provision q. The equations above also imply that $w(\alpha) \geq w(\beta)$.

Now, let us prove that $\delta' = \delta(\alpha) \leq \delta$. Note that if $\delta = 0$, then $w(\beta) \geq d_q$. This implies that $w(\alpha) \geq d_q$ and $\delta' = 0$. Let us assume that both δ and δ' are not null. We recall that

$$\delta = d_r - w(\beta) - \min\left\{f(\beta) - f^*(\beta), c^*(\beta) - c(\beta), \frac{d_r - w(\beta)}{2}\right\}$$

and

$$\delta' = d_r - w(\alpha) - \min\left\{f(\alpha) - f^*(\alpha), c^*(\alpha) - c(\alpha), \frac{d_r - w(\alpha)}{2}\right\}$$

Let us evaluate the difference between δ and δ' according to the following cases:

- If $\min\left\{f(\alpha)-f^*(\alpha), c^*(\alpha)-c(\alpha), \frac{d_r-w(\alpha)}{2}\right\} = f(\alpha)-f^*(\alpha)$, then $\delta' = d_r - w(\alpha)-(f(\alpha)-f^*(\alpha))$, and since $w(\alpha) \geq w(\beta)$ and $f(\alpha)-f^*(\alpha) \geq f(\beta)-f^*(\beta)$, this implies that $\delta' \leq d_r - w(\beta) - (f(\beta)-f^*(\beta)) \leq \delta$.
- If $\min\left\{f(\alpha)-f^*(\alpha), c^*(\alpha)-c(\alpha), \frac{d_r-w(\alpha)}{2}\right\} = c^*(\alpha)-c(\alpha)$, then $\delta' = d_r - w(\alpha) - (c^*(\alpha)-c(\alpha))$. In this case, since $w(\alpha)-c(\alpha) = f(\alpha) = f(\beta) = w(\beta)-c(\beta)$, then $\delta' = d_r - w(\beta) + c(\beta) - c^*(\alpha)$ and since $c^*(\alpha) \geq c^*(\beta)$ we have $\delta' \leq d_r - w(\beta) + c(\beta) - c^*(\beta) \leq \delta$.
- If $\min\left\{f(\alpha)-f^*(\alpha), c^*(\alpha)-c(\alpha), \frac{d_r-w(\alpha)}{2}\right\} = \frac{d_r-w(\alpha)}{2}$, then $\delta' = \frac{d_r-w(\alpha)}{2}$ and since $w(\alpha) \geq w(\beta)$, we have $\delta' \leq \delta$.

□

Algorithm 2: SA-PP with minimum delay

Require: Network $\mathcal{N} = (N, L)$, a set of provisioned requests $\mathcal{R}$ and a new request q with a path P

Ensure: Allocate spectrum to q on P with minimum delay

1: $b(q) = None$, $\beta = \emptyset$ and $\delta = \infty$
2: Find $CS(P)$ the set of requests conflicting with q on P and sort it in the ascending order of $e_{\triangle}$. The sorted list is $< r_1, r_2, \ldots, r_k >$. The corresponding decision-positions are $\alpha_0, \alpha_1, \ldots, \alpha_n$ such that $\alpha_i = (A_i, \bar{A}_i)$
3: **for all** $i \in 0, \ldots, n$ **do**
4: **if** $w^*(\alpha_i) \geq d_q$ **then**
5: Sort the requests in A_i in the descending order of $e(x)$. The sorted list is $< x_1, x_2, \ldots, x_i >$ and $\alpha_i^j = (\{A_i \setminus \{x_1, x_2, \ldots, x_j\}\} \cup \{\bar{A}_i, \{x_1, x_2, \ldots, x_j\}\})$ for $j \in \{1, \ldots, i-1\}$ and $\alpha_i^0 = \alpha_i$
6: **for all** $j \in \{0, \ldots, i-1\}$ **do**
7: **if** $\delta(\alpha_i^j) < \delta$ **then**
8: $\beta = \alpha_i^j$ and $\delta = \delta(\alpha_i^j)$ and $b(q) = \max(f(\beta) - \delta, f^*(\beta))$

5.4 Routing and Spectrum Assignment with Push-Pull

In this section we solve the problem of RSA-PP. We first solve RSA-PP with the shortest path and then solve RSA-PP with minimum delay.

Let q be a request from s to t with spectrum demand d_q. We call *absolute position*, a position in the spectrum range i.e., a value in the interval $[0, S]$ which can be assigned to $b(q)$. We call a *relative position* (A, B), a position between two sets of requests. Allocating position (A, B) to a request r means that request r is above the set of requests A, and below the set of requests B in the spectrum range.

We say that (A, B) is a *valid* relative position if no request of B is constrained to be below a request of A. A relative position (A, B) is *valid* on a link ℓ if it is valid and (A, B) is a partition of the requests using ℓ.

To every relative position (A, B), we associate a *complete* relative position (A_c, B_c) such that A_c contains the requests in A and all the requests constrained to be below them and B_c contains the requests in B and all the requests constrained to be above them. We recall that requests constrained to be above or below a given request r are respectively the requests corresponding to the predecessors and the successors of the vertex r in the dependency DAG.

We say that two relative positions (A, B) and (C, D) are *conflicting* if and only if $A_c \cap D_c \neq \emptyset$ or $C_c \cap B_c \neq \emptyset$.

5.4.1 RSA-PP with the shortest path

Theorem 27. *The RSA-PP problem with shortest path can be solved in pseudo-polynomial time.*

Algorithm 3 is the algorithm we use to solve RSA-PP with shortest path. It is a natural approach to such problems. Since there are $S - d_q + 1$ absolute positions on the spectrum to allocate to request q, then for each absolute position λ, we create the graph consisting of all of the links of $\mathcal{N}$ where the position λ can be freed for request q (by shiftings) and then find the shortest path from s to t in this graph. At the end we choose the shortest path among the ones found for each position and on this path we allocate spectrum with minimum delay.

The correctness of the algorithm is based on the fact that if we can free a position λ on a set of links (i.e. free all slots of $]\lambda, \lambda + d_q]$), then we can free this position on the same set using non-conflicting shiftings or in other words non-conflicting relative positions on the links. This is what we prove with Lemma 5. Note that freeing position λ on link ℓ using the relative position (A, B) means that (A, B) is a valid relative position on ℓ, and requests of A can be shifted below λ and requests of B can be shifted above $\lambda + d_q$.

Lemma 5. *If the absolute position λ can be freed for a request q with demand d_q on a set of links E, then there are valid non-conflicting relative positions on the links of E which can free λ for q.*

Proof. We proceed by induction on the size of E. Let $E = \{\ell, \ell'\}$ and let (A, B) and (A', B') be the two relative positions with which ℓ and ℓ' free λ, respectively. Let (A_c, B_c) and (A'_c, B'_c) be the complete relative positions corresponding to (A, B) and (A', B') , respectively. Let us suppose that (A, B) and (A', B') are conflicting. We suppose without loss of generality that $A_c \cap B'_c = I \neq \emptyset$. This implies that $A'_c \cap B_c = \emptyset$. Let J be the set of the requests I and all the requests that are constrained to be below them. We prove that (C, D), where $C = (A'_c \cup J) \cap CS(\ell')$ and $D = (B'_c \setminus J) \cap CS(\ell')$, is a valid relative position on link ℓ', that frees position λ and that is not conflicting with (A, B).

Algorithm 3: RSA-PP with shortest path

Require: A network $\mathcal{N} = (V, L)$, a set of provisioned requests $\mathcal{R}$ and a new request q with demand d_q, source s and destination t

Ensure: Route and allocate spectrum to q using the shortest path

1: **for all** $\ell \in L$ **do**
2: sort requests using ℓ by the increasing order of their beginning slots $b(r)$; the sorted list is $< r_1, \ldots, r_k >$.
3: $e_{\triangle}(r_0) = 0$ and $b_{\triangledown}(r_{k+1}) = S$
4: **for all** $\lambda \in \{0, 1, \ldots, S - d_q\}$ **do**
5: **for all** $i \in \{0, 1, \ldots, k\}$ **do**
6: **if** $[\lambda, \lambda + d_q] \subseteq [e_{\triangle}(r_i), b_{\triangledown}(r_{i+1})]$ **then**
7: color link ℓ with color λ and break
8: Find the shortest monocolored st-path P. {a monocolored path is a path whose links share a color}
9: Find a position on P using Algorithm 2

- First, (C, D) is a valid position on ℓ'. $C \cup D = (A'_c \cup B'_c) \cap CS(\ell') = CS(\ell')$ and $C \cap D = A'_c \cap B'_c = \emptyset$, so (C, D) is a partition of $CS(\ell')$. Let r_C be a request of C. If $r_C \in A'$ then r_C is below all the requests in D (this conclusion comes from the validity of the position (A', B')). Otherwise, if $r_C \in J$, we know that all the requests constrained to be below r_C are in the same set J and no one of them can be in $D = (B' \backslash J) \cap CS(\ell')$. Position (C, D) is then valid.

- Second, (C, D) can free λ. This means that if we define as before the floor and ceiling of (C, D) after defragmentation as $f^*((C, D)) = max\{e_{\triangle}(r) : r \in C\}$ and $c^*((C, D)) = min\{b_{\triangledown}(r) : r \in D\}$, respectively, then $f^*((C, D)) \leq \lambda$ and $c^*((C, D)) \geq \lambda + d_q$. In fact, we have $D \subseteq B'$, then $c^*((C, D)) \geq c^*((A', B')) \geq \lambda + d_q$ and we have $f^*((C, D)) \leq max\{e_{\triangle}(r) : r \in J\} \leq f^*((A, B)) \leq \lambda$.

- Third, (C, D) is not conflicting with (A, B). (A_c, B_c) and (C_c, D_c) are the complete relative positions corresponding to (A, B) and (C, D), respectively. We prove that $A_c \cap D_c = \emptyset$ and $B_c \cap C_c = \emptyset$. In fact, $D \subseteq B'_c \backslash J$, which implies that $A_c \cap D_c = \emptyset$. Suppose that $B_c \cap C_c \neq \emptyset$ and let r_s be an element of this intersection. Let C' be the set of requests constrained to be below C. Since $C = (A'_c \cup J) \cap CS(\ell')$ and $(A'_c \cup J) \cap B_c = \emptyset$, then $C \cap B_c = \emptyset$. This implies that $r_s \in C' \cap B_c$. Since $r_s \in B_c$, all the requests constrained to be above it should be also in B_c, among which at least one element of C (since $r_s \in C'$), which is not possible since $C \cap B_c = \emptyset$.

Now suppose it is true for $|E| = n - 1$ and let us prove it for $|E| = n$. Let $E = \{\ell_1, \ldots, \ell_n\}$ be a set of links that can all free absolute position λ. With the induction assumption, considering the set $E' = \{\ell_1, \ldots, \ell_{n-1}\}$, for every $\ell_i \in E'$ there is a relative position (A^i, B^i) that can free λ such that all the relative positions

(A^i, B^i), $i \in \{1, \ldots, n-1\}$, are not conflicting. Let (A^i_c, B^i_c) be the complete relative position corresponding to (A^i, B^i), for $i \in \{1, \ldots, n-1\}$. We set $\mathcal{A} = \bigcup_{i\in\{1,\ldots,n-1\}} A^i_c$ and $\mathcal{B} = \bigcup_{i\in\{1,\ldots,n-1\}} B^i_c$.

We know there is a relative position (A^n, B^n) on ℓ_n corresponding to the absolute position λ, let (A^n_c, B^n_c) be the corresponding complete relative position.

If $\mathcal{A} \cap B^n_c = \emptyset$ and $\mathcal{B} \cap A^n_c = \emptyset$, then (A^n, B^n) is not conflicting with any (A^i, B^i), for $i \in \{1, \ldots, n-1\}$. Otherwise, as we did for the case where $n = 2$, we find a position (C^n, D^n) on ℓ_n not conflicting with $(\mathcal{A}, \mathcal{B})$. With the same arguments, it will be a valid position that can free λ and that is not conflicting with any of the positions (A^i, B^i), for $i \in \{1, \ldots, n-1\}$.

□

5.4.2 RSA-PP with minimum delay

Theorem 28. *The RSA-PP problem with minimum delay can be solved in pseudo-polynomial time.*

Let q be a request with demand d_q, source s and destination t. The delay of freeing position λ on link ℓ for request q with relative position (A, B) is given by the formula: $\delta_\ell(A, B) = \max\{0, e(A)-\lambda, \lambda+d_q-b(B)\}$ where $e(A) = \max\{e(x) : x \in A\}$ and $b(B) = \min\{b(x) : x \in B\}$. The delay of insertion in position λ on a path P is $\delta_P = \max_{\ell\in P} \delta_\ell(A_\ell, B_\ell)$ where (A_ℓ, B_ℓ) is the relative position used to free λ on link ℓ.

Algorithm 4 is the algorithm we use to solve RSA-PP with minimum delay. For each absolute position λ, the algorithm checks for each link ℓ of the network if position λ can be freed on ℓ. The link is then weighted with the minimum delay δ_ℓ with which λ can be freed. Afterwards, a path P_λ from s to t is found. This path should use only links which can free λ, and minimize the value $\delta_{P_\lambda} = \max\{\delta_\ell \mid \ell \in P_\lambda\}$. Among all the paths P_λ, the one with the minimum delay δ_λ is chosen for the routing and the corresponding absolute position λ is chosen for the spectrum assignment. The correctness of the algorithm is drawn from Lemmas 6 and 7.

Lemma 6. *For an absolute position λ and a link ℓ, there are at most two relative positions freeing λ on ℓ with minimum delay and if there are two such positions they are of the form (A, B) and $(A \cup \{x\}, B \setminus \{x\})$, where x is a request using ℓ.*

Proof. We have the following two cases.

- If the minimum delay to free λ on ℓ is 0, then there is exactly one corresponding relative position (A, B). Suppose that there are two relative positions (A, B) and (C, D) freeing λ with delay 0, and suppose without loss of generality that $C = A \cup I$ and $D = B \setminus I$, $I \neq \emptyset$. $\delta_\ell(A, B) = 0$ implies that $b(I) \geq \lambda + d_q$ and then $e(I) > \lambda$. On the other hand, $\delta_\ell(C, D) = 0$ implies that $e(I) \leq \lambda$, which is not possible.

- If the minimum delay to free λ on ℓ is $\delta \neq 0$, then there are at most two relative positions freeing λ with delay δ and they are of the form (A, B) and $(A \cup$

Algorithm 4: RSA-PP with minimum delay

Require: Network $\mathcal{N} = (N, L)$, a set of provisioned requests $\mathcal{R}$ and a new request q with demand d_q, source s and destination t.

Ensure: Route and allocate spectrum to q with minimum delay.

1: $P = \emptyset$, $b(q) = None$ and $\delta = \infty$
2: **for all** $\ell \in L$ **do**
3: $\quad\delta_\ell = \infty$ and sort requests using ℓ by the increasing order of their spectrum occupancy; the sorted list $< r_1, \ldots, r_{j_\ell} >$.
4: **for all** $\lambda \in [\![0, S - d_q]\!]$ **do**
5: $\quad$**for all** $\ell \in L$ **do**
6: $\quad\quad$**for all** $i \in [\![0, j_l]\!]$ **do**
7: $\quad\quad\quad$**if** $[\lambda, \lambda + d_q] \subseteq [e_\triangle(r_i), b_\triangledown(r_{i+1})]$ **then**
8: $\quad\quad\quad\quad$**if** ℓ is not colored with λ **then**
9: $\quad\quad\quad\quad\quad$Color link ℓ with color λ
10: $\quad\quad\quad\quad$**if** $\delta_\ell > \delta_\ell(\{r_1, \ldots, r_i\}, \{r_{i+1}, \ldots, r_{j_\ell}\})$ **then**
11: $\quad\quad\quad\quad\quad\delta_\ell = \delta_\ell(\{r_1, \ldots, r_i\}, \{r_{i+1}, \ldots, r_{j_\ell}\})$
12: $\quad$Find shortest path P_λ from s to t using only links colored with λ and which minimizes $\delta_{P_\lambda} = \max\{\delta_\ell \mid \ell \in P_\lambda\}$
13: $\quad$**if** $\delta_{P_\lambda} < \delta$ **then** $P = P_\lambda$, $\delta = \delta_{P_\lambda}$ and $b(q) = \lambda$

$\{x\}, B \setminus \{x\})$. Let (A, B) and (C, D) be two relative positions corresponding to λ with delay δ. Suppose without loss of generality that $C = A \cup I$ and $D = B \setminus I$, $I \neq \emptyset$. Suppose that $|I| \geq 2$ and let J be a proper subset of I and $(E, F) = (A \cup J, B \setminus J)$. First, recall that $\delta = \max\{e(A) - \lambda, \lambda + d_q - b(B)\} = \max\{e(C) - \lambda, \lambda + d_q - b(D)\}$. Since $A \subseteq C$ and $D \subseteq B$ we have $e(A) - \lambda < e(C) - \lambda$ and $\lambda + d_q - b(D) < \lambda + d_q - b(B)$. Thus, $\delta = \lambda + d_q - b(B) = e(C) - \lambda$. Now, since $E \subseteq C$ and $F \subseteq B$, $e(E) - \lambda < e(C) - \lambda$ and $\lambda + d_q - b(F) < \lambda + d_q - b(B)$. This implies that $\delta_\ell(E, F) < \delta$ which is not possible since δ is the minimum delay. So $|I| \leq 1$.

□

By convention, if two positions (A, B) and $(A \cup \{x\}, B \setminus \{x\})$ free λ on ℓ with minimum delay, we will refer to (A, B) as *the* position minimizing the delay.

Remark 3. *Note that if (A, B) is the position minimizing the delay on ℓ, then any relative position on ℓ obtained by shifting up some requests of A has a delay strictly bigger than the minimum delay.*

Lemma 7. *For an absolute position λ and two links ℓ and ℓ', if (A, B) is the relative position that frees λ on ℓ with minimum delay and (C, D) is the relative position that frees λ on ℓ' with minimum delay, then, (A, B) and (C, D) are not conflicting (i.e. their respective complete relative positions are not conflicting).*

Proof. Let λ be an absolute position and ℓ and ℓ' two links in the network. Let (A, B) and (C, D) be the two positions that free λ with minimum delay on ℓ and ℓ', respectively. Suppose that (A, B) and (C, D) are conflicting and suppose without loss of generality that $A_c \cap D_c = I$. Let J be the set containing the requests in I and all the requests constrained to be below them and K the set containing the requests in I and all the requests constrained to be above them.

As shown in the proof of Lemma 5, $(A', B') = (A \setminus (K \cap CS(\ell)), B \cup (K \cap CS(\ell))$ is a valid relative position on ℓ that frees λ and $(C', D') = (C \cup (J \cap CS(\ell')), D \setminus (J \cap CS(\ell')))$ is a valid relative position on ℓ' that frees λ. We have then $\delta_\ell(A, B) < \delta_\ell(A', B')$ (according to Remark 3) and $\delta_{\ell'}(C, D) \leq \delta_{\ell'}(C', D')$. With respect to the sets A, B', D, and C', we have the following two inequalities.

- $e(A) - \lambda < \lambda + d_q - b(B')$; in fact, $\delta_\ell(A, B) < \delta_\ell(A', B')$ means that $\max(e(A) - \lambda, \lambda + d_q - b(B)) < \max(e(A') - \lambda, \lambda + d_q - b(B'))$, and since $e(A) - \lambda \geq e(A') - \lambda$ (because $A' \subset A$), this implies that $\max(e(A') - \lambda, \lambda + d_q - b(B')) = \lambda + d_q - b(B')$ and then that $e(A) - \lambda < \lambda + d_q - b(B')$.
- $\lambda + d_q - b(D) \leq e(C') - \lambda$; in fact, $\delta_{\ell'}(C, D) \leq \delta_{\ell'}(C', D')$ means that $\max(e(C) - \lambda, \lambda + d_q - b(D)) \leq \max(e(C') - \lambda, \lambda + d_q - b(D'))$, and since $\lambda + d_q - b(D') < \lambda + d_q - b(D)$ (because $D' \subset D$), this implies that $\max(e(C') - \lambda, \lambda + d_q - b(D')) = e(C') - \lambda$ and then that $\lambda + d_q - b(D) \leq e(C') - \lambda$.

Now, we know that $e(C') = e(J \cap CS(\ell'))$, $b(D) = b(J \cap CS(\ell'))$, $b(B') = b(K \cap CS(\ell))$ and $e(A) = e(K \cap CS(\ell))$. The two inequalities above will be then written as:

$$e(K \cap CS(\ell)) - \lambda < \lambda + d_q - b(K \cap CS(\ell)) \tag{2}$$

and

$$\lambda + d_q - b(J \cap CS(\ell')) \leq e(J \cap CS(\ell')) - \lambda \tag{3}$$

Since J contains the requests of I and all the requests constrained to be below them, then $e(J) = e(I)$, which implies that $e(J \cap CS(\ell')) \leq e(I)$. Moreover, for any request $r \in I$, if $r \notin K \cap CS(\ell)$, then $r \notin CS(\ell)$ (r is necessarily in K since $I \subseteq K$). Since $I \subseteq A_c$, then r is constrained to be below a request $r' \in CS(\ell)$ which implies that $e(r) \leq e(r')$. The request r' belongs to $K \cap CS(\ell)$ ($r' \in K$ by definition of K). So, $e(I) \leq e(K \cap CS(\ell))$. This implies that

$$e(J \cap CS(\ell')) \leq e(K \cap CS(\ell)) \tag{4}$$

Since K contains the requests of I and all the requests constrained to be above them, then $b(I) = b(K)$, which implies $b(I) \leq b(K \cap CS(\ell))$. Moreover, for any request $r \in I$, if $r \notin J \cap CS(\ell')$, then $r \notin CS(\ell')$ (r is necessarily in J since $I \subseteq J$). Since $I \subseteq D_c$, then r is constrained to be above a request $r' \in CS(\ell')$ which implies that $b(r) \geq b(r')$. The request r' belongs to $J \cap CS(\ell')$ ($r' \in J$ by definition of J). So, $b(J \cap CS(\ell')) \leq b(I)$. This implies that

$$b(J \cap CS(\ell')) \leq b(K \cap CS(\ell)) \tag{5}$$

Inequalities (2), (3), (4) and (5) give

$$e(K \cap CS(\ell)) - \lambda \geq \lambda + d_q - b(K \cap CS(\ell))$$

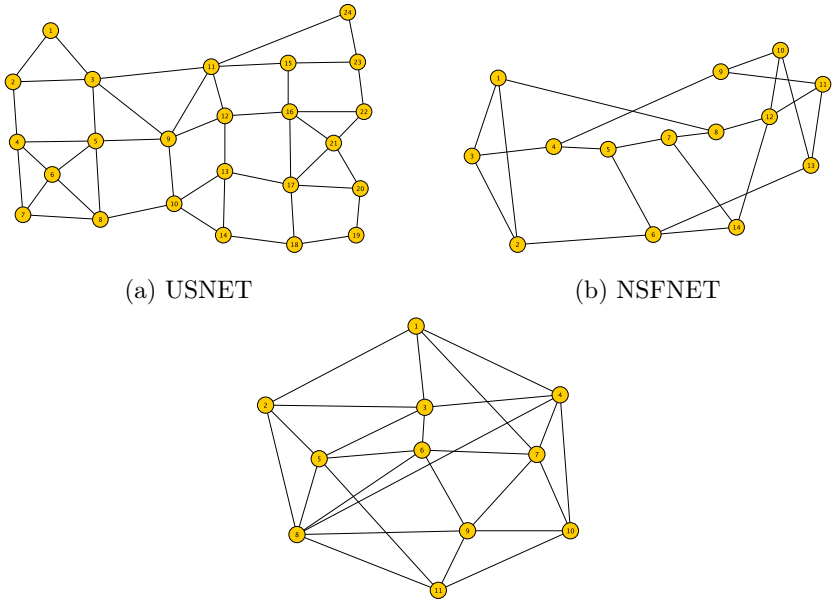
and

$$e(K \cap CS(\ell)) - \lambda < \lambda + d_q - b(K \cap CS(\ell))$$

which is not possible. □

5.5 Simulations

We use settings similar to the ones used in [WM13]. We run our simulations on the following network topologies: USNET (24 nodes and 43 links) in Figure 5.5a, NSFNET (14 nodes and 22 links) in Figure 5.5b, and COST239 (11 nodes and 26 links) in Figure 5.5c.



(a) USNET

(b) NSFNET

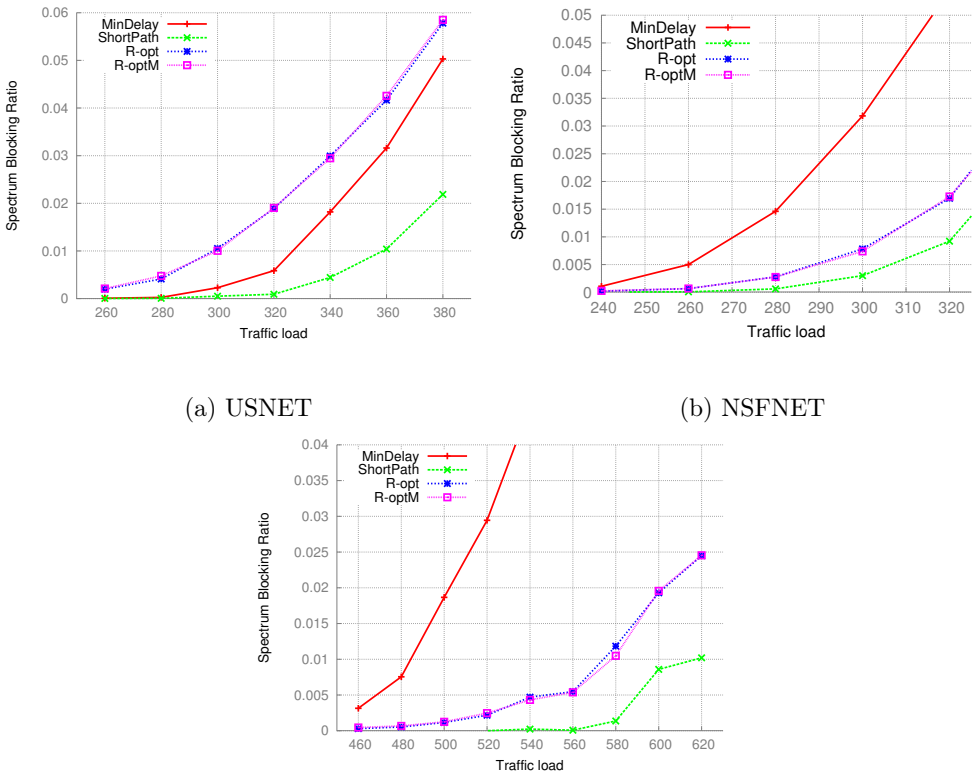
(c) COST239

Figure 5.5: Network topologies

The spectrum is slotted into 380 slots where each slot corresponds to a spectrum interval of width 12.5 GHz. The requests arrive to the network with a Poisson distribution (with arrival rate λ). Each request spends in a network an exponentially distributed duration (with normalized mean $1/\mu = 1$). The offered load is determined by $\rho = \lambda/\mu = \lambda$ (Erlang). For each load, a total of 10^5 requests is simulated. The source and destination are uniformly selected between the nodes of the network. Three types of request demands are considered 100Gb/s , 400Gb/s and 1Tb/s corresponding to 4, 10 and 24 slots respectively. We capture three measurements with

our simulations: the Spectrum Blocking Ratio (SBR) which is equal to the ratio of the spectrum of the blocked requests to the overall spectrum, the Average Delay (AD) which is equal to the ratio of the sum of all delays to the number of accepted requests and the Average Shifted Distance (ASD) which is equal to the ratio of the sum of all performed shiftings to the number of accepted requests.

MinDelay is the exact algorithm which minimizes the delay (Algorithm 4, page 123). *ShortPath* is the exact algorithm which finds the shortest path (Algorithm 3, page 121). *R-opt* is the algorithm proposed by Mukherjee and Wang in [WM13]. In this algorithm, we compute the $k = 3$ shortest paths between each pair of nodes of the network. Afterwards, when a request r arrives, for each path P of the k shortest paths between the source and the destination of r, we solve the spectrum assignment problem on P as explained in Section 5.3.1. Finally, we choose among the k paths, the path and the spectrum assignment with the minimum delay. *R-optM* is an algorithm similar to *R-opt* in which we use Algorithm 2 (page 119) to solve the spectrum allocation problem over a path.



(a) USNET

(b) NSFNET

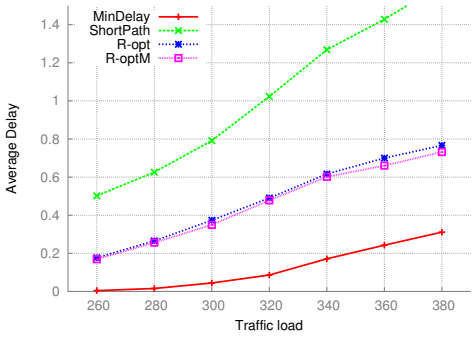
(c) COST239

Figure 5.6: Spectrum Blocking Ratio (SBR)

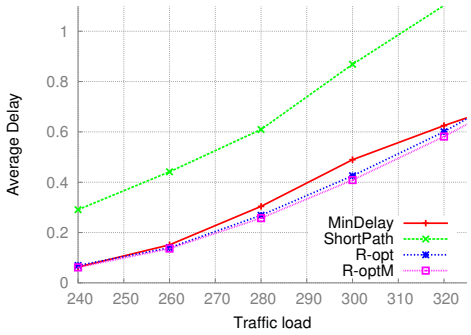
It has already been demonstrated through simulations in [WM13] that *R-opt* performs well in terms of minimizing the blocking probability compared to classical

algorithms using First-Fit and also compared to proactive algorithms using Push-Pull. In what follows, we compare the performance of *R-opt* with our algorithms.

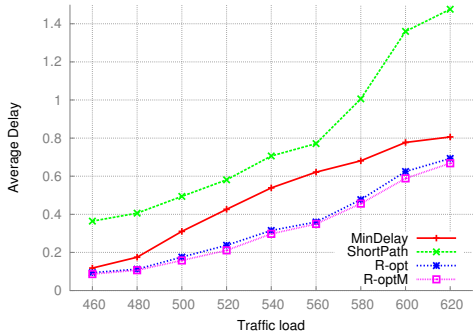
In Figure 5.6, the SBR is plotted as function of the load. We observe that the *Shortpath* algorithm gives the minimum SBR. This due to the fact that we always choose the path with the minimum number of links and hence minimize the utilized resources offering new requests more chances to be accepted. We also see that *R-opt* and *R-optM* have almost the same SBR. This is understandable since the two algorithms differ only in the position chosen to minimize the delay. As for *MinDelay*, we can see that in NSFNET and COST239, it performs poorly, since it can choose very long paths to assign spectrum with minimum delay and hence uses more resources causing the blocking of new requests. This is not the case for the less connected network USNET. This might be due to the fact that the path chosen by *MinDelay* cannot deviate much from the shortest paths due to the limited number of paths between two nodes in the network.



(a) USNET



(b) NSFNET

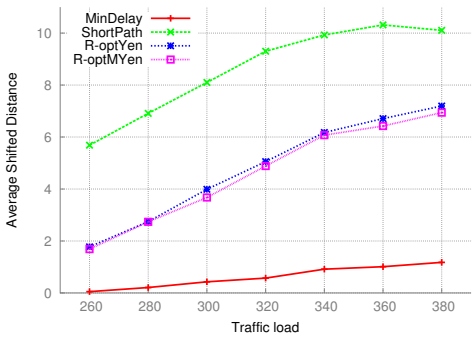


(c) COST239

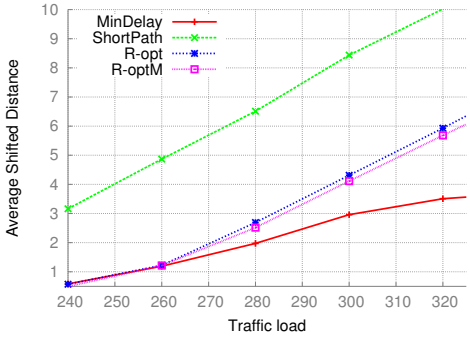
Figure 5.7: Average Delay (AD)

As for the average delay, we observe that *R-optM* performs only slightly better than *R-opt*. The improvement is in fact negligible and seems not to be worth the increased running time (compared to the running time of *R-opt*). *MinDelay* gives

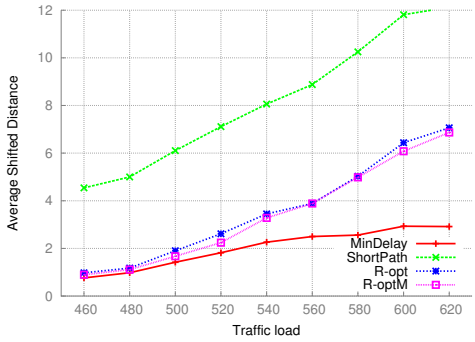
the minimum AD for USNET which is coherent with the blocking probability of the same algorithm. As for NSFNET and COST239, *MinDelay* gives even worst delay than R-opt. This might be because the fragmentation scenarios caused by *MinDelay* are more complex than the ones caused by *R-opt* (this fragmentation might also be the reason behind the high blocking probability). It is important to recall here that, with our algorithms, the local decisions (for a given request) are guaranteed to be optimal in terms of the acceptance of the request and in terms of the delay or the length of the path, however, globally (for all requests), no guarantees are given. Finally for *ShortPath*, the delay presented is the largest since minimizing the delay is not the primary objective.



(a) USNET



(b) NSFNET



(c) COST239

Figure 5.8: Average Shifted Distance (ASD)

For the Average Shifted Distance (ASD), we observe that the algorithm that does the maximum number of shiftings is *ShortPath* and the one performing the minimum number of shiftings is *MinDelay*. It is important to note here that even if *MinDelay* does not give the minimum AD in COST239 and NSFNET, it gives the minimum ASD. This highlights the difference between the two measurements AD and ASD.

Overall, the simulations reveal that choosing the shortest path is always a good

choice for the blocking ratio but not necessarily for the delay. Also, the *R-opt* gives a good trade-off between the blocking probability and the delay especially since the improvement in the delay brought by *R-optM* is negligible.

5.6 Conclusion

We have studied in this chapter dynamic RSA with Push-Pull. We have proposed two exact algorithms to solve the problem with two different criteria, the length of the routing path and the delay of the spectrum assignment. We have also built up an existing algorithm of SA with Push-Pull, to solve SA with Push-Pull with minimum delay. Simulations showed that our algorithms are mainly of theoretical interest and that the existing algorithm offers a good trade-off between the delay, the blocking probability and the running time. We have also developed some Integer Linear Programming (ILP) formulations for the problem that we did not include in the chapter since they have not been validated with simulations. Future research directions for the problem of dynamic RSA with Push-Pull might focus on the three following axes.

- Optimizing another criterion instead of the delay. This criterion might be the number of requests shifted in order to empty space for the new arriving request. This number is significant since it translates the number of messages exchanged at the control plane in order to enable the shiftings.
- Combining the proactive and reactive techniques using Push-Pull in order to decrease even more the spectrum blocking probability. We note here that in the proactive techniques requests are shifted periodically in order to consolidate the free spectrum.
- Examining the case where not only one request but a set of requests arrives at the same time and we need to decide if they can all be provisioned.

Chapter 6

Conclusions and perspectives

Fault-tolerance and efficient resource use, these are two of the highly desired properties in optical networks; the backbone of long-distance communication networks. It is important to provide fault-tolerance services in optical networks in order to protect the huge amounts of data they carry and to avoid costly service disruptions. It is also crucial to make an efficient use of the optical resources so as to meet the exponential growth of the traffic. We have studied in this thesis problems related to these two properties. Namely, we have considered the problem of finding disjoint paths related to the dedicated path protection scheme (DPP). In this scheme, the purpose is to protect connections against network faults by reserving resources on backup paths to reroute the traffic in case of failures. We have also tackled the problem of Routing and Spectrum Assignment (RSA) in Elastic Optical Networks, the new efficient and scalable generation of optical networks. In RSA, the objective is to set up connections in the network so as to make optimum use of the available resources.

Using mainly graph-theoretic models and tools, we established complexity results for these problems under different settings and we have answered questions such that: How hard is it to find a path or disjoint paths in networks with nodes which are not fully connected internally? How hard is it to find disjoint paths in networks where sets of links around a node can fail simultaneously? How hard is RSA in star networks? Can RSA be approximated in special cases of tree networks? How to utilize a non-disruptive defragmentation technique in EONs so as to minimize the spectrum blocking probability? The results we have obtained suggest that both problems DDP and RSA are hard to solve under the settings we consider. However, it is sometimes possible to identify polynomial cases or approximation algorithms. These cases and algorithms can hopefully help design efficient heuristics to tackle these problems for practical cases. We briefly describe in what follows the established results and recall some of the questions we have presented at the end of the chapters.

In Chapter 2, we have used graphs with forbidden transitions to model networks with asymmetric nodes. We have proved that the problem of finding a path avoiding forbidden transitions (PAFT) is NP-complete even in well-structured graphs such as grids. We have also proved that PAFT can be solved in polynomial time in graphs with bounded treewidth. Afterwards, we have tackled the problem of finding disjoint trails. We have shown that when the trails are required to be vertex-disjoint the problem is NP-complete in both directed and undirected graphs. On the positive side, if the paths are only required to be edge-disjoint, the problem can be solved in polynomial time in directed graphs. There are many interesting questions to inves-

tigate in graphs with forbidden transitions. For instance, it would be nice to look deeper into the parameterized complexity of the problem of finding a path avoiding forbidden transitions. In fact, we believe that the PAFT problem is $W[1]$-hard when parameterized by the treewidth. It would be hence interesting to try to prove this conjecture and also to consider other parameters, instead of the treewidth, such as the number of asymmetric nodes. It would be also nice to study the complexity of the problem of finding disjoint trails avoiding forbidden transitions in planar graphs. Finally, it would be interesting to explore the optimization versions of the problems we have studied with the aim of finding good approximations. Namely, the problem of finding a path using a minimum number of forbidden transitions and the problem of finding trails avoiding forbidden transitions which overlap on a minimum number of links of nodes can be considered.

In Chapter 3, we gave an almost complete characterization of the complexity of the problem of finding SRLG-disjoint paths in networks with SRLGs satisfying the star property. We have shown that finding k SRLG-disjoint paths is NP-complete even for $k = 2$. On the positive side, we have proved that the problem can be solved in polynomial time in particular subcases which are relevant in practice. Namely, we solve the problem in polynomial time when the maximum degree is at most 4 or when the input network is a directed acyclic graph. Moreover, we have shown that the problem is fixed-parameter tractable when parameterized by the number of SRLGs. Finally, we have considered the problem of finding the maximum number of SRLG-disjoint paths. We have proved that, under the star property, the problem is hard to approximate within $O(|V|^{1-\varepsilon})$ for any $0 < \varepsilon < 1$, where V is the set of nodes in the network, and we have given polynomial time algorithms for some of the above relevant subcases. The complexity of the problem of finding k SRLG-disjoint paths, under the star property is still open for the cases where the maximum degree of the network is equal to 5, 6 or 7 and for the cases where the number of SRLGs per link or the number of links per SRLG are equal to 2 or 3. Solving these cases will give a complete complexity characterization of the problem with respect to the maximum degree of the network, SRLGs per link and the number of links per SRLG. It would also be interesting to consider the unsolved cases for directed graphs since we only have results for the specific case of DAGs so far.

In Chapter 4, we have studied the problem of spectrum assignment in tree networks. We have proved that SA is NP-complete in undirected stars with 3 links and directed stars with 4 links. We have also shown that there is a 4-approximation algorithm to solve the problem in general stars. Afterwards, we have focused on SA in binary trees with bounded demands and we have designed constant approximation algorithms for several cases using the problem of interval coloring of chordal graphs. As future work, we would like to find constant-factor approximation algorithms for interval coloring in chordal graphs in general and to SA in binary trees in particular, or prove that such algorithms do not exist. Towards this objective, we believe it would be useful to try to adapt the approximation algorithms proposed for the problem of interval coloring in interval graphs.

In Chapter 5, we have studied in this chapter dynamic RSA with Push-Pull. We

have proposed two exact algorithms to solve the problem with two different criteria, the length of the routing path and the delay of the spectrum assignment. We have also built up an existing algorithm of SA with Push-Pull, to solve SA with Push-Pull with minimum delay. Simulations showed that our algorithms are mainly of theoretical interest and that the existing algorithm offers a good trade-off between the delay, the blocking probability and the running time. Future research directions for the problem of dynamic RSA with Push-Pull might focus on optimizing another criterion instead of the delay. It would be also interesting to try to combine the proactive and reactive techniques using Push-Pull in order to decrease even more the spectrum blocking probability. Finally, it would be nice to examine the case where not only one request but a set of requests arrives at the same time and we need to decide if they can all be provisioned.

As a general perspective, we believe that heuristics are a strong tool to confront the difficulty of the problems of DPP and RSA. It would be then interesting to invest in finding good performant heuristics for the problems. These heuristics can be inspired by the polynomial algorithms used to solve exactly or approximately some special cases.

Appendix A

On Static Routing and Spectrum Assignment

Contents

We give in this chapter column generation formulations for the problem of static routing and spectrum assignment. These formulations were written in collaboration with D. Coudert and B. Jaumard.

A.1 Introduction

In the problem of static routing and spectrum assignment, the input is a set of traffic requests and the objective is to allocate to each request, a path in the optical network and an interval of spectrum slots along that path, minimizing the utilized spectrum or maximizing the number of accepted requests for a fixed spectrum interval. The spectrum allocated to a demand has to be contiguous (contiguity constraint), it has to be the same over all links of the routing path (continuity constraint), and on a given link of the network a spectrum slot can be used by at most one request (non-overlapping constraint).

The problem of static RSA is NP-complete even when the routing is fixed and the network has few links as detailed in Chapter 4. Since heuristics and ILPs are usually used to deal with difficult problems, many ILP formulations have been proposed in the literature for RSA [KW11, CTV11, WCP11]. In [VKRC12], these formulations were surveyed and a new more compact formulation was proposed. In this new formulation, the contiguity and continuity constraints which are usually difficult to

model and handle are removed from the ILP and the notion of *channel* is introduced. A channel is a set of contiguous spectrum slots of a given width. With channels, the RSA consists in assigning to each request a path and a channel of width equal to the demand of the request.

The ILP formulations proposed for RSA are not practical to solve moderate size realistic instances because of the large number of variables. To deal with this issue, decomposition methods can be used. In [RPZ+13], *Ruiz et al.* presented an application of a decomposition method called *Column Generation (CG)* to RSA. In CG, the Linear Programming (LP) problem is solved with a small list of the variables (or columns). This list is iteratively extended with variables which can improve the objective function value. To generate these variables (columns), a pricing problem is formulated and solved. CG has been widely studied in the context of RWA [JMT09]. A classical CG-based formulation for RWA consists in generating lightpaths, where a lightpath consists of a path and a wavelength. This formulation is somehow similar to the one proposed in [RPZ+13] which also consists in lightpath generation, where a lighthpath consists of a path and a channel. Another formulation for RWA which performs quite well consists in generating, instead of lightpaths, configurations, where a configuration is a set of paths which can provision a fraction of the traffic over the same wavelength. In this chapter, we examine the applicability of the configuration-based formulation proposed in [JMT09] to the problem of static routing and spectrum assignment.

We first start by stating the problem in Section A.2. In Section A.3, we present the principal of column generation and briefly comment on the CG-based formulations for RWA. Afterwards, in Section A.4, we present the formulation presented in [RPZ+13] and give our configuration-based formulations. Finally in Section A.5, we discuss some numerical results we obtained and give our conclusions.

A.2 Problem Statement

We model the elastic optical network represented by a graph $\mathcal{N} = (V, L)$ with a node set $V = \{v_1, v_2, \ldots, v_n\}$ and link set $L = \{\ell_1, \ell_2, \ldots, \ell_m\}$. The bandwidth over each link is slotted into S small intervals of width 1 each called slots. The traffic is defined by a set $\mathcal{R}$ of requests where each request $r \in \mathcal{R}$ has a source s_r, a destination t_r and a demand in spectrum d_r . We denote by P_r the set of elementary paths from the source of r to its destination and by $\mathcal{P}$ the overall collection of elementary paths.

The RSA problem can be formally stated as follows. Given a graph $\mathcal{N}$ corresponding to an elastic optical network, and a set of requests, find a path and a spectrum allocation for every request respecting the continuity and contiguity constraints. The objective is to minimize the blocking rate, that is equivalent to maximizing the number of accepted requests. We detail in what follows the constraints which need to be respected.

- *Spectrum contiguity:* each request is assigned contiguous spectrum i.e., consecutive slots of the spectrum.

- *Spectrum continuity:* the assigned spectrum slots should be the same for each link of the routing path.
- *Non-overlapping:* each slot in each link can be allocated to at most one request.
- *Link capacity:* the number of slots used at a link cannot exceed the available bandwidth.

We recall that the Routing and Wavelength Assignment (RWA) problem can be considered as the special case of RSA where all requests have unit demands. In this case, there is no contiguity constraint.

A.3 About Column Generation

A linear program (LP) is a problem of optimization (minimization or maximization) of a linear function of variables, called the objective function, subject to a set of linear inequality constraints. If the variables are constrained to be integer, the LP is then called Integer Linear Program (ILP). Some LPs are very large and have a huge number of variables. Since it is hard to consider all these variables explicitly, some techniques have been designed to consider only a subset of the variables. Column generation (CG) is one of these techniques. In CG, the goal is to find the optimal solution without enumerating all of the variables (also referred to as columns), but by generating the columns which have the potential to improve the objective function. In CG, the problem to solve, referred to as the *master problem* (MP), is split into two problems: the *restricted master problem* (RMP) and the *pricing problem* (PP).

- The restricted master problem consists of the master problem over a subset of the variables.
- The pricing problem is a problem created to generate the variables that can improve the objective function if included in the subset over which the RMP is solved. According to whether the MP is a maximization or a minimization problem, the pricing problem consists in general in finding variables with positive (in case of maximization) or negative (in case of minimization) *reduced cost*. This reduced cost depends on the dual optimal solution of the RMP.

The column generation technique consists in solving iteratively the RMP followed by the PP. When the RMP is solved, its dual optimal solution is used to create the PP. If the PP generates new variables then these variables are added to the subset over which the RMP is solved. If no new variable is generated by the PP, then the optimal solution has been achieved and the column generation algorithm terminates.

When we have ILPs instead of LPs (which is the case in this chapter), a *branch-and-price* approach utilizing column generation is used to solve the ILP to optimality [BJN+98]. In this chapter, we will not use branch-and-price, instead we will solve

the relaxation of the MP using the column generation technique. Afterwards, we will use the final set of columns, which gives the optimal value for the relaxed MP, to solve the integer MP. We hope to have, by this approach usually, good bounds on the optimal solution of the ILP. For more details on column generation, the reader is referred to [NW88, DL05].

Column generation for RWA

Many column generation formulations have been proposed for the problem of Routing and Wavelength Assignment. In [JMT09], a review and comparison of several of these formulations are presented.

One of the straightforward and classical formulations for RWA is the path formulation. In this formulation, the variables correspond to lightpaths, where a lightpath is the combination of a routing path and a wavelength. Each request should be assigned a lightpath from its source to its destination such that the non-overlapping and the link capacity constraints are respected. With such formulation, the pricing problem consists in finding lightpaths with positive reduced cost. For RWA, this formulation suffers from the drawback of exhibiting a wavelength symmetry as explained in [JMT09], one can deduce a factorial number (as a function of the number of wavelengths) of alternate solutions for any given solution through wavelength permutations. This symmetry causes any branch-and-bound approach to perform poorly because the problem does not change much after branching.

Other column generation formulations with better performance and qualities have been proposed. One of which is based on the notion of configuration [JMT09]. A configuration is associated with a set of paths that can be used for satisfying a fraction of requests with the same wavelength. The formulation based on configurations is such that the number of used configurations cannot exceed the number of available wavelengths on a link and a request r is satisfied if one of the configurations containing a path of P_r is used. The pricing problem consists in finding a configuration with positive reduced cost. It can be noted that with this formulation the symmetry (with respect to the permutations of wavelength) is not problematic as it is the case for the path formulation. In fact, the wavelengths are assigned to configurations only after the problem is solved to optimality.

Since the configuration-based formulations work well for RWA, our purpose in this chapter is to find configuration-based formulations for the problem of RSA and check their performance with comparison to the CG-formulation proposed by [RPZ+13] and which rather resembles the path formulation of RWA.

A.4 Column generation for RSA

We first present in this section the column generation model presented in [RPZ+13] and then our configuration-based model.

A.4.1 Model of Ruiz *et al.* [RPZ+13]

To the best of our knowledge, only one paper has proposed a column generation formulation for the problem of routing and spectrum allocation so far, and that is [RPZ+13]. The authors of [RPZ+13] use the notions of channel which is a set of contiguous spectrum slots of given width and lightpath which is the combination of a path and a channel. A lightpath (p, c) can provision request r if $p \in P_r$ and the width of the channel c is equal to $d_r s$. They formulate their master problem in terms of lightpaths which allows them to hide the constraints of continuity and contiguity. Their pricing problem consists in generating a new lightpath that can improve the objective function if included in the solution and it is equivalent to finding a shortest path in the network, with weights that depend on the solution of the dual of the master problem. To generate new columns, the shortest path algorithm is ran for every source-destination pair for each possible channel. Note that this formulation is similar to the path formulation for RWA with the difference that a ligthpath for RWA is a path combined with a wavelength and a lightpath in RSA is a path combined with a channel.

A simplified version of the master problem in [RPZ+13] in what follows. In fact, in [RPZ+13], each request has a minimum demand and a maximum demand (these demands translate the bit-rates interval in which the request should operate if satisfied) and the objective of the master problem is to maximize the number of accepted requests (primary objective) and the amount of served bit-rate (secondary objective). In the version we present, we consider requests with fixed demand.

Notations

$P(r)$	set of all the lightpaths (path,channel) that can accommodate request r.
$P(r, \ell, s)$	set of all the lightpaths (path,channel) using link ℓ and slot s, that can accommodate request r.
S	overall set of slots
S_p	set of slots used by lightpath p.
x_r^p	binary decision variable; equal to 1 if request r uses lightpath p and 0 otherwise.

Master Problem

$$\max \quad \sum_{r \in \mathcal{R}} \sum_{p \in P(r)} x_r^p$$

subject to:

$$u_r^{(A.1)} \quad \sum_{p \in P(r)} x_r^p \leq 1 \qquad r \in \mathcal{R} \tag{A.1}$$

$$u_{s\ell}^{(A.2)} \quad \sum_{r \in R} \sum_{p \in P(r,\ell,s)} x_r^p \leq 1 \qquad s \in S, \ell \in L \tag{A.2}$$

$$x_r^p \in \{0,1\} \qquad r \in \mathcal{R}, p \in P(r) \tag{A.3}$$

A.4.2 Our configuration model

A configuration is associated with a set of paths that can be used for satisfying a given fraction of the requests with the same spectrum slot. There is at most one path per request in a configuration. Let C_s be the set of configurations associated to slot s and let C be the set of all configurations:

$$C = \bigcup_{s \in S} C_s.$$

We present in what follows two column generation formulations for the problem of RSA based on configurations: a path formulation and a link formulation. In the path formulation, variables x_r^p are used to decide if a request r can be routed with path p and a link-path flow formulation is used for the pricing problem. In the link formulation, variables x_r^ℓ are used to decide if a request r can be routed using link ℓ and a node-link flow formulation is used for the pricing problem.

A.4.2.1 Path formulation

We use the following notations.

- $a_{r,p}^c$ equal to 1 if request r can use path p in configuration c.
- z_c binary decision variable; equal to 1 if configuration c is used and 0 otherwise.
- x_r^p binary decision variable; equal to 1 if request r uses path p and 0 otherwise.
- y_r^s binary decision variable; equal to 1 if the starting slot of request r is s and 0 otherwise.

The master problem. Maximize the grade of services, i.e., the number of granted requests

$$\max \quad \sum_{r \in \mathcal{R}} \sum_{s \in S} y_r^s$$

subject to:

$$u_s^{(A.4)} \quad \sum_{c \in C_s} z_c \leq 1 \qquad s \in S \tag{A.4}$$

$$u_r^{(A.5)} \quad \sum_{p \in P_r} x_r^p \leq \sum_{s \in S} y_{rs} \qquad r \in \mathcal{R} \tag{A.5}$$

$$u_{rp}^{(A.6)} \quad \sum_{c \in C} a_{r,p}^c z_c \leq |S| x_r^p \qquad r \in \mathcal{R}, p \in P_r \tag{A.6}$$

$$u_r^{(A.7)} \quad \sum_{s \in S} y_r^s \leq 1 \qquad r \in \mathcal{R} \tag{A.7}$$

$$u_{rs}^{(A.8)} \quad d_r y_r^s \leq \sum_{i \in [s, min(|S|, s+d_r-1)]} \sum_{c \in C_i} \sum_{p \in P_r} a_{r,p}^c z_c \qquad r \in \mathcal{R}, s \in S \tag{A.8}$$

$$z_c \in \{0,1\} \qquad c \in C \tag{A.9}$$

$$x_r^p \in \{0,1\} \qquad r \in \mathcal{R}, p \in P_r \tag{A.10}$$

$$y_r^s \in \{0,1\} \qquad r \in \mathcal{R}, s \in S \tag{A.11}$$

Constraints (A.4) ensure that one spectrum slot is used by at most one configuration (non-overlapping constraint). Constraints (A.5) ensure that each request, if granted, uses at most one path. Constraints (A.6) ensure that for a given request r only configurations using the same path for r are used. Constraints (A.7) ensure that a request has at most one starting slot. Constraints (A.8) ensure that if a request has a starting slot, then it is fully provisioned with contiguous spectrum. Note that in constraints (A.8), the contiguous slots are used on the same path thanks to constraints (A.7).

Pricing problem PP(s_c). The reduced cost for variable z_c is (note that the dual variables are introduced next to their corresponding constraints and that s_c is the slot used by configuration c)

$$\overline{c}(z_c) = -u_{s_c}^{(A.4)} - \sum_{r \in \mathcal{R}} \sum_{p \in P_r} a_{r,p}^c u_{rp}^{(A.6)} + \sum_{r \in \mathcal{R}} \sum_{s \in \{max(1, s_c - d_r + 1), \ldots, s_c\}} \sum_{p \in P_r} u_{rs}^{(A.8)} a_{r,p}^c$$

To find whether there is a configuration with a positive reduced cost, let us consider the following problem which needs to be solved for each slot s_c:

$$\max \quad -u_{s_c} - \sum_{r \in \mathcal{R}} \sum_{p \in P_r} u_{rp}^{(A.6)} \alpha_p + \sum_{r \in \mathcal{R}} \sum_{s \in \{max(1, s_c - d_r + 1), \ldots, s_c\}} \sum_{p \in P_r} u_{rs}^{(A.8)} \alpha_p$$

$$\sum_{p \in P} \delta_\ell^p \alpha_p \leq 1 \quad \ell \in L \tag{A.12}$$

$$\sum_{p \in P_r} \alpha_p \leq 1 \quad r \in \mathcal{R} \tag{A.13}$$

$$\alpha_p \in \{0,1\} \quad p \in P \tag{A.14}$$

Constraints A.12 ensure that the paths chosen for the configuration are disjoint ($\delta_\ell^p = 1$ if path p uses link ℓ and 0 otherwise) and constraints A.13 ensure that at most one path is selected for each request.

A.4.2.2 Link Formulation

We use the following notations.

- $a_{r\ell}^c$ is equal to 1 if request r uses link ℓ in configuration c.
- z_c binary decision variable; equal to 1 if configuration c is used and 0 otherwise.
- x_r^ℓ binary decision variable; equal to 1 if request r uses link ℓ and 0 otherwise.
- y_r^s binary decision variable; equal to 1 if the starting slot of request r is s and 0 otherwise.

Master Problem. Maximize the grade of services, i.e., the number of granted requests

$$\max \quad \sum_{r\in\mathcal{R}} \sum_{s\in S} y_r^s$$

subject to:

$$u_s^{(A.15)} \quad \sum_{c\in C_s} z_c \leq 1 \qquad s \in S \qquad \text{(A.15)}$$

$$u_{vr}^{(A.16)} \quad \sum_{\ell\in\omega_v^+} x_r^\ell \leq 1 \qquad v \in V, r \in \mathcal{R} \qquad \text{(A.16)}$$

$$u_{r\ell}^{(A.17)} \quad \sum_{c\in C} a_{r,\ell}^c z_c \leq |S| x_r^\ell \qquad r \in \mathcal{R}, \ell \in L \qquad \text{(A.17)}$$

$$u_r^{(A.18)} \quad \sum_{s\in S} y_r^s \leq 1 \qquad r \in \mathcal{R} \qquad \text{(A.18)}$$

$$u_{rs}^{(A.19)} \quad d_r y_r^s \leq \sum_{i\in[s,min(s+d_r-1,|S|)]} \sum_{c\in C_i} \sum_{\ell\in\omega_{sr}^+} a_{k,\ell}^c z_c \qquad r \in \mathcal{R}, s \in S \qquad \text{(A.19)}$$

$$z_c \in \{0,1\} \qquad c \in C \qquad \text{(A.20)}$$

$$x_r^\ell \in \{0,1\} \qquad r \in \mathcal{R}, \ell \in L \qquad \text{(A.21)}$$

$$y_r^s \in \{0,1\} \qquad r \in \mathcal{R}, s \in S \qquad \text{(A.22)}$$

Constraints (A.15) ensure that each slot is used by at most one configuration. Constraints (A.16) ensure that for each node, a request uses at most one outgoing link. Constraints (A.17) ensure that if a link is not used, all the configuration using it are not. Constraints (A.17) and (A.16) ensure that a request uses at most one path. Constraints (A.18) ensure that a request has at most one starting slot. Constraints (A.19) ensure that if a request has a starting slot, then it is fully provisioned with the same contiguous spectrum over the first link of the path and hence to all other links of the path.

The pricing problem PP(s_c). The reduced cost for variable z_c is (note that the dual variables are introduced next to their corresponding constraints and that s_c is the slot used by configuration c)

$$\overline{c}(z_c) = -u_{s_c} - \sum_{r\in\mathcal{R}}\sum_{\ell\in L} a^c_{r,\ell} u^{(A.17)}_{r\ell} + \sum_{r\in\mathcal{R}} \sum_{s\in\{max(1,s_c-d_r+1),\ldots,s_c\}} \sum_{\ell\in\omega^+_{sr}} u^{(A.19)}_{rs} a^c_{r,\ell}$$

To find whether there is a configuration with a positive reduced cost, let us consider the following problem which needs to be solved for each slot s_c:

$$\max \quad -u_{s_c} - \sum_{r\in\mathcal{R}}\sum_{\ell\in L} u^{(A.17)}_{r\ell} \alpha^r_\ell + \sum_{r\in\mathcal{R}} \sum_{s\in\{max(1,s_c-d_r+1),\ldots,s_c\}} \sum_{\ell\in\omega^+_{sr}} u^{(A.19)}_{rs} \alpha^r_\ell$$

$$\sum_{r\in\mathcal{R}} \alpha^r_\ell \leq 1 \qquad \ell \in L \tag{A.23}$$

$$\sum_{\ell\in\omega^+_v} \alpha^r_\ell = \sum_{\ell\in\omega^-_v} \alpha^r_\ell \qquad r\in\mathcal{R}, v\in V\setminus\{s_r,t_r\} \tag{A.24}$$

$$\sum_{\ell\in\omega^+_{sr}} \alpha^r_\ell = \sum_{\ell\in\omega^-_{t_r}} \alpha^r_\ell \leq 1 \qquad r\in\mathcal{R} \tag{A.25}$$

$$\sum_{\ell\in\omega^-_{sr}} \alpha^r_\ell = \sum_{\ell\in\omega^+_{t_r}} \alpha^r_\ell = 0 \qquad r\in\mathcal{R} \tag{A.26}$$

$$\alpha^r_\ell \in \{0,1\} \qquad \ell \in L \tag{A.27}$$

(A.28)

Constraints A.23 ensure that an link in a configuration is used by at most one request and the other constraints are flow conservation constraints.

A.4.3 Discussion

Compared to the formulation using lightpaths in [RPZ+13] and to the configuration based formulation for RWA in [JMT09], the configuration formulations for RSA seem to be more complex and have more constraints and variables. This is mainly due to the fact that it is difficult to handle the contiguity constraints with configurations. These constraints were successfully hidden with the use of channels in [RPZ+13], but with configurations, we could not ensure them without the use of additional constraints.

For the configuration formulation of RWA, a variable a^c_r is used to indicate if a request r can be provisioned with a configuration c, and only decision variables z_c are used to decide if a configuration is used or not and hence is a request is satisfied or not. In our formulation, we use variables $a^c_{r,p}$ instead of a^c_r since we need to know, not only if a request can be provisioned with a configuration, but also with which path. We also use decision variables x^p_r and x^ℓ_r besides the variables z_c to ensure that the used configurations use the same path for a given request.

Furthermore, while in RWA, configurations are defined independently from wavelengths and wavelengths are assigned to the configurations after the problem is solved, in our formulation, each configuration should be defined with a slot in order to ensure the contiguity constraint.

Because of the complexity of our formulations, we were not optimistic about their performance. We have nonetheless run some simulations to have more insight on how they work in practice.

A.5 Numerical results and conclusion

A.5.1 Numerical results

As in [RPZ+13], we run our simulations with:

- Network: the Spanish Telefonica network which has 21 nodes and 35 links as illustrated in Figure A.1.

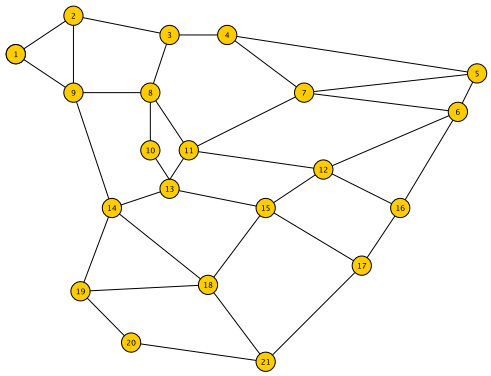


Figure A.1: The Spanish network

- Requests $\mathcal{R}$: randomly generated requests; we use a random selection of the (source,destination) among the nodes of the network and a random selection of the demand from the set $\{1, 2, 4, 8\}$ corresponding to bitrates $\{10, 40, 100, 200\}$ Gbps. For each request r, the set P_r consists of the 5 shortest paths from the source of r to its destination.

- Instances: for each load, we solve the problem for 5 instances and return the average in the tables.

- Slots: we have run the simulations for 10 slots and 40 slots.

- Initial configurations: we proceed similarly to the MaxLeft algortihm presented in [RPZ+13]: we sort requests in the decreasing order of the demand, then create greedily the maximum independent set of requests (i.e. requests whose shortest paths do not intersect), then create configurations from this

set using slots from 1 the size of the biggest request. We repeat this process until all requests have been allocated to some configuration. We refer to the initial set of configurations with C_i and to the final set of configuration (at the termination of the CG algorithm) with C_f.

We have implemented the formulations in Python using the PuLP modeler and the CPLEX 12.4 solver. The simulation were run on a 2,6 GHz Core i7 machine with 8 GB RAM running an OS X 10.8.3.

We capture in the tables the load in Tbps, the number of requests $|\mathcal{R}|$, the number of initial configurations $|C_i|$, the number of final configurations $|C_f|$, the solution of the LP relaxation obtained with the column generation algorithm, solution of the ILP problem with the final set of configurations as variables, the CPLEX time used in the CG algorithm and to solve the ILP, the gap between the LP and ILP solutions and finally the number of iterations in the CG.

Total load (Tbps)	$\|\mathcal{R}\|$	$\|C_i\|$	$\|C_f\|$	LP relax.	Integer lower bound	CG CPLEX time (s)	ILP CPLEX time (s)	Gap (%)	Iter.
2	23	27.8	29.8	23	12	0.265	0.064	47.40	1.2
3	36	41.4	51.4	36	15.4	0.591	0.135	56.09	2
4	46.4	42.6	50.6	46.4	16.6	0.626	0.162	63.99	1.8
5	50.2	49	63	50.2	15	1.011	0.192	56.10	2.2
6	68.8	57.4	77.4	68.8	20.6	1.502	0.283	69.67	3
7	81.2	68.2	94.2	81.2	19	2.194	0.355	76.44	3.6
8	101.4	85.8	113.8	101.4	23	2.713	0.479	77.27	3.8
9	108	83.8	111.8	108	24	2.897	0.516	77.66	3.8

Table A.1: Path formulation with 10 slots

Total load (Tbps)	$\|\mathcal{R}\|$	$\|C_i\|$	$\|C_f\|$	LP relax.	Integer lower bound	CG CPLEX time (s)	ILP CPLEX time (s)	Gap (%)	Iter.
2	23	27.8	31.6	23	14.6	5.625	0.411	47.46	1.4
3	36	41.4	51.4	36	21	13.771	1.377	41.93	2
4	46.4	42.6	50.6	46.4	22.2	17.514	2.152	52.38	1.8
5	50.2	49	63	50.2	21.6	29.572	4.928	45.78	2.2
6	68.8	57.4	75.4	68.8	29.4	47.31	5.60	57.43	2.8
7	81.2	68.2	94.2	81.2	29.8	68.608	31.346	63.55	3.4
8	101.4	85.8	113.8	101.4	37.6	119.033	261.24	63.24	3.8
9	108	83.8	113.8	108	35.8	136.05	176.09	66.76	4

Table A.2: Link formulation with 10 slots

Total load (Tbps)	$\|\mathcal{R}\|$	$\|C_i\|$	$\|C_f\|$	LP relax.	Integer lower bound	CG CPLEX time (s)	ILP CPLEX time (s)	Gap (%)	Iter.
2	23	27.8	29.6	23	12	1.08	0.167	47.40	1.2
3	36	41.4	81.4	36	18.2	2.287	0.315	48.83	2
4	46.4	42.6	74.6	46.4	18	2.7036	0.799	60.77	1.8
5	50.2	49	105	50.2	22.8	3.978	3.076	43.96	2.2
6	68.8	57.4	145.4	68.8	30	6.226	4.640	55.86	3.2
7	81.2	68.2	164.2	81.2	35	7.655	8.296	56.64	3.4
8	101.4	85.8	197.8	101.4	45.8	8.555	16.164	54.90	3.8
9	108	83.8	203.8	108	44.6	11.787	14.503	58.96	4

Table A.3: Path formulation with 40 slots

Total load (Tbps)	$\|\mathcal{R}\|$	$\|C_i\|$	$\|C_f\|$	LP relax.	Integer lower bound	CG CPLEX time (s)	ILP CPLEX time (s)	Gap (%)	Iter.
2	23	27.8	37.6	23	17	23.154	0.522	28.72	1.4
3	36	41.4	81.4	36	21.6	55.588	2.714	40.27	2
4	46.4	42.6	74.6	46.4	22.8	69.951	4.241	50.8	1.8
5	50.2	49	97	50.2	24	121.346	7.816	41.98	2
6	68.8	57.4	129.4	68.8	31.6	233.729	11.851	54.52	2.8
7	81.2	68.2	148.2	81.2	39.2	247.77	26.93	51.501	3
8	101.4	85.8	197.8	101.4	52.2	452.876	77.964	52.2	3.8
9	108	83.8	203.8	108	46.2	528.462	240.71	46.2	4

Table A.4: Link formulation with 40 slots

Since the link formulation (Tables A.2, A.4) has more constraints and variables than the path formulation (Tables A.1, A.3), it performs slower. However, it gives better integer solutions and hence better gaps than the path formulation. The important thing to note here is that in both cases, the gap between the linear and integer solutions is big. In order to reduce this gap, we have tried an implementation of the path formulation with rounding off: we run a column generation iteration, examine the solution to see the variables y that are close enough to one (≥ 0.7), round them off to one (which implies forcing the acceptance of a given request on a given starting slot), and then run another column generation iteration. This is repeated until no new variables y can be rounded off i.e. no variables are close enough to one. However, this has not yielded better gaps.

Afterwards, we have implemented the channel-based compact formulation (the master problem presented in Section A.4.1) proposed in [RPZ+13] to check the quality of the integer solutions we can have. To obtain the set of channels, for

every request, the 5 shortest paths are computed and then for each path all possible channels are created (considering all possible contiguous sets of slots of width equal to the demand of the request). Results of the experiments captured in Tables A.5,A.6 show a very small gap between the linear and integer solutions.

The fact that the gap is small with the compact formulation of [RPZ+13] indicates that there are indeed configurations which can give good integer solutions. These configurations were not generated with our formulations.

Total load (Tbps)	$\|\mathcal{R}\|$	|Channels|	Linear relax.	Integer solution	Time to solve LP (s)	Time to solve ILP	Gap (%)
2	19	626	19	18.66	0.09	0.89	1.55
3	35	1243	33.87	32.66	0.33	1.42	4.21
4	46	1660	42.79	42	0.92	21.42	1.61
5	61	2226	52.84	51	1.15	287.36	3.47
6	72	2603	60.6	58.33	4.18	1934	3.74
7	78	2703	60.25	58	4.06	1159.5	3.77
8	104	3982	86.625	84	3.48	20229	3.01

Table A.5: Ruiz *et al.* compact formulation with 10 slots

Total load (Tbps)	$\|\mathcal{R}\|$	|Channels|	Linear relax.	Integer solution	Time to solve LP (s)	Time to solve ILP	Gap (%)
2	19	3526	19	19	0.36	1.65	0
3	35	6493	35	35	0.45	2.1	0
4	46	8660	46	46	0.57	2.72	0
5	61	11426	61	61	0.78	3.33	0
6	72	13453	72	72	0.77	4.27	0
7	78	14353	78	78	16.75	29.75	0
8	104	19456	104	104	27.88	57.8	0
9	110	20486	110	110	21.19	60.53	0
10	113	20926	113	113	25.53	112.04	0
20	244	45605	229.7		565	$> 10h$	

Table A.6: Ruiz *et al.* compact formulation with 40 slots

A.5.2 Conclusion

Our approach of using configuration-based column generation formulations for the problem of Routing and Spectrum Assignment, added complexity to the model instead of simplifying it as discussed in Section A.4.3. Furthermore, preliminary numerical results show that while there exist configurations which can give good

integer solutions for the problem, the configurations generated by our formulations are not of good quality in terms of the integer solution they provide. This has discouraged us from investing more in these formulations by trying a branch-and-price approach.

Appendix B

Minimum Size Tree-Decompositions

Contents

In this chapter, we consider the problem of computing a tree-decomposition of a graph with width at most k and minimum *size*, i.e. the number of bags in the tree-decomposition. More precisely, we focus on the following problem: given a fixed $k \geq 1$, what is the complexity of computing a tree-decomposition of width at most k with minimum size in the class of graphs with treewidth at most k? The results of this chapter are in a collaboration with B. Li and N. Nisse. They were presented in the conference ICGT 2014 [LMN14] and LAGOS 2015 [LMNS15].

B.1 Introduction

A *tree-decomposition* of a graph [RS86] G, as we have defined it in Section 2.4.2 of Chapter 2, is a way to represent G by a family of subsets of its vertex-set organized in a tree-like manner and satisfying some connectivity property. The *treewidth* of G measures the proximity of G to a tree. More formally, a tree-decomposition of $G = (V, E)$ is a pair $(T, \mathcal{X})$ where $\mathcal{X} = \{X_t | t \in V(T)\}$ is a family of subsets of V, called *bags*, and T is a tree, such that:

- $\bigcup_{t \in V(T)} X_t = V$;
- for any edge $uv \in E$, there is a bag X_t (for some node $t \in V(T)$) containing both u and v;

- for any vertex $v \in V$, the set $\{t \in V(T) | v \in X_t\}$ induces a subtree of T.

The *width* of a tree-decomposition $(T, \mathcal{X})$ is $max_{t \in V(T)} |X_t| - 1$ and its *size* is the order $|V(T)|$ of T. The treewidth of G, denoted by $tw(G)$, is the minimum width over all possible tree-decompositions of G. If T is constrained to be a path, $(T, \mathcal{X})$ is called a *path-decomposition* of G. The pathwidth of G, denoted by $pw(G)$, is the minimum width over all possible path-decompositions of G.

Tree-decompositions are the corner-stone of many dynamic programming algorithms for solving graph problems. For example, the famous Courcelle's Theorem states that any problem expressible in MSOL can be solved in linear-time in the class of bounded treewidth graphs [Cou90]. Another framework based on graph decompositions is the *bi-dimensionality theory* that allowed the design of sub-exponential-time algorithms for many problems in the class of graphs excluding some fixed graph as a minor (e.g., [DH08]). Given a tree-decomposition with width w and size n, the time-complexity of most of such dynamic programming algorithms can often be expressed as $O(2^w n)$ or $O(2^{w \log w} n)$. These algorithms have mainly theoretical interest because their time-complexity depends exponentially on the treewidth and, on the other hand, no practical algorithms are known to compute a "good" tree-decomposition for graphs with treewidth at least 5.

Since the computation of tree-decompositions is a challenging problem, we propose in this chapter to study it from a new point of view. Namely, we aim at minimizing the number of bags of the tree-decomposition when the width is bounded. This new perspective is interesting on its own and we hope it will allow to gain more insight into the difficulty of designing practical algorithms for computing tree-decompositions.

We consider the problem of computing tree-decompositions with minimum size. If the width is not constrained, then a trivial solution is a tree-decomposition of the graph with one bag (the full vertex-set). Hence, given a graph G and an integer $k \geq tw(G)$, we consider the problem of minimizing the size of a tree-decomposition of G with width at most k.

Related work The problem of computing "good" tree-decompositions has been extensively studied. Computing optimal tree-decomposition - i.e., with width $tw(G)$ - is NP-complete in the class of general graphs G [ACP87]. For any fixed $k \geq 1$, Bodlaender designed an algorithm that computes, in time $O(k^{k^3} n)$, a tree-decomposition of width k of any n-vertex graph with treewidth at most k [Bod96]. Very recently, a single-exponential (in k) algorithm has been proposed that computes a tree-decomposition with width at most $5k$ in the class of graphs with treewidth at most k [BDD+13]. As far as we know, the only practical algorithms for computing optimal tree-decompositions hold for graphs with treewidth at most 1 (trivial since $tw(G) = 1$ if and only if G is a tree), 2 (graphs excluding K_4 as a minor) [WC83], 3 [AP86, KIU86, MT91] and 4 [San96].

In [DKZ13], Dereniowski *et al.* consider the problem of size-constrained path-decompositions. Given any positive integer k and any graph G with pathwidth at

most k, let $l_k(G)$ denote the smallest size (length) of a path-decomposition of G with width at most k. For any fixed $k \geq 4$, computing l_k is NP-complete in the class of general graphs and it is NP-complete, for any fixed $k \geq 5$, in the class of connected graphs [DKZ13]. Moreover, computing l_k can be solved in polynomial-time in the class of graphs with pathwidth at most k for any $k \leq 3$. Finally, the "dual" problem is also hard: for any fixed $s \geq 2$, it is NP-complete in general graphs to compute the minimum width of a tree-decomposition with size s [DKZ13][1]. We have generalized the problem of minimum size path-decomposition presented in [DKZ13], and introduced the problem of minimum size tree-decomposition in [LMN14, LMNS15]. To the best of our knowledge, no other paper has dealt with the computation of tree-decompositions with minimum size before [LMN14, LMNS15]. However, very recently, following the work in [LMN14] and [DKZ13], Bodlaender et al. [BN15] have proposed exact subexponential time algorithms to solve the problems of minimum size tree-decomposition and minimum size path-decomposition for a fixed width k in $2^{O(n/\log(n))}$ time and showed that the two problems cannot be solved in $2^{o(n/\log(n))}$ time, assuming the Exponential Time Hypothesis.

Contribution Let k be any positive integer and G be any graph. If $tw(G) > k$, let us set $s_k(G) = \infty$. Otherwise, let $s_k(G)$ denote the minimum size of a tree-decomposition of G with width at most k. See a simple example in Figure B.1. We first prove in Section B.2 that, for any (fixed) $k \geq 4$, the problem of computing s_k is NP-hard in the class of graphs with treewidth at most k. Moreover, the computation of s_k for $k \geq 5$ is NP-hard in the class of connected graphs with treewidth at most k. Furthermore, the computation of s_4 is NP-complete in the class of planar graphs with treewidth 3. In Section B.3, we present a general approach for computing s_k for any $k \geq 1$. In the rest of the chapter, we prove that computing s_2 can be solved in polynomial-time. Finally, we prove that s_3 can be computed in polynomial time in the class of trees and 2-connected outerplanar graphs.

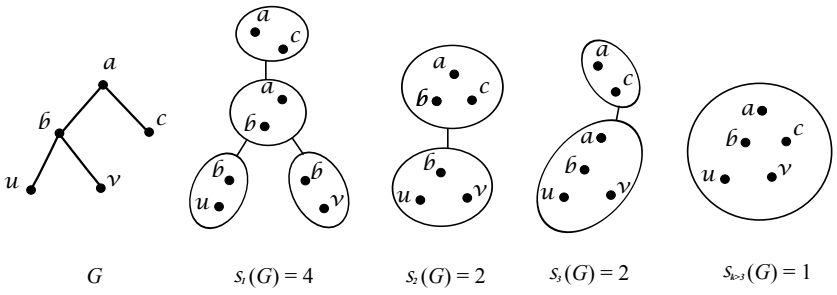


Figure B.1: Given a tree G with five vertices, for any $k \geq 1$, a minimum size tree-decomposition of width at most k is illustrated: $s_1(G) = 4$, $s_2(G) = s_3(G) = 2$, and $s_{k>3}(G) = 1$.

[1] This result was proved in [DKZ13] in terms of path-decomposition but it is straightforward to extend it to tree-decomposition.

B.2 NP-hardness in the class of bounded treewidth graphs

In this section, we prove that:

Theorem 29. *For any fixed integer $k \geq 4$ (resp., $k \geq 5$), the problem of computing s_k is NP-complete in the class of graphs (resp., of connected graphs) with treewidth at most k.*

Note that the corresponding decision problem is clearly in NP. Hence, we only need to prove it is NP-hard.

Our proof mainly follows the one of [DKZ13] for size-constrained path-decompositions. Hence, we recall here the two steps of the proof in [DKZ13]. First, it is proved that, if computing l_k is NP-hard for any $k \geq 1$ in general graphs, then the computation of l_{k+1} is NP-hard in the class of connected graphs. Second, it is shown that computing l_4 is NP-hard in general graphs with pathwidth 4. In particular, this implies that computing l_5 is NP-hard in the class of connected graphs with pathwidth 5. The second step consists of a reduction from the 3-PARTITION problem [GJ79] to the one of computing l_4. Precisely, for any instance $\mathcal{I}$ of 3-PARTITION, a graph $G_{\mathcal{I}}$ is built such that $\mathcal{I}$ is a YES instance if and only if $l_4(G_{\mathcal{I}})$ equals a defined value $\ell_{\mathcal{I}}$.

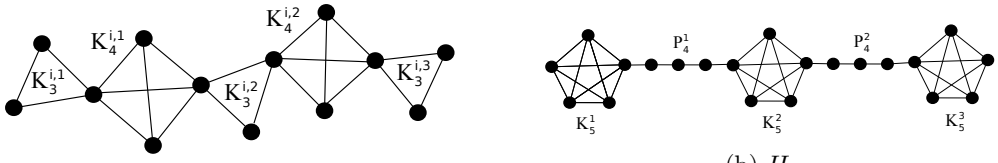
Our contribution consists first in showing that the first step of [DKZ13] directly extends to the case of tree-decompositions. That is, it directly implies that, if computing s_k is NP-hard for some $k \geq 4$ in general graphs, then so is the computation of s_{k+1} in the class of connected graphs. Our main contribution of this section is to show that, for the graphs $G_{\mathcal{I}}$ built in the reduction proposed in [DKZ13], any tree-decomposition of $G_{\mathcal{I}}$ with width at most 4 and minimum size is a path-decomposition. Hence, in this class of graphs, $l_4 = s_4$ and, for any instance $\mathcal{I}$ of 3-PARTITION, $\mathcal{I}$ is a YES instance if and only if $s_4(G_{\mathcal{I}})$ equals a defined value $\ell_{\mathcal{I}}$. We describe the details in what follows.

Lemma 8. *If the problem of computing s_k for an integer $k \geq 1$ is NP-complete in general graphs, then the computation of s_{k+1} is NP-complete in the class of connected graphs.*

Proof. Let G be any graph. We construct an auxiliary connected graph G' from G by adding a vertex a adjacent to all vertices in $V(G)$. Given two integers $k, s \geq 1$, in the following, we prove that there is a tree-decomposition of G with width at most k and size at most s if and only if there is a tree-decomposition of G' with width at most $k+1$ and size at most s.

First, let us assume that $(T, \mathcal{X})$ is a tree-decomposition of G with width at most k and size at most s. By adding a in each bag of $\mathcal{X}$, we obtain a tree-decomposition of G' with width at most $k+1$ and size at most s.

Now let $(T', \mathcal{X}')$ be a tree-decomposition of G' with width at most $k+1$ and size at most s. We are going to find a tree-decomposition of G with width at most



(a) H_i, where $w_i = 3$

(b) $H_{2,4}$

Figure B.2: Examples of gadgets in graph $G(S, b)$ [DKZ13]

k and size at most s. Let $\mathcal{X}_a$ be the set of all bags in $\mathcal{X}'$ containing a. Let T_a be the subtree of T' induced by the bags in $\mathcal{X}_a$. Every vertex $v \in V(G)$ is contained in a bag in $\mathcal{X}_a$ because $va \in E(G')$. For any edge $uv \in E(G)$, there is a bag $X \supseteq \{a, u, v\}$ in $\mathcal{X}'$ since $\{a, u, v\}$ induces a clique in G'. This implies that $X \in \mathcal{X}_a$. We delete a from each bag of $\mathcal{X}_a$ and denote by $\mathcal{X}^-$ the obtained set of bags. So $(T_a, \mathcal{X}^-)$ is a tree-decomposition of G with width at most k and size at most s. □

Before doing the reduction from the 3-Partition problem to the problem of computing s_4, let us first recall its definition.

Definition 4. [3-Partition]
Instance: A set S of $3m$ positive integers $S = (w_1, \ldots, w_{3m})$ and an integer b.
Question: Is there a partition of the set $\{1, \ldots, 3m\}$ into m sets $S_1, \ldots, S_m$ such that $\sum_{i \in S_j} w_i = b$ for each $j = 1, \ldots, m$?

This problem is NP-complete even if $|S_j| = 3$ for all $j = 1, \ldots, m$ [GJ79].

Given an instance of 3-Partition, in the following, we construct a disconnected graph $G(S, b)$ as in [DKZ13]. First, for each $i \in \{1, \ldots, 3m\}$, we construct a connected graph H_i as follows. We take w_i copies of K_3, denoted by $K_3^{i,q}$, $q = 1, \ldots, w_i$, and $w_i - 1$ copies of K_4, denoted by $K_4^{i,q}$, $q = 1, \ldots, w_i - 1$ (the copies are mutually disjoint). Afterwards, for each $q = 1, \ldots, w_i - 1$, we identify two different vertices of $K_4^{i,q}$ with a vertex of $K_3^{i,q}$ and with a vertex of $K_3^{i,q+1}$, respectively. This is done in such a way that each vertex of each $K_3^{i,q}$ is identified with at most one vertex from other cliques. Informally, the cliques form a 'chain' in which the cliques of size 3 and 4 alternate. See Figure B.2a for an example of H_i for $w_i = 3$.

Second, we construct a graph $H_{m,b}$ as follows. We take $m + 1$ copies of K_5, denoted by $K_5^1, \ldots, K_5^{m+1}$, and m copies of the path graph P_b of length b (P_b has b edges and $b + 1$ vertices), denoted by $P_b^1, \ldots, P_b^m$ (again, the copies are mutually disjoint). Now, for each $j = 1, \ldots, m$, we identify one of the endpoints of P_b^j with a vertex of K_5^j, and identify the other endpoint with a vertex of K_5^{j+1}. Moreover, we do this in a way that ensures that, for each j, no vertex of K_5^j is identified with the endpoints of two different paths. See Figure B.2b for an example of $H_{2,4}$.

Let $G(S, b)$ be the graph obtained by taking the disjoint union of the graphs $H_1, \ldots, H_{3m}$ and the graph $H_{m,b}$. In the following, we prove that there is a tree-decomposition of $G(S, b)$ of width 4 and size at most $s = 1 - 2m + 2\sum_{i=1}^{3m} w_i$ if and

only if there is a partition of the set $\{1,\dots,3m\}$ into m sets $S_1,\dots,S_m$ such that $\sum_{i\in S_j} w_i = b$ for each $j = 1,\dots,m$ in the instance of 3-Partition.

In Lemma 2.2 of [DKZ13], a path-decomposition of $G(S,b)$ of width 4 and length $1-2m+2\sum_{i=1}^{3m} w_i$ is constructed if there is a partition of the set $\{1,\dots,3m\}$ into m sets $S_1,\dots,S_m$ such that $\sum_{i\in S_j} w_i = b$ for each $j = 1,\dots,m$ in the instance of 3-Partition. Obviously, this path-decomposition is also a tree-decomposition of $G(S,b)$ of width 4 and size s. So we have the following lemma.

Lemma 9. *Given a multiset S of $3m$ positive integers $S = (w_1,\dots,w_{3m})$ and an integer b, if there is a partition of the set $\{1,\dots,3m\}$ into m sets $S_1,\dots,S_m$ such that $\sum_{i\in S_j} w_i = b$ for each $j = 1,\dots,m$, then $G(S,b)$ has a tree-decomposition of width at most 4 and size at most $s = 1-2m+2\sum_{i=1}^{3m} w_i$.*

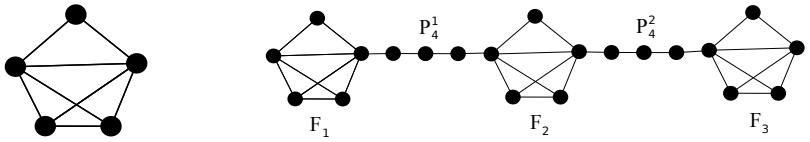
Now we prove the other direction.

Lemma 10. *If $G(S,b)$ has a tree-decomposition $(T,\mathcal{X})$ of width at most 4 and size at most $s = 1-2m+2\sum_{i=1}^{3m} w_i$, then there is a partition of the set $\{1,\dots,3m\}$ into m sets $S_1,\dots,S_m$ such that $\sum_{i\in S_j} w_i = b$ for each $j = 1,\dots,m$.*

Proof. Lemma 2.6 in [DKZ13] proved that if $G(S,b)$ has a path-decomposition $(T,\mathcal{X})$ of width at most 4 and length at most $1-2m+2\sum_{i=1}^{3m} w_i$, then there is a partition of the set $\{1,\dots,3m\}$ into m sets $S_1,\dots,S_m$ such that $\sum_{i\in S_j} w_i = b$ for each $j = 1,\dots,m$. In what follows, we prove that any tree-decomposition (T,X) of $G(S,b)$ of width at most 4 and size at most $s = 1-2m+2\sum_{i=1}^{3m} w_i$ is a path-decomposition of $G(S,b)$.

As proved in Lemma 2.3 of [DKZ13], each bag in (T,X) contains exactly one of the cliques $K_3^{i,q}, K_4^{i,q}, K_5^j$. Indeed, each of these cliques has size at least 3. Moreover, any two of them share at most one vertex, and no two cliques of size 3 ($K_3^{i,q}$) share a vertex. So each bag of $(T,\mathcal{X})$ contains at most one of the cliques $K_3^{i,q}, K_4^{i,q}, K_5^j$. Moreover, any clique of the graph is fully contained in a bag of $(T,\mathcal{X})$. Since s equals the number of the cliques $K_3^{i,q}, K_4^{i,q}, K_5^j$, each bag of $(T,\mathcal{X})$ contains exactly one of them.

Now, let us prove that any edge in $K_4^{i,q}, K_5^j, P_b^j$ (i.e. both the two endpoints of the edge) is contained in exactly one bag. Since each bag in (T,X) contains exactly one of the cliques $K_3^{i,q}, K_4^{i,q}, K_5^j$, the two endpoints of any edge in the paths $P_b^1,\dots,P_b^m$ are contained in a bag containing some $K_3^{i,q}$. In fact, the bags containing a $K_4^{i,q}$ (resp., K_5^j) cannot contain two additional vertices (resp., one vertex) since $(T,\mathcal{X})$ is a tree-decomposition of width at most 4. Every bag containing some $K_3^{i,q}$ contains at most one edge in the paths $P_b^1,\dots,P_b^m$, because the bag can contain at most two additional vertices and $K_3^{i,q}$ and P_b^j are disjoint. There are mb edges in the paths $P_b^1,\dots,P_b^m$ and there are mb bags containing some $K_3^{i,q}$, so every bag containing a $K_3^{i,q}$ contains exactly one edge in the paths $P_b^1,\dots,P_b^m$. Any edge in the paths $P_b^1,\dots,P_b^m$ is then contained in exactly one bag. Also each bag containing some $K_3^{i,q}$ contains 5 vertices, so it does not contain any edge (i.e. both its endpoints) in $K_4^{i,q}$ or K_5^j. Therefore, any edge on $K_4^{i,q}, K_5^j$ is contained in exactly one bag.



(a) F (b) $H_{2,4}$

Figure B.3: Example of the new gadget in $G(S, b)$.

Now we prove that there are only two leaves in T and so T is a path. If a bag containing some $K_3^{i,q}$ and an edge uv on some path P_b^j is a leaf bag in T, then its neighbor bag also contains u, v because both u and v are incident to other edges in $G(S, b)$. This is a contradiction with the fact any edge (its two endpoints) on P_b^j is contained in only one bag. Hence, any bag containing some $K_3^{i,q}$ is not a leaf bag in T. Similarly, we can prove that any bag containing any $K_4^{i,q}$ or K_5^j for $1 < j < m+1$ is not a leaf bag in T. Thus there are only two bags containing K_5^1 and K_5^{m+1} which are leaves in T. □

Thus, we obtain the following corollary.

Corollary 7. *It is NP-complete to compute s_4 in the class of graphs of treewidth at most 4.*

Theorem 29 follows from Lemma 8 and Corollary 7. We furthermore modify the reduction to prove theorem 30.

Theorem 30. *It is NP-complete to compute s_4 in the class of planar graphs of treewidth at most 3.*

Proof. As in the previous reduction, we build a graph $G(S, b)$ for an instance of 3-PARTITION; we keep the subgraphs H_i as they are and modify the graph $H_{m,b}$ as follows. We replace the $m + 1$ copies of K_5 by $m + 1$ copies of the graph F that consists of a K_4 and a K_3 sharing an edge as depicted in Figure B.3a. We denote the copies by $F_1, F_2, \ldots, F_{m+1}$. The new graph $G(S, b)$ we obtain is planar and has treewidth 3.

Lemma 9 is still true and for Lemma 10 to be correct, we need to prove that if $G(S, b)$ has a tree-decomposition $(T, \mathcal{X})$ of width at most 4 and size at most $s = 1 - 2m + 2\sum_{i=1}^{3m} w_i$, then there is a bag of $(T, \mathcal{X})$ containing F_i, for each F_i, $i \in \{1, \ldots, m + 1\}$. Let us denote by K_3^i and K_4^i the two cliques sharing exactly one edge that form F_i. Each of these cliques, should appear in one bag. Note that among all the cliques of $G(S, b)$, the only cliques that can coexist in a bag are of the form K_3^i and K_4^i since the sum of the number of vertices of any other two cliques is more than 5. Let us suppose that there exists $j \in \{1, \ldots, m+1\}$ such that no bag of $(T, \mathcal{X})$ contains F_j, i.e. K_3^j and K_4^j are not in the same bag. In this case the number of bags of $(T, \mathcal{X})$ is at least the number of the cliques $K_3^{i,q}, K_4^{i,q}, K_4^{i'}$ $(i' \neq j)$, plus the two bags containing K_3^j and K_4^j. This gives a size of at least $2 - 2m + 2\sum_{i=1}^{3m} w_i$ wich is not possible.

□

B.3 Preliminaries

In this section, we present the definitions and notations used throughout the chapter and some well-known facts about tree-decompositions.

B.3.1 Notations

Let $G = (V, E)$ be a graph. Throughout this chapter we refer to an edge of E as uv instead of $\{u, v\}$, for ease of presentation. Given a subset $S \subseteq V$, and two vertices $a, b \in V \setminus S$, we say that S *separates* a and b if any path between a and b contains a vertex in S. A subset $S \subset V$ is a *separator* in G if there exists two vertices $a, b \in V \setminus S$ such that S separates a and b in G. For an integer $c \geq 0$, G is *c-connected* if $|V| > c$ and no subset $V' \subseteq V$ with $|V'| < c$ is a separator in G. A *2-connected component* of G is a maximal 2-connected subgraph.

Let $(T, \mathcal{X})$ be any tree-decomposition of G. Abusing the notations, we will identify a node $t \in V(T)$ and its corresponding bag $X_t \in \mathcal{X}$. This means that, e.g., instead of saying $t \in V(T)$ is adjacent to $t' \in V(T)$ in T, we can also say that $X_t \in \mathcal{X}$ is adjacent to $X_{t'} \in \mathcal{X}$ in T. A bag $B \in \mathcal{X}$ is called a *leaf-bag* if B has degree one in T. Let G be a graph with $tw(G) \leq k$ ($k \geq 1$). A subset $B \subseteq V(G)$ is a *k-potential-leaf* if there is a tree-decomposition $(T, \mathcal{X})$ with width at most k and size $s_k(G)$ such that B is a leaf bag of $(T, \mathcal{X})$. A subgraph $H \subseteq V$ is a *k-potential-leaf* of G if $V(H)$ is a k-potential-leaf of G. Note that a k-potential-leaf has size at most $k + 1$. Given a class of graphs $\mathcal{C}$ and integer $k \in \mathbb{N}^*$, a set of graphs $\mathcal{P}$ is called a *complete set of k-potential-leaves* of $\mathcal{C}$, if for any graph $G \in \mathcal{C}$, there exists a graph $H \in \mathcal{P}$ such that H is a k-potential-leaf of G.

A tree-decomposition is *reduced* if no bag is contained in another one. It is straightforward that, in any leaf-bag B of a reduced tree-decomposition, there is $v \in V$ such that v appears only in B and so $N[v] \subseteq B$. Note that it implies that any reduced tree-decomposition has at most $n - 1$ bags.

In the following we define two *transformation rules* which take a tree-decomposition $(T, \mathcal{X})$ of a graph G, and computes another one without increasing the width nor the size.

Leaf. Let $X \in \mathcal{X}$ and $N_T(X) = \{X_1, \cdots, X_d\}$. Assume that, for any $1 < i \leq d$, $X_i \cap X \subseteq X_1$. Let $(T^*, \mathcal{X}^*) = Leaf(X, X_1, (T, \mathcal{X}))$ denote the tree-decomposition of G obtained by replacing each edge $X_iX \in E(T)$ by an edge X_iX_1 for any $1 < i \leq d$. Note that X becomes a leaf-bag after the operation. See in Figure B.4.

Reduce. Let $XX' \in E(T)$ with $X \subseteq X'$. Let $(T^*, \mathcal{X}^*) = Reduce(X, X', (T, \mathcal{X}))$ denote the tree-decomposition of G obtained by deleting the bag X from the tree-decomposition $Leaf(X, X', (T, \mathcal{X}))$. Note that the size of the tree-decomposition is decreased by one after the operation.

From any tree-decomposition of G with width k and size s, it is easy to obtain a reduced tree-decomposition of G with width at most k and size at most $s - 1$ by

applying the Reduce operation if it is possible (i.e., if a bag is contained in another one). In particular, any minimum size tree-decomposition is reduced.

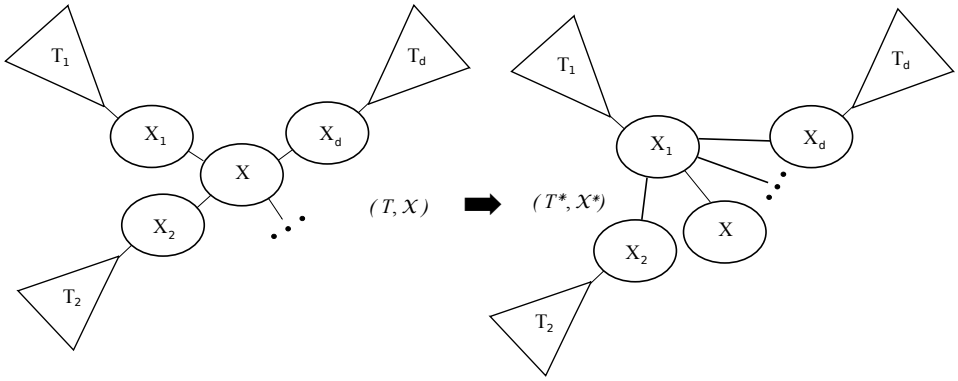


Figure B.4: In a tree-decomposition $(T, \mathcal{X})$, $N_T(X) = \{X_1, \cdots, X_d\}$ and for any $1 < i \leq d$, $X_i \cap X \subseteq X_1$. For $1 \leq i \leq d$, $T_i \cup X_i$ induces the subtree containing X_i in $T \setminus \{XX_i\}$. We replace each edge $X_iX \in E(T)$ by an edge X_iX_1 for any $1 < i \leq d$. This gives a tree-decomposition $(T^*, \mathcal{X}^*) = Leaf(X, X_1, (T, \mathcal{X}))$. X is a leaf-bag in $(T^*, \mathcal{X}^*)$.

We conclude this section by a general lemma on tree-decompositions. This lemma is known as folklore, we recall it for completness.

Lemma 11. *Let $(T, \mathcal{X})$ be a tree-decomposition of a graph G. Let $X \in \mathcal{X}$ and $v, w \in X$. If there exists a connected component in $G \setminus X$ containing a neighbor of v and a neighbor of w, then there is a neighbor bag of X in $(T, \mathcal{X})$ containing v and w.*

Proof. First, let us note that, for any connected subgraph H of G, the set of bags of T which contain a vertex of H induces a subtree of T (the proof can be done by induction on $|V(H)|$).

Let C be a connected component in $G \setminus X$ containing a neighbor of v and a neighbor of w. Let T'_C be the subtree of T induced by the bags that contain at least a vertex of C. Because no vertices of C are contained in the bag X, then T'_C is a subtree of $T \setminus X$. Let T_C be the connected component of $T \setminus X$ that contains T'_C. Let $Y \in V(T_C)$ be the bag of T_C which is a neighbor of X in T. Let $x \in N(v) \cap C$ be a neighbor of v in C. Then there exists a bag $Z \in \mathcal{X}$ in T_C containing both x and v. So both X and Z contain vertex v. Then the bag Y, which is on the path between X and Z in T, also contains v. Similarly, we can prove that $w \in Y$. □

Corollary 8. *Let $(T, \mathcal{X})$ be a tree-decomposition of a 2-connected graph G. Let $X \in \mathcal{X}$ and $|X| \leq 2$. Then there is a neighbor bag Y of X in $(T, \mathcal{X})$ such that $X \subseteq Y$.*

Proof. Since G is 2-connected, $|V(G)| \geq 3$. So there exist at least another bag except X in $\mathcal{X}$.

If $|X| = 1$, let $X = \{v\}$. Then there is a neighbor bag Y of X containing v, since G is 2-connected and v is adjacent to some vertices in G. If $X = 2$, let $X = \{v, w\}$. Let G_1 be any connected component in $G \setminus X$. If v is not adjacent to any vertex in G_1, then $\{w\}$ separates $V(G_1)$ from $\{v\}$. This contradicts with the assumption that G is 2-connected. So any connected component in $G \setminus X$ contains a neighbor of v and a neighbor of w. From Lemma 11, there is a neighbor bag Y of X containing v, w, i.e. $X \subseteq Y$. □

B.3.2 General approach

In what follows, we present the general approach used to design polynomial-time algorithms to compute minimum-size tree-decompositions of graphs with small treewidth. Our algorithms mainly use the notion of potential-leaf.

Let $k \geq 1$ and $G = (V, E)$ be a graph with $tw(G) \leq k$. The key idea of our algorithms is to identify a finite complete set of potential-leaves. Then, our algorithms are recursive: given a graph G and a k-potential-leaf H from the complete set, we compute a minimum-size tree-decomposition of G by adding H to a minimum-size tree-decomposition of a smaller graph.

The next lemmas formalize the above paragraph.

Lemma 12. *Let $k \geq 1$ and $G = (V, E)$ be a graph with $tw(G) \leq k$. Let $B \subseteq V$ be a k-potential-leaf of G and $S \subset B$ be the set of vertices of B that have a neighbor in $V \setminus B$. Then $s_k(G) = s_k(G_S \setminus (B \setminus S)) + 1$.*

Proof. Let us first prove $s_k(G) \leq s_k(G_S \setminus (B \setminus S)) + 1$. Suppose that $(T_S, \mathcal{X}_S)$ is a minimum size tree-decomposition of width at most k of the graph $G_S \setminus (B \setminus S)$. Then there exists a bag $X \in \mathcal{X}_S$ containing S because S induces a clique in the graph $G_S \setminus (B \setminus S)$. We add the bag B and make it adjacent to X in the tree-decomposition $(T_S, \mathcal{X}_S)$. We obtain then a tree-decomposition of width at most k for graph G of size $s_k(G_S \setminus (B \setminus S)) + 1$.

Now we prove that $s_k(G) \geq s_k(G_S \setminus (B \setminus S)) + 1$. Let $(T, \mathcal{X})$ be a minimum size tree-decomposition of G of width at most k such that B is a leaf bag in $(T, \mathcal{X})$. Note that, if $B = V$ then $G_S \setminus (B \setminus S)$ is the empty graph. Let us assume that $B \subset V$. Then $(T, \mathcal{X})$ is also a tree-decomposition of G_S. Let B be adjacent to the bag Y in $(T, \mathcal{X})$. Then $S \subset Y$ since each vertex in S is contained in another bag in $(T, \mathcal{X})$. Let $(T', \mathcal{X}')$ be the tree-decomposition obtained by deleting the vertices in $B \setminus S$ in all the bags of $(T, \mathcal{X})$. Then B is changed to $B' = S \in \mathcal{X}'$ and let Y be changed to $Y' \in \mathcal{X}'$. So $B' \subseteq Y'$. Then the tree-decomposition $Reduce(B', Y', (T', \mathcal{X}'))$ is a tree-decomposition of $G_S \setminus (B \setminus S)$ of size $s_k(G) - 1$. So $s_k(G) - 1 \geq s_k(G_S \setminus (B \setminus S))$. □

This lemma implies the following corollary:

Corollary 9. *Let $k \in \mathbb{N}^*$ and $\mathcal{C}$ be the class of graphs with treewidth at most k. If there is a $g(n)$-time algorithm $\mathcal{A}_k$ that, for any n-vertex-graph $G \in \mathcal{C}$, computes a k-potential-leaf of G. Then s_k can be computed in $O(g(n) \cdot n)$ time in the class*

of n-vertex graphs in $\mathcal{C}$. Moreover, a minimum size tree-decomposition of width at most k can be constructed in the same time.

Proof. Let $G \in \mathcal{C}$ be a n-vertex-graph. Let us apply Algorithm $\mathcal{A}_k$ to find a subgraph H of G in $g(n)$ time, which is a k-potential-leaf of G. Let $S \subset V(H)$ be the set of vertices having a neighbor in $G \setminus H$ and $G' = G_S \setminus (V(H) \setminus S)$. Then, by Lemma 12, $s_k(G) = s_k(G') + 1$. Finally, $|V(G')| \leq n - 1$ and G' has treewidth at most k. We then proceed recursively. So the total time complexity is $O(g(n) \cdot n)$. Moreover, for any minimum size ($s_k(G')$) tree-decomposition $(T', \mathcal{X}')$ of G' of width k, there is a bag X containing S since S induces a clique in G'. Add a new bag $N = V(H)$ adjacent to X in $(T', \mathcal{X}')$. The obtained tree-decomposition is a minimum size ($s_k(G) = s_k(G') + 1$) tree-decomposition of G of width at most k. □

B.4 Graphs with treewidth at most 2

In this section, we describe the algorithm $\mathcal{A}_2$ which computes a 2-potential-leaf of a given graph. In particular, all graphs considered in this section have treewidth at most 2, i.e. partial 2-trees. Please see a complete set of 2-potential-leaves of graphs of treewidth at most 2 in Figure B.5. We are going to prove that any of the subgraphs in Figure B.5 is a potential-leaf and then that each non-empty graph of treewidth at most 2 contains one of them as a 2-potential-leaf.

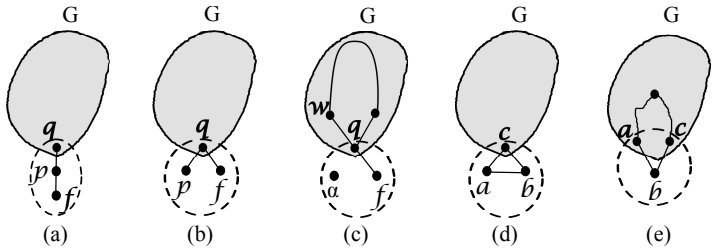


Figure B.5: Complete set of 2-potential-leaves of graphs of treewidth at most 2.

Lemma 13. *Let G be a graph with treewidth at most 2 and $p \in V(G)$ such that $N(p) = \{f, q\}$ and f has degree one (see Figure B.5(a)). Then $\{f, p, q\}$ is a 2-potential-leaf of G.*

Proof. Let $(T, \mathcal{X})$ be any tree-decomposition of G with width at most 2 and size at most $s \geq 1$. We show how to modify $(T, \mathcal{X})$ to obtain a tree-decomposition with width at most 2 and size at most s and in which $\{f, p, q\}$ is a leaf bag.

Since $fp \in E(G)$, there is a bag B in $(T, \mathcal{X})$ containing both f and p. We may assume that B is the single bag containing f (otherwise, we delete f from any other bag). Similarly, since $pq \in E(G)$, let X be a bag in $(T, \mathcal{X})$ containing both p and q.

First, let us assume that $X = B = \{f, p, q\}$. In this case, we may assume that X is the single bag containing p (otherwise, we delete p from any other bag). If X is a leaf bag, then the lemma is proved. Otherwise, let $X_1, \cdots, X_d$ be the neighbors of

X in T. Since f and p appear only in X, then $X \cap X_i \subseteq \{q\}$ for any $1 \leq i \leq d$. By definition of the operation $Leaf$, the tree-decomposition $Leaf(X, X_1, (T, \mathcal{X}))$ has width at most 2 and the same size as $(T, \mathcal{X})$, and X is a leaf.

Second, consider the case when $X \neq B$. There are two cases to consider. Either $B = \{f, p\}$ or $B = \{f, p, x\}$ with $x \neq q$. In the latter case, note that there is another bag B', neighbor of B, that contains x unless x is an isolated vertex of G. In the former case or if x appears only in B (in which case, x is an isolated vertex), let B' be any neighbor of B. Let $(T', \mathcal{X}')$ be obtained by deleting f, p in all bags of $(T, \mathcal{X})$. Then, we contract the edge BB' in T', i.e., we remove B and make any neighbor of B adjacent to B'. Note that, in the resulting tree-decomposition of $G \setminus \{f, p\}$, there is a bag X' containing q and with $|X'| \leq 2$ (the bag that results from X). Finally, we add a bag $\{f, p, q\}$ adjacent to X' and, if node x was only in B, then we add x to X'. The result is the desired tree-decomposition. □

Lemma 14. *Let G be a graph with treewidth at most 2 and $q \in V(G)$ such that q has at least two one-degree neighbors f and p (see Figure B.5(b)). Then $\{f, p, q\}$ is a 2-potential-leaf of G.*

Proof. Let $(T, \mathcal{X})$ be any tree-decomposition of G with width at most 2 and size at most $s \geq 1$. We show how to modify $(T, \mathcal{X})$ to obtain a tree-decomposition with width at most 2 and size at most s and in which $\{f, p, q\}$ is a leaf bag.

Since $fq \in E(G)$, there is a bag B in $(T, \mathcal{X})$ containing both f and q. We may assume that B is the single bag containing f (otherwise, we delete f from any other bag). Similarly, since $pq \in E(G)$, let X be a bag in $(T, \mathcal{X})$ containing both p and q. Again, we may assume that X is the single bag containing p (otherwise, we delete p from any other bag).

First, let us assume that $X = B = \{f, p, q\}$. If X is a leaf bag, then the lemma is proved. Otherwise, let $X_1, \cdots, X_d$ be the neighbors of X in T. Since f and p appear only in X, then $X \cap X_i \subseteq \{q\}$ for any $1 \leq i \leq d$. By definition of the operation $Leaf$, the tree-decomposition $Leaf(X, X_1, (T, \mathcal{X}))$ has width at most 2, and the same size as $(T, \mathcal{X})$, and X is a leaf.

Second, let us assume that $X = \{f, q\}$ or $B = \{p, q\}$. In the former case, we remove p from any bag and add p to X. In the latter case, we remove f from any bag and add f to B. In both cases, we obtain a bag $\{f, p, q\}$ as in the first case.

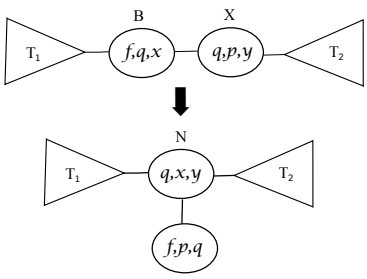
Otherwise, let $B = \{f, q, x\}$, $x \neq p$, and $X = \{p, q, y\}$, $y \neq f$.

- If B and X are adjacent in T, then we add a new bag $N = \{q, x, y\}$, remove B and X and make each of their neighbors adjacent to the new bag N and, finally, add a leaf-bag $\{f, p, q\}$ adjacent to N. see Figure B.6a. The obtained tree-decomposition has the desired properties.

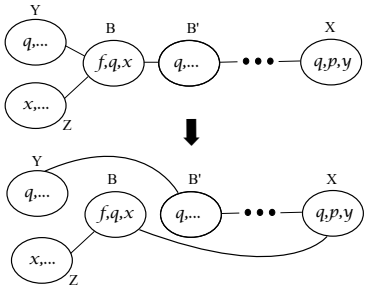
- Otherwise, if there is a neighbor B' of B with $q, x \in B'$, then we remove B, make all neighbors of B adjacent to B' and finally add a leaf-bag $\{f, p, q\}$ adjacent to X. The obtained tree-decomposition has the desired properties.

- Otherwise, let B' be the neighbor of B on the path between B and X. In this case, $q \in B'$ and $x \notin B'$. Moreover, q does not belong to any neighbor of B that contains x and the other way around. For any neighbor Y of B with $q \in Y$ (and hence $x \notin Y$), we replace the edge $YB \in E(T)$ with the edge YB'. Finally, we replace the edge $BB' \in E(T)$ by the edge BX. See Figure B.6b. In the resulting tree-decomposition of G, B and X are adjacent and we are back to the first case.

□



(a) In the tree-decomposition $(T, \mathcal{X})$, let $T_1 \cup B$ (resp. $T_2 \cup B$) induce the subtree containing B (resp. X) in $T \setminus \{B\}$. We delete B and X, make each of their neighbors adjacent to the new bag $N = \{q, x, y\}$, and add a leaf-bag $\{f, p, q\}$ adjacent to N.



(b) For the sake of simplicity, we show only the induced path from B to X in T and two neighbors $Y, Z \neq B'$ of B. Y contains q and Z contains x. Then we just make Y adjacent to B' instead of B and make B adjacent to X instead of B'.

Figure B.6: Examples illustrating the proof of Lemma 14.

Lemma 15. *Let G be a graph with treewidth at most 2 and $q \in V(G)$ such that q has one neighbor f with degree 1 and for any vertex $w \in N(q) \setminus \{f\}$, $\{w, q\}$ belongs to a 2-connected component of G.*

If G has an isolated vertex α, then $\{q, f, \alpha\}$ is a 2-potential-leaf; otherwise $\{q, f\}$ is a 2-potential-leaf (see Figure B.5(c)).

Proof. Let $(T, \mathcal{X})$ be any tree-decomposition of G with width at most 2 and size at most $s \geq 1$. We show how to modify $(T, \mathcal{X})$ to obtain a tree-decomposition with width at most 2 and size at most s and in which $\{f, q, \alpha\}$ is a leaf bag if G has an isolated vertex α; and $\{f, q\}$ is a leaf bag otherwise.

Since $fq \in E(G)$, there is a bag B in $(T, \mathcal{X})$ containing both f and q. We assume that B is the single bag containing f (otherwise, delete f from any other bag).

1. If $B = \{f, q\}$, then the intersection of B and any of its neighbor in T is empty or $\{q\}$. If there is a neighbor of B containing q, then let X be such a neighbor; otherwise let X be any neighbor of B. By definition of the operation

$Leaf$, the tree-decomposition $Leaf(B, X, (T, \mathcal{X}))$ has width at most 2, same size as $(T, \mathcal{X})$, and B is a leaf. If there are no isolated vertices, we are done. Otherwise, if there is an isolated vertex α in G, then we delete α in all bags of the tree-decomposition $Leaf(B, X, (T, \mathcal{X}))$ and add α to bag B, i.e. make $B = \{f, p, \alpha\}$. The result is the desired tree-decomposition.

2. Otherwise let $B = \{f, q, x\}$.

   (a) If x is a neighbor of q, then x and q are in a 2-connected component of G. So there exists a connected component in $G \setminus B$ containing a vertex adjacent to x and a vertex adjacent to q. From Lemma 11, there is a neighbor X of B in $(T, \mathcal{X})$ containing both x and q. By definition of the operation $Leaf$, the tree-decomposition $Leaf(B, X, (T, \mathcal{X}))$ has width at most 2, same size as $(T, \mathcal{X})$, and B is a leaf. Then we delete x in B, i.e. $B = \{f, q\}$. Finally, if α is an isolated vertex of G, we remove it from any other bag and add it to B. The result is the desired tree-decomposition.

   (b) If x is not adjacent to q. If there is a neighbor X of B in $(T, \mathcal{X})$ containing both x and q, then $(T, \mathcal{X})$ is modified as in case 2a. Otherwise, any neighbor of B in $(T, \mathcal{X})$ contains at most one of the vertices q and x.

   If there is a neighbor of B in T containing q, then let Y be such a neighbor of B; otherwise let Y be any neighbor of B. We delete the edges between B and all its neighbors not containing x except Y in $(T, \mathcal{X})$ and make them adjacent to Y.

   If there is no neighbor of B containing x, then x is an isolated vertex and we obtain a tree-decomposition of the same size and width as $(T, \mathcal{X})$, in which there is a leaf bag $B = \{f, q, x\}$. It is a required tree-decomposition.

   Otherwise, let Z be a neighbor of B in $(T, \mathcal{X})$ containing x, then we delete the edges between B and all its neighbors containing x except Z in $(T, \mathcal{X})$ and make them adjacent to Z. Now B has only two neighbors Y and Z and $B \cap Y \subseteq \{q\}$, $B \cap Z = \{x\}$ and $Y \cap Z = \emptyset$. We delete the edge between B and Z and make Z adjacent to Y. We delete x in B, i.e. make $B = \{f, q\}$. See the transformations in Figure B.7. Then we obtain a tree-decomposition of the same size and width as $(T, \mathcal{X})$, in which $B = \{f, q\}$ is a leaf bag. Again, if α is an isolated vertex of G, we remove it from any other bag and add it to B. The result is the desired tree-decomposition.

□

Lemma 16. *Let G be a graph of treewidth at most 2. Let $b \in V(G)$ with $N(b) = \{a, c\}$. If $N(a) = \{b, c\}$ (see Figure B.5(d)) or if there is a path, with at least one internal vertex, between a and c in $G \setminus \{b\}$ (see Figure B.5(e)), then $\{a, b, c\}$ is a 2-potential-leaf of G.*

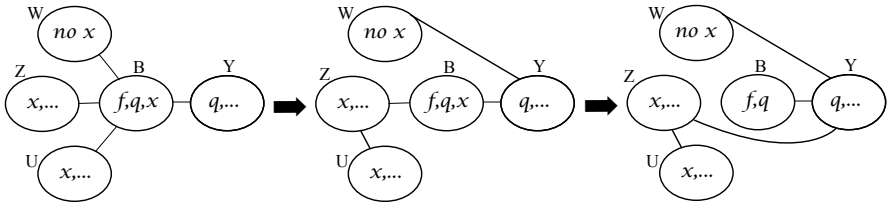


Figure B.7: To the sake of simplicity, we show only the subtree induced by B, Y and three neighbors Z, W, U of B. Y contains q; Z, U both contain x and W does not contain x. First we make the bag not containing x, e.g. W adjacent to Y instead of B; and make the bag containing x except Z, e.g. U adjacent to Z instead of B. Second, we make Z adjacent to Y instead of B and delete x in B. Then $B = \{f, q\}$ is a leaf-bag.

Proof. Let $G = (V, E)$ be a graph of treewidth at most 2. Let $b \in V$ with exactly 2 neighbors $a, c \in V$ satisfying the hypotheses of the lemma. If $V = \{a, b, c\}$, the result holds trivially, so let us assume that $|V| \geq 4$.

Let $(T, \mathcal{X})$ be a reduced tree-decomposition of width at most 2 of G. From $(T, \mathcal{X})$, we will compute a tree-decomposition $(T^*, \mathcal{X}^*)$ of G without increasing the width nor the size and such that $\{a, b, c\}$ is a leaf-bag of $(T^*, \mathcal{X}^*)$.

Let X be any bag of $(T, \mathcal{X})$ containing $\{a, b\}$ and Y be any bag containing $\{b, c\}$. The bags X, Y exist because $ab, bc \in E$. If $X = \{a, b\}$, then there exists a connected component in $G \setminus X$ containing a neighbor of a and a neighbor of b. By Lemma 11, there is a neighbor of X in $(T, \mathcal{X})$ that contains both a and b, contradicting the fact that $(T, \mathcal{X})$ is reduced. So $|X| = 3$ and, similarly, $|Y| = 3$.

- Let us first assume that $X = Y = \{a, b, c\}$. In particular, it is the case when $N(a) = \{b, c\}$ since $\{a, b, c\}$ induces a clique. We may assume that b only belongs to bag X (otherwise, we remove b from any other bag).

  If $N(a) = \{b, c\}$, then we can also assume that a only belongs to X. Let Z be any neighbor of B containing c if it exists; otherwise let Z be any neighbor of B (Z exists since $|V| \geq 4$).

  Otherwise, there exists a path P between a and c in $G \setminus \{b\}$ with at least one internal vertex. In this latter case, there exists a connected component in $G \setminus X$ containing a neighbor of a and a neighbor of c. So by Lemma 11, there is a neighbor bag Z of X in $(T, \mathcal{X})$ containing both a and c. In both cases, $Leaf(X, Z, (T, \mathcal{X}))$ is the desired tree-decomposition.

- $X = \{a, b, x\}$ and $Y = \{b, c, y\}$ with $x \neq c$ and $y \neq a$; and there exists a path P between a and c in $G \setminus \{b\}$ with at least one internal vertex. Let Q be the path between X and Y in $(T, \mathcal{X})$. We may assume that b only belongs to the bags in Q, because otherwise b can be removed from any other bag.

  - If X is adjacent to Y, then by properties of tree-decomposition, $X \cap Y$ separates a and c. Since $\{b\}$ does not separate a and c, $X \cap Y = \{b, x\}$,

i.e. $x = y$. In this case, $(T^*, \mathcal{X}^*)$ is obtained by making $X = \{a, c, x\}$ and removing Y from $(T, \mathcal{X})$, then making all neighbors of Y adjacent to X and finally, adding a bag $\{a, b, c\}$ adjacent to X.

– Otherwise, let X' be the bag in the path Q containing a, which is closest to Y. Similarly, let Y' be the bag in the path Q containing c, which is closest to X. Finally, let Q' be the path from X' to Y' in T and note that b belongs to each bag in Q' and a and c do not belong to any internal bag in Q'. Also we may assume that b only belongs to the bags in Q', because otherwise b can be removed from any other bag.

  If X' and Y' are adjacent in T, the proof is similar to the one in previous item. Otherwise, let Z be the neighbor of X' in Q'. By properties of tree-decompositions, $X' \cap Z$ separates a and c. Since $\{b\}$ does not separate a and c, let $X' \cap Z = \{b, x'\}$. Since $Z \neq \{b, x'\}$ because $(T, \mathcal{X})$ is reduced, then $Z = \{b, x', z\}$ for some $z \in V$. We replace b with a in all the bags. By doing this $(T, \mathcal{X})$ is changed to a tree-decomposition $(T^c, \mathcal{X}^c)$ of the graph G/ab obtained by contracting the edge ab in G. In $(T^c, \mathcal{X}^c)$, the bag X' has become $X^c = \{a, x'\}$ and Z is changed to be $Z^c = \{a, x', z\}$. So X^c can be reduced in $(T^c, \mathcal{X}^c)$. Moreover Y is changed to $Y^c = \{a, c, y\}$. To conclude, let us add the bag $\{a, b, c\}$ adjacent to Y^c in the tree-decomposition $Reduce(X^c, Z^c, (T^c, \mathcal{X}^c))$. see Figure B.8. The result is the desired tree-decomposition $(T^*, \mathcal{X}^*)$ of G.

□

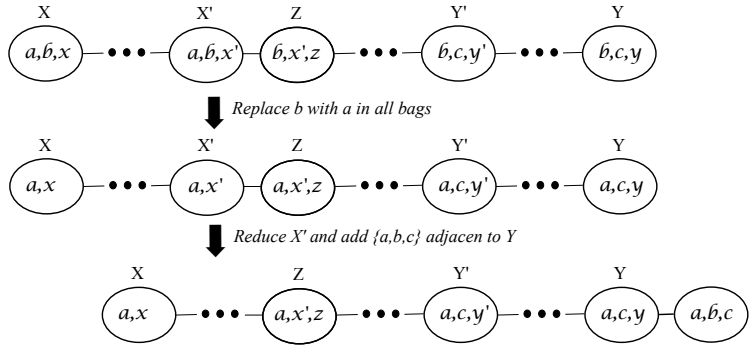


Figure B.8: For the sake of simplicity, we show only the path from X to Y. After the two transformations, $\{a, b, c\}$ is a leaf-bag.

Before going further, let us introduce some notations. A *bridge* in a graph $G = (V, E)$ is any subgraph induced by two adjacent vertices u and v of G (i.e., $uv \in E$) such that the number of connected components strictly increases when deleting the edge uv, but not the two vertices u, v in G, i.e., $G' = (V, E \setminus \{uv\})$ has strictly more connected components than G. A vertex $v \in V$ is a *cut vertex* if $\{v\}$ is a separator in G. A maximal connected subgraph without a cut vertex is called a

block. Thus, every block of a graph $G = (V, E)$ is either a 2-connected component of G or a bridge or an isolated vertex. Conversely, every such subgraph is a block. Different blocks of G intersect in at most one vertex, which is a cut vertex of G. Hence, every edge of G lies in a unique block, and G is the union of its blocks.

Let $G = (V, E)$ be a connected graph and let $r \in V$. A spanning tree T of G is a BFS-tree of G if for any $v \in V(G)$, the distance from r to v in G is the same as the one in T. Let $\mathcal{B} = \{C : C \text{ is a block of } G\}$. The *block graph* of G is the graph $B(G)$ whose vertices are the blocks of G and two block-vertices of $B(G)$ are adjacent if the corresponding blocks intersect, that is, $B(G) = (\mathcal{B}, \{C_1C_2 : C_1, C_2 \in \mathcal{B} \text{ and } C_1 \cap C_2 \neq \emptyset\})$. Note that $B(G)$ is connected. Finally, a *block-tree* of G is any BFS-tree F (with any arbitrary root) of $B(G)$. See an example in Figure B.9.

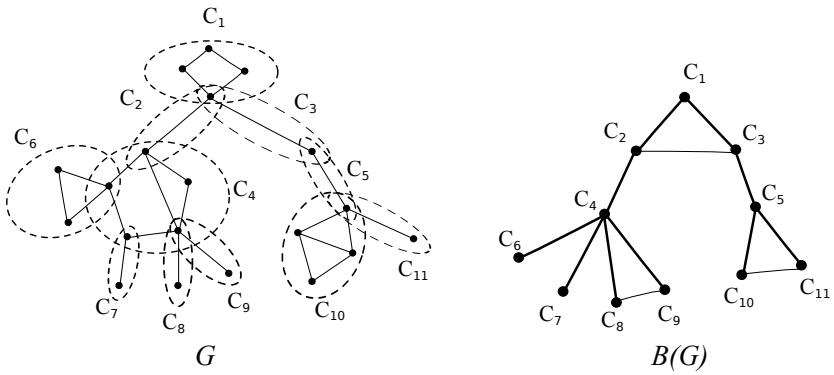


Figure B.9: Graph G is connected. For $i = 1, \ldots, 11$, each C_i is a block of G. $B(G)$ is the block graph of G. The BFS tree of $B(G)$ with bold edges is a block tree of G with root C_1.

There is a linear (in the number of edges) algorithm for computing all blocks in a given graph [HT73]. Also a BFS-tree can be found in linear (in the number of vertices plus the number of edges) time. So given a graph $G = (V, E)$, we can compute a block tree F of G in $O(|V| + |E|)$ time.

Now we are ready to prove the next theorem by using the Lemmas 13-16.

Theorem 31. *There is an algorithm that, for any n-vertex-m-edge-graph G with treewidth at most 2, computes a 2-potential-leaf of G in time $O(n + m)$.*

Proof. If $n \leq 3$, then $V(G)$ is a 2-potential-leaf of G. Let us assume that $n \geq 4$. First, let us compute the set of isolated vertices in G, which can be done in $O(n)$ time. If G has only isolated vertices, then any three vertices induce a 2-potential-leaf of G. Otherwise, there is at least one edge in G.

Let G_1 be any connected component of G containing at least one edge. If $|V(G_1)| = 2$, then from Lemma 16, either G has an isolated vertex α and $\{\alpha, u, v\}$ is a 2-potential-leaf or $\{u, v\}$ is a 2-potential leaf.

Otherwise, $|V(G_1)| \geq 3$. We compute a block tree F of G_1 rooted in an arbitrary block R. This can be done in time $O(n + m)$. Note that any node in F corresponds

to either a 2-connected component of G or a bridge $uv \in E(G)$. Let C be a leaf block in F, which is furthest from R and $|V(C)|$ is maximum. There are several cases to be considered.

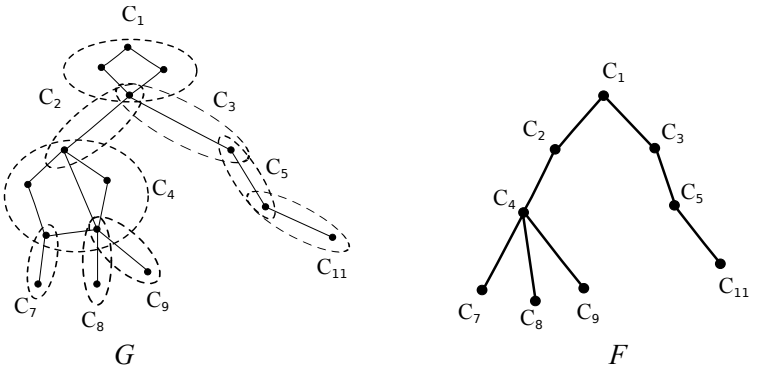


Figure B.10: This graph G is an induced subgraph of the graph in Figure B.9. Its block tree F, with root C_1, has two blocks less than the one in Figure B.9 (the blocks C_6 and C_{10}). Each one of the leaf blocks, C_7, C_8, C_9, C_{11}, in F contains two vertices of G.

- let us first assume that C is a bridge in G, i.e. C consists of one edge $fp \in E(G)$ and p is a cut vertex. Then f has degree one in G because C is a leaf block in F. Let P be the parent block of C in F. Then any child block A of P in F consists of one edge because C has the maximum number of vertices among all the children of P; and A is a leaf block in F because C is a furthest leaf from the root block R.

  If P has another child block except C in F containing the cut vertex p, then this child block also consists of one edge $f'p \in E(G)$, where f' has degree one in G because this child is also a leaf block in F. For example, in Figure B.10, we take C as C_8, which intersects C_9 with a cut vertex. From Lemma 14, $\{f, p, f'\}$ is a 2-potential-leaf.

  Otherwise, P has only one child block C in F containing the cut vertex p. Then any vertex in $N_G(p) \setminus \{f\}$ belongs to P. If P is also a bridge in G, i.e., P consists of one edge $pq \in E(G)$, then p has degree 2 in G. (For example, in Figure B.10, take C as C_{11}, whose parent C_5 is also a bridge in G.) From Lemma 13, $\{f, p, q\}$ is a 2-potential-leaf of G. Otherwise, P is a 2-connected component of G and $p \in V(G)$ satisfies the hypothesis of Lemma 15. For example, in Figure B.10, we take C as C_7, whose parent C_4 is a 2-connected component of G. Hence, either G has an isolated vertex α and $\{\alpha, f, p\}$ is a 2-potential-leaf or $\{f, p\}$ is a 2-potential-leaf.

- Finally, let us assume that C is a 2-connected component of G. It is known that any graph with at least two vertices of treewidth k contains at least two vertices of degrees at most k [BK11]. There is no degree one vertex in

C because C is 2-connected. So there exists two vertices with degree 2 in C. Since C is a leaf in F, there is only one cut vertex of G in C. So there exists a vertex b in C which has degree two in G. If $|V(C)| \geq 4$, then there exists a path between two neighbors a, c of b in $G \setminus \{b\}$ containing at least one internal vertex. For example, in Figure B.9, we take C as C_{10}. From Lemma 16, $\{a, b, c\}$ is a 2-potential-leaf. Otherwise C is a triangle $\{a, b, c\}$ with at least two vertices with degree 2 in G. Again from Lemma 16, $\{a, b, c\}$ is a 2-potential-leaf.

So the total time complexity is $O(n + m)$. □

Corollary 10. *s_2 can be computed in polynomial-time in general graphs. Moreover, a minimum size tree-decomposition can be constructed in polynomial-time in the class of partial 2-trees.*

Proof. Let G be any graph. It can be checked in polynomial-time whether $tw(G) \leq 2$ (e.g. see [WC83]). If $tw(G) > 2$, then $s_2 = \infty$. Otherwise $tw(G) \leq 2$, then the result follows from Theorem 31 and Corollary 9. □

B.5 Minimum-size tree-decompositions of width at most 3

In this section, we present algorithms to compute s_3 in the class of trees and 2-connected outerplanar graphs.

B.5.1 Computation of s_3 in trees

In this subsection, given a tree G, we show how to find a 3-potential-leaf in G. We characterize a complete set of 3-potential-leaves of trees in Figure B.11. We first prove that each of the subgraphs in Figure B.11 is a 3-potential-leaf and then that any tree with at least four vertices contains one of them.

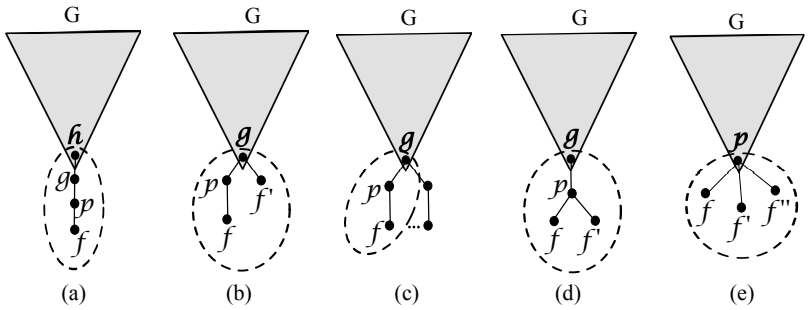


Figure B.11: Complete set of 3-potential-leaves of trees.

Lemma 17. *Let $(T, \mathcal{X})$ be a tree-decomposition of a tree G. Let $X \in \mathcal{X}$ and $N_T(X) = \{X_1, \ldots, X_d\}$, $d \geq 1$. Suppose that for any $1 \leq i \leq d$, $X_i \cap X \subseteq \{x\}$. Then there is a tree-decomposition $(T', \mathcal{X}')$ of G of the same width and size as $(T, \mathcal{X})$ such that X is a leaf bag.*

Proof. If there is a bag X_i for $1 \leq i \leq d$ containing x, then let B be X_i. Otherwise let B be any neighbor of X. By definition of the operation $Leaf$, the tree-decomposition $Leaf(X, B, (T, \mathcal{X}))$ is the desired tree-decomposition. □

Lemma 18. *Let G be a tree rooted at $r \in V(G)$. Let f be a leaf in G, p be the parent of f and g be the parent of p in G. Let p have degree 2 in G. Let $(T, \mathcal{X})$ be a tree-decomposition of G of width at most 3 and size at most $s \geq 1$. If there is no bag in $(T, \mathcal{X})$ containing all of f, p, g, then there is a tree-decomposition $(T', \mathcal{X}')$ of G of width at most 3 and size at most s such that $\{f, p, g\} \in \mathcal{X}'$ is a leaf bag.*

Proof. Since $fp \in E(G)$, there is a bag B in $(T, \mathcal{X})$ containing both f and p. We may assume that B is the single bag containing f (otherwise, we delete f from any other bag). Similarly, since $pg \in E(G)$, let X be a bag in $(T, \mathcal{X})$ containing both p and g. Let P be the path in T from B to X. Then p is contained in all bags on P and we may assume that p is not contained in any other bags (otherwise, we delete p from any other bag). Let B' be the neighbor of B on P. Then $\{p\} \subseteq B \cap B'$. Note that it is possible that $B' = X$.

If $B = \{f, p\}$, then we make all other neighbors of B adjacent to B' and delete B. We add a bag $\{f, p, g\}$ adjacent to X. The result is the desired tree-decomposition $(T', \mathcal{X}')$.

Otherwise, B contains at least one vertex not in $\{f, p\}$. If $B \cap B' = \{p\}$, then $\{p\}$ separates g from any vertex in $B \setminus \{p\}$. So $B \setminus \{p\} = \{f\}$, i.e., $B = \{f, p\}$. This contradicts the assumption.

So $|B \cap B'| \geq 2$ and let $\{p, x\} \subseteq B \cap B'$. Then we create a bag $Z = (B \setminus \{f, p\}) \cup (B' \setminus \{p, x\})$ (note that $x \in Z$ since $x \in B$.) So $|Z| \leq 4$. We make Z adjacent to all neighbors of B and all neighbors of B', delete the two bags B and B', and delete f, p from all bags. Finally, we add another new bag $N = \{f, p, g\}$ adjacent to some bag containing g. The obtained tree-decomposition has width at most 3, same size as $(T, \mathcal{X})$, and a bag $N = \{f, p, g\}$ as a leaf. □

Lemma 19. *Let G be a tree rooted at $r \in V(G)$ and $|V(G)| \geq 4$. Let f be a leaf in G, p be the parent of f and g be the parent of p in G. Suppose that both p and g have degree 2. Let h be the parent of g (see Figure B.11(a)), then $H = G[\{f, p, g, h\}]$ is a 3-potential-leaf of G.*

Proof. Let $(T, \mathcal{X})$ be any reduced tree-decomposition of width at most 3 and size at most $s \geq 1$ of G. We show how to modify $(T, \mathcal{X})$ to obtain a tree-decomposition with width at most 3 and size at most s and in which $\{f, p, g, h\}$ is a leaf bag.

From Lemma 18, we can assume that there is a bag B in $(T, \mathcal{X})$ containing all f, p, g. We may assume that B is the single bag containing f, p (otherwise, we delete

f, p from any other bag). Since $gh \in E(G)$, let Y be a bag in $(T, \mathcal{X})$ containing both h and g.

1. If $B = Y = \{f, p, g, h\}$, then the intersection of B and any of its neighbor in T is contained in $\{h\}$. A desired tree-decomposition can be obtained from Lemma 17.

2. If $B = \{f, p, g\}$, then the intersection of B and any of its neighbors in T is contained in $\{g\}$. From Lemma 17, there is a tree-decomposition $(T', \mathcal{X}')$ of the same width and size as the ones of $(T, \mathcal{X})$ such that $B = \{f, p, g\}$ is a leaf. Then we delete B in the tree-decomposition $Leaf(B, B', (T, \mathcal{X}))$ and add a new bag $N = \{f, p, g, h\}$ adjacent to Y. The obtained tree-decomposition has the desired properties.

3. Otherwise, $B = \{f, p, g, x\}$ where $x \neq h$. Then the intersection of B and any of its neighbor in T is contained in $\{g, x\}$. Let P be the path in T from B to Y. Then g is contained in all bags on P. Let B' be the neighbor of B on P. Note that it is possible that $B' = Y$. If $B \cap B' = \{g\}$, then $\{g\}$ separates h from x. So $x \in \{f, p\}$ i.e. $B = \{f, p, g\}$, a contradiction with the assumption. So we have $B \cap B' = \{g, x\}$. By definition of the operation $Leaf$, the tree-decomposition $Leaf(B, B', (T, \mathcal{X}))$ has width at most 3, same size as $(T, \mathcal{X})$, and $B = \{f, p, g, x\}$ is a leaf. Then we delete B in the tree-decomposition $Leaf(B, B', (T, \mathcal{X}))$ and add a new bag $N = \{f, p, g, h\}$ adjacent to Y. The obtained tree-decomposition has the desired properties since $\{g, x\} \subseteq B'$ and $\{g, h\} \subseteq Y$.

□

Lemma 20. *Let G be a tree rooted at $r \in V(G)$ and $|V(G)| \geq 4$. Let f be a leaf in G, p be the parent of f and g be the parent of p in G. If p has degree 2 and g has a child f', which is a leaf in G (see Figure B.11(b)), then $H = G[\{f, p, g, f'\}]$ is a 3-potential-leaf of G.*

Proof. Let $(T, \mathcal{X})$ be any reduced tree-decomposition of width at most 3 and size at most $s \geq 1$ of G. We show how to modify $(T, \mathcal{X})$ to obtain a tree-decomposition with width at most 3 and size at most s and in which $\{f, p, g, f'\}$ is a leaf bag.

From Lemma 18, we can assume that there is a bag B in $(T, \mathcal{X})$ containing all of the vertices f, p, g. We may assume that B is the only bag containing f, p (otherwise, we delete f, p from any other bag). Since $gf' \in E(G)$, let Y be a bag in $(T, \mathcal{X})$ containing both f and g. We may assume that Y is the single bag containing f' (otherwise, we delete f' from any other bag).

- If $B = Y = \{f, p, g, f'\}$, then the intersection of B and any of its neighbors in T is contained in $\{g\}$. A desired tree-decomposition can be obtained from Lemma 17.

- If $B = \{f, p, g\}$, then we delete f' in Y and add f' in B; we will be back then to the previous case.
- Otherwise, $B = \{f, p, g, x\}$ where $x \neq f'$. The intersection of B and any of its neighbors in T is contained in $\{g, x\}$. Let P be the path in T from B to Y. Then g is contained in all bags on P. Let B' be the neighbor of B on P. If $B \cap B' = \{g, x\}$, then by definition of the operation $Leaf$, the tree-decomposition $Leaf(B, B', (T, \mathcal{X}))$ has width at most 3, same size as $(T, \mathcal{X})$, and $B = \{f, p, g, x\}$ is a leaf. In the tree-decomposition $Leaf(B, B', (T, \mathcal{X}))$, we delete f' in Y, remove x from B, and add f' to B, i.e. make $B = \{f, p, g, f'\}$. The obtained tree-decomposition has the desired properties since $\{g, x\} \subseteq B'$. Otherwise, if $B \cap B' = \{g\}$. We delete f' from the bag Y, add x to Y, delete x from B, and add f' in B, i.e., make $B = \{f, p, g, f'\}$. Finally, we make all neighbors of B except B' adjacent to Y since now $\{g, x\} \subseteq Y$. The result is the desired tree-decomposition.

□

Lemma 21. *Let G be a tree rooted at $r \in V(G)$ and $|V(G)| \geq 3$. Let f be one of the furthest leaves from r, p be the parent of f and g be the parent of p in G. If g has degree at least 3 and any child of g has degree 2 in G (see Figure B.11(c)), then $H = G[\{f, p, g\}]$ is a 3-potential-leaf of G.*

Proof. Let $(T, \mathcal{X})$ be any reduced tree-decomposition of width at most 3 and size at most $s \geq 1$ of G. We show how to modify $(T, \mathcal{X})$ to obtain a tree-decomposition with width at most 3 and size at most s, and in which $\{f, p, g, f'\}$ is a leaf bag.

From Lemma 18, we can assume that there is a bag B in $(T, \mathcal{X})$ containing all the vertices f, p, g. We may assume that B is the only bag containing f, p (otherwise, we delete f, p from any other bag).

1. If $B = \{f, p, g\}$, then the intersection of B and any of its neighbors in T is contained in $\{g\}$. The desired tree-decomposition can be obtained from Lemma 17.
2. Otherwise, $B = \{f, p, g, x\}$. In this case, the intersection of B and any of its neighbors in T is contained in $\{g, x\}$.
   (a) If there is a neighbor B' of B such that $B \cap B' = \{g, x\}$, then by definition of the operation $Leaf$, the tree-decomposition $Leaf(B, B', (T, \mathcal{X}))$ has width at most 3, same size as $(T, \mathcal{X})$, and $B = \{f, p, g, x\}$ is a leaf. We delete x in B in the tree-decomposition $Leaf(B, B', (T, \mathcal{X}))$ since $\{g, x\} \subseteq B'$. The obtained tree-decomposition has the desired properties.
   (b) Otherwise any neighbor of B contains at most one of the vertices g and x. If x is not adjacent to g, then there is a connected component in $G \setminus B$ containing a neighbor of g and a neighbor of x. From Lemma 11, there exists a neighbor bag of B in $(T, \mathcal{X})$ containing g and x. This is a contradiction and x is hence adjacent to g in this case.

i. x is a child of g. Then x has exactly one child y, which is a leaf in G since f is one of the furthest leaves from r. Since $yx \in E(G)$, there is a bag Y in $(T, \mathcal{X})$ containing both y and x. We may assume that Y is the only bag containing y (otherwise, we delete y from any other bag). Since $\{g, x\} \subset B$ and any neighbor of B contains at most one of the vertices g and x, any bag except B contains at most one of the vertices g and x. Then $g \notin Y$ because $x \in Y$. The vertices y, x, g are hence not contained in one bag. From Lemma 18, we can modify $(T, \mathcal{X})$ to obtain a tree-decomposition $(T', \mathcal{X}')$ of width at most 3 and size at most s having a leaf bag $X = \{y, x, g\}$. Note that x (resp. y) plays the same role as p (resp. f) in G, i.e., g, p, f and g, x, y are symmetric in G. Hence, the result is the desired tree-decomposition.

ii. x is the parent of g. Let p' be another child of g and let f' be the child of p', which is a leaf in G. Let B' be the bag in $(T, \mathcal{X})$ containing both f' and p'. We may assume that B' is the only bag containing f' (otherwise, we delete f' from any other bag). Let X' be a bag containing both p' and g. Then we have $X' \neq B$ (because $p' \notin B$). Since $g \in X'$, any bag except B contains at most one of the vertices g and x, we have $x \notin X'$. In the following, we modify $(T, \mathcal{X})$ to obtain a tree-decomposition $(T', \mathcal{X}')$ with width at most 3 and size at most s having a bag $\{f', p', g\}$. We will be back then to case 1, since g, p, f and g, p', f' are symmetric in G.
If $B' = X' = \{f', p', g\}$ then, we are done. If $B' = X' = \{f', p', g, x'\}$. Then $x' \neq x$, which is the parent of g, since $x \notin X'$. So we can do as in case 2a or case 2(b)i.
Otherwise, $B' \neq X'$. From Lemma 18, we can modify $(T, \mathcal{X})$ to obtain a tree-decomposition with width at most 3 and size at most s having a leaf bag $\{f', p', g\}$.

□

Lemma 22. *Let G be a tree rooted at $r \in V(G)$ and $|V(G)| \geq 4$. Let f a leaf in G, p be the parent of f, and g be the parent of p. If p has exactly two children f, f' in G (see Figure B.11(d)), then $H = G[\{f, f', p, g\}]$ is a 3-potential-leaf of G.*

Proof. Let $(T, \mathcal{X})$ be any reduced tree-decomposition of width at most 3 and size at most $s \geq 1$ of G. We show how to modify $(T, \mathcal{X})$ to obtain a tree-decomposition with width at most 3 and size at most s and in which $\{f, f', p, g\}$ is a leaf bag.

Since $fp \in E(G)$, there is a bag B in $(T, \mathcal{X})$ containing both f and p. We may assume that B is the only bag containing f (otherwise, we delete f from any other bag). Similarly, let B' be the only bag in $(T, \mathcal{X})$ containing both f' and p. Let X be a bag containing both p and g.

1. If $B = B' = X = \{f, f', p, g\}$, then we can assume that B is the only bag containing p (otherwise, we delete p from any other bag). The intersection

of B and any of its neighbor in T is contained in $\{g\}$. The desired tree-decomposition can then be obtained from Lemma 17.

2. If $B = B' = \{f, f', p\}$, then the intersection of B and any of its neighbors in T is contained in $\{p\}$. Let Y be a neighbor of B in T containing p. By definition of the operation $Leaf$, the tree-decomposition $Leaf(B, Y, (T, \mathcal{X}))$ has width at most 3, same size as $(T, \mathcal{X})$, and $B = \{f, f', p\}$ is a leaf. We delete B and add a new bag $N = \{f, f', p, g\}$ adjacent to X. The result is the desired tree-decomposition.

3. If $B = B' = \{f, f', p, x\}$ and $x \neq g$, then the intersection of B and any of its neighbors in T is contained in $\{p, x\}$. Since $x \notin \{f, f', g\}$, p is not adjacent to x. There is a connected component in $G \setminus B$ containing a neighbor of p and a neighbor of x. From Lemma 11, there exists a neighbor bag of B in $(T, \mathcal{X})$ containing p and x. Let Y be such a neighbor of B in T. By definition of the operation $Leaf$, the tree-decomposition $Leaf(B, Y, (T, \mathcal{X}))$ has width at most 3, same size as $(T, \mathcal{X})$, and $B = \{f, f', p, x\}$ is a leaf. We delete x from B and obtain a tree-decomposition having a bag $\{f, f', p\}$. We will be back then to case 2.

4. If $B \neq B'$ and $|B| \leq 3$, then we delete f' from B and add f' to B. We will be back then to case 2 or 3. The proof is similar if $B \neq B'$ and $|B'| \leq 3$.

5. Otherwise, if $B \neq B'$ and $|B| = |B'| = 4$, let $B = \{f, p, x, y\}$ and $B' = \{f', p, x', y'\}$. Let P be the path in T from B to B'. Then p is contained in all bags on P. Let Y be the neighbor of B on P. If $B \cap Y = \{p\}$, then $\{p\}$ separates x from x'. But p is not a separator for any two vertices in $V(G) \setminus \{f, f'\}$. This is a contradiction. So w.l.o.g. we can assume that $B \cap Y \supseteq \{p, x\}$. We delete f, f', p in all bags of $(T, \mathcal{X})$, add a new bag $Z = \{x, y\} \cup Y \setminus \{p, x\}$ adjacent to all neighbors of the two bags B, Y and delete B and Y. Finally, we add another new bag $N = \{f, f', p, g\}$ adjacent to a bag containing g. The obtained tree-decomposition has the desired properties.

□

Lemma 23. *Let G be a tree rooted at $r \in V(G)$ and $|V(G)| \geq 4$. Let all children of p be leaves in G and p have at least three children f, f', f'' (see Figure B.11(e)). Then $H = G[\{p, f, f', f''\}]$ is a 3-potential-leaf of G.*

Proof. Let $(T, \mathcal{X})$ be any reduced tree-decomposition of width at most 3 and size at most $s \geq 1$ of G. We show how to modify $(T, \mathcal{X})$ to obtain a tree-decomposition with width at most 3 and size at most s and in which $\{p, f, f', f''\}$ is a leaf bag.

Since $fp \in E(G)$, there is a bag B in $(T, \mathcal{X})$ containing both f and p. We may assume that B is the only bag containing f (otherwise, we delete f from any other bag). Similarly, let B' (resp. B'') be the only bag in $(T, \mathcal{X})$ containing both f' (resp. f') and p.

1. If $B = B' = B'' = \{f, f', f'', p\}$, then the intersection of B and any of its neighbors in T is contained in $\{p\}$. A desired tree-decomposition can be obtained from Lemma 17.

2. If $B = B' = \{f, f', p\}$, then we delete f'' from B'' and add f'' to B. We will be back then to case 1. The proof is similar for $B = B'' = \{f, f'', p\}$ or $B' = B'' = \{f', f'', p\}$.

3. If $B = B' = \{f, f', p, x\}$ and $x \neq f''$, then the intersection of B and any of its neighbors in T is contained in $\{p, x\}$. If x is a child of p, then x is also a leaf in G and x play the same role as f''. We are then in case 1. Therefore, in the following we assume that x is not a child of p.

   If x is not the parent of p, then p is not adjacent to x. So there is a connected component in $G \setminus B$ containing a neighbor of p and a neighbor of x. From Lemma 11, there exists a neighbor bag of B in $(T, \mathcal{X})$ containing p and x. Let Y be such a neighbor of B in T. By definition of the operation $Leaf$, the tree-decomposition $Leaf(B, Y, (T, \mathcal{X}))$ has width at most 3, same size as $(T, \mathcal{X})$, and $B = \{f, f', p, x\}$ is a leaf. We delete x from B and obtain a tree-decomposition having a bag $\{f, f', p\}$. We are back then to case 2.

   Otherwise, if x is the parent of p, let P be the path in T from B to B''. Then p is contained in all bags on P. Let Y be the neighbor of B on P. If $B \cap Y = \{p, x\}$, then by definition of the operation $Leaf$, the tree-decomposition $Leaf(B, Y, (T, \mathcal{X}))$ has width at most 3, same size as $(T, \mathcal{X})$, and $B = \{f, f', p, x\}$ is a leaf. By deleting x from B we will be back to case 2. Otherwise, if $B \cap Y = \{p\}$, then $\{p\}$ separates x from all vertices in $B'' \setminus \{p\}$. All vertices in $B'' \setminus \{p\}$ are children of p and so they are leaves in G. We can assume then that any vertex in $B'' \setminus \{p\}$ is contained only in B'' (otherwise we can delete it in any other bag). We delete f, f' from B, add vertices of $B'' \setminus \{f'', p\}$ in B, and make $B'' = \{f, f', f'', p\}$. We will be back then to case 1.

   The cases $B = B'' = \{f, f'', p, x\}$ and $x \neq f'$ or $B' = B'' = \{f', f'', p, x\}$ and $x \neq f$ can be proved in a similar way.

4. Otherwise, no two vertices of f, f', f'' are contained in a same bag.

   If $|B| \leq 3$, then we delete f' in B' and add f' in B. We will be then in case 2 or 3. The proof is similar if $|B'| \leq 3$ or $|B''| \leq 3$.

   Otherwise $|B| = |B'| = |B''| = 4$. In the following, we are going to modify $(T, \mathcal{X})$ to obtain a tree-decomposition with width at most 3 and size at most s having a bag X containing at least two of the vertices f, f', f'' or $f \in X$ and $|X| \leq 3$. We are then in the above cases. Note that all children of p play the same role (they are all leaves) in G. So it is enough to have that X contains at least two children of p or that X contains one child of p and $|X| \leq 3$.

   Let T_p be the subtree in T induced by all the bags containing p. If $|V(T_p)| \leq 2$, there exists one bag containing at least two children of p since p has at least

three children. We assume then that $|V(T_p)| \geq 3$. There is a bag $R \in V(T_p)$ containing both p and g. We root T_p at R. Let $L \in V(T_p)$ be one of the furthest leaf bags in T_p from R. If there is no child of p in L, then we can delete p from L and consider $T_p \setminus \{L\}$. We can assume then that there is a vertex $l \in L$, which is a child of p in G. Let Y be the neighbor of L in T_p. If the intersection of $L \cap Y = \{p\}$, then p separate any vertex in $L \setminus \{p\}$ and any vertex in $Y \setminus \{p\}$. So at least one of the bags L, Y contains only p and children of p. We denote this bag by X. Either X contains at least two children of p or X contains only one children and $|X| = 2$. So $(T, \mathcal{X})$ and X satisfy the desired properties.

Otherwise, $|L \cap Y| \geq 2$. If Y has no other child except L in T_p, then $Y \neq R$ since $|V(T_p)| \geq 3$. Let $X = \{p, l\}$ if Y contains no child of p and $X = \{p, l, l'\}$ if Y contains one child l' of p. We add a new bag $Z = Y \cup L \setminus X$. Since $|Y \cap L| \geq 2$, we have $|Y \cup L| \leq 6$. Also $|Z| \leq 4$, since $X \subseteq Y \cup L$ and $|X| \geq 2$. We make Z adjacent to all neighbors of Y, L in T and delete Y, L. Finally, we make X adjacent to R. The obtained tree-decomposition and X have the desired properties.

Otherwise, Y has at least another child L' in T_p. Then L' is also a furthest leaf from R in T_p, since L is a furthest leaf from R. For the same reason as L, there is a vertex $l' \in L$, which is a child of p in G. Let $L = \{l, p, x, y\}$ and $L' = \{l', p, x', y'\}$. The intersection of L (resp. L') and any of its neighbors in T except Y is contained in $\{x, y\}$ (resp. $\{x', y'\}$). We create a new bag $N = \{x, y, x', y'\}$ adjacent to all neighbors of L and L' and delete L and L'. Finally, we add another bag $X = \{p, l, l'\}$ adjacent to Y. The obtained tree-decomposition and X have the desired properties.

□

From Lemmas 19- 23 and Corollary 9, we obtain the following result.

Corollary 11. *s_3 and a minimum size tree-decomposition of width at most 3 can be computed in polynomial-time in the class of trees.*

Proof. From Corollary 9, it is enough to prove that we can find a 3-potential-leaf in any tree in polynomial time.

Let G be any tree. If $|V(G)| \leq 4$, then $V(G)$ is a 3-potential-leaf. Let us assume that $|V(G)| \geq 5$. We root G at any vertex r. Let f be one of the furthest leaves from r in G. Let p, g, h be the first three vertices on the path from f to r in G (if they exist), i.e. p is f's parent; g is the parent of p, and h is the parent of g in G.

- If g, p both have only one child in G, then $\{f, p, g, h\}$ is a 3-potential-leaf of G from Lemma 19;
- If p has only one child and g has a child f', which is a leaf in G, then $\{f, p, g, f'\}$ is a 3-potential-leaf of G from Lemma 20;

- If p has only one child and any child of g has exactly one child, then $\{f, p, g\}$ is a 3-potential-leaf of G from Lemma 21;
- If p has only one child and there exists a child p' of g, which has exactly two children f_1, f_2, then $\{f_1, f_2, p', g\}$ is a 3-potential-leaf of G from Lemma 22;
- If p has only one child and there exists a child p' of g, which has at least three children f_1, f_2, f_3, then $\{f_1, f_2, f_3, p'\}$ is a 3-potential-leaf of G from Lemma 23;
- If p has exactly two children f, f', then $\{f, f', p, g\}$ is a 3-potential-leaf of G from Lemma 22;
- Otherwise, if p has at least three children f, f', f'', then $\{f, f', f'', p\}$ is a 3-potential-leaf of G from Lemma 23.

□

In fact, the algorithm for trees can be extended to forests by considering their connected component, i.e., trees. The only difference is in Lemma 21 the 3-potential-leaf becomes $\{f, p, g, \alpha\}$ if there is an isolated vertex α in the given forest.

B.5.2 Computation of s_3 in 2-connected outerplanar graphs

In this subsection, given a 2-connected outerplanar graph G, we show how to find a 3-potential-leaf in G. We give in Figure B.12 a complete set of 3-potential-leaves of 2-connected outerplanar graphs. We first prove that each subgraph in the Figure B.12 is a 3-potential-leaf and then we show that any 2-connected outerplanar graph contains one of them.

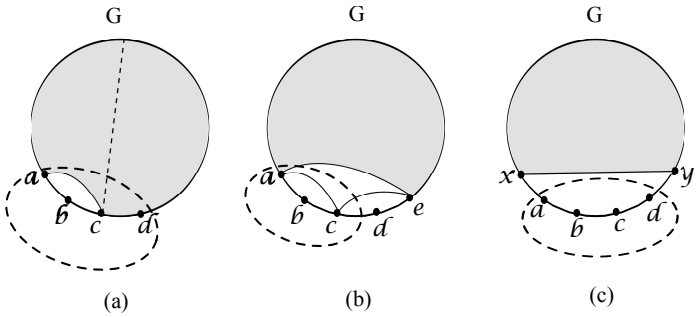


Figure B.12: Complete set of 3-potential-leaves of 2-connected outerplanar graphs.

The following fact is well known for 2-connected outerplanar graphs.

Lemma 24. *[Sys79] A 2-connected outerplanar graph has a unique Hamiltonian cycle.*

In the rest of this subsection, let G be a 2-connected outerplanar graph and C be the Hamiltonian cycle in G.

Definition 5. *Any edge in $E(G) \setminus E(C)$ is called a chord in G.*

The vertices $v_1, \ldots, v_j \in V(G)$, for $2 \leq j \leq |V(G)|$, are *consecutive* in C (we also say that they are consecutive in G) if $v_i v_{i+1} \in E(C)$ for $1 \leq i \leq j-1$.

Lemma 25. *Let $a, b, c, d \in V(G)$ be consecutive vertices in C. If $\{a, b, c\}$ induces a clique and c has degree 3 in G (see Figure B.12(a)), then $H = G[\{a, b, c, d\}]$ is a 3-potential-leaf of G.*

Proof. Let $(T, \mathcal{X})$ be any tree-decomposition of width at most 3 and size at most $s \geq 1$ of G. We show how to modify $(T, \mathcal{X})$ to obtain a tree-decomposition with width at most 3 and size at most s, and in which $\{a, b, c, d\}$ is a leaf bag.

Since $\{a, b, c\}$ induces a clique in G, there is a bag B containing all of the vertices a, b, c. Let X be a bag in $(T, \mathcal{X})$ containing both c and d (such bag exists since $cd \in E(G)$). Note that b is not incident to any chords, i.e. has degree 2. In fact, if $by \in E(G)$ is a chord in G, then deleting all chords except ac, by in G and contracting the edges in C except ab, bc we get a K_4-minor in G. This is a contradiction with the fact that G is outerplanar.

We replace vertices b, c with vertex a in all bags of $(T, \mathcal{X})$. Then $(T, \mathcal{X})$ becomes a tree-decomposition $(T', \mathcal{X}')$ of the graph G' obtained by contracting the edges ab and bc. The bag X becomes X', which contains both a and d, and B becomes $B' = \{a\}$ if $B = \{a, b, c\}$ or $B' = \{a, x\}$ if $B = \{a, b, c, x\}$. From Corollary 8, in both cases there exists a neighbor Y of B' such that $B' \subseteq Y$. So B' can be reduced in $(T', \mathcal{X}')$. The tree-decomposition $\mathit{Reduce}(B', Y, (T', \mathcal{X}'))$ has one bag less than $(T, \mathcal{X})$. Finally, add a new bag $N = \{a, b, c, d\}$ adjacent to X', which contained both a and d, in the tree-decomposition $\mathit{Reduce}(B', Y, (T', \mathcal{X}'))$. The result is the desired tree-decomposition, because b, c are not adjacent to any vertices in $V(G) \setminus N$. □

Lemma 26. *Let $a, b, c, d, e \in V(G)$ be consecutive vertices in C. If $\{a, b, c\}$ and $\{c, d, e\}$ induce two cliques respectively in G and $ae \in E(G)$ (see Figure B.12(b)), then $H = G[\{a, b, c\}]$ is a 3-potential-leaf of G.*

Proof. Let $(T, \mathcal{X})$ be any tree-decomposition of width at most 3 and size at most $s \geq 1$ of G. We show how to modify $(T, \mathcal{X})$ to obtain a tree-decomposition with width at most 3 and size at most s and in which $\{a, b, c\}$ is a leaf bag.

Since $\{a, b, c\}$ (resp. $\{c, d, e\}$) induces a clique in G, there is a bag X (resp. Y) containing all the vertices a, b, c (resp. c, d, e). Note that b, c, d are not adjacent to any vertices in $V(G) \setminus \{a, b, c, d, e\}$.

We delete b, c, d in all bags of $(T, \mathcal{X})$. Then $(T, \mathcal{X})$ becomes a tree-decomposition $(T', \mathcal{X}')$ of the graph $G' = G \setminus \{b, c, d\}$. The bag X becomes becomes $X' = \{a\}$ if $X = \{a, b, c\}$ or $X' = \{a, x\}$ if $B = \{a, b, c, x\}$. From Corollary 8, in both cases there exists a neighbor A of X' such that $X' \subseteq A$. So X' can be reduced in $(T', \mathcal{X}')$. Similarly, the bag Y becomes Y', which can also be reduced in $(T', \mathcal{X}')$. After reducing the two bags X', Y' in $(T', \mathcal{X}')$, let the obtained tree-decomposition be $(T'', \mathcal{X}'')$. Finally, add two new bags $N_1 = \{a, b, c\}$ and $N_2 = \{a, c, d, e\}$; make N_1 adjacent to N_2 and make N_2 adjacent to a bag Z containing both a and e in

the tree-decomposition $(T'', \mathcal{X}'')$ (Z exists because $ae \in E(G')$.) The result is the desired tree-decomposition. □

Lemma 27. *Let C_l be a cycle of $l \geq 4$ vertices. Let $(T, \mathcal{X})$ be a tree-decomposition of C_l of width at most 3. Then there exist either a bag containing all vertices of $V(C_l)$ (only if $l = 4$) or two bags $X, Y \in \mathcal{X}$ such that X (resp. Y) contains at least three consecutive vertices x_1, x_2, x_3 (resp. y_1, y_2, y_3) and the two edge sets $\{x_1x_2, x_2x_3\}$ and $\{y_1y_2, y_2y_3\}$ are disjoint i.e. $\{x_1x_2, x_2x_3\} \cap \{y_1y_2, y_2y_3\} = \emptyset$.*

Proof. The treewidth of any cycle is bigger than 1, so there exists a bag in any tree-decomposition of a cycle (with at least 4 vertices) containing two vertices which are not consecutive (not adjacent in the cycle). We prove the lemma by induction on l in the following.

First let us prove that it is true for $l = 4$. Let a, b, c, d be the four consecutive vertices in C_4. Let $(T, \mathcal{X})$ be a tree-decomposition of width at most 3. Then there exists a bag containing a, c or b, d. W.l.o.g we assume that a, c are contained in one bag. So $(T, \mathcal{X})$ is also a tree-decomposition of the graph H obtained from C_4 by adding the edge ac. The set $\{a, b, c\}$ induces a clique in H. So there is a bag X containing a, b, c. For the same reason, there is a bag Y containing c, d, a. If $X = Y$ then there is a bag containing all a, b, c, d of $V(C_4)$. Otherwise there are two bags X, Y such that $X \supseteq \{a, b, c\}$ and $Y \supseteq \{c, d, a\}$. We see that $\{ab, bc\} \cap \{cd, da\} = \emptyset$. So the lemma is true for $l = 4$.

Now, let us suppose it is true for $l \leq n - 1$ and prove it for $l = n \geq 5$. Note that since $(T, \mathcal{X})$ has width 3 and $l \geq 5$, there is no bag containing all vertices of $V(C_l)$. So in the following we prove that there always exist two bags X, Y with the desired properties. Let $v_1, \ldots, v_n$ be the n consecutive vertices in C_n. Let (T, X) be a tree-decomposition of width at most 3 of C_n. Then there exists a bag containing two non-adjacent vertices v_i, v_j for $1 \leq i < j \leq n$. So $(T, \mathcal{X})$ is also a tree-decomposition of the graph H obtained from C_n by adding the edge v_iv_j. The graph H is also the union of two subcycles C^1 induced by $\{v_i, \ldots, v_j\}$ and C^2 induced by $\{v_j, \ldots, v_n, \ldots, v_i\}$. Then $\max\{|C^1|, |C^2|\} \leq n - 1$. Let (T^1, X^1) (resp. (T^2, X^2)) be the tree-decomposition of C^1 (resp. C^2) obtained by deleting all vertices not in C^1 (resp. C^2) in the bags of (T, X).

If $|V(C^1)| = 3$ then there is a bag in (T^1, X^1) containing $V(C^1) = \{v_i, v_{i+1}, v_j = v_{i+2}\}$. So $v_iv_j \notin \{v_iv_{i+1}, v_{i+1}v_j\}$.

If $|V(C^1)| \geq 4$ then, by induction, there exist either a bag in (T^1, X^1) containing all vertices of $V(C^1) = \{v_i, v_{i+1}, v_{i+2}, v_j = v_{i+3}\}$ or two bags A, B in (T^1, X^1) containing three consecutive vertices a_1, a_2, a_3 and b_1, b_2, b_3 respectively in C^1; moreover, $\{a_1a_2, a_2a_3\} \cap \{b_1b_2, b_2b_3\} = \emptyset$. So we have either $v_iv_j \notin \{a_1a_2, a_2a_3\}$ or $v_iv_j \notin \{b_1b_2, b_2b_3\}$.

In both cases ($|V(C^1)| = 3$ and $|V(C^1)| \geq 4$), there is at least one bag X in (T^1, X^1) containing three consecutive vertices in C^1, denoted as x_1, x_2, x_3, such that $v_iv_j \notin \{x_1x_2, x_2x_3\}$. So x_1, x_2, x_3 are also consecutive in C. Similarly, there is at least one bag Y in (T^2, X^2) containing three consecutive vertices in C^2, denoted as y_1, y_2, y_3, such that $v_iv_j \notin \{y_1y_2, y_2y_3\}$. So y_1, y_2, y_3 are also consecutive in C.

Finally, we have $\{x_1x_2, x_2x_3\} \cap \{y_1y_2, y_2y_3\} = \emptyset$ because $E(C^1) \cap E(C^2) = \{v_iv_j\}$ and $v_iv_j \notin \{x_1x_2, x_2x_3\}$. □

Lemma 28. *Let xy be a chord in G. Let C' be the set of all the consecutive vertices from x to y in C and $|C'| \geq 4$. If each vertex in $C' \setminus \{x, y\}$ has degree 2 in G, then for any consecutive vertices $a, b, c, d \in C'$ (see Figure B.12(c)), $H = G[\{a, b, c, d\}]$ is a 3-potential-leaf of G.*

Proof. Let $(T, \mathcal{X})$ be any tree-decomposition of width at most 3 and size at most $s \geq 1$ of G. We show how to modify $(T, \mathcal{X})$ to obtain a tree-decomposition of G, which has width at most 3, size at most s and a leaf bag $\{a, b, c, d\}$.

Note that the vertices of C' induce a cycle in G. Without confusion, we denote this cycle by C'. Let (T', X') be the tree-decomposition of C' obtained by deleting all vertices not in C' in the bags of (T, X). From Lemma 27, there is either a bag containing all vertices in C' (only if $|C'| = 4$), or two bags X, Y containing three consecutive vertices in C' respectively and the two corresponding edge sets do not intersect.

In the former case, $V(C') = \{a, b, c, d\}$ and so $(T, \mathcal{X})$ is also a tree-decomposition of $G \cup \{ac\}$, from Lemma 25, $\{a, b, c, d\}$ is a 3-potential-leaf of G.

In the latter case, let $X \supseteq \{u, v, w\}$ and $Y \supseteq \{u', v', w'\}$, where u, v, w (resp. u', v', w') are consecutive in C'. Since $\{uv, vw\} \cap \{u'v', v'w'\} = \emptyset$, we have either $xy \notin \{uv, vw\}$ or $xy \notin \{u'v', v'w'\}$. W.l.o.g. we assume that $xy \notin \{uv, vw\}$. Then u, v, w are also consecutive in C and at least one of u, w has degree 2 in G. W.l.o.g. we suppose that w has degree 2 in G, i.e.$w \notin \{x, y\}$ (since x, y have degree at least 3 in G). Let $z \in C'$ be the other neighbor (except v) of w in C' (z exists because $w \notin \{x, y\}$.) $(T, \mathcal{X})$ is also a tree-decomposition of $G \cup \{uw\}$, which is still an outerplanar graph by our assumptions. Note that w has degree 3 in the graph $G \cup \{uw\}$. So from Lemma 25, we can modify $(T, \mathcal{X})$ to obtain a tree-decomposition $(T', \mathcal{X}')$ of $G \cup \{uw\}$, which has width at most 3, size at most s and a leaf bag L containing four consecutive vertices $\{u, v, w, z\}$. Note that $(T', \mathcal{X}')$ is also a tree-decomposition of G. So we obtain a tree-decomposition where a leaf bag contains 4 consecutive vertices of C'. It remains to show how to modify it to obtain a tree-decomposition with a leaf bag $\{a, b, c, d\}$.

Let B be the neighbor of L in T. Then $u, z \in B$ since each of u, z is adjacent to some vertices in $G \setminus L$. We can assume that L is the single bag containing v, w in $(T', \mathcal{X}')$, because otherwise we can delete them in any other bags. Thus, by deleting the bag L in $(T', \mathcal{X}')$, we get a tree-decomposition $(T_1, \mathcal{X}_1)$ of the graph G_1, which is the graph obtained by deleting v, w and adding an edge uz in G. So $(T_1, \mathcal{X}_1)$ has width at most 3 and size at most $s - 1$. Note that the graph G_1 is isomorphic to the graph $G_2 \equiv G \cup \{ad\} \setminus \{b, c\}$ since $z \in C'$. So from the tree-decomposition $(T_1, \mathcal{X}_1)$ of G_1 we can obtain a tree-decomposition $(T_2, \mathcal{X}_2)$ of G_2 with the same width and size. Note that since $ad \in E(G_2)$, there is a bag Y containing both a and d. Finally, we add a new bag $N = \{a, b, c, d\}$ adjacent to Y in $(T_2, \mathcal{X}_2)$. The result is the desired tree-decomposition. □

Lemma 29. *There is an algorithm that, for any 2-connected outerplanar graph G, computes a 3-potential-leaf of G in polynomial time.*

Proof. Let G be a 2-connected outerplanar graph and C be the unique Hamiltonian cycle of G. If $|V(G)| \leq 4$, then $V(G)$ is a 3-potential-leaf of G. Otherwise, $|V(G)| \geq 5$ and we consider the outerplanar embedding of G.

- If there exists an inner face f with at most one chord of G and f has at least four vertices, then from Lemma 28, the set of any four consecutive vertices in f, which are also consecutive in C, is a 3-potential-leaf in G.
- If there is an inner face $f = \{a, b, c\}$ with only one chord ac of G and c has degree 3, then let d be the other neighbor of c except b, a. From Lemma 25, the set of four consecutive vertices a, b, c, d, is a 3-potential-leaf in G.
- Otherwise, let $\mathcal{F}$ be the set of all inner faces with only one chord of G. Then any face $f \in \mathcal{F}$ has three vertices and both the two endpoints of the chord in f have degree at least 4, i.e., they are incident to some other chords except this one. We can prove by induction on $|V(G)|$ that:

  Claim 19. *There exist two faces $f_1, f_2 \in \mathcal{F}$ such that (1)$f_1 = \{a, b, c\}$; (2) $f_2 = \{c, d, e\}$; (3) a, b, c, d, e are consecutive in G; (4) there is a face f_0 containing both ac and ce and at most one chord, which is not in any face of $\mathcal{F}$. see Figure B.13.*

  This is true when $|V(G)| = 5$. Assume that it is true for $|V(G)| \leq n - 1$. We prove that it is true for $|V(G)| = n$. Note that $\mathcal{F} \neq \emptyset$ if there is at least one chords in G, which is valid in this case. Let $f \in \mathcal{F}$ have three consecutive vertices x, y, z and let $xz \in E(G)$ be the single chord in f. Then the graph $G \backslash y$ is a 2-connected outerplanar graph with $n - 1$ vertices. From the assumption, we have the desired faces f_0', f_1', f_2' in $G \setminus y$. If xz is not an edge in any face of f_1', f_2', then these faces are also the desired faces in G. Otherwise, let xz be an edge of f_1' or $f_2' = \{x, z, t\}$. Then z has degree 3 in G, i.e. it is not incident to any other chords except xz, since $xt \in E(G)$. So we are in second case above, which contradicts with the assumption.

  In the following, let f_0, f_1, f_2 be the faces as in Claim 19. If $ae \in E(G)$, then from Lemma 26, $\{a, b, c\}$ is a 3-potential-leaf of G.

  Otherwise, we can prove that any tree-decomposition of G of width at most 3 can be modified to a tree-decomposition of $G \cup \{ae\}$ with the same width and size in the following. So $\{a, b, c\}$ is a 3-potential-leaf of G.

  Let $(T, \mathcal{X})$ be a tree-decomposition of width at most 3 and size at most $s \geq 1$ of G. Let $(T_0, \mathcal{X}_0)$ be the tree-decomposition obtained by deleting all vertices not in f_0. Then $(T_0, \mathcal{X}_0)$ is a tree-decomposition of f_0 (f_0 is used to denote the face and the cycle induced by vertices in f_0 as well). From Lemma 27, there is a bag containing three consecutive vertices u, v, w in f_0 and uv, vw

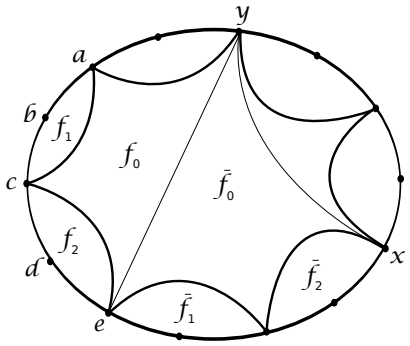


Figure B.13: $\mathcal{F}$ is the set of all inner faces with only one chord of G, such as $f_1, f_2, \bar{f}_1, \bar{f}_2$. The faces f_0, f_1, f_2 satisfy the properties in Claim 19. But $\bar{f}_0, \bar{f}_1, \bar{f}_2$ does not satisfy the properties since $\bar{f}_0$ contains two edges ey, xy which are not in any face of $\mathcal{F}$.

are edges of some faces in $\mathcal{F}$ (note that u, v, w are not consecutive in C). So $(T, \mathcal{X})$ is also a tree-decomposition of $G \cup uw$. The graph $G \cup uw$ and the graph $G \cup ae$ are isomorphic. So from $(T, \mathcal{X})$ we can obtain a tree-decomposition $(T', \mathcal{X}')$ of $G \cup ae$ with the same width and size. Then $(T', \mathcal{X}')$ is the desired tree-decomposition.

□

From Lemmas 29 and Corollary 9, we obtain the following result.

Corollary 12. *s_3 can be computed and a minimum size tree-decomposition of width at most 3 can be constructed in polynomial-time in the class of 2-connected outerplanar graphs.*

B.6 Conclusion

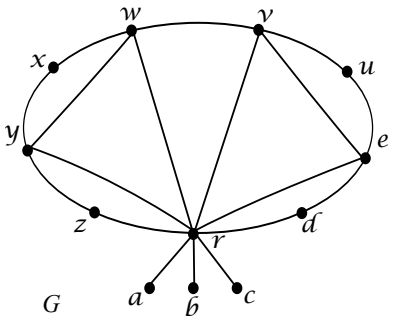
In this chapter, we gave preliminary results on the complexity of minimizing the size of tree-decompositions with given width. Table B.1 summarizes our results as well as the remaining open questions.

As future research direction, we would like to investigate the problem of computing s_3 in the class of connected graphs with treewidth 2 or 3. We have already solved the problem for trees and 2-connected outerplanar graphs. However, solving the problem for the general case seems to be more tricky. It seems that a global view of the graph needs to be considered to decide wether a subgraph is a 3-potential-leaf. The example in Figure B.14a illustrates this fact. In the example, G is a connected outerplanar graph and $\{r, a, b, c\}$ is not a 3-potential-leaf of G, but it is a 3-potential-leaf of $G \setminus \{yw\}$. Let $G' \equiv G \setminus \{a, b, c\}$, G' is 2-connected outerplanar. Using the algorithm of computing s_3 in 2-connected outerplanar graphs presented in subsection B.5.2, we have $s_3(G') = 5$. So if $\{r, a, b, c\}$ is a potential-leaf of G,

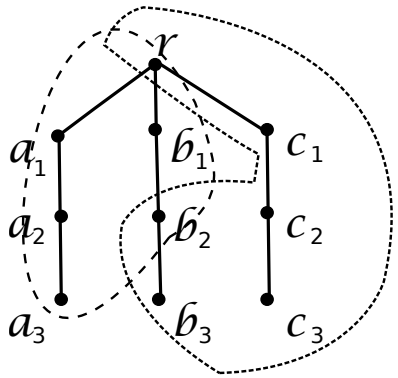
	s_1	s_2	s_3	s_4	s_k $k = \max\{tw+1, 5\}$
Graphs of treewidth at most $tw = 1$	P (trivial)	P	P	?	?
Graphs of treewidth at most $tw = 2$	-	P	?	?	?
Graphs of treewidth at most $tw = 3$	-	-	?	NP-hard	?
Graphs of treewidth at most $tw \geq 4$	-	-	-	NP-hard	NP-hard

Table B.1: Complexity of MSTD (P=Polynomial)

then $s_3(G) = 6$. However, there exists a tree-decomposition of G of width 3 and size 5, where the bags are $\{a, r, z, y\}, \{r, y, x, w\}, \{b, r, w, v\}, \{r, v, u, e\}, \{c, r, d, e\}$. This implies that $\{r, a, b, c\}$ is not a 3-potential-leaf of G. Now, let us consider the graph $G'' \equiv G \setminus \{yw\}$. We can prove that $s_3(G'') = 5$ and there is a minimum size tree-decomposition containing $\{r, a, b, c\}$ as a leaf bag. This implies that $\{r, a, b, c\}$ is 3-potential-leaf of G''. Therefore, the existence of the edge yw, not incident to any vertex in $\{r, a, b, c\}$, has an influence on whether $\{r, a, b, c\}$ is a 3-potential-leaf or not.



(a) $\{r, a, b, c\}$ is not a 3-potential-leaf of G, but it is a 3-potential-leaf of $G \setminus \{yw\}$. The five bags $\{a, r, z, y\}, \{r, y, x, w\}, \{b, r, w, v\}, \{r, v, u, e\}, \{c, r, d, e\}$ connected as a path in this order forms a tree-decomposition of G.



(b) In any minimum size tree-decomposition of width 5 (and size 2) of this tree, there exists a bag inducing a non-connected subgraph. For example, in a tree-decomposition of width 5 and size 2, one bag is $\{r, a_1, a_2, a_3, b_1, b_2\}$ and the other one is $\{r, b_2, b_3, c_1, c_2, c_3\}$.

Figure B.14

The problem of computing s_k, for $k \geq 4$, seems more intricate already in the

case of trees. Indeed, our polynomial-time algorithms to compute s_k, $k \leq 3$, in trees mainly rely on the fact that, for any tree T, there exists a minimum-size tree-decomposition of T with width at most 3, where each bag induces a connected subtree. This is unfortunately not true anymore in the case of minimum size tree-decompositions with width 5. The example in Figure B.14b illustrates this fact. In the example, we have a tree G (with 10 nodes) obtained from a star with three 3 leaves by subdividing twice each edge. For G, $s_5(G) = 2$ and any minimum size tree-decomposition has a bag X such that $G[X]$ is disconnected.

Appendix C

Résumé des travaux de thèse

Contents

C.1 Contexte et motivation

Les réseaux optiques sont au cœur des communications longues distances [Sen92, RS02]. Les données y sont codées en impulsions de lumière et transportées sur un fil très fin en verre ou en plastique, appelé fibre optique. Cette fibre constitue un support de transmission très efficace grâce à sa large bande passante et ses propriétés de faibles dispersion et atténuation [Sen92, GT98]. Elle est généralement composée d'un noyau en verre ou en plastique entouré d'une gaine de verre ou de plastique d'un indice de réfraction moins élevé. Le signal lumineux se déplace dans le cœur de la fibre à travers une série de réflexions, comme illustré sur la Figure C.1.

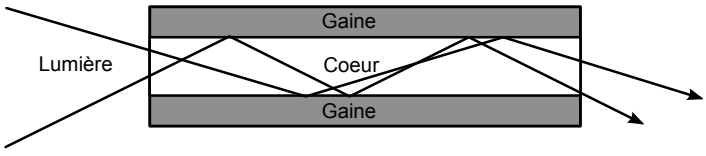


Figure C.1: Lumière transmise dans une fibre optique [Wik]

La fibre optique permet de véhiculer des données à très haut débit sur de longues distances. C'est pour cela que la quasi-totalité du trafic internet est transportée

dans des réseaux optiques. Ce trafic est cependant en croissance exponentielle; selon un rapport de Cisco [CS15], le trafic IP mondial a quintuplé au cours des 5 dernières années et devrait tripler au cours des 5 prochaines années. Pour satisfaire cette demande croissante en bande passante et protéger le trafic, il est indispensable d'utiliser les ressources optiques plus efficacement et de fournir plus de tolérance aux pannes dans les réseaux optiques. Dans cette thèse, nous abordons principalement des problèmes d'optimisation liés à ces deux propriétés très importantes dans les réseaux optiques: la tolérance aux pannes et l'efficacité.

C.2 La tolérance aux pannes dans les réseaux optiques

Les réseaux optiques transportent de gros volumes de données et toute panne peut entraîner la perte d'un trafic important et la perturbation de nombreux services. Il est donc essentiel de fournir des services de tolérance aux pannes dans ces réseaux [RS02]. Les pannes dans les réseaux optiques peuvent être totales ou partielles, uniques ou multiples. Elles peuvent être déclenchées par une coupure de fibre ou par une panne de courant, par une erreur humaine ou une catastrophe naturelle. Les fournisseurs de services doivent tenir compte de tous ces types de pannes et concevoir des stratégies qui assurent la survie de leurs réseaux. Ces stratégies sont généralement de deux types : les stratégies de protection et celles de restauration [RM99]. Les stratégies de protection consistent à utiliser des ressources prédéterminées et réservées pour rerouter le trafic interrompu, alors que dans les stratégies de restauration, les ressources utilisées pour rétablir le trafic sont calculées au moment de la panne. Nous nous intéressons dans la première partie de cette thèse aux stratégies de protection et plus précisément à la protection avec chemin dédié (DPP). La DPP consiste à calculer pour chaque demande deux chemins. Un chemin, dit chemin principal, est utilisé pour transmettre les données dans les conditions normales et l'autre, chemin secondaire ou chemin de protection, est réservé pour rerouter le trafic en cas de panne. Ces deux chemins doivent être disjoints de sorte qu'au moins un d'entre eux peut survivre en cas de défaillance dans le réseau. Dans la première partie de cette thèse, nous étudions le problème DPP dans deux contextes différents. D'abord, dans le chapitre 2, nous considérons des scénarios de pannes d'un lien ou d'un nœud dans des réseaux avec nœuds asymétriques; nous examinons le problème de trouver un seul chemin et puis le problème de trouver des chemins disjoints. Ensuite, dans le chapitre 3, nous nous focalisons sur un autre type de pannes, à savoir, les pannes simultanées multiples localisées autour des nœuds (les SRLGs en étoile).

C.2.1 Chapitre 2 : Les chemins dans les réseaux optiques avec nœuds asymétriques

Dans ce chapitre, nous étudions le problème de trouver des chemins et des chemins disjoints dans les réseaux optiques avec nœuds asymétriques [BLGM09]. Les nœuds asymétriques sont des nœuds qui ne sont pas entièrement connectés à l'intérieur; cela veut dire que le signal entrant d'un certain port ne peut pas arriver à tous les

autres ports. Dans [BLGM09, CHW+13, HTTN11], il est expliqué qu'un nœud peut être configuré de façon asymétrique pour plusieurs raisons comme le nombre limité de ports dans les commutateurs optiques et le coût réduit des nœuds asymétriques par rapport aux nœuds symétriques. La Figure C.2 de [CHW+13] présente un diagramme d'un multiplexeur optique d'insertion-extraction reconfigurable (ROADM) de degré 4. Dans la Figure C.2-a, le ROADM est symétrique et d'un port d'entrée, on peut atteindre n'importe quel port de sortie. Dans la Figure C.2-b, une architecture asymétrique du ROADM est présentée. Dans cette architecture, des équipements moins chers sont utilisés au niveau des ports. Le signal optique provenant d'une direction peut atteindre seulement deux parmi les trois autres directions.

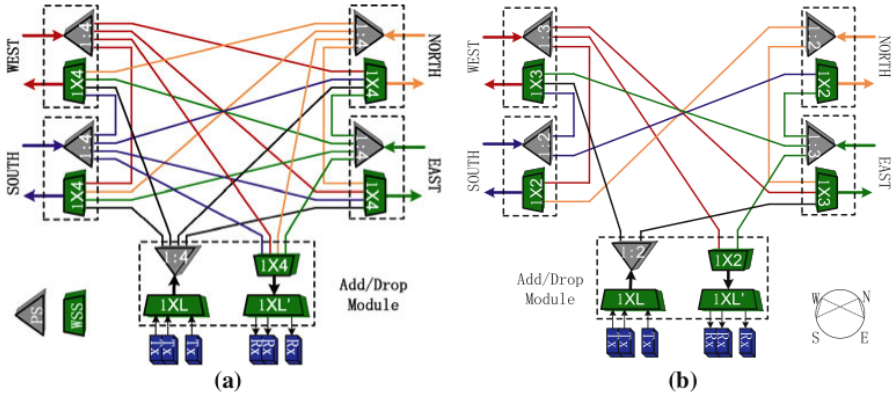


Figure C.2: Diagramme d'un ROADM [CHW+13]: **a.** cas symétrique; **b.** cas asymétrique

Nous modélisons les réseaux avec nœuds asymétriques avec des graphes avec transitions interdites. Une transition dans un graphe est une paire d'arêtes adjacentes. Etant donnés un graphe $G = (V, E)$ et un ensemble associé de transitions interdites $\mathcal{F} \subseteq E \times E$, le problème de trouver un chemin avec transitions interdites (PAFT) consiste à trouver un chemin qui n'utilise aucune des transitions de $\mathcal{F}$. Nous étudions le problème de trouver un chemin élémentaire avec transitions interdites dans les graphes planaires. Nous prouvons que le problème est NP-difficile dans les graphes planaires et plus particulièrement dans les grilles. Nous montrons également que le problème est FPT quand le paramètre est la somme de la largeur arborescente et le degré maximum. Nous traitons ensuite les problèmes de trouver k nœud-disjoints chemins non élémentaires évitant les transitions interdites (k-VDT) et k lien-disjoints chemins non élémentaires évitant les transitions interdites (k-EDT). Nous démontrons que k-VDT ($k \geq 2$) est NP-difficile dans les graphes orientés et non-orientés et que k-EDT est polynomial dans les graphes orientés. La Table C.1 résume les résultats de complexité obtenus pour les problèmes considérés.

	Graphes orientés	**Graphes non-orientés**
Trouver un chemin élémentaire	• NP-complet en général [Sze03] • NP-complet dans les grilles orientées • Polynomial dans les DAGs	• NP-complet en général [Sze03] • NP-complet dans les grilles • Polynomial dans les graphes avec treewidth borné
Trouver un chemin non-élémentaire	Polynomial [Win02, ABP96]	Polynomial
k-VDT	• NP-complet dans les graphes orientés avec un degré ≤ 8 • Polynomial dans les DAGs	• NP-complet dans les graphes non-orientés avec un degré ≤ 8 • NP-complet dans les graphes non-orientés avec un degré ≤ 4 [GLMM12]
k-EDT	Polynomial	NP-complet [ADF+08]

Table C.1: Résultats de complexité des problèmes dans les réseaux avec transitions interdites

C.2.2 Chapitre 3 : Chemins disjoints dans les réseaux avec des SRLGs en étoile

Dans ce chapitre, nous nous focalisons sur les groupes de liens partageant un risque (*Shared Risk Link Groups*, (SRLGs)). La notion de SRLG a été introduite pour modéliser des problèmes de tolérance aux pannes simultanées d'ensembles de liens d'un réseau. Un SRLG peut correspondre par exemple à un ensemble de liens enterrés dans la même tranchée et qui peuvent tous tomber en panne à cause d'un coup de pelleteuse ou qui peuvent tous être affectés par un tremblement de terre ou par la pluie.

Dans un réseau avec des SRLGs, le problème du *routage diversifié* consiste à trouver un ensemble de chemins SRLG-disjoints entre une paire donnée de nœuds du réseau. Ce problème a été prouvé NP-complet en général [Hu03] et certains cas polynomiaux ont été caractérisés [CDP+07]. Nous avons étudié le problème du routage diversifié dans les réseaux satisfaisant la *propriété d'étoile* [LW05]. Dans un réseau satisfaisant la propriété d'étoile, un lien peut être affecté par plusieurs SRLGs, mais tous les liens affectés par un même SRLG sont incidents à un même sommet. Un exemple de tels SRLGs est présenté dans la Figure C.3. Dans l'exemple, des liens incidents à la même carte d'un routeur forment un SRLG puisqu'ils peuvent tous être affectés par une panne de la carte.

Nous avons trouvé des contre-exemples à l'algorithme polynomial proposé dans [LW05] pour le calcul de paires de chemins SRLG-disjoints dans les réseaux

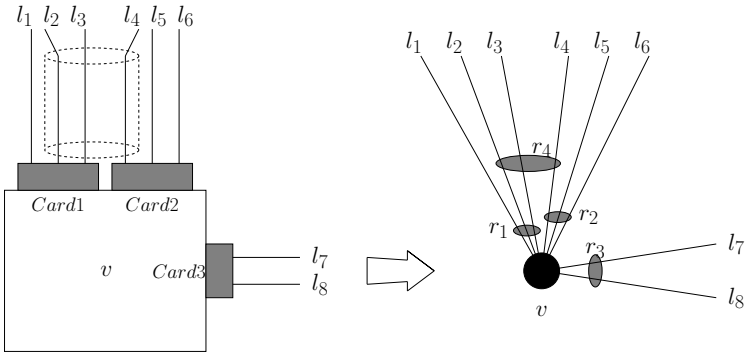


Figure C.3: Exemple de risques localisés : le lien l_4 partage un risque r_2, correspondant à une panne de la carte 2, avec les liens l_5 et l_6, et il partage un risque r_4, correspondant à une coupure du conduit, avec les liens l_2 et l_3.

satisfaisant la propriété d'étoile. Puis, en utilisant un modèle de graphes colorés où les couleurs correspondent aux SRLGs, nous avons prouvé que ce problème est en effet NP-difficile. Plus généralement, nous avons montré que le problème du routage diversifié dans les réseaux avec la propriété d'étoile est NP-difficile, APX-difficile, et W[1]-difficile lorsque le paramètre est le nombre de chemins SRLG-disjoints. Enfin, nous avons caractérisé des instances polynomiales, en particulier lorsque le degré maximum des sommets est 4, ou lorsque le réseau est acyclique. La Table C.2 résume les résultats de complexité obtenus pour les problèmes considérés. Dans la table, V, E, $\mathcal{C}$, EPC, CPE, k-DCP, et MDCP dénotent respectivement l'ensemble des nœuds, l'ensemble des liens, l'ensemble des SRLGs, le nombre de liens atteints par le même SRLG, le nombre de SRLGs auxquels peut appartenir un lien, le problème de trouver k chemins SRLG-disjoints, et le problème de trouver le nombre maximum de chemins SRLG-disjoints.

C.3 L'utilisation efficace du spectre dans les réseaux optiques

Pour faire face à la croissance exponentielle du trafic Internet, une nouvelle génération de réseaux optiques est en cours de développement; les réseaux optiques élastiques (les EONs), aussi connus sous le nom des réseaux Flex-grid [GJLY12, JTK+09]. Les EONs sont proposés pour remplacer les réseaux Wavelength Division Multiplexing (WDM) actuellement déployés. Dans les réseaux WDM, le spectre optique est divisé en canaux de largeur fixe (par exemple 50 GHz) appelés longueurs d'onde, formant la grille fixe traditionnelle de WDM. Chaque longueur d'onde de cette grille fixe peut assurer une connexion à un débit qui peut atteindre 100 Gbps. Même si une connexion nécessite moins de 100 Gbps, on y alloue une longueur d'onde complète. Ce qui résulte en une utilisation inefficace du spectre. D'autre part, quand une connexion exige plus de 100Gbps, on y alloue plusieurs

	Δ	EPC	CPE	k-DCP	MDCP
Graphes non orientés	non borné	1	non borné	$O(\|V\|+\|E\|)$	$O(\Delta\|E\|)$
		non borné	1	$O(\|V\|+\|E\|)$ [CDP+07]	$O(\Delta_{\mathcal{C}}\|E\|)$ [CDP+07]
	≥ 8	≥ 2	≥ 4	NP-difficile pour $\Delta \geq \max\{8,k\}$	Non approximable à $O(\|V\|^{1-\varepsilon})$, pour $0<\varepsilon<1$
		≥ 4	≥ 2		
	≤ 3	non borné	non borné	$O(\|V\|+\|E\|)$	Optimum en $O(\|V\|+\|E\|)$
	$=4$	non borné	non borné	$O(\|V\|+\|E\|)$ pour $k=2$	2-approximation en $O(\|V\|+\|E\|)$
$\|\mathcal{C}\|=O(1)$	non borné	non borné	non borné	$O(f(\|\mathcal{C}\|)(\|V\|+\|E\|))$, FPT si paramétré par $\|\mathcal{C}\|$	Optimum en $O(f(\|\mathcal{C}\|)(\|V\|+\|E\|)\log\Delta)$, FPT si paramétré par $\|\mathcal{C}\|$
DAG	non borné	≥ 3	≥ 3	$O(\text{CPE}^2\|V\|\|E\|^{2k})$	NP-difficile
		≥ 2	≥ 6		
		≥ 2	non borné		Non approximable à $O(\|V\|^{1-\varepsilon})$, pour tout $0<\varepsilon<1$
		non borné	$=3$		APX-difficile
		non borné	non borné		$W[1]$-difficile si paramétré par le nombre de chemins

Table C.2: Résultats de complexité des problèmes dans les réseaux avec des SRLGs en étoile.

longueurs d'onde. Ceci mène à l'utilisation de plus de spectre que si la connexion a été satisfaite par un seul canal contigu de spectre.

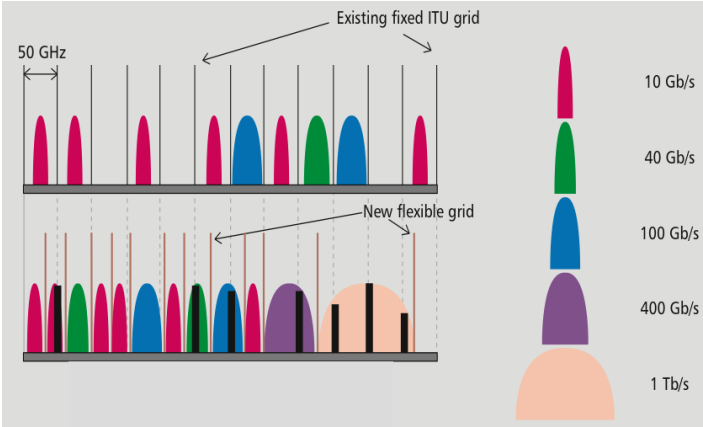


Figure C.4: Fixed-grid vs Flexgrid [GJLY12]

Dans les EONs, des nouvelles technologies telles que l'OFDM (Orthogonal Frequency Division Multiplexing) optique, les techniques de modulation adaptative (tels que le QPSK (quadrature phase-shift keying) et la modulation d'amplitude en quadrature (16-QAM)), et les commutateurs sélectifs en longueur d'onde, sont utilisées pour assurer une utilisation efficace des ressources optiques et pour transformer la grille fixe des réseaux WDM en une grille souple. Dans cette grille souple, le spectre optique est divisé en un ensemble de bandes de largeur spectrale fixe et fine (25 GHz, voire 12.5GHz ou 6.25GHz), appelées *slots* ou *minigrids*. Ainsi, les

connexions à petits débits ne sont pas sur-provisionnées et les grands débits peuvent être satisfaits sur des slots contigus multiples. En d'autres termes, au lieu de forcer toutes les connexions à utiliser la même largeur de spectre, les EONs permettent aux connexions d'utiliser un certain nombre de slots optiques contigus en fonction de leurs besoins.

La Figure C.4, tiré de [GJLY12], illustre le différence entre la grille fixe de WDM (en haut) et la grille souple des EONs (en bas). Dans la grille fixe, un canal n'est pas pleinement utilisé par un petit débit, et les débits de 400 Gbps et 1 Tbps ne peuvent pas être pris en charge par un seul canal. De l'autre côté, dans la grille souple, les grands débits peuvent être approvisionnés et le spectre est utilisé de manière plus efficace pour les petits débits. D'autres avantages des EONs sont détaillés dans [GJLY12, WLV13, JTK+09].

La flexibilité offerte par les EONs n'est cependant pas gratuite; elle est accompagnée de plus de complexité dans les problèmes d'allocation de ressources. Dans les réseaux WDM, le poblème clé d'allocation de ressources est dit routage et allocation de longueurs d'ondes (RWA) [RS95, ZJ00]. Ce problème consiste à attribuer à chaque connexion un chemin optique, c'est-à-dire un chemin et une longueur d'onde de la source à la destination. Chaque connexion doit utiliser la même longueur d'onde sur tous les liens de son chemin (contrainte de continuité) et deux connexions traversant le même lien ne peuvent pas utiliser la même longueur d'onde (contrainte de non-chevauchement). Dans les EONs, puisqu'il est possible d'attribuer des bandes de spectre de largeurs quelconques, RWA est remplacé par le problème de routage et d'allocation de spectre (RSA). Le problème RSA consiste à attribuer à chaque requête un chemin et un intervalle de spectre. Le spectre attribué à une demande doit être contigu (contrainte de contiguïté), il doit être le même sur tous les liens du chemin de routage (contrainte de continuité) et les intervalles attribués à deux requêtes traversant le même lien doivent être disjoints (contrainte de non-chevauchement). RWA peut être considéré comme le cas particulier de RSA où toutes les requêtes ont la même demande.

On peut distinguer deux versions du problème RSA par rapport au type du trafic: RSA statique et RSA dynamique.

- *Le RSA statique.* Dans ce problème, l'ensemble des requêtes est connu à l'avance. L'objectif est d'allouer des chemins et du spectre à ces requêtes en minimisant le spectre utilisé ou en minimisant le nombre de requêtes bloquées. Puisque le problème RWA statique est NP-complet [CGK92], le RSA statique est NP-complet aussi. Pour le résoudre, il est généralement divisé en deux sous-problèmes qui sont résolus séquentiellement: le sous-problème de routage où le but est de trouver des chemins pour les requêtes et le sous-problème d'allocation de spectre (SA). Nous avons traité le problème SA dans des réseaux en arbre dans le chapitre 4.

- *Le RSA dynamique.* Dans ce problème, les requêtes arrivent et quittent le réseau dynamiquement. L'objectif est de minimiser le taux de blocage des requêtes dans le réseau. La dynamicité du trafic peut causer la fragmentation du

spectre. Ceci veut dire l'accumulation de petites bandes de spectre qui deviennent inutilisables, soit parce qu'elles ne sont pas continues ou parce qu'elles ne sont pas contiguës. Dans le chapitre 5, nous avons étudié le problème RSA dynamique avec une technique de défragmentation non-perturbatrice appelée *Push-Pull*.

C.3.1 Chapitre 4 : Allocation de spectre dans les réseaux en arbre

Dans le problème d'allocation de spectre (SA), nous avons en entrée un réseau et un ensemble de requêtes tel que chaque requête a un chemin et une demande en spectre. L'objectif est d'allouer du spectre aux requêtes sous les contraintes de la contiguïté et le non-chevauchement. Dans ce chapitre, nous traitons le problème d'allocation de spectre dans les réseaux optiques élastiques en arbre. Nous démontrons que le problème est NP-difficile dans les étoiles non-orientées à trois liens et les étoiles orientées à deux liens entrants et deux liens sortants. Ensuite, nous présentons une 4-approximation du problème dans les étoiles orientées et non-orientées. Nous étudions également le problème dans les graphes binaires. Nous démontrons qu'il y a des approximations à facteurs constants pour le problème de SA dans les arbres binaires avec des profils de demandes spécifiques. Ces approximations sont présentées dans la Table C.3. Dans la table, $\mathcal{OPT}$ est la valeur optimale du spectre qui peut être utilisée pour résoudre le problème.

Demandes	**Bornes supérieures**
$\boldsymbol{\{k, kX\}}$ $\boldsymbol{k, X} \in \mathbb{N}^*$	$(2 - \frac{1}{X})\mathcal{OPT} + k$
$\boldsymbol{\{kX, k(X+1)\}}$ $\boldsymbol{k, X} \in \mathbb{N}^*$	$(1 + \frac{1}{X})\mathcal{OPT}$
$\leq \mathbf{3}$	$\frac{19}{10}\mathcal{OPT} + \frac{8}{5}$
$\leq \mathbf{4}$	$\frac{59}{27}\mathcal{OPT} + \frac{67}{27}$
$\leq \mathbf{5}$	$\frac{859}{336}\mathcal{OPT} + \frac{229}{56}$
$\leq \mathbf{6}$	$\frac{287}{100}\mathcal{OPT} + \frac{885}{200}$
$\leq \boldsymbol{W}$	$2\lceil \log_2(W) \rceil \mathcal{OPT}$

Table C.3: Bornes supérieures sur le spectre utilisé par nos algorithmes d'approximation pour résoudre SA dans des arbres binaires avec des profils de demandes spécifiques.

C.3.2 Chapitre 5 : RSA dynamique avec Push-Pull

Pour remédier au problème de la fragmentation, des techniques de défragmentation sont utilisées. Nous avons étudié le problème RSA dynamique avec une technique de défragmentation qui s'appelle le Push-Pull. Cette technique permet de décaler les bandes de spectre utilisées par d'autres requêtes en vue de libérer de la place pour les nouvelles. D'abord, nous avons travaillé avec un algorithme proposé par Wang and Mukherjee dans [WM13] pour résoudre le problème d'allocation de spectre avec Push-Pull. Nous avons modifié l'algorithme pour qu'il retourne la position dans le spectre qui minimise le délai d'attribution du spectre à la nouvelle demande. Ensuite, nous avons proposé des algorithmes pseudo-polynomiaux pour résoudre exactement le problème de routage et d'allocation de spectre avec Push-Pull. Le premier algorithme minimise la longueur du chemin alloué à la nouvelle requête alors que le deuxième minimise le délai d'attribution du spectre.

C.4 La décomposition arborescente

Un autre problème traité durant cette thèse, présenté en annexe, est les décompositions arborescentes de taille minimum. Une décomposition arborescente d'un graphe est une manière de le représenter sous forme d'un arbre (chaque sommet de l'arbre est appelé '*sac*'), en préservant des propriétés de connexité. Les décompositions arborescentes ont été beaucoup étudiées pour leurs applications algorithmiques qui utilisent, en particulier, la programmation dynamique.

En annexe B, nous étudions les décompositions de taille minimum, c'est-à-dire avec un nombre minimum de sacs. Etant donné un entier $k \geq 4$ fixé, nous prouvons que le problème de calculer une décomposition arborescente de largeur au plus k et de taille minimum est NP-complet dans les graphes de largeur arborescente au plus 4. Nous décrivons ensuite des algorithmes qui calculent des décompositions de taille minimum dans certaines classes de graphes de largeur arborescente au plus 3 (en particulier dans les arbres et les graphes planaires extérieurs). La Table C.4 résume nos résultats sur le problème de décomposition arborescente de taille minimum (MSTD). Dans la table, s_k dénote le problème de calculer une décomposition arborescente de largeur au plus k et de taille minimum.

C.5 Conclusion

La tolérance aux pannes et l'utilisation efficace des ressources sont des propriétés bien souhaitées dans les réseaux optiques. Il est important de fournir des services de tolérance aux pannes dans les réseaux optiques en vue de protéger les quantités énormes de données qu'ils transportent. Il est également essentiel d'utiliser avec efficacité le spectre optique pour pouvoir répondre à la croissance exponentielle du trafic. Nous avons étudié dans cette thèse des problèmes liés à ces deux propriétés. Nous avons d'abord examiné le problème de trouver des chemins disjoints. Ce problème est essentiel dans la stratégie de protection par chemin dédié (DPP). Dans

	s_1	s_2	s_3	s_4	s_k $k = \max\{tw+1, 5\}$
Graphes de treewidth $tw = 1$	P	P	P	?	?
Graphes de treewidth $tw = 2$	-	P	?	?	?
Graphes de treewidth $tw = 3$	-	-	?	NP-difficile	?
Graphes de treewidth $tw \geq 4$	-	-	-	NP-difficile	NP-difficile

Table C.4: Complexité of MSTD (P=Polynomial)

cette stratégie, pour protéger le trafic entre deux nœuds, deux chemins disjoints sont réservés : un chemin pour assurer la connexion et un chemin pour rerouter le trafic en cas de panne du premier. Nous avons également abordé le problème de routage et d'allocation de spectre (RSA) dans les réseaux optiques élastiques; la nouvelle génération des réseaux optiques. Dans RSA, l'objectif est d'établir des connexions dans le réseau de manière à assurer une utilisation optimale des ressources optiques.

En utilisant des modèles et des outils de la théorie des graphes, nous avons établi des résultats de complexité de ces problèmes sous différentes hypothèses. Nous avons répondu à des questions telles que: quelle est la difficulté de trouver un ou plusieurs chemins disjoints dans les réseaux avec des nœuds avec restrictions sur leur connectivité interne ? Quelle est la difficulté de trouver des chemins disjoints dans les réseaux où des ensembles de liens partageant un nœud peuvent tomber en panne simultanément ? Quelle est la difficulté du problème RSA dans les réseaux sous forme d'étoiles ? Le problème RSA peut-il être approché dans des cas particuliers de réseaux d'arbres ? Comment utiliser une technique de défragmentation non-perturbatrice dans les EONs de façon à minimiser la probabilité de blocage du spectre ? Les résultats que nous avons obtenus suggèrent que les deux problèmes DDP et RSA sont difficiles à résoudre sous les conditions que nous avons considérées. Cependant, il est parfois possible de trouver des cas polynomiaux et des algorithmes d'approximation. Ces cas et des algorithmes pourraient peut être aider à concevoir des heuristiques efficaces. Nous décrivons brièvement dans ce qui suit les résultats établis et nous présentons quelques questions ouvertes.

Dans le chapitre 2, nous avons utilisé les graphes avec transitions interdites pour modéliser les réseaux avec nœuds asymétriques. Nous avons prouvé que le problème de trouver un chemin évitant les transitions interdites (PAFT) est NP-complet, même dans des graphes bien structurés comme les grilles. Nous avons également prouvé que PAFT peut être résolu en temps polynomial lorsque la largeur arborescente est bornée. Nous avons aussi abordé le problème de trouver des chemins disjoints. Nous avons montré que lorsque les chemins doivent être nœud-disjoints le problème est NP-complet à la fois dans les graphes orientés et non-orientés. Sur le

côté positif, si les chemins sont lien-disjoints, le problème peut être résolu en temps polynomial dans les graphes orientés. Il serait intéressant d'étudier plus pronfondément la complexité paramétrée du problème PAFT. En effet, nous pensons que PAFT est $W[1]$-difficile quand il est paramétré par la largeur arborescente. Il serait aussi intéressant d'étudier la complexité du problème de trouver des chemins nœud-disjoints évitant les transitions interdites dans les graphes planaires. Enfin, il serait aussi possible d'explorer les versions d'optimisation des problèmes que nous avons étudiés dans le but de trouver de bonnes approximations. A savoir, le problème de trouver un chemin utilisant un nombre minimum de transitions interdites et le problème de trouver des chemins évitant les transitions interdites et partageant un nombre minimum de liens ou de nœuds.

Dans le chapitre 3, nous avons donné une caractérisation presque complète de la complexité du problème de trouver des chemins SRLG-disjoints dans les réseaux avec SRLGs satisfaisant la propriété de l'étoile. Nous avons montré qu'il est NP-complet de trouver k chemins SRLG-disjoints même quand $k = 2$. Sur le côté positif, nous avons prouvé que le problème peut être résolu en temps polynomial dans des cas particuliers. À savoir, nous avons résolu le problème en temps polynomial lorsque le degré maximum est au plus 4 ou lorsque le réseau est un graphe orienté acyclique. En outre, nous avons montré que le problème est FPT quand il est paramétré par le nombre de couleurs dans $\mathcal{C}$. Enfin, nous avons considéré le problème de trouver le nombre maximum de chemins SRLG-disjoints. Nous avons prouvé que, le problème est difficile à approcher à un facteur de $O(|V|^{1-\varepsilon})$ pour tout $0 < \varepsilon < 1$, où V est l'ensemble des nœuds du réseau, et nous avons donné des algorithmes polynomiaux pour certains cas. La complexité du problème de trouver k chemins SRLG-disjoints, sous la propriété d'étoile, est toujours ouverte pour les cas où le degré maximum du réseau est égal à 5, 6 ou 7 et pour les cas où le nombre de SRLGs par lien (ou le nombre de liens par SRLG) est égal à 2 ou 3. La résolution de ces cas donnera une caractérisation complète de la complexité du problème par rapport au degré maximum du réseau, au nombre de SRLGs par lien et au nombre de liens par SRLG.

Dans le chapitre 4, nous avons étudié le problème d'allocation de spectre (SA) dans les réseaux sous forme d'arbre. Nous avons prouvé que SA est NP-complet dans les étoiles non orientées à 3 liens et les étoiles orientées à 4 liens. Nous avons également montré qu'il existe un algorithme d'approximation d'un facteur 4 pour résoudre le problème dans les étoiles orientées et non-orientées. Ensuite, nous nous sommes focalisés sur le problème SA dans les arbres binaires avec des profils de demandes spécifiques. Nous avons conçu des algorithmes d'approximation de facteurs constants en utilisant le problème de coloration par intervalles des graphes chordaux. Comme piste de recherche pour ce problème, il serait intéressant de trouver un algorithme d'approximation de facteur constant pour le problème de coloration par intervalles des graphes chordaux en général ou de prouver qu'un tel algorithme n'existe pas. Pour ce faire, il serait utile d'essayer d'utiliser les algorithmes d'approximation conçus pour le problème de la coloration par intervalles des graphes d'intervalles.

Dans le chapitre 5, nous avons étudié le problème RSA dynamique avec Push-Pull. Nous avons proposé deux algorithmes exacts pour résoudre le problème en optimisant deux critères différents, la longueur du chemin de routage et le délai d'attribution du spectre. Nous avons également modifié un algorithme existant de SA avec Push-Pull, pour qu'il retourne la position dans le spectre qui minimise le délai. Les résultats d'expérimentations ont montré que l'intérêt de nos algorithmes est plutôt de nature théorique car un algorithme existant offre un meilleur compromis entre le délai, la probabilité de blocage et le temps d'exécution. Les pistes de recherche pour le problème RSA dynamique avec Push-Pull pourraient inclure l'optimisation d'autres critères à la place du délai et de la longueur du chemin de routage. Il serait également intéressant d'essayer de combiner les techniques proactives et réactives de défragmentation avec Push-Pull afin de diminuer encore plus la probabilité de blocage du spectre. Enfin, il serait bien d'examiner le cas où non seulement une demande, mais plusieurs demandes arrivent en même temps et nous avons besoin de décider si elles peuvent toutes être satisfaites.

C.6 Liste des publications

Journal international

[BCDM15] J.-C. Bermond, D. Coudert, G. D'Angelo, and F. Z. Moataz. Finding disjoint paths in networks with star shared risk link groups. *Theoretical Computer Science*, 579(0):74 – 87, 2015

Conférences internationales

[BCDM12] J.-C. Bermond, D. Coudert, G. D'Angelo, and F. Z. Moataz. Diverse routing in networks with star SRLGs. In *Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop*, CoNEXT Student '12, pages 23–24, New York, NY, USA, 2012. ACM

[BCDM13a] J.-C. Bermond, D. Coudert, G. D'Angelo, and F. Moataz. SRLG-diverse routing with the star property. In *Design of Reliable Communication Networks (DRCN), 2013 9th International Conference on the*, pages 163–170, March 2013

[LMN14] B. Li, F. Z. Moataz, and N. Nisse. Minimum Size Tree-Decompositions. In *9th International colloquium on graph theory and combinatorics (ICGT)*, Grenoble, France, 2014

[LMNS15] B. Li, F. Z. Moataz, N. Nisse, and K. Suchan. Minimum Size Tree-decompositions. In *LAGOS 2015 – VIII Latin-American Algorithms, Graphs and Optimization Symposium*, Beberibe, Ceará, Brazil, May 2015

[KMMN15a] M. M. Kanté, F. Z. Moataz, B. Momège, and N. Nisse. Finding Paths in Grids with Forbidden Transitions. In *WG 2015, 41st International Workshop on Graph-Theoretic Concepts in Computer Science*, Munich, Germany, June 2015

Conférences nationales

[BCDM13b] J.-C. Bermond, D. Coudert, G. D'Angelo, and F. Z. Moataz. Diverse Routing with Star SRLGs. In *15èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (AlgoTel)*, pages 1–4, Pornic, France, May 2013

[CJM14] D. Coudert, B. Jaumard, and F. Z. Moataz. Dynamic Routing and Spectrum Assignment with Non-Disruptive Defragmentation. In *ALGOTEL 2014 – 16èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, pages 1–4, Le Bois-Plage-en-Ré, France, June 2014

[KMMN15b] M. M. Kanté, F. Z. Moataz, B. Momège, and N. Nisse. On paths in grids with forbidden transitions. In *ALGOTEL 2015 - 17èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, Beaune, France, June 2015

[Moa15] F. Z. Moataz. On Spectrum Assignment in Elastic Optical Tree-Networks. In *ALGOTEL 2015 - 17èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, Beaune, France, June 2015

Bibliography

[ABG+01] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. RSVP-TE: Extensions to RSVP for LSP tunnels, 2001. (Cited in page 111.)

[ABP96] J. Añez, T. D. L. Barra, and B. Pérez. Dual graph representation of transport networks. *Transportation Research Part B: Methodological*, 30(3):209 – 216, 1996. (Cited in pages 15, 18, 19, 53 and 186.)

[ACP87] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, April 1987. (Cited in pages 34 and 150.)

[ADF+08] A. Abouelaoualim, K. Das, L. Faria, Y. Manoussakis, C. Martinhon, and R. Saad. Paths and trails in edge-colored graphs. In *LATIN 2008: Theoretical Informatics*, volume 4957 of *Lecture Notes in Computer Science*, pages 723–735. Springer Berlin Heidelberg, 2008. (Cited in pages 41, 53 and 186.)

[ADGW13] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. Alternative routes in road networks. *J. Exp. Algorithmics*, 18:1.3:1.1–1.3:1.17, April 2013. (Cited in page 16.)

[ADP80] G. Ausiello, A. D'Atri, and M. Protasi. Structure preserving reductions among convex optimization problems. *J. Comput. Syst. Sci.*, 21(1):136–153, August 1980. (Cited in page 72.)

[AL13] M. Ahmed and A. Lubiw. Shortest paths avoiding forbidden subpaths. *Networks*, 61(4):322–334, 2013. (Cited in page 15.)

[AP86] S. Arnborg and A. Proskurowski. Characterization and recognition of partial 3-trees. *SIAM J. Algebraic Discrete Methods*, 7(2):305–314, April 1986. (Cited in page 150.)

[BCDM12] J.-C. Bermond, D. Coudert, G. D'Angelo, and F. Z. Moataz. Diverse routing in networks with star SRLGs. In *Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop*, CoNEXT Student '12, pages 23–24, New York, NY, USA, 2012. ACM. (Cited in pages 5, 7, 55 and 194.)

[BCDM13a] J.-C. Bermond, D. Coudert, G. D'Angelo, and F. Moataz. SRLG-diverse routing with the star property. In *Design of Reliable Communication Networks (DRCN), 2013 9th International Conference on the*, pages 163–170, March 2013. (Cited in pages 5, 7, 55 and 194.)

[BCDM13b] J.-C. Bermond, D. Coudert, G. D'Angelo, and F. Z. Moataz. Diverse Routing with Star SRLGs. In *15èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (AlgoTel)*, pages 1–4, Pornic, France, May 2013. (Cited in pages 5, 8, 55 and 195.)

[BCDM15] J.-C. Bermond, D. Coudert, G. D'Angelo, and F. Z. Moataz. Finding disjoint paths in networks with star shared risk link groups. *Theoretical Computer Science*, 579(0):74 – 87, 2015. (Cited in pages 5, 7, 55 and 194.)

[BDD+13] H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk. A $o(c^k n)$ 5-approximation algorithm for treewidth. *CoRR*, abs/1304.6321, 2013. (Cited in pages 34 and 150.)

[BDGS11] R. Bader, J. Dees, R. Geisberger, and P. Sanders. Alternative route graphs in road networks. In *Proceedings of the First International ICST Conference on Theory and Practice of Algorithms in (Computer) Systems*, TAPAS'11, pages 21–32, Berlin, Heidelberg, 2011. Springer-Verlag. (Cited in page 16.)

[Bea00] B. Beauquier. *Communication dans les réseaux optiques par multiplexage en longueur d'onde*. PhD thesis, Université de Nice Sophia Antipolis, 2000. (Cited in page 89.)

[BEJ+06] A. L. Buchsbaum, A. Efrat, S. Jain, S. Venkatasubramanian, and K. Yi. Restricted strip covering and the sensor cover problem. *CoRR*, abs/cs/0605102, 2006. (Cited in pages 84, 86 and 88.)

[Bha97] R. Bhandari. Optimal physical diversity algorithms and survivable networks. In *Computers and Communications, 1997. Proceedings., Second IEEE Symposium on*, pages 433–441, Jul 1997. (Cited in page 16.)

[Bha98] R. Bhandari. *Survivable Networks: Algorithms for Diverse Routing*. Kluwer Axademic Publishers, 1998. (Cited in page 59.)

[BHK09] A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009. (Cited in page 65.)

[BJN+98] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.*, 46(3):316–329, March 1998. (Cited in page 137.)

[BK11] H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations ii. lower bounds. *Inf. Comput.*, 209(7):1103–1119, 2011. (Cited in page 166.)

[BKK+04] A. L. Buchsbaum, H. Karloff, C. Kenyon, N. Reingold, and M. Thorup. Opt versus load in dynamic storage allocation. *SIAM J. Comput.*, 33(3):632–646, March 2004. (Cited in pages 87, 88, 92 and 108.)

[BLGM09] G. Bernstein, Y. Lee, A. Gavler, and J. Martensson. Modeling WDM wavelength switching systems for use in GMPLS and automated path computation. *Optical Communications and Networking, IEEE*, 1(1):187–195, June 2009. (Cited in pages 13, 184 and 185.)

[BLS99] A. Brandstädt, V. Le, and J. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, 1999. (Cited in page 19.)

[BLWZ05] H. Broersma, X. Li, G. Woeginger, and S. Zhang. Paths and cycles in colored graphs. *Australas. J. Combin.*, 31:299–311, 2005. (Cited in page 57.)

[BM08] A. Bondy and U. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2008. (Cited in page 3.)

[BN15] H. Bodlaender and J. Nederlof. Subexponential time algorithms for finding small tree and path decompositions. In N. Bansal and I. Finocchi, editors, *Algorithms ? ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 179–190. Springer Berlin Heidelberg, 2015. (Cited in page 151.)

[Bod96] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. (Cited in pages 34 and 150.)

[Bod97] H. Bodlaender. Treewidth: Algorithmic techniques and results. In *Mathematical Foundations of Computer Science 1997*, volume 1295 of *Lecture Notes in Computer Science*, pages 19–36. Springer Berlin Heidelberg, 1997. (Cited in page 34.)

[CDG02] G. Confessore, P. Dell'Olmo, and S. Giordani. An approximation result for the interval coloring problem on claw-free chordal graphs. *Discrete Appl. Math.*, 120(1-3):73–90, August 2002. (Cited in pages 88 and 108.)

[CDKM00] R. D. Carr, S. Doddi, G. Konjevod, and M. Marathe. On the red-blue set cover problem. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 345–353, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics. (Cited in page 57.)

[CDP+07] D. Coudert, P. Datta, S. Perennes, H. Rivano, and M.-E. Voge. Shared risk resource group: Complexity and approximability issues. *Parallel*

Processing Letters, 17(2):169–184, June 2007. (Cited in pages 56, 57, 75, 186 and 188.)

[CGK92] I. Chlamtac, A. Ganz, and G. Karmi. Lightpath communications: an approach to high bandwidth optical WAN's. *Communications, IEEE Transactions on*, 40(7):1171–1182, 1992. (Cited in pages 80, 84 and 189.)

[Cha03] V. T. Chakaravarthy. New results on the computability and complexity of points–to analysis. In *ACM SIGPLAN-SIGACT symposium on principles of programming languages (POPL)*, pages 115–125. ACM, 2003. (Cited in pages 43, 44, 68 and 69.)

[CHW+13] Y. Chen, N. Hua, X. Wan, H. Zhang, and X. Zheng. Dynamic lightpath provisioning in optical WDM mesh networks with asymmetric nodes. *Photonic Network Com.*, 25(3):166–177, 2013. (Cited in pages 13, 14 and 185.)

[CJM14] D. Coudert, B. Jaumard, and F. Z. Moataz. Dynamic Routing and Spectrum Assignment with Non-Disruptive Defragmentation. In *ALGOTEL 2014 – 16èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, pages 1–4, Le Bois-Plage-en-Ré, France, June 2014. (Cited in pages 6, 8, 109 and 195.)

[CNP+11] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *CoRR*, abs/1103.0534, 2011. (Cited in page 34.)

[Cou90] B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12 – 75, 1990. (Cited in page 150.)

[CPM+13] F. Cugini, F. Paolucci, G. Meloni, G. Berrettini, M. Secondini, F. Fresi, N. Sambo, L. Poti, and P. Castoldi. Push-pull defragmentation without traffic disruption in flexible grid optical networks. *Lightwave Technology, Journal of*, 31(1):125–133, Jan 2013. (Cited in page 111.)

[CPRV14] D. Coudert, S. Pérennes, H. Rivano, and M.-E. Voge. Combinatorial optimization in networks with Shared Risk Link Groups. Research report RR-8575, Inria, July 2014. (Cited in page 57.)

[CS88] M. Chrobak and M. Slusarek. On some packing problem related to dynamic storage allocation. *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, 22(4):487–499, 1988. (Cited in pages 86 and 108.)

[CS15] I. Cisco Systems. Cisco visual networking index: Forecast and methodology, 2014-2019. Technical report, Cisco Systems, Inc., May 2015. Available at http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html. (Cited in pages 1 and 184.)

[CSS+12] F. Cugini, M. Secondini, N. Sambo, G. Bottari, G. Bruno, P. Iovanna, and P. Castoldi. Push-pull technique for defragmentation in flexible optical networks. In *Optical Fiber Communication Conference and Exposition (OFC/NFOEC), 2012 and the National Fiber Optic Engineers Conference*, pages 1–3, 2012. (Cited in pages 6 and 111.)

[CTV11] K. Christodoulopoulos, I. Tomkos, and E. Varvarigos. Elastic bandwidth allocation in flexible ofdm-based optical networks. *Lightwave Technology, Journal of*, 29(9):1354–1366, May 2011. (Cited in page 135.)

[DF95] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141(1-2):109–131, 1995. (Cited in page 74.)

[DF13] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. (Cited in page 74.)

[DG05] J. Doucette and W. Grover. Shared-risk logical span groups in span-restorable optical networks: Analysis and capacity planning model. *Photonic Network Communications*, 9(1):35–53, 2005. (Cited in page 56.)

[DGSB10] J. Dees, R. Geisberger, P. Sanders, and R. Bader. Defining and computing alternative routes in road networks. Technical report, Fakultät für Informatik, Karlsruher Institut für Technologie, 2010. (Cited in page 16.)

[DH08] E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008. (Cited in page 150.)

[DKZ13] D. Dereniowski, W. Kubiak, and Y. Zwols. Minimum length path decompositions. *CoRR*, abs/1302.2788, 2013. (Cited in pages 150, 151, 152, 153 and 154.)

[DL05] J. Desrosiers and M. Lübbecke. A primer in column generation. In *Column Generation*, pages 1–32. Springer US, 2005. (Cited in page 138.)

[Dor10] F. Dorn. Dynamic programming and planarity: Improved tree-decomposition based algorithms. *Discrete Applied Mathematics*, 158(7):800 – 808, 2010. Third Workshop on Graph Classes, Optimization, and Width Parameters Eugene, Oregon, USA, October 2007. (Cited in page 39.)

[DPBF05] F. Dorn, E. Penninkx, H. Bodlaender, and F. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions. In *Algorithms ? ESA 2005*, volume 3669 of *Lecture Notes in Computer Science*, pages 95–106. Springer Berlin Heidelberg, 2005. (Cited in pages 39 and 40.)

[DS08] P. Datta and A. Somani. Graph transformation approaches for diverse routing in shared risk resource group (SRRG) failures. *Computer Networks*, 52(12):2381–2394, August 2008. (Cited in pages 56 and 57.)

[Far06] A. Faragó. A graph theoretic model for complex network failure scenarios. In *8th INFORMS Telecommunications Conference*, Dallas, Texas, March 2006. (Cited in pages 56 and 57.)

[FF57] L. R. Ford and D. R. Fulkerson. A simple algorithm for finding maximal network flows and an application to the hitchcock problem. *Canadian Journal of Mathematics*, 9:210–218, 1957. (Cited in pages 66 and 75.)

[FGK10] M. R. Fellows, J. Guob, and I. Kanj. The parameterized complexity of some minimum label problems. *Journal of Computer and System Sciences*, 76(8):727–740, December 2010. (Cited in pages 56 and 57.)

[FHW80] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111 – 121, 1980. (Cited in page 42.)

[Ger96] J. Gergov. Approximation algorithms for dynamic storage allocation. In *Algorithms ? ESA '96*, volume 1136 of *Lecture Notes in Computer Science*, pages 52–61. Springer Berlin Heidelberg, 1996. (Cited in page 86.)

[Ger99] J. Gergov. Algorithms for compile-time memory optimization. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, pages 907–908, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics. (Cited in pages 87 and 108.)

[GHP95] P. Galinier, M. Habib, and C. Paul. Chordal graphs and their clique graphs. In *Graph-Theoretic Concepts in Computer Science*, volume 1017 of *Lecture Notes in Computer Science*, pages 358–371. Springer Berlin Heidelberg, 1995. (Cited in page 107.)

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. (Cited in pages 65, 152 and 153.)

[GJ85] M. C. Golumbic and R. E. Jamison. Edge and vertex intersection of paths in a tree. *Discrete Mathematics*, 55(2):151 – 159, 1985. (Cited in pages 92 and 93.)

[GJLY12] O. Gerstel, M. Jinno, A. Lord, and S. Yoo. Elastic optical networking: a new dawn for the optical layer? *Communications Magazine, IEEE*, 50(2):s12–s20, February 2012. (Cited in pages 2, 79, 80, 187, 188 and 189.)

[GL07] L. Guo and L. Li. A novel survivable routing algorithm with partial shared-risk link groups (SRLG)-disjoint protection based on differentiated reliability constraints in WDM optical mesh networks. *IEEE/OSA Journal of Lightwave Technology*, 25(6):1410–1415, June 2007. (Cited in page 57.)

[GLMM12] L. Gourvčs, A. Lyra, C. Martinhon, and J. Monnot. On paths, trails and closed trails in edge-colored graphs. *Discrete Mathematics and Theoretical Computer Science*, 14(2), 2012. (Cited in pages 4, 41, 42, 46, 53 and 186.)

[GLMM13] L. Gourvès, A. Lyra, C. A. Martinhon, and J. Monnot. Complexity of trails, paths and circuits in arc-colored digraphs. *Discrete Applied Mathematics*, 161(6):819 – 828, 2013. (Cited in page 15.)

[GLS08] M. C. Golumbic, M. Lipshteyn, and M. Stern. Representing edge intersection graphs of paths on degree 4 trees. *Discrete Mathematics*, 308(8):1381 – 1387, 2008. Third European Conference on Combinatorics Graph Theory and Applications Third European Conference on Combinatorics. (Cited in page 92.)

[GM08] E. Gutiérrez and A. L. Medaglia. Labeling algorithm for the shortest path problem with turn prohibitions with application to large-scale road networks. *Annals of Operations Research*, 157(1):169–182, 2008. (Cited in pages 15 and 19.)

[GMO76] H. Gabow, S. N. Maheshwari, and L. Osterweil. On two problems in the generation of program test paths. *Software Engineering, IEEE Transactions on*, SE-2(3):227–231, Sept 1976. (Cited in page 15.)

[Goe95] M. X. Goemans. An approximation algorithm for scheduling on three dedicated machines. *Discrete Applied Mathematics*, 61(1):49 – 59, 1995. (Cited in page 86.)

[Gol04] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57)*. North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 2004. (Cited in pages 87, 88 and 108.)

[GT98] A. Ghatak and K. Thyagarajan. *An Introduction to Fiber Optics*. Cambridge University Press, 1998. (Cited in pages 1 and 183.)

[HCC00] J. Huang, J. Chen, and S. Chen. A simple linear-time approximation algorithm for multi-processor job scheduling on four processors. In *Algorithms and Computation*, volume 1969 of *Lecture Notes in Computer Science*, pages 60–71. Springer Berlin Heidelberg, 2000. (Cited in page 86.)

[HCD09] C.-C. Hsu, D.-R. Chen, and H.-Y. Ding. An efficient algorithm for the shortest path problem with forbidden paths. In *Algorithms and Architectures for Parallel Processing*, volume 5574 of *Lecture Notes in Computer Science*, pages 638–650. Springer Berlin Heidelberg, 2009. (Cited in page 15.)

[HMS07] R. Hassin, J. Monnot, and D. Segev. Approximation algorithms and hardness results for labeled connectivity problems. *Journal of Combinatorial Optimization*, 14(4):437–453, November 2007. (Cited in page 57.)

[HŮ99] J. Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Mathematica*, 182(1):105–142, 1999. (Cited in page 72.)

[HT73] J. Hopcroft and R. Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973. (Cited in pages 67 and 165.)

[HTTN11] T. Hashiguchi, K. Tajima, Y. Takita, and T. Naito. Node-disjoint paths search in WDM networks with asymmetric nodes. In *Optical Network Design and Modeling (ONDM), 2011 15th International Conference on*, pages 1–6, Feb 2011. (Cited in pages 13, 16 and 185.)

[Hu03] J. Hu. Diverse routing in mesh optical networks. *IEEE Transactions on Communications*, 51:489–494, March 2003. (Cited in pages 57 and 186.)

[HvdVV94] J. Hoogeveen, S. van de Velde, and B. Veltman. Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Applied Mathematics*, 55(3):259 – 272, 1994. (Cited in page 86.)

[JMT09] B. Jaumard, C. Meyer, and B. Thiongane. On column generation formulations for the rwa problem. *Discrete Appl. Math.*, 157(6):1291–1308, March 2009. (Cited in pages 136, 138 and 143.)

[JRN04] Q. Jiang, D. Reeves, and P. Ning. Improving robustness of PGP keyrings by conflict detection. In *RSA Conference Cryptographers' Track (CT-RSA)*, volume 2964 of *LNCS*, pages 194–207. Springer, February 2004. (Cited in page 56.)

[JTK+09] M. Jinno, H. Takara, B. Kozicki, Y. Tsukishima, Y. Sone, and S. Matsuoka. Spectrum-efficient and scalable elastic optical path network: architecture, benefits, and enabling technologies. *Communications Magazine, IEEE*, 47(11):66–73, November 2009. (Cited in pages 2, 79, 187 and 189.)

[Kan91] V. Kann. Maximum bounded 3-dimensional matching is MAX SNP-complete. *Inf. Process. Lett.*, 37(1):27–35, January 1991. (Cited in page 72.)

[Kie88] H. Kierstead. The linearity of first-fit coloring of interval graphs. *SIAM Journal on Discrete Mathematics*, 1(4):526–530, 1988. (Cited in page 86.)

[Kie91] H. Kierstead. A polynomial time approximation algorithm for dynamic storage allocation. *Discrete Mathematics*, 88(2-3):231 – 237, 1991. (Cited in page 86.)

[KIU86] Y. Kajitani, A. Ishizuka, and S. Ueno. Characterization of partial 3-trees in terms of three structures. *Graphs and Combinatorics*, 2(1):233–246, 1986. (Cited in page 150.)

[Klo94] T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994. (Cited in page 34.)

[KMMN15a] M. M. Kanté, F. Z. Moataz, B. Momège, and N. Nisse. Finding Paths in Grids with Forbidden Transitions. In *WG 2015, 41st International Workshop on Graph-Theoretic Concepts in Computer Science*, Munich, Germany, June 2015. (Cited in pages 4, 7, 13 and 195.)

[KMMN15b] M. M. Kanté, F. Z. Moataz, B. Momège, and N. Nisse. On paths in grids with forbidden transitions. In *ALGOTEL 2015 - 17èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, Beaune, France, June 2015. (Cited in pages 4, 8, 13 and 195.)

[KP09] P. Kolman and O. Pangrác. On the complexity of paths avoiding forbidden pairs. *Discrete Applied Mathematics*, 157(13):2871 – 2876, 2009. (Cited in page 15.)

[KW11] M. Klinkowski and K. Walkowiak. Routing and spectrum assignment in spectrum sliced elastic optical path network. *Communications Letters, IEEE*, 15(8):884–886, August 2011. (Cited in page 135.)

[LLQ04] S. Li, H. Leong, and S. Quek. New approximation algorithms for some dynamic storage allocation problems. In *Computing and Combinatorics*, volume 3106 of *Lecture Notes in Computer Science*, pages 339–348. Springer Berlin Heidelberg, 2004. (Cited in pages 87, 88 and 97.)

[LMN14] B. Li, F. Z. Moataz, and N. Nisse. Minimum Size Tree-Decompositions. In *9th International colloquium on graph theory and combinatorics (ICGT)*, Grenoble, France, 2014. (Cited in pages 7, 149, 151 and 194.)

[LMNS15] B. Li, F. Z. Moataz, N. Nisse, and K. Suchan. Minimum Size Tree-decompositions. In *LAGOS 2015 – VIII Latin-American Algorithms, Graphs and Optimization Symposium*, Beberibe, Ceará, Brazil, May 2015. (Cited in pages 7, 149, 151 and 194.)

[LS12] D. Luxen and D. Schieferdecker. Candidate sets for alternative routes in road networks. In *Experimental Algorithms*, volume 7276 of *Lecture Notes in Computer Science*, pages 260–270. Springer Berlin Heidelberg, 2012. (Cited in page 16.)

[LW05] X. Luo and B. Wang. Diverse routing in WDM optical networks with shared risk link group (SRLG) failures. In *Proc. DRCN*, pages 445–452. IEEE, October 2005. (Cited in pages 4, 56, 57, 59, 60 and 186.)

[Mar10] D. Marx. Fixed parameter algorithms. Available at `http://www.cs.bme.hu/~dmarx/papers/marx-warsaw-fpt2`, 2010. (Cited in page 35.)

[MB99] P. K. Murthy and S. S. Bhattacharyya. Approximation algorithms and heuristics for the dynamic storage allocation problem. Technical report, Univ. Maryland Inst. Adv. Comput. Studies, College Park, 1999. (Cited in pages 87, 88 and 93.)

[MG02] C. Murthy and M. Gurusamy. *WDM Optical Networks: Concepts, Design, and Algorithms*. Prentice Hall PTR, 2002. (Cited in page 79.)

[Moa15] F. Z. Moataz. On Spectrum Assignment in Elastic Optical Tree-Networks. In *ALGOTEL 2015 - 17èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, Beaune, France, June 2015. (Cited in pages 5, 8, 83 and 195.)

[MT91] J. Matousek and R. Thomas. Algorithms finding tree-decompositions of graphs. *Journal of Algorithms*, 12(1):1 – 22, 1991. (Cited in page 150.)

[Muk97] B. Mukherjee. *Optical Communication Networks*. McGraw-Hill series on computer communications. McGraw-Hill, 1997. (Cited in pages 1 and 79.)

[Nie06] R. Niedermeier. *Invitation to fixed-parameter algorithms*. Oxford lecture series in mathematics and its applications. Oxford University Press, 2006. (Cited in page 74.)

[NW88] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY, USA, 1988. (Cited in page 138.)

[PG13] L. D. P. Pugliese and F. Guerriero. Shortest path problem with forbidden paths: The elementary version. *European Journal of Operational Research*, 227(2):254 – 267, 2013. (Cited in page 15.)

[PPR05] S. V. Pemmaraju, S. Penumatcha, and R. Raman. Approximating interval coloring and max-coloring in chordal graphs. *J. Exp. Algorithmics*, 10, December 2005. (Cited in pages 96 and 108.)

[RM99] S. Ramamurthy and B. Mukherjee. Survivable WDM mesh networks. part i-protection. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 744–751 vol.2, Mar 1999. (Cited in pages 2, 11 and 184.)

[RPZ+13] M. Ruiz, M. Pióro, M. Zotkiewicz, M. Klinkowski, and L. Velasco. Column generation algorithm for rsa problems in flexgrid optical networks. *Photonic Network Communications*, 26(2-3):53–64, 2013. (Cited in pages x, 135, 136, 138, 139, 143, 144, 146 and 147.)

[RS86] N. Robertson and P. D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. (Cited in pages 33 and 149.)

[RS95] R. Ramaswami and K. N. Sivarajan. Routing and wavelength assignment in all-optical networks. *IEEE/ACM Trans. Netw.*, 3(5):489–500, October 1995. (Cited in pages 79 and 189.)

[RS02] R. Ramaswami and K. Sivarajan. *Optical Networks: A Practical Perspective*. Morgan Kaufmann series in networking. Morgan Kaufmann Publishers, 2002. (Cited in pages 1, 2, 11, 183 and 184.)

[San96] D. P. Sanders. On linear recognition of tree-width at most four. *SIAM J. Discret. Math.*, 9(1):101–117, 1996. (Cited in page 150.)

[SAX14] M. Sekiya, Y. Aoki, and W. Xi. Photonic network defragmenttaion technology improving resource utilization during operation. *FUJITSU Sci. Tech. J., Vol. 50, No. 1*, January 2014. (Cited in page 111.)

[Sen92] J. M. Senior. *Optical Fiber Communications (2Nd Ed.): Principles and Practice*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1992. (Cited in pages 1 and 183.)

[Sha15] M. Shalom. On the interval chromatic number of proper interval graphs. *Discrete Mathematics*, 338(11):1907 – 1916, 2015. (Cited in page 88.)

[SJK03] L. Speičvcys, C. S. Jensen, and A. Kligys. Computational data modeling for network-constrained moving objects. In *Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems*, GIS '03, pages 118–125, New York, NY, USA, 2003. ACM. (Cited in page 15.)

[ST84] J. W. Suurballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14(2):325–336, 1984. (Cited in pages 55 and 59.)

[Suu74] J. W. Suurballe. Disjoint paths in a network. *Networks*, 4(2):125–145, 1974. (Cited in pages 55 and 59.)

[SYR05] L. Shen, X. Yang, and B. Ramamurthy. Shared risk link group (SRLG)-diverse path provisioning under hybrid service level agreements in wavelength-routed optical mesh networks. *IEEE/ACM Transactions on Networking*, 13:918–931, August 2005. (Cited in page 57.)

[Sys79] M. M. Syslo. Characterizations of outerplanar graphs. *Discrete Mathematics*, 26(1):47 – 53, 1979. (Cited in page 175.)

[SZDS13] S. Shirazipourazad, C. Zhou, Z. Derakhshandeh, and A. Sen. On routing and spectrum allocation in spectrum-sliced optical networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 385–389, April 2013. (Cited in pages 84, 87, 88 and 92.)

[Sze03] S. Szeider. Finding paths in graphs avoiding forbidden transitions. *Discrete Applied Mathematics*, 126(2 - 3):261 – 273, 2003. (Cited in pages 15, 17, 18, 19, 20, 21, 53, 60 and 186.)

[Tam87] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, June 1987. (Cited in page 32.)

[TBL+14] S. Talebi, E. Bampis, G. Lucarelli, I. Katib, and G. N. Rouskas. Spectrum assignment in optical networks: A multiprocessor scheduling perspective. *J. Opt. Commun. Netw.*, 6(8):754–763, Aug 2014. (Cited in pages 84, 85, 88 and 89.)

[TP97] J. A. Telle and A. Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Discret. Math.*, 10(4):529–550, August 1997. (Cited in page 34.)

[TR04a] A. Todimala and B. Ramamurthy. IMSH: an iterative heuristic for SRLG diverse routing in WDM mesh networks. In *IEEE International Conference on Computer Communications and Networks (ICCCN)*, pages 199 – 204. IEEE, 2004. (Cited in page 57.)

[TR04b] A. Todimala and B. Ramamurthy. Survivable virtual topology routing under shared risk link groups in WDM networks. In *International Conference on Broadband Communications, Networks and Systems (BroadNets)*, pages 130–139, San Jose, CA, USA, October 2004. IEEE. (Cited in page 57.)

[TY84] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, July 1984. (Cited in page 93.)

[Val81] L. G. Valiant. Universality considerations in VLSI circuits. *IEEE Transactions on Computers*, 30(2):135–140, 1981. (Cited in page 32.)

[VD05] D. Villeneuve and G. Desaulniers. The shortest path problem with forbidden paths. *European Journal of Operational Research*, 165(1):97 – 107, 2005. (Cited in page 15.)

[VKRC12] L. Velasco, M. Klinkowski, M. Ruiz, and J. Comellas. Modeling the routing and spectrum allocation problem for flexgrid optical networks. *Photonic Network Communications*, 24(3):177–186, 2012. (Cited in page 135.)

[WC83] J. A. Wald and C. J. Colbourn. Steiner trees, partial 2-trees, and minimum ifi networks. *Networks*, 13(2):159–167, 1983. (Cited in pages 150 and 167.)

[WCP11] Y. Wang, X. Cao, and Y. Pan. A study of the routing and spectrum allocation in spectrum-sliced elastic optical path networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 1503–1511, April 2011. (Cited in page 135.)

[Wik] `https://en.wikipedia.org/wiki/File:Optical-fibre.png`. (Cited in pages 1 and 183.)

[Win02] S. Winter. Modeling costs of turns in route planning. *GeoInformatica*, 6(4):345–361, 2002. (Cited in pages 15, 18, 19, 53 and 186.)

[WLV13] P. Wright, A. Lord, and L. Velasco. The network capacity benefits of flexgrid. In *Optical Network Design and Modeling (ONDM), 2013 17th International Conference on*, pages 7–12, April 2013. (Cited in pages 79, 109 and 189.)

[WM13] R. Wang and B. Mukherjee. Provisioning in elastic optical networks with non-disruptive defragmentation. *Lightwave Technology, Journal of*, 31(15):2491–2500, 2013. (Cited in pages x, 109, 111, 112, 113, 115, 116, 125, 126 and 191.)

[WM14] R. Wang and B. Mukherjee. Spectrum management in heterogeneous bandwidth optical networks. *Optical Switching and Networking*, 11, Part A(0):83 – 91, 2014. (Cited in pages 97 and 110.)

[YVJ05] S. Yuan, S. Varma, and J. Jue. Minimum-color path problems for reliability in mesh networks. In *Proc. IEEE INFOCOM*, volume 4, pages 2658–2669, Houston, TX, USA, March 2005. (Cited in pages 56 and 57.)

[ZJ00] H. Zang and J. P. Jue. A review of routing and wavelength assignment approaches for wavelength-routed optical WDM networks. *Optical Networks Magazine*, 1:47–60, 2000. (Cited in pages 79 and 189.)

[ZSXT07] Q. Zhang, J. Sun, G. Xiao, and E. Tsang. Evolutionary algorithms refining a heuristic: A hybrid method for shared-path protections in WDM networks under SRLG constraints. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 37(1):51–61, February 2007. (Cited in page 57.)