Exact and approximation algorithms for some packing and covering problems in graphs

Cédric Bentz

Habilitation à Diriger les Recherches

in the field of

Computer Science (Informatique)

entitled

Exact and approximation algorithms for some packing and covering problems in graphs

submitted at

Université Pierre et Marie Curie

by

Cédric BENTZ

and publicly defended on

November 13th, 2017

in front of the following committee members:

Éric COLIN DE VERDIÈRE CNRS Research Director at LIGM, Marne-La-Vallée	*Reviewer*
Thomas ERLEBACH Professor at Univ. Leicester, UK	*Reviewer*
Irena RUSU Professor at Univ. Nantes	*Reviewer*
Bruno ESCOFFIER Professor at Univ. Pierre et Marie Curie, Paris	*Advisor*
Cristina BAZGAN Professor at Univ. Paris Dauphine	
Alain BILLIONNET Professor Emeritus at ENSIIE, Évry	
Yannis MANOUSSAKIS Professor at Univ. Paris Sud	

Contents

Chapter 1

Introduction

1.1 Foreword

Long before mobile computing changed our daily lives drastically, the development of computers, which began in the middle of the 20th century, already led scientists to study how to make use of them as efficiently as possible. In particular, it led them to ask new questions such as: what can be computed by a computer? And how fast can it be computed?

Computer scientists studied such questions with the help of formal models (on which standard computers, even the most recent ones, are based), namely *Turing machines*. In particular, it enabled theorists to define complexity classes for combinatorial problems, such as **P** and **NP** (both contain only decision problems) [41]. The complexity class a given problem belongs to then reflects the kind of algorithms (in terms of running time or of required space) it can be solved with. Problems in **NP** can be solved in time polynomial in the size of the input by a non deterministic Turing machine, and problems in **P** can be solved in time polynomial in the size of the input by a deterministic Turing machine. Lots of practical combinatorial problems can be shown to be in **NP**, but few of them are likely to belong to **P**.

The work of S. Cook and R. Karp showed the existence of *hard* problems for **NP**, i.e., problems harder than any other problem in **NP**, as well as relations between problems in **NP**. Such relations are obtained via the notion of *reductions*, which can also be seen as *transformations*. If any instance of a given problem Π_1 can be transformed in polynomial time into an instance of a given problem Π_2, then Π_1 can be polynomially reduced to Π_2, and hence Π_2 is more general, or harder, than Π_1. The hardest problems in **NP**, also called ***NP-complete*** problems, are then the ones to which any problem in **NP** can be polynomially reduced. Given an optimization problem, one can always define an associated decision problem (called its *decision version*), and a problem that is not in **NP** (as, for instance, it is not a decision problem), but is nevertheless more general than any problem in **NP**, is called ***NP-hard***.

One of the main conjectures in theoretical computer science is that **P** is not equal to **NP**, i.e., that there exist problems in **NP** (in particular, all the **NP**-complete problems) that cannot be solved in *polynomial time* (i.e., in time polynomial in the size of the input by a deterministic Turing machine). When studying a practical problem (or its decision version), the first question to be answered is whether this given problem is **NP**-complete (and hence, from this conjecture, unlikely to be in **P**) or not, and, if the answer is "yes", then we can try to solve it practically in several ways.

First, if the problem at hand is an optimization problem, then we can try to compute in polynomial time a "good", but not necessarily optimal, solution. An algorithm that computes such a solution is generally called an *approximation algorithm* if the quality of the computed solution is guaranteed a priori (before actually running the algorithm), and a *heuristic* otherwise. When considering algorithms for maximization problems, the *approximation ratio* of such algorithms is defined as the ratio between the optimal value and the value of the returned solution [66]. (For minimization problems, just use the inverse ratio.) If this ratio is equal to 1, then the algorithm computes an optimal solution. Otherwise, the closer to 1, the better. In order to use these concepts to classify optimization problems, theoretical computer scientists have defined approximation classes for optimization problems, which are similar to complexity classes for decision problems. For instance, the class of problems that admit approximation algorithms having a ratio as close to 1 as wanted (possibly without reaching 1) is called **PTAS** (for *Polynomial-Time Approximation Schemes*). Another example is the class of problems that admit approximation algorithms having a constant ratio, which is called **APX**. The hardest problems for **APX**, which are not in **PTAS** (unless **P** equals **NP**), are called ***APX-hard*** (or ***APX-complete*** if they are in **APX**).

Second, we can express the problem that must be solved with the help of some classical model in mathematical programming, such as linear programming with integral variables (also called integer linear programming), and then solve the problem using suitable algorithms (such as *Branch & Bound*). Obviously, the running time of such algorithms is not guaranteed to be polynomial, but in practice one can hope they run faster than a basic exhaustive search, in particular when implemented in a commercial solver.

Third, one can solve a more restricted problem, by considering real-life additional constraints or imposing a special (but relevant) structure on the instances to be solved, in the hope that this may yield an easier problem (e.g., one that belongs to **P**). Another way of restricting the problem is to assume that, in practice, some part of the input (called a *parameter*) is *small*. This is the basic idea in the field of *parameterized complexity*, which extends classical computational complexity [30]. Given a parameter k for a problem Π, an algorithm solving Π is **FPT** (or *Fixed-Parameter Tractable*) with respect to k if it runs in time $O(f(k)n^c)$, where $f(\cdot)$ is any computable function, n is the size of the input, and c is a constant. In other words, the non-polynomial

part in the running time only depends on k. The class **FPT** is then the one of parameterized problems admitting **FPT** algorithms. Other parameterized classes named **W[1]**, **W[2]**, etc., have been defined, and contain problems "harder" than the ones in **FPT**. Actually, the relationship between **FPT** and **W[1]**/**W[2]**/etc. is similar to the one between **P** and **NP**: a parameterized problem that is **W[1]**-hard/**W[2]**-hard/etc. is unlikely to be **FPT**, unless some very likely conjecture in parameterized complexity collapses.

We shall present in Section 1.2 the class of combinatorial problems that we will be interested into throughout this report, as well as its general structure, and then provide in Section 1.3 some basic **NP**-complete problems, as well as some notions from graph theory that will be useful.

1.2 General presentation of the report

In this report, I shall review all the results, related to several covering and packing problems in graphs, that were obtained with different co-authors since 2003. Assume we are given an $m \times n$ matrix $A = (a_{ij})$ and two vectors b and c of respective sizes m and n, which are such that all a_{ij}'s, all b_i's and all c_j's are positive rational numbers. A *packing problem* is a problem that can be modelled as the following (integer) linear program:

$$\begin{array}{lll} \max & \sum_{j=1}^{n} c_j x_j & \\ \text{s. t.} & \sum_{j=1}^{n} a_{ij} x_j \leq b_i & \forall i \in \{1, \ldots, m\} \\ & x_j \geq 0 \ (x_j \in \mathbb{N}) & \forall j \in \{1, \ldots, n\} \end{array} \tag{1.1}$$

A *covering problem* is a problem that can be modelled as the following (integer) linear program:

$$\begin{array}{lll} \min & \sum_{j=1}^{n} c_j x_j & \\ \text{s. t.} & \sum_{j=1}^{n} a_{ij} x_j \geq b_i & \forall i \in \{1, \ldots, m\} \\ & x_j \geq 0 \ (x_j \in \mathbb{N}) & \forall j \in \{1, \ldots, n\} \end{array} \tag{1.2}$$

In both cases, the x_j's can take any positive real values, or only positive integral values, depending on the problem that is considered. When the x_j's are not required to take positive integral values, the problem can be solved in time *polynomial in n and m* as a linear program. Otherwise, the problem generally becomes hard. We shall give examples of such packing and covering problems in the next section of this chapter, and also throughout the report.

The first of my articles that needs to be mentioned here is [BCR13]. This article deals with the issue of modeling and solving packing and covering problems in graphs using linear programming: it was written with Denis Cornaz and Bernard Ries, and greatly inspired the presentation of this report. Indeed, the general idea on which this article is based is that all kinds of packing and covering problems in graphs share, to some extent, some common principles, such as the fact that, while finding (near-)optimal solutions is often a hard task, finding provably good feasible solutions may be easy.

Any constraint having the same form as Constraints (1.1) will be called a packing constraint, and similarly any constraint having the same form as Constraints (1.2) will be called a covering constraint. In this report, we shall also consider packing problems with additional covering constraints, or covering problems with additional packing constraints. In other words, we shall consider packing problems, covering problems, as well as combinatorial problems with both packing and covering constraints.

The general structure of the present report is as follows. In the next section, we describe the basic notions and problems that will be needed to read and understand the remainder of the report. In Chapter 2, we study a pair of related packing and covering problems, which respectively generalize the maximum flow and minimum cut problems: namely, the maximum integral multiflow problem and the minimum multicut problem, as well as some of their variants. In Chapter 3, we study several problems consisting of coloring the vertices of a graph, i.e., covering them by "colors", under different types of cardinality constraints, which can naturally be seen as packing constraints. In Chapter 4, we study d-transversal problems, i.e., problems consisting of covering a given number d of times all the optimal solutions of a given graph optimization problem. Finally, in Chapter 5, we study several variants of the Steiner tree problem, which asks to cover, by a connected subgraph (called a Steiner tree), some pre-specified vertices (called terminals) of a graph.

1.3 Basic notions and problems

We shall now review the basic notions and problems that we will need in the remainder of the report.

Cook's Theorem states that the *satisfiability problem* (called SAT), a basic problem in propositional logic, is **NP**-complete [41]. Assume we are given a set of boolean variables $x_1, x_2, \ldots$ (each variable x_i having two associated literals, namely x_i itself and its negation $\bar{x}_i$) and a set of clauses, each one of them being a disjunction of some the literals associated with the boolean variables. Then, each clause among this set of clauses is satisfied, i.e., of value *true*, if at least one of its literals is *true*. The problem SAT consists in determining whether there exists a truth assignment (i.e., a way of giving a value *true* or *false* to each variable), called *satisfying truth assignment*, for

which *all* clauses are satisfied. Variants of this problem can be obtained, in particular, by restricting the number of literals by clause and/or adding an objective function. For instance, the problem called 3SAT is the special case of SAT where each clause contains at most three literals, and the problem called MAX2SAT is the variant of SAT where each clause contains at most two literals, and the problem consists in determining the maximum number of clauses that can be satisfied by a truth assignment. Both these variants, as well as other similar variants, remain **NP**-hard (**NP**-complete for 3SAT). This is the case, for instance, for the problem MONOTONE1-IN-3SAT, a variant of 3SAT that will be detailed when needed.

Throughout the report, we shall also need some basic graph notions. We will describe the main ones here, but, for any other basic notion not defined here, the reader can refer to any book about graph theory, should it be needed. Given a graph G, the *line graph of* G is the graph where there is one vertex for each edge of G, and an edge between two of its vertices if the associated two edges in G share a vertex. A graph G is *bipartite* if its vertex set can be partitioned into two sets L (for *Left*) and R (for *Right*) such that, for any edge uv of G, we have $u \in L$ and $v \in R$. A graph is *complete* if there is an edge between any two of its vertices, and we define the *complement* of a graph G with n vertices as the graph obtained from G by taking a complete graph on the n vertices of G, and then removing from this complete graph the edges of G. A *subgraph* (i.e., a subset of vertices and/or edges) of a graph G is a *clique* if it is a complete graph in its own right.

Given a graph G, apart from cliques, we can define other types of subgraphs: for instance, a *stable set* in G is the complement of a clique in G, a *matching* in G is a set of vertex-disjoint edges, and a *vertex cover* in G is a set of vertices of G that contains at least one vertex of each edge of G.

In fact, for each of these four types of subgraphs, we can define an associated graph optimization problem, by searching, in a graph G, the maximum size (or minimum size, for a vertex cover) of such a subgraph, which will simply be called the *stability number* of G in the case of a stable set. All the problems obtained in this way are **NP**-hard in general, except finding a matching of maximum size, which can be done in polynomial time. However, they can all be solved in polynomial time in bipartite graphs: in particular, the famous *König's Theorem* for bipartite graphs states that, in such graphs, the maximum size of a matching equals the minimum size of a vertex cover.

A notion related to stable sets is the one of *proper colorings*: a proper (vertex) coloring of a graph G is an assignment of *colors* (for instance, numbers) on the vertices of G, such that any two adjacent vertices receive different colors. Such a coloring can also be seen as a partition of the vertices of G into stable sets (one for each color). The minimum number of colors needed for a proper coloring of a graph G is called the *chromatic number* of G, and is denoted by $\chi(G)$: computing $\chi(G)$ is **NP**-hard in general, but we have $\chi(G) \leq 2$ in any bipartite graph G (as L and R are two stable sets).

We end this section by describing a graph parameter that can be seen as a measurement of the tree-likeness of a graph. Indeed, not all graphs are trees, and bipartite graphs are not the only relevant generalizations of trees.

A *tree decomposition* of a graph $G = (V, E)$ is a pair $\big(\{X_i | i \in I\}, T\big)$ where $X_i \subseteq V, \forall i \in I$, are *bags* of vertices of G, $T = (I, F)$ is a tree, and:

(1) $\bigcup_{i \in I} X_i = V$,

(2) For every edge $uv \in E$, there is an $i \in I$ such that $u, v \in X_i$,

(3) For all $i, j, l \in I$, if j lies on the path between i and l, then $X_i \cap X_l \subseteq X_j$.

The *width* of a given tree decomposition of a graph G equals $\max_{i \in I} |X_i| - 1$. The *tree-width* of a graph G, denoted by $tw(G)$, is the minimum width of a tree decomposition of G, taken over all tree decompositions of G. Note that trees (and hence chains and stars) have tree-width 1. Without loss of generality, we can also assume that the tree decomposition is *nice* [55], i.e.:

- T is rooted at some node r,
- T is binary and has $O(|V|)$ nodes,
- If a node i has two children j and k then $X_i = X_j = X_k$ (join node)
- If a node i has one child j, then either
  - (a) $|X_i| = |X_j| - 1$ and $X_i \subset X_j$ (forget node)
  - (b) $|X_i| = |X_j| + 1$ and $X_j \subset X_i$ (introduce node)

Given two vertices i, j of T, we will use the notation $j \succeq i$ to denote the fact that j is either i or a descendant of i with respect to r. Given a node $i \in I$, let $Y_i = E \cap (X_i \times X_i)$, i.e., Y_i is the subset of E induced by the vertices in X_i. Moreover, let $T_i = \bigcup_{j \succeq i} X_j$ and let $G[T_i]$ be the subgraph of G induced by T_i.

Chapter 2

Integral multiflows & multicuts

2.1 Introduction

In this chapter, we review results about integral multiflow and multicut problems, which form a pair of related packing and covering problems, respectively. I started studying such problems during my PhD thesis [4], and I have continued to work on some related open questions after I defended it in 2006. In both problems, we are given an edge-capacitated graph $G = (V, E)$ (directed or not) and a list of source-sink pairs (s_i, s'_i).

The basic versions of such problems can be modeled by two linear programs with integral variables (ILP). The first one models the *maximum integral multiflow problem* (MaxIMF), and is as follows:

$$\textbf{(ILP-IMF)} \left| \begin{array}{lll} \max & \sum_{j=1}^{|\mathcal{P}|} f_j & \\ \text{s. t.} & \sum_{j \text{ s. t. } e \in p_j} f_j \leq capa(e) & \forall e \in E \\ & f_j \in \mathbb{N} & \forall j \in \{1, \ldots, |\mathcal{P}|\} \end{array} \right. \quad (2.1)$$

where $\mathcal{P} = \bigcup_i \{\text{simple paths linking } s_i \text{ to } s'_i\}$, $capa(e)$ is the capacity of edge e, p_j is the jth path of $\mathcal{P}$, and f_j is the amount of flow routed on p_j.

The maximum integral multiflow problem hence consists in routing as much flow units (i.e., integral amounts of flow) as possible on the paths of $\mathcal{P}$ (i.e., linking one source s_i to its corresponding sink s'_i) while respecting the capacity constraints on the edges, and therefore it is a packing problem.

Note that the case where all capacities are equal to 1 is actually an important special case of MaxIMF. In this case, no path can carry more than 1 unit of flow, and at most one path from $\mathcal{P}$ carrying 1 unit of flow can go through each edge. This case hence consists in routing the maximum number of edge-disjoint paths from $\mathcal{P}$, and is known as MaxEDP.

Here is the second ILP. It models the *minimum multicut problem* (MinMC):

$$\textbf{(ILP-MC)} \quad \left| \begin{array}{lll} \min & \sum_{e \in E} capa(e) z_e & \\ \text{s. t.} & \sum_{e \in p_j} z_e \geq 1 & \forall j \in \{1, \ldots, |\mathcal{P}|\} \\ & z_e \in \{0,1\} & \forall e \in E \end{array} \right. \tag{2.2}$$

where z_e is equal to 1 if edge e is selected, and 0 otherwise. MinMC hence consists in selecting a set of edges of minimum total capacity, such that any path in $\mathcal{P}$ (i.e., linking one source s_i to its corresponding sink s'_i) contains at least one such edge, and therefore it is a covering problem.

Actually, the continuous relaxations of these ILPs are two dual linear programs (simply associate a nonnegative dual variable z_e to every constraint (2.1)). Moreover, the number of variables in (*ILP-IMF*) is exponential, as is the number of constraints in (*ILP-MC*). There also exist alternative ILPs for these two problems that have a polynomial number of variables and constraints (by using flow-based variables), but we will not need them. However, these other ILPs imply that the continuous relaxations of MaxIMF and MinMC can be solved in polynomial time: in the case of MaxIMF, this corresponds to the maximum multiflow problem, i.e., for each pair (s_i, s'_i), the amount of flow routed from s_i to s'_i on each edge can be non integral.

One of the main consequences of the duality between these formulations is that many researchers have worked on finding relevant special cases where MaxIMF and MinMC have the same optimal value (as in the continuous case, by strong duality in linear programming) or optimal values that are "close" to each other (if one is only interested in "good" solutions for both problems), as well as exhibiting sufficient conditions ensuring such properties.

In the next section, we shall review some of our results related to such questions, and in the subsequent sections we shall deal with the complexity of both MaxIMF and MinMC in different cases. Also note that some open problems related to this chapter are given in Appendix A.

2.2 Solving MaxIMF & MinMC by using duality

The first remark of this section is simply a direct consequence of the duality relationship between the continuous relaxations of the ILPs modeling MaxIMF and MinMC, described above. If, given an edge-capacitated graph G (along with a list of source-sink pairs), one knows both a feasible solution F to the instance of MaxIMF defined on G and a feasible solution C to the instance of MinMC defined on G that are such that $val(C) \leq \alpha \cdot val(F)$, where $val(C)$ and $val(F)$ denote the respective *values* of C and F, then

F and C are α-approximate solutions for the instances of MaxIMF and MinMC (respectively) defined on G. For instance, Garg *et al.* designed a polynomial-time algorithm that computes such a pair of solutions with $\alpha = 2$, when G is a tree [43]. Their algorithm is "guided" by the complementary slackness (or optimality) conditions in linear programming, in order to compute C from F: such algorithms are called *primal-dual algorithms*.

Thus, when α is small enough (e.g., a constant), designing approximation algorithms in such a way implies in particular that the associated optimal values of MaxIMF and MinMC are not too far away from each other.

The case where $\alpha = 1$ simply implies that the associated instances of MaxIMF and MinMC have the same optimal values: in such a case, the reasons behind this fact often lie in some properties of the underlying linear programming formulation, which we shall discuss.

When trying to solve a hard combinatorial problem in the context of a real-life application, one tends to use simple (and easy-to-implement) heuristics, with or without guaranteed quality performance. As far as MaxIMF is concerned, with applications ranging from routing through telecommunication networks to wiring in VLSI circuits, one of the best known and most used heuristics is the simple following one [54]:

SPF

While $\mathcal{P}$ is not empty do:

1. Route as much flow as possible on the shortest path p in $\mathcal{P}$,
2. Update the residual capacities, and delete any edge with residual capacity equal to 0, as well as any vertex that becomes isolated,
3. Remove from $\mathcal{P}$ the path p and any other path containing at least one deleted edge.

As indicated, we will refer to this heuristic as SPF (for *Shortest Path First*). It should be noticed that, although $\mathcal{P}$ may contain an exponential number of paths, SPF can be implemented to run in polynomial time, since in practice it amounts to computing a shortest path in a residual graph at most $O(|E|)$ times, without actually deleting paths from $\mathcal{P}$ (the deletion of an edge usually resulting in the deletion of an exponential number of paths from $\mathcal{P}$).

It is known for a long time that SPF has an approximation ratio of $O(\sqrt{|E|})$ in terms of $|E|$ [54], which means $O(\sqrt{|V|})$ in sparse graphs where $|E| = O(|V|)$ (e.g., planar graphs). However, it was highlighted in [Ben09a] that SPF can behave very badly even on some simple families of trees, by providing such families where the ratio of $O(\sqrt{|V|})$ is actually reached. Hence, even for what seems like "simple" instances, relying on a heuristic

based solely on this principle may result in providing very bad solutions, although such a heuristic does work well in practice whenever sufficiently short paths are available for routing.

To overcome this shortcoming, a heuristic that essentially routes flow iteratively along a set of spanning trees was proposed in [Ben09a], and its performance was studied theoretically under some specific conditions. Moreover, a variant of the following approximation algorithm, combining SPF with the previously mentioned heuristic, was also suggested:

AH4MAXIMF

- For λ ranging from 0 to $|V|$ do:
  1. While there remain paths in $\mathcal{P}$ of length at most λ, run SPF on G. Let G' be the graph obtained from G at the end of this step.
  2. Compute a spanning tree with maximum total capacity on each connected component of G', using essentially Kruskal's algorithm.
  3. Run the approximation algorithm of Garg *et al.* [43] on each tree S obtained at Step 2, to get a feasible integral multiflow F_S and a feasible multicut C_S on S such that $val(C_S) \leq 2 \cdot val(F_S)$. Update the residual capacities, and delete from G' any edge with residual capacity equal to 0, as well as any vertex that becomes isolated.
  4. If $val(F_S) > 0$ for some S, then go to Step 2.
- Output the best solution obtained, where each solution is the combination of the solution computed at Step 1 for a given λ and of the solution computed by going through Steps 2 to 4 for this same λ.

We will refer to this heuristic as $AH4MAXIMF$ (for *Another Heuristic For MAXimum Integral MultiFlow*), and to the heuristic consisting only of Steps 3 to 5 (i.e., without running SPF first) as $MST + GA$ (for *Maximum Spanning Tree + Garg et al.'s Algorithm*), or simply $ST + GA$ if the spanning tree obtained at Step 3 is not computed as a maximum spanning tree (in which case we shall give details about how it is computed).

Note that the solution computed by $AH4MAXIMF$ is at least as good as the one computed by SPF (thanks to the case $\lambda = |V|$) and as the one computed by $MST + GA$ (thanks to the case $\lambda = 0$). Moreover, apart from SPF and $AH4MAXIMF$, there are not many heuristics specifically known for MaxIMF. However, it should be mentioned that Chekuri, Khanna and Shepherd have written an impressive series of papers where they investigated approximation algorithms for MaxIMF when the tree-width of the graph is small, or when the minimum capacity is a small integer larger than or

equal to 2. Unfortunately, these algorithms are quite involved, quite hard to implement, and thus unlikely to be used in practice [13, 14, 15, 16].

The initial intuition behind $ST+GA$ was that, thanks to Garg *et al.*'s 2-approximation algorithm for MaxIMF in trees [43], we know an efficient and easy way of routing a good integral multiflow in a tree, whereas MaxIMF remains **APX**-hard in this case. Moreover, this algorithm uses only basic graph algorithms (essentially a *breadth-first search*, and the computation of a set of *least common ancestors*), so it is rather simple to implement. Therefore, using this algorithm as a subroutine inside a more general one seemed like a sound idea, provided that under relevant assumptions this would still yield good theoretical approximation ratios.

The seminal paper of Garg *et al.* studying MaxIMF and MinMC on trees [43] inspired many research papers, which generalized or improved these results in some way. For instance, one can notice that their approximation ratio of 2 is, in this case, $2(\gamma(G)+1)$, where $\gamma(G)$ is the *cyclomatic number* of G (with $\gamma(G) = 0$ if G is a tree, by definition). Actually, the first motivation for introducing $MST + GA$ was to prove the following set of results:

Theorem 2.1 ([Ben05, Ben09a])**.** *All the tractability and approximability results proved in the paper of Garg, Vazirani and Yannakakis* [43] *can be generalized to undirected graphs with bounded cyclomatic number.*

In other words, the approximation results in [Ben09a] were proved by making use of $MST + GA$. However, Garg *et al.* also proved other results, which we will not fully describe here, such as the polynomial-time solvability of MaxEDP, i.e., the special case where all capacities are 1, and so where we look for edge-disjoint paths: this theorem states that these other results hold in undirected graphs with bounded cyclomatic number as well, while this is not true for other natural generalizations. For instance, MaxEDP becomes **APX**-hard even in some specific families of cacti (i.e., graphs where any two cycles share no common edge). It should be noticed that a graph G with cyclomatic number $\gamma(G)$ has tree-width $O(\gamma(G))$ (so having bounded cyclomatic number is a special case of having bounded tree-width), and that the complete graph K_t has tree-width $t-1$ and cyclomatic number $\Theta(t^2)$. Hence, the family of graphs G with cyclomatic number $\gamma(G) = O(1)$ contains graphs with (constant but) arbitrarily large tree-width: a graph G with cyclomatic number $\gamma(G)$ can have tree-width $\Omega(\sqrt{\gamma(G)})$.

In contrast with Theorem 2.1, there exist cases where applying Garg *et al.*'s algorithm on a maximum spanning tree of the input graph does not always yield a good solution for the whole graph. This is why the variant $ST + GA$ was also considered: the choice of the spanning tree on which Garg *et al.*'s algorithm will be applied must then be made according to some rules, which can depend on the structure of the underlying graph. Let us now describe an application of this idea, which also appeared in [Ben09a].

There exist several ways of expressing the fact that a graph is "tree-like", and the more a graph resembles a tree, the more applying ideas that worked well for trees is likely to yield good results in such a graph as well. One such way (actually a rather classical one) is to consider graphs with bounded tree-width, and one of the most widely known classes of planar graphs with bounded tree-width is the class of *k-outerplanar graphs* for $k = O(1)$.

Roughly speaking, a planar graph is a k-outerplanar graph if removing at most k times its vertices lying on the outer face (and updating the outer face accordingly after each such removal) results in a graph with no vertex. In [Ben09a], I introduced a new subclass of k-outerplanar graphs called *k-edge-outerplanar graphs*. The definition of such graphs is easily obtained from the previous one by replacing "vertex" by "edge".

It is rather easy to see that any k-edge-outerplanar graph is also a k-outerplanar graph, but the converse is not true in general (for instance, the complete bipartite graph $K_{2,t}$ is 2-outerplanar but $\lceil \frac{t}{2} \rceil$-edge-outerplanar). However, it was proved in the same paper that, if the degree of any vertex is bounded inside each block of a k-outerplanar graph with $k = O(1)$, then this graph is also k'-edge-outerplanar for some $k' = O(1)$. A wheel graph, or any *Halin* graph (i.e., any planar graph obtained by connecting the leaves of a tree into a cycle), provides an example showing that such a condition is not necessary for a k-outerplanar graph with $k = O(1)$ to be k'-edge-outerplanar for some $k' = O(1)$. Also note that, if a graph is either a k-outerplanar graph or a k-edge-outerplanar graph, then it has tree-width $O(k)$.

There exist simple and classical families of graphs which are k-edge-outerplanar (and hence also k-outerplanar) for some $k = O(1)$. For instance, any rectilinear grid with $\Theta(k)$ rows and columns has tree-width $\Theta(k)$, and is both a $\Theta(k)$-outerplanar and a $\Theta(k)$-edge-outerplanar graph. Such graphs provide a way of showing, in particular, that the class of k-edge-outerplanar graphs with $k = O(1)$ contains graphs with (constant but) *arbitrarily* large tree-width, which means that, for any constant K, there is a $\Theta(K)$-edge-outerplanar graph whose tree-width is at least K.

Using $ST + GA$, where the spanning tree is computed level by level (and so is not a maximum spanning tree), as well as an additional step during which a feasible multicut for the whole graph is computed based on the one for the spanning tree, we obtain the following result:

Theorem 2.2. *[Ben09a] Given any k-edge-outerplanar graph with capacities 1, a feasible integral multiflow F and a feasible multicut C satisfying $val(C) \leq 4k \cdot val(F)$ can be computed in polynomial time.*

An easy consequence is the following result:

Corollary 2.1. *There exists a family of graphs of bounded (but arbitrarily large) tree-width, containing the rectilinear grids with at least one bounded dimension, in which the ratio between the optimal values of* MinMC *and* MaxIMF *is $O(1)$, provided that all capacities are $O(1)$.*

To the best of our knowledge, this result provides the only known class of graphs of bounded (but arbitrarily large) tree-width in which the optimal values of MinMC and MaxIMF are "close", i.e., for which the ratio between these two optimal values is $O(1)$, and moreover for which this ratio is linearly related to $tw(G)$, the tree-width of the input graph G, as this ratio is actually $O(tw(G))$. Indeed, the algorithm of Chekuri, Khanna and Shepherd in [15] shows that, in graphs of bounded tree-width, this ratio is $O(\log(|V|))$, not $O(1)$. Moreover, applying it to k-outerplanar graphs with $k = O(1)$ cannot help in improving this ratio to $O(1)$ either.

Note that Theorem 2.1 also provides such a class of graphs (with no restriction on the capacities), but a far less general one, and in addition one that does not contain rectilinear grids with only one bounded dimension (i.e., in which either the number of rows or the number of columns is bounded).

Later, Naves *et al.* did manage to prove that the ratio between the optimal values of MinMC and MaxIMF is actually $O(1)$ in *all* graphs of bounded tree-width, by using a much more complex approach [12]. However, they were not able to prove a linear dependence with respect to $tw(G)$, the tree-width of the input graph G; the ratio they obtained is $O(2^{tw(G)})$.

It should be noticed that the approach used to prove Theorem 2.2 only works for MaxEDP (i.e., MaxIMF with capacities 1), or for MaxIMF if all capacities are $O(1)$. However, Garg *et al.* established that MaxEDP essentially captures the hardness of the general case, by providing a family of undirected planar graphs with capacities 1 in which the ratio between the optimal values of MinMC and MaxIMF (i.e., MaxEDP) is $\Omega(\sqrt{|V|})$ [43]. Such a family of graphs is constructed from rectilinear grids, by replacing vertices by edges in order to obtain degree-3 vertices, and Theorem 2.2 shows that, if one dimension of the grid is bounded, then this ratio shrinks to $O(1)$ (note that Theorem 2.1 would not be sufficient to prove such a result). Later, the authors of [65] confirmed the intuition that MaxEDP is actually the "hardest" case by proving that, if the minimum capacity is 2, essentially excluding the case of MaxEDP, then the ratio between the optimal values of MinMC and MaxIMF is $O(1)$ in undirected planar graphs. (Recently, a polylogarithmic ratio was proved in general undirected graphs [23].)

However, applying $MST + GA$ on 1-edge-outerplanar graphs, which we shall simply call edge-outerplanar graphs (or cacti), together with an additional step during which a feasible multicut for the whole graph is computed based on the one for the maximum spanning tree, yields both a feasible integral multiflow F and a feasible multicut C satisfying $val(C) \leq 4 \cdot val(F)$ [Ben09a], while Erlebach obtained a 3-approximation for MaxEDP in 2-edge-connected edge-outerplanar graphs (also called *trees of rings*) [34].

We now go back to the case of rectilinear grids. A lot of decision problems related to the existence of edge-disjoint paths in such graphs and slightly more general ones have been studied before 1990 [39, 40, 53]: when the

sources and sinks can lie anywhere inside the grid, the problem of deciding whether edge-disjoint paths linking s_i to s'_i for each i simultaneously exist is **NP**-complete, even if some parity condition on the degrees (after adding an edge between each pair (s_i, s'_i)), called *Eulerian condition*, holds [59]. However, the problem is easier if all sources and sinks lie on the outer face. In particular, Frank gave necessary and sufficient conditions for the existence of such disjoint paths in this case [39], and later extended this characterization to planar graphs where all sources and sinks lie on the outer face, and any vertex not on the outer face has even degree [40].

In [BCR07], we studied MaxIMF and MinMC in particular rectilinear grids called *two-sided*, in the hope that the strong results concerning the existence of disjoint paths in grids could be used to solve related optimization problems efficiently. A two-sided grid is a rectilinear grid where all sources and sinks lie on the uppermost and lowermost rows of the grid, and at most one source/sink can lie on each vertex of these two rows. Furthermore, as Frank, we implicitly assume that, for each i, s_i and s'_i can be linked by at most one path: therefore, in this case, solving MaxEDP is equivalent to finding the maximum number of source-sink pairs (s_i, s'_i) that can simultaneously be linked by edge-disjoint paths in the grid.

A well-known necessary condition for the existence of edge-disjoint paths between s_i and s'_i for each i in a graph $G = (V, E)$ is the *cut condition*: for each $W \subset V$, the number of edges of E having one endpoint in W and one endpoint in $V \backslash W$ must be greater than or equal to the number of pairs (s_i, s'_i) such that one of $\{s_i, s'_i\}$ lies in W and the other one in $V \setminus W$. Okamura and Seymour showed in particular that, in a planar graph that satisfies the Eulerian condition and where all sources and sinks lie on the outer face, this condition is necessary and sufficient for the existence of disjoint paths [61].

Note that a two-sided grid does not necessarily satisfy the Eulerian condition; however, some of the necessary and sufficient conditions stated by Frank in this case are in fact equivalent to the cut condition on some particular subsets of V. One example of such subsets is the set of all vertices lying on the left of any given column c (called a *vertical cut*): requiring the cut condition on all such sets of vertices is then equivalent to requiring that the number of rows in the grid is greater than or equal to the *congestion* of the grid, i.e., the maximum number of pairs having one endpoint on the left of c (included) and one endpoint on the right of c, over all columns c. Such a condition will be called the *congestion condition*.

The first step of the approach we proposed in [BCR07] was to prove that determining the maximum number of source-sink pairs (s_i, s'_i) that can be selected in a two-sided grid so as to satisfy the congestion condition can be done efficiently, by solving a linear program having a totally unimodular constraint matrix. Alternatively, it was also observed that this can be done by solving a call control on a chain, a problem that was solved in linear time by Adamy, Ambuehl, Sai Anand and Erlebach [1].

The second step of this approach was to use Frank's results in order to prove that, in some cases, satisfying the congestion condition is in fact sufficient to simultaneously route all the selected source-sink pairs along edge-disjoint paths, while in *any* case "sacrificing" at most one of the selected source-sink pairs ensures that all the other selected ones can be simultaneously routed along edge-disjoint paths. The link with a linear program having a totally unimodular constraint matrix, established during the previous step, also allowed us to prove, by means of the Okamura-Seymour theorem, that the dual of this linear program actually models MinMC in this special case. A hole in the proof of one of the main lemmas was later pointed out by Guyslain Naves, but we were able to fix it, and to write down the details in a proper erratum [BCR]. We obtained in particular the following result:

Theorem 2.3 ([BCR07, BCR13])**.** *In any two-sided grid with capacities* 1 *that does not satisfy the congestion condition and has an odd number of rows, the optimal values of* MaxIMF *(i.e.,* MaxEDP*) and* MinMC *are the same, and can be computed in linear time.*

The results concerning MaxEDP can be generalized to MaxIMF when all capacities are equal to a common value κ (*uniform* capacities), provided that the amount of flow routed from s_i to s'_i is not allowed to be greater than κ, by using the pair of linear programs mentioned above, as well as a variant of the Okamura-Seymour theorem that applies to edge-capacitated planar graphs. More precisely, we gave a characterization of the cases where MaxIMF and MinMC have the same optimal value when $\kappa \geq 2$, and showed that, in all other cases, the difference between these optimal values is 1 (as this was already established when $\kappa = 1$). This yields:

Theorem 2.4 ([BCR07])**.** *In any two-sided grid with uniform capacities, the difference between the optimal values of* MaxIMF *and* MinMC *is at most* 1*, and these optimal values can be computed in polynomial time.*

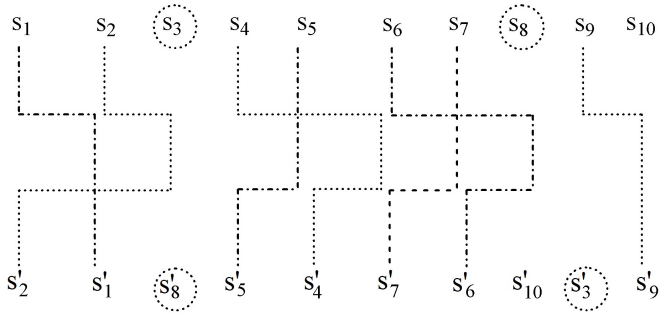

Figure 2.1: Edge-disjoint paths in a rectilinear grid.

Figure 2.1 shows a rectilinear grid with 10 columns, 2 rows, and 10 source-sink pairs. All the sources s_i lie on the uppermost row, and all the sinks s'_i lie on the lowermost row. For instance, s'_2, s'_8 and s'_{10} lie respectively on the 1st, the 3rd and the 8th column.

The congestion of the whole grid is 4 (the value is reached for $c = 4$ or $c = 6$), and the minimum number of source-sink pairs that we have to "sacrifice" in order to satisfy the congestion condition, i.e., in order to obtain a congestion that does not exceed 2, is 2. Moreover, the only way to achieve this is to "sacrifice" (s_3, s'_3) and (s_8, s'_8), so any of the two linear programs mentioned above has optimal value $10 - 2 = 8$.

However, in the rectilinear grid obtained in this way, Frank's conditions show that it is not possible to simultaneously route the 8 remaining source-sink pairs along edge-disjoint paths, and so at least one additional source-sink pair needs to be "sacrificed". Here, sacrificing (s_{10}, s'_{10}) is sufficient to ensure that the 7 remaining source-sink pairs can be simultaneously routed along edge-disjoint paths (these 7 paths are the 7 dotted lines in the figure): thus, MaxEDP has optimal value 7, although MinMC has optimal value 8.

We also proved in [BCR07] that MinMC is **NP**-hard in rectilinear grids that are not two-sided, even if all sources and sinks lie on the outer face and all capacities are 1. Finally, note that, while finding edge-disjoint paths in planar graphs where any vertex not on the outer face has even degree and where all sources and sinks lie on the outer face can be done efficiently thanks to the results that Frank proved in [40], which can be seen as a generalization of both the Okamura-Seymour theorem and Frank's previous results concerning rectilinear grids, MaxEDP is **APX**-hard in this case, as it is **APX**-hard in 1-outerplanar graphs (also called outerplanar graphs). Actually, as we already mentioned earlier in this section, it is even **APX**-hard in edge-outerplanar graphs. Therefore, it seems very unlikely that results such as ours can be extended, in some way, to this more general case.

Now, we turn our attention towards directed graphs. Indeed, apart from trees, Garg, Vazirani and Yannakakis also studied the use of primal-dual algorithms for variants of MaxIMF and MinMC in directed graphs. In the *multiterminal* variants of these problems, we are given, in an edge-capacitated graph, a set of terminal vertices $\mathcal{T} = \{t_1, t_2, \ldots, t_{|\mathcal{T}|}\}$ with $|\mathcal{T}| \geq 2$, and the source-sink pairs are (t_i, t_j) for all pairs (i, j) with $i \neq j$.

In [42], Garg *et al.* described a divide-and-conquer primal-dual algorithm that, in any multiterminal instance defined on a directed graph, computes in polynomial time a feasible integral multiflow F and a feasible multicut C such that $val(C) \leq 2\log_2(|\mathcal{T}|) \cdot val(F)$. Moreover, Costa *et al.* showed in [27] that, in any multiterminal instance defined on a directed acyclic graph, any optimal integral multiflow F and any optimal multicut C satisfy $val(F) = val(C)$, by reducing MaxIMF and MinMC in this case to classical maximum flow and minimum cut problems, respectively.

In [Ben06, Ben07], I introduced a new parameter $\mathcal{T}_L \subseteq \mathcal{T}$ for these problems: $\mathcal{T}_L$ is the set of *lonely* terminals, which is defined as the set of terminals that lie on at least one directed cycle containing no other terminal. Note that $|\mathcal{T}_L|$, the number of lonely terminals, is upper-bounded both by $|\mathcal{T}|$ and by the number of directed cycles in the digraph, and that it can be arbitrarily smaller than these two bounds. Thanks to this parameter, I was able to slightly improve the approximation result of Garg *et al.*, as well as to provide some kind of dichotomy results for computing optimal solutions to such problems. The first result can be stated as follows:

Theorem 2.5 ([Ben07]). *Given any multiterminal instance defined on a arc-capacitated digraph, a feasible integral multiflow F and a feasible multicut C such that $val(C) \leq 2\log_2(|\mathcal{T}_L| + 2) \cdot val(F)$ can be computed in polynomial time, by preprocessing the digraph and then applying Garg et al.'s algorithm.*

I also described in [Ben07] a family of instances showing that the ratios $2\log_2(|\mathcal{T}|)$ and $2\log_2(|\mathcal{T}_L| + 2)$ are essentially tight. However, this does not necessarily mean that there exists no way of determining a feasible integral multiflow F and a feasible multicut C satisfying $val(C) = O(val(F))$, and so this very question is still an open one. Here is the second set of results:

Theorem 2.6 ([Ben07]). *If we consider the restriction of* MaxIMF *and* MinMC *to multiterminal instances defined on directed graphs, we have:*

- *If $|\mathcal{T}_L| = 0$, or if $|\mathcal{T}_L| = 1$ and $|\mathcal{T}| = 2$, then any optimal integral multiflow F and any optimal multicut C satisfy $val(F) = val(C)$, and such optimal solutions can be computed in polynomial time,*
- *In all other cases,* MaxIMF *is* ***APX****-hard.*

Note, in particular, that if the input graph is a directed *ring* (i.e., a graph consisting only of one directed cycle), then $|\mathcal{T}_L| = 0$. However, when switching from multiterminal instances to non multiterminal ones, things are a bit different in this case. Directed rings have been studied in [BCLR09], together with undirected rings (i.e., graphs consisting only of one cycle), since any undirected ring can be transformed into a directed one by directing the edges clockwise, and doubling the number of source-sink pairs.

This was not stated explicitly in [BCLR09], but, in a directed ring, the optimal value of MinMC is at most twice the one of the continuous relaxation of (*ILP-MC*). Indeed, if there is an optimal multicut consisting of only one arc, then it is easy to see that there exists an optimal solution for MaxIMF where this arc will be saturated, and so the optimal values of MinMC and of the continuous relaxation of (*ILP-MC*) will be the same. Otherwise, any optimal multicut contains at least two arcs. By removing the arc having the smallest capacity in such a solution, we are left with a path. It is well-known that, in a path, MinMC and the continuous relaxation of (*ILP-MC*) have the same optimal value (which is, by assumption, greater than or

equal to the capacity of the arc that was removed to obtain a path), since the constraint matrix of this linear programming formulation is an interval matrix, and hence is totally unimodular. This concludes our claim.

What was stated and proved explicitly in [BCLR09], however, is the following result about the optimal value of MaxIMF:

Theorem 2.7 ([BCLR09]). *Let F be a maximum multiflow in a directed ring. Then, the optimal value of* MaxIMF *in this ring is $\lfloor val(F) \rfloor$.*

Moreover, the proof shows that an integral multiflow of value $\lfloor val(F) \rfloor$ can be computed efficiently by using a linear program with a totally unimodular constraint matrix. We also proved in [BCLR09] that, when all capacities are equal, an optimal integral multiflow can be computed in linear time. Since we have $val(F) \leq 2\lfloor val(F) \rfloor$, the worst case being when $val(F)$ lies in the interval [1,2[, the previous theorem immediately implies, together with the above discussion, that we have the following result:

Corollary 2.2. *In directed rings, the ratio between the optimal values of* MinMC *and* MaxIMF *is at most* 4.

Note, however, that this may be a rough estimate. The simple example of a directed ring with three vertices a, b, c, three arcs $(a, b), (b, c), (c, a)$ with capacity 1, and three source-sink pairs $(a, c), (b, a), (c, b)$, shows that this ratio can be at least as high as 2. Indeed, in this case, we have $val(F) = \frac{3}{2}$, $\lfloor val(F) \rfloor = 1$, and any optimal multicut has value 2 (it consists of two arcs).

2.3 Solving MinMC with few source-sink pairs

During my PhD thesis, I studied the complexity and inapproximability of MinMC in directed acyclic graphs, as well as in some generalizations. Indeed, the related problem of simultaneously routing the maximum number of source-sink pairs (s_i, s'_i) along arc-disjoint paths in a directed acyclic graph is known to be polynomial-time solvable when the number of source-sink pairs is $O(1)$, while in general digraphs both this problem and MinMC have been proved to be **NP**-hard even with only two source-sink pairs [38, 42].

Drawing inspiration from a proof detailed in [8], I proved, by proposing a reduction from 3SAT, that MinMC is **NP**-hard in directed acyclic graphs, even under very specific restrictions:

Theorem 2.8 ([Ben08]). MinMC *is **NP**-hard in directed acyclic graphs, even if all weights are* 1*, all vertex degrees are at most* 3*, and the underlying undirected graph is a bipartite cactus.*

It was also highlighted that, actually, MinMC in directed acyclic graphs is equivalent to MinMC in *layered* digraphs, by means of a simple transformation. A layered digraph is a directed graph $G = (V, E)$ whose vertex set

can be partitioned into $q \geq 2$ sets $V_1, \ldots, V_q$ such that, for each arc (u, v), we have $u \in V_i$ and $v \in V_{i+1}$ for some $i \in \{1, \ldots, q-1\}$. The V_i's will be called *layers*, and we let $p = \max_i |V_i|$. The proof of Theorem 2.8 showed that this **NP**-hardness result still holds if either $p = O(1)$ or $q = O(1)$.

However, if both p and the number of source-sink pairs are $O(1)$, then an efficient dynamic programming algorithm was proposed, yielding:

Theorem 2.9 ([Ben08]). *In layered digraphs,* MinMC *is* ***FPT*** *with respect to* $p = \max_i |V_i|$ *and the number of source-sink pairs.*

Actually, the dynamic programming algorithm used in this case even shows that MinMC becomes *linear-time solvable* when both $p = \max_i |V_i|$ and the number of source-sink pairs are $O(1)$.

These results raised two types of questions: on the one hand, what can be said about inapproximability results, and, on the other hand, what happens if we bound (only) the number of source-sink pairs? In [Ben08], the former question was answered partially. In order to show how, we first need to make some comments on directed acyclic graphs and their possible generalizations.

Although the definition of the "right" notion of tree-width is rather clear in undirected graphs, this is far from being the case in directed graphs. Actually, several notions have been proposed during the years, and each one shares some properties with its undirected counterpart, but none manages to gather *all* the nice properties that would be needed [5, 49, 51]. This is why it is much harder to compare results related to such widths (in particular, tractability results when "some" width is small) in directed graphs. However, to the best of our knowledge, *all* these widths share the fact that the digraphs with the smallest width are exactly the directed acyclic graphs. Hence, proving a hardness result in directed acyclic graphs would allow to prove the same hardness result in all digraphs with bounded width, whatever this width is (provided that it is among the ones that we mentioned).

Unfortunately, there was no result in [Ben08] concerning the inapproximability of MinMC in directed acyclic graphs, but I showed that, when one of these widths is bounded, an **APX**-hardness result holds:

Theorem 2.10 ([Ben08]). MinMC *is* ***APX****-hard in digraphs with bounded directed tree-width.*

One of the ten main open questions that I mentioned in my PhD thesis was: what is the complexity of MinMC in directed acyclic graphs with $O(1)$ source-sink pairs? I kept on working on this question, and was finally able to deal with both of the above questions at the same time in [Ben11]:

Theorem 2.11 ([Ben11]). MinMC *is* ***APX****-hard in directed acyclic graphs, even when there are only two source-sink pairs.*

The reduction used to prove this theorem is from the problem Max2SAT (an alternative proof, which works only for three source-sink pairs, uses a

reduction from the minimum vertex cover problem), and also shows that this hardness result remains true in layered digraphs where all capacities are 1 and the number of layers q is $O(1)$, hence complementing Theorem 2.9.

It should also be noticed that, up to a constant factor, this result is best possible when the number of source-sink pairs is $O(1)$ (to see this, simply take as a heuristic solution the union over all i of a minimum cut between s_i and s'_i), and that it matches the best inapproximability result known for MinMC in general digraphs having an arbitrary number of source-sink pairs *that is based only on the assumption* $\boldsymbol{P \neq NP}$. Indeed, there exist slightly stronger inapproximability results for the general case, but to the best of our knowledge they are all based on stronger complexity assumptions [11, 20].

Let us now turn to undirected graphs. When there are only two source-sink pairs (s_1, s'_1) and (s_2, s'_2), MinMC is known to be tractable. Indeed, it suffices to compute two minimum cuts, one separating $\{s_1, s_2\}$ from $\{s'_1, s'_2\}$, and the other separating $\{s_1, s'_2\}$ from $\{s'_1, s_2\}$: the best solution among these two cuts is an optimal solution for MinMC [67]. However, the problem has been shown to be **APX**-hard when there are three pairs (or more) [28]. So, it is natural to study its complexity in relevant special cases when there are $O(1)$ pairs, as otherwise the problem is **APX**-hard even in stars [43].

In [Ben08], I showed, by using a result of Dahlhaus *et al.* concerning the multiterminal variant of the problem, that the following result holds:

Theorem 2.12 ([Ben08])**.** *When the number of source-sink pairs is* $O(1)$*,* MinMC *is polynomial-time solvable in graphs of bounded tree-width.*

The proof relies on a reduction from MinMC to its multiterminal variant, which is polynomial and increases the tree-width of the graph by a constant when the number of source-sink pairs is $O(1)$. (We shall give more details about a variant of this reduction in the next paragraph, as we will need it for subsequent results.) I later discovered that this result has been proved independently in [46], using a slightly different proof.

I also studied the case of planar graphs. I proved in particular that the case where sources and sinks lie on the outer face can be reduced efficiently to a tractable case of the multiterminal variant. Actually, any optimal solution to an instance of MinMC can be seen as a partition of the input graph into connected components such that, for each i, no connected component contains both s_i and s'_i. By linking all the sources and sinks in each connected component to a dedicated terminal, using edges with sufficiently large capacities, one obtains an instance of the multiterminal variant. If the number of source-sink pairs is $O(1)$, then obviously the number of partitions to enumerate is $O(1)$. When the input graph is planar and all the sources and sinks lie on the outer face, I showed that the partitions that need to be enumerated satisfy some strong properties, which ensure that the multiterminal instances that are generated during the algorithm remain *planar*.

As the multiterminal variant of MinMC is known to be polynomial-time solvable in planar graphs if the number of terminals is $O(1)$ [28], this yields:

Theorem 2.13 ([Ben09c]). *When there are $O(1)$ source-sink pairs and all sources and sinks lie on the outer face,* MinMC *is tractable in planar graphs.*

The algorithm used to prove this theorem even shows that, in this case, the problem can be solved in linear time when there are two source-sink pairs. This result left as open the existence of an FPT algorithm for MinMC with respect to the number of source-sink pairs in this case (actually, a very recent proof of mine, which extends ideas already mentioned in [Ben09b] but is still unpublished, shows that such an algorithm does exist [Ben17a]), as well as the complexity of MinMC in planar graphs when the number of source-sink pairs is $O(1)$, a case that was listed as one among ten open questions in my PhD thesis. In order to settle this question, I tried to adapt some of the efficient algorithms that were known for the multiterminal variant of the problem in this case. Among the three algorithms I was aware of, two were based on strong properties linked to the multiterminal nature of the problem at hand: on the one hand, a matroid structure [47], and, on the other hand, the fact that, in any optimal solution to a multiterminal instance, any connected component of the associated partition contains only one terminal [28]. Therefore, it seemed unlikely that there was a way to adapt these two ones to the more general case of MinMC.

By studying the third algorithm, due to Yeh [68], I finally discovered that its proof was wrong. Roughly speaking, one of the main properties used in this proof stated that, if we enumerate enough vertices, replacing some part of an optimal solution by a minimum cut separating a terminal and the enumerated vertices from the other terminals yields another solution. In my PhD thesis, I provided a counter-example to this statement (see Figure 2.2).

In this multiterminal instance, the five terminals, t_1 to t_5, are the big black vertices at the center and in the four corners of the grid, and the only optimal solution is given by the 32 edges crossed by the dashed lines, which all have capacity 1. Any edge incident to t_1 has capacity 2, while any other bold edge has capacity 1. Any edge not mentioned yet has a sufficiently large capacity. Yeh's algorithm identifies the five big black vertices incident to the bold edges (including t_1), and the proof claims that replacing the 28 edges of the current solution not lying on the outer face by a minimum cut separating these five black vertices from the four other terminals (t_2 to t_5) yields a new feasible solution. However, the only such minimum cut is obtained by removing the bold edges (for a total capacity of 24), and these edges clearly do not yield a feasible solution to the considered multiterminal instance, as t_2 is not separated from t_j, for each $j \in \{3, 4, 5\}$, for instance.

This counter-example was also mentioned in [Ben09b], and later it was proved in [18] that the algorithm itself (and not only its proof) is not correct.

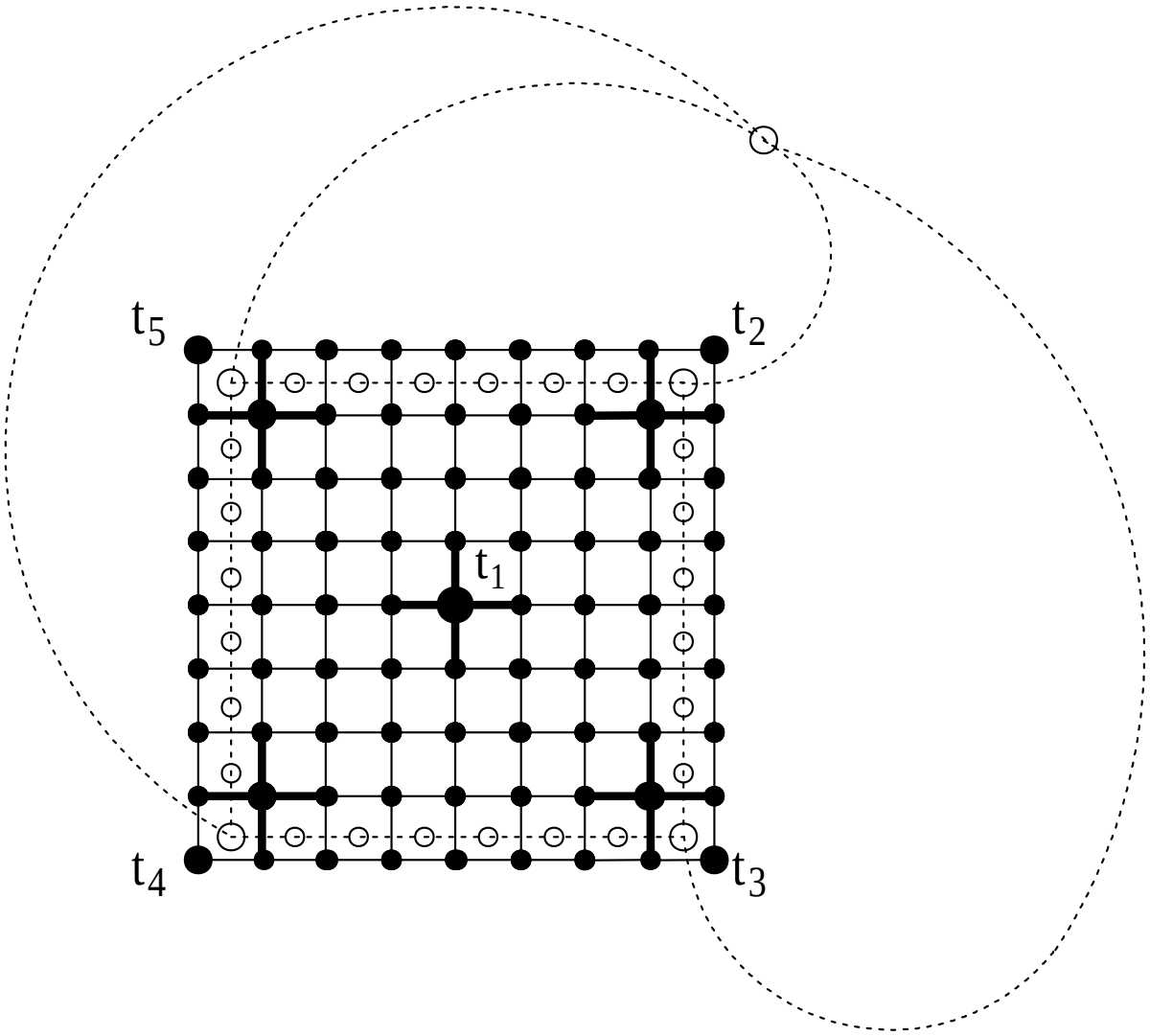

Figure 2.2: A counter-example to the proof of Yeh's algorithm.

In [Ben12], I proposed another efficient algorithm for solving MinMC in this case, based on the geometric notion of *homotopic paths*. Unfortunately, a hole in the proof of one of the lemmas was found by Éric Colin de Verdière, who later provided his own algorithm to solve MinMC in polynomial time when the graph is planar and the number of source-sink pairs is $O(1)$ [24].

2.4 Solving variants of MaxIMF and MinMC

In this section, we describe some algorithmic results concerning problems that can be seen as variants or generalizations of MaxIMF or MinMC.

In the first variant we shall consider, we are given a list of source-sink pairs in an edge-capacitated rectilinear grid, all capacities being equal to a common integer $c \geq 1$. More precisely, we are interested in the case of a so-called *dense channel*: all sources (resp. sinks) lie on the uppermost (resp. lowermost) row, and the number of source-sink pairs equals the number of columns. Therefore, all vertices on the uppermost (resp. lowermost) row are sources (resp. sinks), and any other vertex is neither a source nor a sink.

Such a setting arises, for instance, in the context of VLSI layout: nodes need to be connected together by wires on a rectilinear board, and there is a limit on the number of wires that can be installed in parallel. Moreover, because installing wires is costly, one wants to install wires so that their total length is as small as possible. Formally, we want to link each source to its corresponding sink with a path, while respecting the capacity constraints on the edges, and while minimizing the total length of the paths we use.

When $c = 1$, this problem amounts to finding edge-disjoint paths linking each source to its corresponding sink, while minimizing the total length of these paths. Frank's results on edge-disjoint paths in rectilinear grids, mentioned in Section 2.2, imply that such edge-disjoint paths cannot exist in a dense channel, except in the very special case where every source-sink pair can be routed along a vertical path. In all other cases, Formann, Wagner and Wagner showed in [37] that, if one adds an additional column on one side (left or right) of the grid, then a routing of minimum total length can be computed in polynomial time, and cannot use shortest paths.

For all other values of c, we proved in [BCPZ07] that this problem can be solved in polynomial time, and that, in any optimal solution, any source-sink pair is routed along a shortest path. Actually, we proved a slightly more general result, extending partial results previously obtained by three of the authors of [BCPZ07]: this remains true even if all "horizontal" edges have the same capacity, and all "vertical" edges have the same capacity (not necessarily equal to the one of the horizontal edges), provided that at least one of these two capacities is not equal to 1 (otherwise, we are in the case studied by Formann *et al.*). Summing up, we proved the following result:

Theorem 2.14 ([BCPZ07]). *In a dense channel where all horizontal edges have the same capacity, all vertical edges have the same capacity, and not all capacities are equal to 1, one can compute in polynomial time, if it exists, a set of paths linking each source to its corresponding sink, while satisfying all capacity constraints and minimizing the total length of the paths in this set. Moreover, any path in this set of paths is a shortest path.*

Now, we shall focus our attention on two generalizations of MinMC.

The first one consists, in its simplest version, in adding a cardinality constraint (i.e., bounding the number of edges used in a solution) to MinMC. That is, we add a packing constraint to a covering problem. The results presented here were obtained during the Master's internship that N. Derhy did under the supervision of M.-C. Costa, F. Roupin and myself [29].

When there is only one source-sink pair, MinMC is simply the minimum cut problem. We proved in [BCDR09] that, even in this special case, adding a cardinality constraint makes the problem much harder:

Theorem 2.15 ([BCDR09]). *Computing the minimum capacity of a cut that separates a given source from a given sink and uses no more than a given number of edges is a strongly **NP**-hard problem.*

When there is more than one source-sink pair, it is known that MinMC is polynomial-time solvable in directed trees [27], since in particular the constraint matrix of the ILP formulation (*ILP-MC*) is totally unimodular. Thanks to a small example, we showed in [BCDR09] that adding a cardinality constraint can destroy this property, even in directed stars (i.e., directed graphs whose underlying undirected graph is a star). Actually, we proved that the problem becomes much harder even in this very special case:

Theorem 2.16 ([BCDR09]). *The problem obtained by adding a cardinality constraint to* MinMC *is strongly **NP**-hard, even in directed stars.*

However, in another well-known special case (i.e., paths), we proved that adding a cardinality constraint does *not* destroy the total unimodularity of the constraint matrix of (*ILP-MC*). This implies:

Theorem 2.17 ([BCDR09]). *The problem obtained by adding a cardinality constraint to* MinMC *is polynomial-time solvable in paths.*

Actually, we also provided a (combinatorial) polynomial-time dynamic programming algorithm for MinMC with cardinality constraint in this case.

We also noticed in [BCDR09] that the problem obtained by adding a cardinality constraint to MinMC can be seen as a special case of a multicriteria multicut problem. Given a set of bounds $B_1, \ldots, B_\nu$ and a list of source-sink pairs in a graph where each edge e is weighted by a weight vector $w(e)$ of size ν, the multicriteria multicut problem consists in deciding whether there

exists a multicut whose total edge weight with respect to the ith components of the weight vectors w is smaller than or equal to B_i, for each $i \in \{1, \ldots, \nu\}$. When $\nu = 2$ and the second component of each weight vector is 1, the problem amounts to deciding whether there is a multicut using at most B_2 edges, and whose total capacity (given by the first components of the weight vectors w) is at most B_1. For this more general problem, we showed:

Theorem 2.18 ([BCDR09]). *When weight vectors are arbitrary, the multicriteria multicut problem is **NP**-complete in paths, in the weak sense if $\nu = O(1)$ (even if $\nu = 2$), and in the strong sense otherwise.*

The second generalization of MinMC that we shall consider will be called *partial* MinMC. Consider a MinMC instance, together with an integer p: the problem is then to compute the minimum capacity of a set of edges whose removal leaves no path from s_i to s'_i, for at least p source-sink pairs (s_i, s'_i). If p equals the number of source-sink pairs, then this is exactly MinMC. Moreover, partial MinMC can be seen as a partial covering problem.

A natural question is then: can we exhibit cases where MinMC and partial MinMC do not share the same complexity status? The answer to this question was unknown until D. Chalu and I proved the following result, during the Master's internship he did under my supervision:

Theorem 2.19 ([9]). Partial MinMC *is strongly **NP**-hard in directed stars.*

Since the full proof of this result is available only in D. Chalu's Master thesis, we provide the main steps here.

First, we show that partial MinMC in directed stars is polynomially equivalent to another partial covering problem, called the *Partial Weighted Vertex Cover* problem (or PWVC), in bipartite graphs. In this problem, we are given a node-weighted graph, and the goal is to compute the weight of a minimum-weight set of vertices that covers at least a given number of edges. It is clearly **NP**-hard in general graphs (as optimally covering *all* edges is), and we will show that it remains **NP**-hard in bipartite graphs.

Take any instance of partial MinMC in a directed star. We can assume without loss of generality that all sources have out-degree 1 and in-degree 0, and that all sinks have out-degree 0 and in-degree 1. We associate an undirected bipartite graph to this directed star by taking as vertices the sources and sinks (the weight of each vertex being the weight of the arc incident to the corresponding source or sink), and by adding an edge between two vertices if the corresponding source and sink in the directed star belong to the same source-sink pair. Then, in the partial MinMC instance defined in a directed star, removing an arc is clearly equivalent, in the PWVC instance defined in a bipartite graph, to selecting the vertex (source or sink) this arc is incident to. Conversely, any PWVC instance in a bipartite graph can be polynomially reduced to a partial MinMC instance in a directed star, using the same correspondence (edges become source-sink pairs).

It remains to show that PWVC is strongly **NP**-hard in bipartite graphs. We reduce from the following problem, which was shown to be strongly **NP**-complete in [BCDR09]: given a connected bipartite graph G with bipartition (L, R) and an integer $l \geq 1$, decide whether there exists a vertex cover of G having l vertices in L and $r = \nu(G) - l$ vertices in R, where $\nu(G)$ is the minimum size of a vertex cover in G. Take any instance of this problem in a graph G with m edges, and link $m + 1 \geq \nu(G)$ pendant vertices to each vertex in L (this yields a new bipartite graph G'). Moreover, in G', set the weight of any vertex to 1 if it belongs to R, and to $m + 1$ otherwise. Then, it is easy to see that in G' there exists a vertex cover of weight at most $l(m + 1) + (\nu(G) - l) = lm + \nu(G)$ that covers at least $m + l(m + 1)$ edges if and only if in G there exists a vertex cover having l vertices in L and $r = \nu(G) - l$ vertices in R. The "if" part is trivial, and the "only if" part comes from the fact that taking at least l vertices in L is the only way to cover at least $l(m + 1)$ edges, taking more than l vertices in L would cost at least $(l + 1)(m + 1) > lm + \nu(G)$, and covering the m edges of G using l vertices in L (which already cover $l(m+1)$ edges incident to pendant vertices) and $r = \nu(G)-l$ vertices in R is the only way to cover $m+l(m+1)$ edges in G'. This ends the proof of Theorem 2.19.

Although we do not provide all the details here, it was also shown in [9] that partial MinMC can be solved in polynomial time in rooted trees, using a dynamic programming algorithm. Therefore, if **P**≠**NP**, partial MinMC does not have the same complexity status in directed stars and in rooted trees (while MinMC is polynomial-time solvable in both cases, as it is polynomial-time solvable in directed trees).

During the PhD thesis of P. Le Bodic, we further studied the complexity of partial MinMC, and of some related problems. It was shown, in particular, that the partial version of another cut problem does not behave like its non partial counterpart. Assume we are given an integer p and a set of k terminal sets of vertices $T_1, \ldots, T_k$ (or *clusters*) in a graph. A cluster T_i is *isolated* from a cluster T_j if, for each pair of vertices $u \in T_i$ and $v \in T_j$, there is no path from u to v. The *partial* MinVMCC (for *partial* minimum vertex multi-cluster cut) problem then consists in computing the minimum number of vertices that need to be removed in order to isolate at least p of the clusters from all the others. On the one hand, it was proved in [57, BB12] that:

Theorem 2.20 ([57, BB12])**.** Partial MinVMCC *is* ***NP****-hard in the strong sense in undirected trees.*

On the other hand, note that the non partial version of MinVMCC (i.e., the case where $p = k$) can be solved in polynomial time in undirected graphs of bounded tree-width (and hence in undirected trees), by using a dynamic programming algorithm similar to the one mentioned in [28].

Finally, let us mention that a general result regarding (partial) MinMC was proved in [57, BB12]. Given a list of source-sink pairs in a graph G, let H denote the graph whose vertex set is composed of the sources and sinks in G, and where there is an edge between two vertices if and only if there exists a source-sink pair they both belong to. Moreover, let $G + H$ denote the graph obtained from G by adding the edges in H (except when such an edge already exists in G), and similarly let $G + \bar{H}$ denote the graph obtained from G by adding the edges in $\bar{H}$, the complement of H (except when such an edge already exists in G). Then, the following holds:

Theorem 2.21 ([57, BB12])**.** Partial MinMC *is polynomial-time solvable when* $\min(tw(G+H), tw(G+\bar{H})) = O(1)$*,* $tw(\cdot)$ *being the related tree-width.*

Note that this theorem unifies and generalizes to partial MinMC two results previously known for MinMC. First, it was noticed in [28] that the multiterminal variant of MinMC can be solved by a polynomial-time dynamic programming algorithm in graphs of bounded tree-width (in this case we have $tw(G + \bar{H}) = O(1)$, since H is a clique). Second, it was proved in [44] that MinMC is polynomial-time solvable when $tw(G + H) = O(1)$, hence generalizing the result of Theorem 2.12. Also note that a simple modification of the **APX**-hardness proof described in [43] for MinMC shows that only requiring that $tw(G) + tw(H) = O(1)$ is not sufficient to ensure that (partial) MinMC becomes polynomial-time solvable.

Chapter 3

Vertex and edge colorings with cardinality constraints

3.1 Introduction

In this chapter, we review results about vertex and edge coloring problems having some sort of additional cardinality constraints.

Coloring problems can be seen as covering problems, as basically they consist in covering the vertices (or edges) of a graph using only some specified type of subgraphs (each vertex, or each edge, must be covered in this way). The type of subgraphs that will be used depends on the coloring problem that is considered: for instance, recall that the classical vertex coloring problem (also called proper vertex coloring) consists in covering the vertices of a graph using the minimum number (or a given number) of stable sets (each stable set in such a solution being then referred to as a *color*).

Adding cardinality constraints to such a problem thus amounts to adding packing constraints to a covering problem. In the next three sections, we will consider three different special cases of this general setting. In Section 3.2, we will consider non proper edge coloring problems in complete bipartite graphs, in which we define a bound on the number of edges of each color incident to each vertex, and also require that the subgraph induced by the edges of each color has a special structure. In Section 3.3, we will consider vertex (and edge) coloring problems in which we are given a set of (not necessarily disjoint) chains in a graph, together with bounds (i.e., cardinality constraints) on the number of vertices (or edges) of each color in each chain, and we look for a coloring (proper or not) satisfying these bounds. Finally, in Section 3.4, we will consider vertex (and edge) colorings when a partition of the vertex set is given, and we look for a proper coloring satisfying given bounds on the number of vertices of each color in each set of the partition.

We will also show that the problems considered in Sections 3.2 and 3.3 generalize the following problem, which was in fact what motivated their

study in the first place. We are given a list of $p_1 + p_2$ integers $h_1, \ldots, h_{p_1}$, $v_1, \ldots, v_{p_2}$, and we consider the problem of deciding whether there exists a $p_1 \times p_2$ matrix containing 0's and 1's such that, for each $i \in \{1, \ldots, p_1\}$, the number of 1's on row i is exactly h_i, and, for each $j \in \{1, \ldots, p_2\}$, the number of 1's on column j is exactly v_j. This problem, known as one of the basic problems in *discrete tomography*, was studied by Ryser, who gave necessary and sufficient conditions for such a matrix to exist [64]. It should be noticed that these conditions can be checked in polynomial time. The complexity status of natural generalizations of this problem was open several years ago, so we became interested in them, and started working on characterizing or at least identifying tractable special cases of these generalizations.

The papers to which this chapter is related contain numerous results, and we will not describe them all. We will only highlight and comment those that we consider as the most interesting ones. Any reader interested in having knowledge of other results not presented here can refer to the respective journals where these papers have been published.

3.2 Degree-constrained edge colorings of complete (bipartite) graphs

In this section, we consider an edge coloring problem. We are given a complete bipartite graph with bipartition (L, R), three colors, and, for each vertex, the number of edges of each color that must be incident to it. The problem that we consider, denoted by DC-EdgeCol, then consists in deciding whether there exists an (non necessarily proper) edge coloring with three colors that satisfies all the degree requirements.

We can assume that these requirements are given as upper bounds, as the sum of the three requirements for each vertex v is exactly its degree in the whole bipartite graph, i.e., $|L|$ if $v \in R$ and $|R|$ if $v \in L$. Therefore, these requirements can actually be expressed as packing constraints.

Also note that DC-EdgeCol can actually be viewed as the problem of deciding whether there exists a (non necessarily complete) bipartite graph that satisfies the degree requirements for the colors 1 and 2 (the edges of color 3 being only useful to obtain a complete bipartite graph).

For similar reasons, it is not hard to see that, when there is no edge of color 3 (i.e., there are only two colors), DC-EdgeCol is equivalent to deciding whether there exists a (non necessarily complete) bipartite graph where the degree of each vertex is the number of edges of color 1 that must be incident to it. This problem, in turn, is equivalent to deciding whether there exists a $|L| \times |R|$ matrix containing 0's (when there is no edge) and 1's (when there is an edge) such that, for each $i \in \{1, \ldots, |L|\}$, the number of 1's on row i is exactly the degree of the ith vertex in L, and, for each $j \in \{1, \ldots, |R|\}$, the number of 1's on column j is exactly the degree of the

jth vertex in R. That is, it is equivalent to the basic problem solved by Ryser in [64]. However, when there are three colors (and not two) in a complete bipartite graph, the complexity status of the problem remained open for a long time (more details on this case are given below).

When we became interested in DC-EdgeCol due to its equivalence with this open problem in discrete tomography, we wanted to identify additional (graph) requirements that could make the problem easier, or harder. For instance, imposing a special structure or condition on the subgraph induced by the edges of color 1, by the ones of color 2, or by the ones of color 1 or 2, may precisely lead to either easier or harder problems.

In the remaining of this section, instead of considering three colors in a complete bipartite graph, we will consider two colors in the bipartite graph (to be determined) obtained by removing the edges of color 3. Moreover, we shall also consider the problem DC-EdgeCol in complete graphs (and not complete bipartite anymore) with three colors: let us refer to this variant as DC-Complete-EdgeCol. Again, instead of reasoning directly on a complete graph with three colors, we will consider two colors in the graph (to be determined) obtained by removing the edges of color 3.

We first established a link between the hardness of both problems:

Theorem 3.1 ([BCP+09])**.** DC-Complete-EdgeCol *is at least as hard as* DC-EdgeCol*, as the latter can be reduced to the former.*

As DC-EdgeCol is known to be **NP**-complete since 2012 [33] (before this paper, only the variant with four or more colors was known to be intractable [19]), so is DC-Complete-EdgeCol.

We also proposed a sufficient condition for an instance of DC-EdgeCol or DC-Complete-EdgeCol to have a solution: more precisely, if the number of edges of color 1 or 2 is not smaller than some lower bound that we computed, then there always exists a solution (complete bipartite graph with three colors). Then, we studied DC-Complete-EdgeCol and DC-EdgeCol under the assumption that the subgraph induced by the edges of color 1 or 2 has a special structure, and provided necessary and sufficient conditions for a solution to exist. Note that all these conditions can be checked in polynomial time. More precisely, we have proved:

Theorem 3.2 ([BCP+09])**.** *When we require that the graph induced by the edges of color* 1 *or* 2 *must be a tree, there exist necessary and sufficient conditions, which can all be checked in polynomial time, for an instance of* DC-EdgeCol *or* DC-Complete-EdgeCol *to have a solution.*

This theorem is essentially proved with the help of two key lemmas, called the *recoloring* and *recycling* lemmas. The recoloring lemma states that we can merge two connected components of the subgraph induced by the edges of color 1 or 2 into a single connected component, provided that each of these

connected components contains one edge of a given color (1 or 2), and at least one of these two edges lies on a cycle. The recycling lemma basically states that, under some weak assumptions, an edge not lying on a cycle can be put on a cycle, without modifying the number of connected components.

In the proof of Theorem 3.2, we apply the recoloring lemma while we can (decreasing by one the number of connected components each time we use it). Then, we show that the recycling lemma can be used, and use it. After that, the recoloring lemma can be used again, which concludes the proof.

The next result deals with disjoint chains covering the whole graph:

Theorem 3.3 ([BCP+09]). *When we require that the graph induced by the edges of color* 1 *or* 2 *must be a set of vertex-disjoint chains spanning all vertices, there are necessary and sufficient conditions, which can all be checked in polynomial time, for an instance of* DC-EdgeCol *to have a solution.*

For DC-Complete-EdgeCol, we can even require that the edges of color 1 or 2 induce a *hamiltonian* cycle (i.e., simple and spanning all vertices):

Theorem 3.4 ([BCP+09]). *When we require that the edges of color* 1 *or* 2 *must induce a hamiltonian cycle, the following conditions are necessary and sufficient for an instance of* DC-Complete-EdgeCol *to have a solution:*

- *the number of edges of color* 1 *or* 2 *incident to each vertex is two,*
- *there is a vertex incident to an edge of color* 1 *and to one of color* 2*.*

These conditions can be checked in polynomial time and are clearly necessary (the second one being necessary for a connected solution to exist). The sufficiency comes from the recoloring lemma: the only case where this lemma cannot be applied is when the edges of color 1 or 2 induce two monochromatic vertex-disjoint cycles, but this is impossible from the second condition.

Now we turn to the case where both the edges of color 1 and the edges of color 2 must have a special structure. We first study hamiltonian chains:

Theorem 3.5 ([BCP+09]). *When we require that the edges of each color (*1 *and* 2*) must induce a hamiltonian chain, there exist necessary and sufficient conditions, which can all be checked in polynomial time, for an instance of* DC-EdgeCol *or* DC-Complete-EdgeCol *to have a solution.*

Then, we study the case of cycles:

Theorem 3.6 ([BCP+09]). *When we require that the edges of each color (*1 *and* 2*) must induce a cycle, the following conditions are necessary and sufficient for an instance of* DC-EdgeCol *to have a solution:*

- *the edges of each color (*1 *and* 2*) induce a set of disjoint cycles,*
- *we have neither the forbidden configuration given in Figure* 3.1*, nor the one obtained by exchanging edges of color* 1 *and edges of color* 2*.*

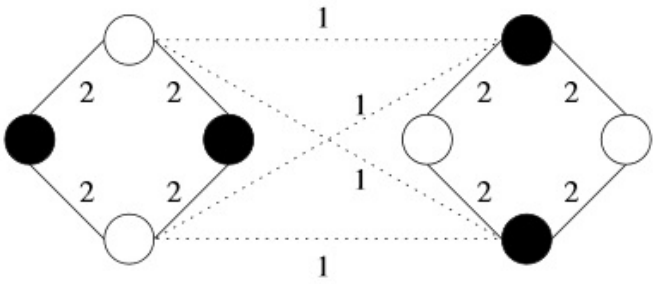

Figure 3.1: A forbidden configuration for Theorem 3.6. The black vertices are in L, the white ones are in R, and the number on each edge is its color.

In [25], the first condition of Theorem 3.6 is shown to be equivalent to a set of necessary and sufficient conditions, which can be checked in polynomial time, corresponding to a set of "pathological cases" that must be forbidden.

Finally, DC-Complete-EdgeCol behaves in a similar way in this case, although the associated necessary and sufficient conditions are a bit simpler:

Theorem 3.7 ([BCP+09]). *When we require that the edges of each color (1 and 2) must induce a cycle, there exist necessary and sufficient conditions, which can all be checked in polynomial time, for an instance of* DC-Complete-EdgeCol *to have a solution.*

3.3 Colorings with cardinality constraints on a given set of chains

In this section, we consider the following problem Col-CC: we are given a list of colors, a graph, and a list of (not necessarily disjoint) chains in this graph, together with the number of vertices of each color that each chain must contain, and we are thus looking for a (not necessarily proper) vertex coloring such that each chain given in the input contains the right number of vertices of each color. Note that these numbers of vertices can as well be given as upper bounds in the input since, for each chain, the sum of these numbers is exactly the total number of vertices on the chain (so, each of these numbers must be reached anyway in any feasible coloring). This way, we can define these constraints as packing (or cardinality) constraints.

Assume for instance that we are given a rectilinear grid with p_1 rows and p_2 columns, and two colors. For each $i \in \{1, \ldots, p_1\}$, the ith row (one of the chains given in the input) must contain h_i vertices of color 1 and $p_2 - h_i$ vertices of color 2. Similarly, for each $j \in \{1, \ldots, p_2\}$, the jth column (one of the chains given in the input) must contain v_j vertices of color 1 and $p_1 - v_j$ vertices of color 2. Then, finding a (non proper) coloring

satisfying the cardinality constraints on the $p_1 + p_2$ chains in this special case is equivalent to finding a $p_1 \times p_2$ matrix containing 0's (corresponding to vertices of color 2) and 1's (corresponding to vertices of color 1) such that, for each $i \in \{1, \ldots, p_1\}$, the number of 1's on row i is exactly h_i, and, for each $j \in \{1, \ldots, p_2\}$, the number of 1's on column j is exactly v_j.

In the above special case, which is Ryser's problem [64], the input graph is a rectilinear grid, and the number of colors is two. When there are four (or more) colors, the problem is known to be **NP**-complete (even in rectilinear grids where the set of chains is as above), as the corresponding problem in discrete tomography (reconstructing a matrix with four colors, knowing the number of times each color appears in each row and each column) is [19]. However, the complexity of the problem remained open in other special cases (for instance, in other families of bipartite graphs), or when there are three colors. We studied the former question, and the latter one was settled in 2012 (the problem being actually **NP**-complete) by Dürr *et al.* [33].

The first noticeable result that we obtained in [BCdW$^+$08] concerning Col-CC is the following negative one:

Theorem 3.8 ([BCdW$^+$08])**.** Col-CC *is **NP**-complete in trees with maximum degree three, even when there are only two colors.*

The proof is made by reduction from the **NP**-complete problem 1-in-3SAT with no negated literal (also called Monotone1-in-3SAT), a variant of 3SAT where each clause must contain *exactly* one true literal, and which will be described formally in the next section. Informally, in any instance of this problem, we have a set of variables and a set of clauses of size 3 defined on these variables (with only positive literals), and the idea of the reduction is to associate a leaf to each variable and a gadget to each clause. Then, we define chains in each clause gadget in order to ensure that each such gadget has only 3 possible colorings: for each of these 3 colorings, exactly one of the 3 associated variable leaves has color 1. Having color 1 can thus be associated with being *true* in a truth assignment, and we prove the equivalence between the solutions to the Col-CC and 1-in-3SAT instances in this way.

However, restricting the way the chains given in the input can interact with each other makes the problem easier when there are two colors:

Theorem 3.9 ([BCdW$^+$08])**.** *When there are two colors,* Col-CC *can be solved in polynomial time when any chain given in the input has at most two vertices in common with other chains given in the input.*

Imposing strong restrictions on the lengths of the chains given in the input also makes the problem easier:

Theorem 3.10 ([BCdW$^+$08])**.** Col-CC *can be solved in polynomial time when any chain given in the input contains at most two vertices.*

It was also proved in [BCdW$^+$08] that the problem becomes **NP**-complete if chains containing 3 (or more) vertices are allowed, even with only 3 colors.

When the graph is a tree and the chains given in the input are *almost* disjoint, a rather intricate algorithm, called the *Forced and Forbidden Colors* procedure, and based on several non trivial properties, can be used to show:

Theorem 3.11 ([BCdW$^+$08]). Col-CC *can be solved in polynomial time in trees when any two chains given in the input share at most one vertex.*

When we impose in addition that any solution must be a proper coloring, the problem (which is trivial with two colors) becomes a bit harder to deal with. However, one can show the following (positive) result:

Theorem 3.12 ([BCdW$^+$08]). *When any solution is required to be a proper coloring,* Col-CC *can be solved in polynomial time in trees when any two chains given in the input share at most one common vertex.*

A reduction inspired by the one used to prove Theorem 3.8 yields:

Theorem 3.13 ([BCdW$^+$08]). *When any solution is required to be a proper coloring,* Col-CC *is* ***NP****-complete in trees with maximum degree three, even when there are only three colors.*

Finally, if we consider the case of (non necessarily proper) edge colorings, then we impose the number of *edges* of each color that each chain given in the input must contain. In this case, the complexity of the problem (denoted by EdgeCol-CC) is open in trees, but we were able to prove that the problem is hard in graphs that are hardly more general than trees:

Theorem 3.14 ([BCdW$^+$08]). EdgeCol-CC *is* ***NP****-complete in cacti with maximum degree three and no cycle of length four or more, even when there are only two colors.*

3.4 Colorings with "local" cardinality constraints on a fixed partition

In this section, we consider the following problem Col-LocalBounds: we are given a list of k colors, a graph $G = (V, E)$, a partition $(V_1, V_2, \ldots, V_p)$ of the vertex set V, and a list of $p{\cdot}k$ bounds $n_{11}, \ldots, n_{1k}, \ldots, n_{p1}, \ldots, n_{pk}$ such that $\sum_{j=1}^{k} n_{ij} = |V_i|$ for each i, and we want to decide whether there exists a proper coloring such that, for each $i \in \{1, \ldots, p\}$ and for each $j \in \{1, \ldots, k\}$, the number of vertices of color j in V_i is n_{ij}. Again, note that, because of the condition $\sum_{j=1}^{k} n_{ij} = |V_i|$ for each i, the numbers n_{ij} can as well be given as upper bounds in the input. This way, we can, as in the previous section, define these constraints as packing (or cardinality) constraints.

When $p = 1$, the problem is known as the *Capacitated Coloring* problem (see [7]): Gravier *et al.* showed that this problem is polynomial-time solvable in graphs of bounded tree-width when the number of colors k is $O(1)$ [45]. Other very well-known special cases of *Capacitated Coloring* also include the *Equitable Coloring* problem (where the sizes of any pair of color classes must differ by at most 1), and the *Bounded Coloring* problem (where all the bounds n_{ij} are equal), for which numerous results are known (see in particular [6, 7, 45], as well as additional links referenced in these papers).

In [BP09], we proved the following result, with the help of a dynamic programming algorithm:

Theorem 3.15 ([BP09]). *When both p and the number of colors k are $O(1)$, we can solve* Col-LocalBounds *in polynomial time in trees.*

Applications of Col-LocalBounds, and in particular links with scheduling problems, were also discussed in this paper.

Later, I continued working on a generalization of Col-LocalBounds, named WeightedListCol-LocalBounds. In this more general problem, any vertex v has a weight $w(v)$ and a list of possible colors $L(v)$, and the $p \cdot k$ bounds are denoted by $W_{11}, \ldots, W_{1k}, \ldots, W_{p1}, \ldots, W_{pk}$, and are such that $\sum_{j=1}^{k} W_{ij} = \sum_{v \in V_i} w(v)$ for each i. The problem consists in deciding whether there exists a proper coloring such that, for each i and for each j, the total weight of the vertices of color j in V_i is W_{ij}, and which is a valid *list-coloring*, i.e., the color of any vertex v in this coloring belongs to $L(v)$. For this problem, the following set of (unpublished) results can be shown:

Theorem 3.16. *When both p and the number of colors k are $O(1)$, we can solve* WeightedListCol-LocalBounds *in pseudo-polynomial time in graphs of bounded tree-width. If, in addition, all vertex weights are bounded by a polynomial in the input size, then we can solve it in polynomial time.*

Proof. In order to design a dynamic programming algorithm that solves WeightedLocallyBoundedListColoring in an efficient way provided that a tree decomposition $\big(\{X_i | i \in I\}, T\big)$ of the input graph G having minimum width $tw(G)$ is given, let us define the following function f:

- $f(i, c_i, \omega_{11}, \ldots, \omega_{1k}, \ldots, \omega_{p1}, \ldots, \omega_{pk}) =$ *true* if there is a list-coloring of $G[T_i]$ where each vertex $u \in X_i$ has color $c_i(u)$ and the total weight of vertices of $G[T_i]$ having color c in V_h is ω_{hc}, and *false* otherwise.

To describe the algorithm, one now simply needs to write down the induction equations defining the values of $f(\cdot)$, for each type of nodes of T: forget, introduce, join and leaf (see Section 1.3). Then, these values will be computed in a bottom-up fashion, starting from the leaves of T.

Note that, by the definition of tree decompositions, any two vertices linked by an edge in G must both belong to at least one common bag of the

tree decomposition (Condition (2)). This implies that a (list-)coloring that is proper in each bag is also proper in the whole graph G, provided that each vertex has the same color in each bag it belongs to. Moreover, Condition (3) ensures that the subgraph of T induced by the bags any given vertex belongs to is connected, and hence in all these bags this vertex will have the same color if we do not change its color whenever we move (in T) from one bag to an adjacent one. Therefore, in order to show that a vertex list-coloring is proper in T_i, it suffices to ensure that it is a proper list-coloring in X_i, and that the color of any vertex remains the same when moving from one bag of T_i to an adjacent one. Here are the equations computing the values of $f(\cdot)$:

If i is a forget node. Let j denote its child such that $X_j = X_i \cup \{v\}$:

$$f(i, c_i, \omega_{11}, \ldots, \omega_{pk}) = \bigvee_{c_j:(c_j(u)=c_i(u)\ \forall u\in X_i)\wedge(c_j(v)\in L(v))} f(j, c_j, \omega_{11}, \ldots, \omega_{pk})$$

Proof. Each vertex must keep its color when moving from X_j to X_i. □

If i is an introduce node. Let j denote its child such that $X_j = X_i \setminus \{v\}$, and assume that $v \in V_h$:

$$f(i, c_i, \omega_{11}, \ldots, \omega_{pk}) = \mathcal{E} \bigwedge f(j, c_j, \omega_{11}, \ldots, \omega_{hc_i(v)} - w(v), \ldots, \omega_{pk})$$

where $\mathcal{E} = (c_i(v) \in L(v)) \bigwedge (c_i(v) \neq c_i(u)\ \forall u : uv \in Y_i) \bigwedge (c_j(u) = c_i(u)\ \forall u \in X_j) \bigwedge (\omega_{hc_i(v)} \geq w(v))$.

Proof. The conditions $c_i(v) \in L(v)$ and $c_i(v) \neq c_i(u)$ ensure that the list-coloring is valid in the subgraph of G induced by X_i. □

If i is a join node. Let j and l denote its two children, and let w^i_{hc} be the total weight of vertices $v \in X_i \cap V_h$ such that $c_i(v) = c$, for each h and c.

$$f(i, c_i, \omega_{11}, \ldots, \omega_{pk}) = \bigvee_{(q_{11},\ldots,q_{pk})\in Q^i} \left(f(j, c_i, \omega_{11} + w^i_{11} - q_{11}, \ldots, \omega_{pk} + w^i_{pk} - q_{pk}) \wedge f(l, c_i, q_{11}, \ldots, q_{pk})\right)$$

where $Q^i = \{(q_{11}, \ldots, q_{pk}) : w^i_{hc} \leq q_{hc} \leq \omega_{hc}\ \forall h\ \forall c\}$.

Proof. We try all the possible combinations of vertex weights between the two children of node i, and take into account the fact that the weights of the vertices in X_i must be counted both in T_j and in T_l. □

If i is a leaf node. In this case, we just have to check that the coloring function c_i provides a valid locally bounded list-coloring of $G[X_i]$.

$$f(i, c_i, \omega_{11}, \ldots, \omega_{pk}) = \mathcal{E} \bigwedge (w(\{v \in X_i \cap V_h : c_i(v) = c\}) = \omega_{hc}\ \forall h\ \forall c)$$

where $\mathcal{E} = (c_i(v) \in L(v)\ \forall v \in X_i) \bigwedge (c_i(v) \neq c_i(u)\ \forall u, v \in X_i : uv \in Y_i)$.

Root value. The value computed at the root r of T is:

$$\bigvee_{c_r : c_r(u) \in L(u) \ \forall u \in X_r} f(r, c_r, W_{11}, \ldots, W_{pk})$$

We conclude the proof by noticing that the overall running time we obtain is $O(n(\max_{h,c} W_{hc})^{pk} k^{tw(G)+1}((\max_{h,c} W_{hc})^{pk} + tw(G)(tw(G) + pk)))$. □

Note that Theorem 3.16 generalizes both Theorem 3.15 (where $w(v) = 1$ for each vertex v, the tree-width is 1, and any vertex can take any color) and the results in [45]. This theorem can also easily be extended to the case where one wants to properly color edges instead of vertices.

Furthermore, if the tree-width of the input graph is arbitrary, then Col-LocalBounds is **NP**-complete (even if $k = 3$ and $p = 1$), as the basic problem of deciding whether a graph can be colored with 3 colors is.

Actually, one can show that the conditions $p = O(1)$ and $k = O(1)$ are necessary in Theorem 3.16, since otherwise the problem is strongly **NP**-complete, and that the problem is weakly **NP**-complete even if they hold.

Assume that the vertex weights are arbitrary, and set $k = 2$ and $p = 1$. Take any instance I of the weakly **NP**-complete Partition problem, where we are given $n+1$ positive integers $a_1, \ldots, a_n, B$ such that $\sum_{i=1}^{n} a_i = 2B$, and we want to decide whether there exists $I' \subset \{1, \ldots, n\}$ such that $\sum_{i \in I'} a_i = B$. Consider a set of n isolated vertices $v_1, \ldots, v_n$ (a graph with tree-width 0) that can take any color, and define $W_{11} = W_{12} = B$, as well as $w(v_i) = a_i$ for each i. Then, the answer to instance I is *yes* if and only if the answer to this instance of WeightedListCol-LocalBounds is *yes*. This implies that WeightedListCol-LocalBounds is weakly **NP**-complete in this case.

Assume now that the number of colors k is arbitrary. Bodlaender and Fomin have shown that deciding whether there exists an equitable coloring with k colors when some vertices are already colored is strongly **NP**-complete in trees [6]. However, ensuring that a vertex v takes color c can be done by imposing that the list of possible colors for v contains only c. Hence, WeightedListCol-LocalBounds generalizes this problem, and is also strongly **NP**-complete in trees, even if $w(v) = 1$ for each vertex v and $p = 1$.

Finally, assume that p is arbitrary. In this case, it can be shown that WeightedListCol-LocalBounds remains strongly **NP**-complete, by using a reduction from the following well-known **NP**-complete problem, which is mentioned in the previous section, but which we now describe formally [41]:

Monotone1-in-3SAT
Instance: A set X of ν boolean variables $x_1, \ldots, x_\nu$, and a set of μ clauses, each one containing exactly three (non negated) boolean variables from X.
Question: Decide whether there exists a truth assignment for the variables in X such that in every clause there is exactly one variable equal to *true*.

More precisely, the following result holds:

Theorem 3.17. WeightedListCol-LocalBounds *is* ***NP****-complete in the strong sense in forests, even if* $k = 2$ *and* $w(v) = 1$ *for each* $v \in V$.

Proof. Given an instance of Monotone1-in-3SAT, we construct an instance of WeightedListCol-LocalBounds as follows: the graph G consists of ν vertex-disjoint stars, one star per variable x_i. We denote by v_i the central vertex of the ith star, and by $u_i^0, \ldots, u_i^{occ(i)}$ its leaves, where $occ(i)$ is the number of occurrences of x_i in the set of clauses. Then, we define $k = 2$, $p = \nu + \mu$, and, for each vertex v, $w(v) = 1$. Moreover, any vertex can take any color. The partition of the vertices of G and their target weights are given by: $V_h = \{v_h, u_h^0\}$ and $W_{h1} = W_{h2} = 1$ for each $h \in \{1, \ldots, \nu\}$, and $V_h = \{u_i^a, u_j^b, u_k^c\}$, $W_{h1} = 1$ and $W_{h2} = 2$ for each $h \in \{\nu + 1, \ldots, \nu + \mu\}$, where we consider that the $(h - \nu)$th clause consists of the ath occurrence of the variable x_i, of the bth occurrence of the variable x_j, and of the cth occurrence of the variable x_k.

It is easy to check that we have the following equivalence between this WeightedListCol-LocalBounds instance and the initial Monotone1-in-3SAT instance: for each $i \in \{1, \ldots, \nu\}$, variable x_i is equal to *true* if and only if v_i has color 2, which concludes the proof. □

Complementary results for this problem in other classes of graphs (such as cographs and split graphs) have been obtained recently, but will not be discussed here; the interested reader can refer to [Ben17b].

Chapter 4

Blockers and d-transversals in graphs

4.1 Introduction

In this chapter, we review results about minimum d-transversals for several graph problems. Formally, given a weighted graph G, an integer d, and a graph parameter $\pi(G)$ that cannot increase under vertex (resp. edge) deletions and that is equal to the optimal weight of a set of vertices (resp. edges) satisfying a given graph property Π, a *d-transversal* for $\pi(G)$ is a set of vertices (resp. edges) that intersects at least d times *any* set of vertices (resp. edges) of weight $\pi(G)$ satisfying property Π in G. A related notion is the following: a *blocker* for a graph parameter $\pi(G)$ as defined previously is a set B of vertices (resp. edges) such that $\pi(G-B) < \pi(G)$, where $G-B$ is the graph obtained from G by removing the vertices (resp. edges) in B [26].

In fact, it is not hard to see that we have the following equivalence:

Proposition 4.1 ([ZRP+09]). *Given a weighted graph G and a graph parameter $\pi(G)$ that cannot increase under vertex (resp. edge) deletions and that is equal to the optimal weight of a set of vertices (resp. edges) satisfying a given graph property, a set of vertices (resp. edges) is a 1-transversal for $\pi(G)$ if and only if it is a blocker for $\pi(G)$.*

Examples of graph parameters that we shall consider are $\alpha(G)$, the stability number of G, where the property Π consists in being a stable set, and $\nu(G)$, the maximum size of a matching in G (which can be efficiently computed), the property Π then consisting in being a matching. The general problem that we shall study consists in determining the minimum size (or cost, if each vertex or edge also has a cost) of a d-transversal or blocker. Proposition 4.1 shows that d-transversals can be seen as generalizations of blockers: other generalizations or variants of blockers for different graph parameters were considered in [3, 26, 69], but will not be discussed here.

Note that, basically, such a problem is hard in two ways: first, computing the value of $\pi(G)$ may be hard (this is the case, for instance, for $\pi(G) = \alpha(G)$ in general graphs); second, even if $\pi(G)$ can be computed efficiently, there may be a huge number of optimal vertex (or edge) sets satisfying the property Π. In particular, proving that the decision version of such a problem belongs to **NP** (or not) may be a hard task. In the case of blockers (i.e., $d = 1$), this fact may remain hard to prove whenever computing $\pi(G)$ is **NP**-hard, but the decision version of the problem is obviously in **NP** otherwise.

In any case, this clearly defines a family of covering problems, as one wants to cover at least d times, in an optimal way, *any* set of vertices (resp. edges) of weight $\pi(G)$ satisfying property Π in G.

4.2 Minimum d-transversals for matchings

This section reviews results about the case where the property Π consists in being a matching, and all edge weights are 1. In other words, one wants to determine the minimum size of a set of edges intersecting at least d times any maximum matching of a graph G, for $d \leq \nu(G)$.

In [ZRP+09], we first established, with the help of a rather intricate reduction from the **NP**-hard maximum clique problem, the complexity of finding a minimum-size blocker for $\nu(G)$ in arbitrary bipartite graphs:

Theorem 4.1 ([ZRP+09]). *Determining the minimum size of a blocker for $\nu(G)$ is **NP**-hard in bipartite graphs.*

From this theorem and Proposition 4.1, finding the minimum size of a 1-transversal for $\nu(G)$ is also **NP**-hard in bipartite graphs. Actually, this can also be extended to d-transversals for $d > 1$:

Theorem 4.2 ([ZRP+09]). *For any $d \geq 1$, determining the minimum size of a d-transversal for $\nu(G)$ is **NP**-hard in bipartite graphs.*

This latter result is obtained by reducing the case $d = 1$ to the case where d can take any value, by adding to any initial bipartite instance a matching of size $d - 1$. Moreover, note that, for any $d \geq 1$, the decision version of the problem consisting in determining the minimum size of a d-transversal for $\nu(G)$ is actually in **NP**. More precisely, given any potential d-transversal $\mathcal{T}_d$ in a graph G with m edges, checking that $\mathcal{T}_d$ is indeed a d-transversal for $\nu(G)$ can be done by giving a weight 1 to any edge in $\mathcal{T}_d$ and a weight $1 + \frac{1}{m}$ to any other edge, and then computing a maximum matching of maximum weight, which can be done in polynomial time. If this maximum weight is at most $\nu(G)(1 + \frac{1}{m}) - \frac{d}{m}$, then any maximum matching must use at least d edges from $\mathcal{T}_d$, establishing that $\mathcal{T}_d$ is a d-transversal for $\nu(G)$.

Since the problem of determining the minimum size of a d-transversal for $\nu(G)$ is **NP**-hard in arbitrary bipartite graphs, we studied its complexity in several families of bipartite graphs in [ZRP$^+$09] and [RBP$^+$10]. For instance, the cases of chains and rings can be solved easily (the details, which are left to the reader, are also given in [ZRP$^+$09]). Another natural family of bipartite graphs is the class of regular bipartite graphs, for which we have:

Theorem 4.3 ([ZRP$^+$09]). *Given any bipartite graph G such that all n vertices have the same degree Δ, a minimum-size d-transversal for $\nu(G)$ can be found in polynomial time, by taking the $d\Delta$ edges incident to the vertices of any independent set of size d, for any $d \in \{1, \ldots, \frac{n}{2}\}$.*

The proof is divided into two parts. First, as a direct consequence of *Hall's Theorem*, we know that such a graph G contains Δ disjoint perfect (and hence maximum) matchings, meaning that any d-transversal must contain at least $d\Delta$ edges. Second, as any vertex is incident to an edge of any perfect matching, taking all the edges incident to the vertices of any independent set of size d indeed yields a d-transversal of size $d\Delta$.

We also provided a family of instances showing that, in this case, there may exist minimum-size d-transversals that are not obtained as in Theorem 4.3. Figure 4.1 illustrates the case where $\Delta = 3$ and $d = 4$: the solution obtained by taking the bold edges is a d-transversal of size $d\Delta = 12$, and hence of minimum size, but it is also a matching.

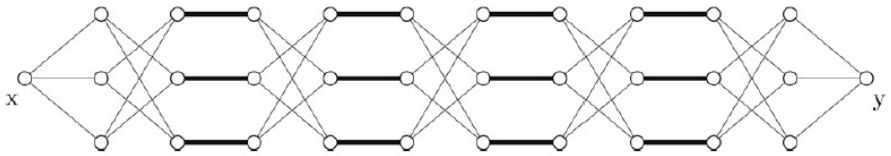

Figure 4.1: A matching of size 12 (in bold edges), which is also a minimum-size 4-transversal in a regular bipartite graph of degree 3.

A natural way of obtaining bipartite graphs that are not regular (but not by far) is by considering the rectilinear grids. We studied them in [RBP$^+$10], and proved that the structure of minimum-size d-transversals can be very different from the one described in Theorem 4.3.

We shall deal with three different main cases, establishing closed formulas for each one of them, before concluding:

- The numbers of rows and columns are even,
- The numbers of rows and columns are odd,
- The number of rows is even and the number of columns is odd.

The following lemma deals with the first main case:

Lemma 4.1 ([RBP+10]). *Let $G_{m,n}$ be a rectilinear grid with m rows and n columns, m and n being even. A minimum-size d-transversal for $\nu(G_{m,n})$*

1. *(when $m = 2$ or $n = 2$)*
   - *has size $2d$ for $d = 1$ or $d = 2$.*
   - *has size $3d - 2$ for $d \in \{3, \ldots, \nu(G_{m,n})\}$.*
2. *(when $m \geq 4$ and $n \geq 4$)*
   - *has size $2d$ for $d \in \{1, 2, 3, 4\}$.*
   - *has size $3d - 4$ for $d \in \{5, \ldots, m + n - 4\}$ and $\max\{m, n\} > 4$.*
   - *has size $4d - m - n$ for $d \in \{m + n - 3, \ldots, \nu(G_{m,n})\}$.*

The construction associated with each of these five cases is also given explicitly, and the solutions obtained mainly consist in taking all the edges incident to a particular set of vertices, as in Theorem 4.3.

We now turn to the second main case:

Lemma 4.2 ([RBP+10]). *Let $G_{m,n}$ be a rectilinear grid with m rows and n columns, m and n being odd. A minimum-size d-transversal for $\nu(G_{m,n})$*

1. *(when $m = 1$ or $n = 1$)*
   - *has size $2d$ for $d \in \{1, \ldots, \nu(G_{m,n})\}$.*
2. *(when $m \geq 3$ and $n \geq 3$)*
   - *has size 3 for $d = 1$.*
   - *has size $2d + 2$ for $d = 2$ or $d = 3$.*
   - *has size 12 for $d = 4$ and $m = n = 3$.*
   - *has size $3d - 1$ for $d \in \{4, \ldots, m + n - 3\}$ and $\max\{m, n\} > 3$.*
   - *has size $4d - m - n + 2$ for $d \in \{m + n - 2, \ldots, \nu(G_{m,n})\}$.*

Again, the construction associated with each of these six cases is also given explicitly, and the solutions obtained mainly consist in taking all the edges incident to a particular set of vertices, as in Theorem 4.3.

Finally, we deal with the third main case:

Lemma 4.3 ([RBP+10]). *Let $G_{m,n}$ be a rectilinear grid with m rows and n columns, m being even and n odd. A minimum-size d-transversal for $\nu(G_{m,n})$*

1. *(when $n = 1$)*
   - *has size d for $d \in \{1, \ldots, \frac{m}{2} = \nu(G_{m,1})\}$.*
2. *(when $m = 2$ and $n \geq 3$)*

- *has size* $2d$ *for* $d = 1$ *or* $d = 2$.
- *has size* $3d - 2$ *for* $d \in \{3, \ldots, n = \nu(G_{2,n})\}$.

3. *(when* $m \geq 4$ *and* $n = 3$*)*

- *has size* $2d$ *for* $d \in \{1, \ldots, \frac{m}{2} + 2\}$.
- *has size* $3d - \frac{m}{2} - 2$ *for* $d \in \{\frac{m}{2} + 3, \ldots, m\}$.
- *has size* $5d - \frac{5m}{2} - 3$ *for* $d \in \{m + 1, \ldots, \frac{3m}{2} = \nu(G_{m,3})\}$.

4. *(when* $m \geq 4$ *and* $n \geq 5$*)*

- *has size* $2d$ *for* $d \in \{1, 2, 3, 4\}$.
- *has size* $3d - 4$ *for* $d \in \{5, \ldots, m + n - 3\}$.
- *has size* $4d - m - n - 1 - \left\lfloor \frac{d-(m+n-3)}{\frac{n-3}{2}} \right\rfloor$ *for* $d \in \{m+n-2, \ldots, \frac{mn}{4} + \frac{m}{4} + \frac{n-5}{2}\}$.
- *has size* $4d - m - n - \left\lfloor \frac{\frac{mn}{2}-d}{\frac{n-1}{2}} \right\rfloor$ *for* $d \in \{\frac{mn}{4} + \frac{m}{4} + \frac{n-3}{2}, \ldots, \nu(G_{m,n})\}$.

As in the two previous lemmas, the construction associated with each case is given, but this time the solutions obtained make use of special structures called *dancats* (for *dancing caterpillars*), shown in Figure 4.2:

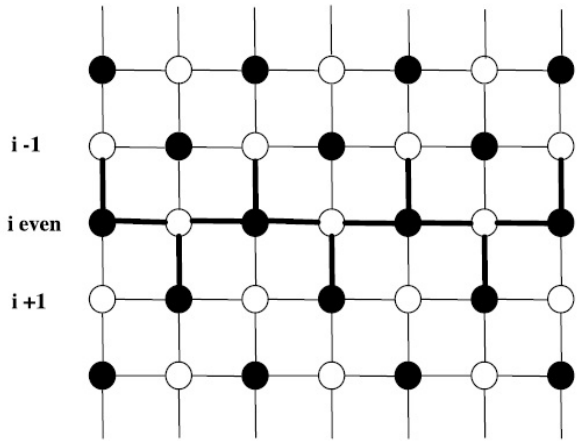

Figure 4.2: A dancat (in bold edges) on a grid.

Depending on the case that is considered, the dancats are then completed, either with a matching or with the set of all edges incident to some vertices, in order to obtain a minimum-size d-transversal. As Lemmas 4.1, 4.2, and 4.3 cover all possible cases, this immediately yields:

Theorem 4.4 ([RBP+10]). *Given any rectilinear grid $G_{m,n}$ with m rows and n columns, a minimum-size d-transversal for $\nu(G_{m,n})$ can be computed in polynomial time, for any $d \in \{1, \ldots, \nu(G_{m,n})\}$.*

Unlike the bipartite graphs that were considered above, the last family of bipartite graphs that we studied has arbitrary maximum degree, but no cycles. In other words, we studied trees, and, using a rather intricate dynamic programming approach, we obtained the following result:

Theorem 4.5 ([RBP+10]). *For any $d \in \{1, \ldots, \nu(G)\}$, the minimum size of a d-transversal for $\nu(G)$ can be computed in polynomial time in trees.*

More precisely, the algorithm used to prove Theorem 4.5 runs in $O(n^5)$ time in any tree with n vertices. Later, Zenklusen even managed to solve this problem in polynomial time in graphs of bounded tree-width [69].

One last result concerns the minimum size of a d-transversal for $\nu(G)$ in non bipartite graphs, and more precisely in complete graphs:

Theorem 4.6 ([ZRP+09]). *Let n and $d \geq 1$ be two integers such that $2d \leq n$, and let $r = \lfloor \frac{n}{2} \rfloor - d$. Let us denote by K_n the complete graph on n vertices. The minimum size of a d-transversal for $\nu(K_n)$ is then:*

- $\binom{n}{2} - \binom{2r+1}{2}$ *if $d \leq \lfloor \frac{n}{2} \rfloor - \frac{2}{5}n + \frac{3}{5}$, and a d-transversal of this size can then be obtained by taking all the edges incident to $n - 2r - 1$ vertices.*
- $\binom{n-r}{2}$ *if $d \geq \lfloor \frac{n}{2} \rfloor - \frac{2}{5}n + \frac{3}{5}$, and a d-transversal of this size can then be obtained by taking a clique on $n - r$ vertices.*

The proof of this theorem is a bit technical, and shows that the constructions described in the theorem are actually the only ones that can lead to minimum-size d-transversals. This implies, in particular, that the case $d = \lfloor \frac{n}{2} \rfloor - \frac{2}{5}n + \frac{3}{5}$ is the only one where K_n contains two (and exactly two) non-isomorphic minimum-size d-transversals. For instance, when $n = 4$ and $d = 1$, a minimum-size 1-transversal (or equivalently, from Proposition 4.1, a minimum-size blocker) has size 3, and can be obtained either by taking a triangle, i.e., a clique on 3 vertices (leaving only a claw in the remaining graph), or by taking a claw corresponding to the 3 edges incident to an arbitrary vertex (leaving only a triangle in the remaining graph).

4.3 Minimum d-transversals for stable sets

This section reviews results about the case where the property Π consists in being a stable set. In other words, given a graph G where each vertex has both a cost and a weight, one wants to determine the minimum cost (or size, if all costs are 1) of a set of vertices intersecting at least d times any maximum-weight stable set of G, for some $d \geq 1$.

Note that the maximum value that d can take is $\alpha(G)$ if all weights are 1, whereas it is equal to the minimum size of a maximum-weight stable set otherwise: both this value and $\alpha(G)$ may be hard to determine. However, we shall study the problem in bipartite graphs, and in this case $\alpha(G)$ can be computed in polynomial time. Also note that Theorem 4.2 implies that, for any fixed $d \geq 1$, determining the minimum size of a d-transversal for $\alpha(G)$ is **NP**-hard in line graphs of bipartite graphs (as a matching in any graph corresponds to a stable set in the associated line graph).

In [BCP+12], we showed the following theorem:

Theorem 4.7 ([BCP+12]). *Given a bipartite graph G where each vertex has both a cost and a weight, a minimum-cost set of vertices intersecting at least d times any maximum-weight stable set of G can be found in polynomial time, for any possible value of $d \geq 1$.*

As this result is based on strong properties of maximum-weight stable sets in bipartite graphs, which can be of independent interest, we describe the main steps of the proof, without providing all the details. In the remainder of the section, let G be a bipartite graph where each vertex v has a cost $c(v)$ and a weight $w(v)$, and let (L, R) be its bipartition and E its edge set.

The first step consists in partitioning G into three sets of vertices:

- The *forced* vertices, which belong to all maximum-weight stable sets,
- The *excluded* vertices, which belong to no maximum-weight stable set,
- The other vertices of G, called *free* vertices.

Note that any excluded vertex can be removed from G, as it belongs neither to a maximum-weight stable set nor to an optimal d-transversal for $\alpha(G)$. Moreover, such a partition can be computed in polynomial time, as the maximum weight of a stable set in a bipartite graph (either G or a graph obtained from G by removing some vertex) can be determined by computing a maximum flow in an associated directed graph. To see this, orient the edges of G from L to R, and assign to each such arc an infinite capacity. Then, add two vertices s and s', an arc of capacity $w(v)$ from s to any vertex v in L, and an arc of capacity $w(v)$ from any vertex v in R to s'. Clearly, the value of a maximum flow from s to s' in this directed graph G' equals, in G, the total weight of vertices minus the maximum weight of a stable set.

Also note that a *forced* vertex does not necessarily belong to a minimum-cost d-transversal for $\alpha(G)$, as its cost can be prohibitive. However, we can remove it from G for now, as we shall show later how to deal with it. Hence, we can assume, without loss of generality, that all vertices of G are free, which implies that L and R are two disjoint maximum-weight stable sets.

The second step consists in identifying *forbidden* arcs in G', i.e., arcs that carry no flow unit in any maximum flow from s to s'. In particular, we show that these arcs can be identified in polynomial time, and that, when G is connected and there are no such arcs in G', L and R are the only maximum-weight stable sets in G.

Therefore, we define a (unique) partition of G in the following way: we remove from G all the edges associated with the forbidden arcs of G', and denote by $G_1, \ldots, G_h$ the h connected components of the resulting graph. Then, it is clear that, for each i, the only two maximum-weight stable sets in G_i are obtained by taking either the vertices both in L and in G_i or the vertices both in R and in G_i. As L is a maximum-weight stable set in G (containing all the vertices both in L and in G_i, for each i), this implies that we have the following characterization: a stable set in G has maximum weight if and only if, for each i, it contains either all the vertices both in L and in G_i or all the vertices both in R and in G_i.

We construct another directed graph from G, denoted by $\overrightarrow{G}$: there is a vertex i for each G_i, and an arc between two vertices i and j if and only if there exists an edge uv in G such that u is both in G_i and in L, and v is both in G_j and in R. Note that, by the definition of $\overrightarrow{G}$ and the characterization of maximum-weight stable sets in G, if a maximum-weight stable set in G contains all the vertices both in G_i and L for some i, then it must contain all the vertices both in G_j and L, for all successors j of i in $\overrightarrow{G}$.

We also define the following relationship between two vertices $u \in R$ and $v \in L$ of G: $u \preccurlyeq v$ if and only if u and v belong to G_i for some i, or $u \in G_i$ and $v \in G_j$, and there exists a directed path from i to j in $\overrightarrow{G}$. From the above discussion, for any pair of vertices u, v such that $u \preccurlyeq v$, any maximum-weight stable set in G contains either u or v. Hence, taking any d pairs of vertices u_i, v_i such that $u_i \preccurlyeq v_i$ for each i yields a d-transversal for $\alpha(G)$ of size $2d$. Note that, since L and R are two disjoint maximum-weight stable sets in G, any d-transversal for $\alpha(G)$ must contain at least $2d$ vertices.

The third step consists in constructing another bipartite graph associated with G, denoted by $\tilde{G}$: the vertices of $\tilde{G}$ are the vertices of G, and there is an edge between two vertices u and v with cost $c(u) + c(v)$ if and only if $u \preccurlyeq v$. We also add in $\tilde{G}$ an isolated edge of cost $c(v)$ for each forced vertex v that was in G. Then, it is not hard to see that there is an equivalence between d-transversals for $\alpha(G)$ obtained as described above (when, in addition, we allow forced vertices) and matchings of size d in $\tilde{G}$.

A rather technical proof allowed us to establish that the minimum size of a maximum-weight stable set in G is equal to $\nu(\tilde{G})$, which implies that computing a minimum-cost d-transversal for $\alpha(G)$ containing only forced vertices and pairs of vertices u_i, v_i such that $u_i \preccurlyeq v_i$ for each i is equivalent to finding a minimum-cost matching of size d in $\tilde{G}$. Another technical proof

establishes that this is the only way of obtaining inclusion-wise minimal d-transversals for $\alpha(G)$ in bipartite graphs, concluding the whole proof.

We make two last remarks. On the one hand, we previously obtained similar results for trees in [BCdW+11], before we managed to generalize them to bipartite graphs. On the other hand, the above characterization of maximum-weight stable sets in bipartite graphs enabled us to obtain in [BCP+12] similar results when the graph parameter is not $\alpha(G)$, but the minimum weight of a vertex cover (a well-known related parameter).

4.4 Minimum blockers for the chromatic number

This section reviews results about a variant of the blocker notion as defined in Section 4.1: here, we want to decrease by at least 1 the chromatic number $\chi(G)$ of a graph G by removing as few edges as possible. A set of edges whose removal decreases the chromatic number of a graph G by at least 1 will be called a *chromatic blocker for* G. A related problem is the following: we want to decrease by at least 1 the stability number of a graph G by *adding* as few edges as possible. A set of non-edges whose addition decreases the stability number $\alpha(G)$ of a graph G by at least 1 will be called a *stability blocker for* G. As in the previous sections, and for similar reasons, proving that such a problem belongs to **NP** (or not) may be a hard task.

Note that, when looking for a minimum-size chromatic blocker for a graph G, we can assume without loss of generality that $\chi(G) \geq 3$, as the problem is trivial otherwise: a graph can be colored with only one color if and only if it has no edge. For similar reasons, when looking for a minimum-size stability blocker for G, we can assume without loss of generality that $\alpha(G) \geq 3$.

It is easy to see that finding a minimum-size chromatic blocker in a complete graph is trivial, as one edge is enough. Similarly, any minimum-size stability blocker in a graph with no edge consists of only one edge. However, we showed in [BBPR15] that both problems are **NP**-hard in general:

Theorem 4.8 ([BBPR15])**.** *Determining the minimum size of a chromatic blocker (resp. of a stability blocker) is an* ***NP****-hard problem.*

For the stability blocker, the proof uses a polynomial Turing reduction from the problem of computing $\alpha(G)$ for a graph G. Let $G_1, \ldots, G_n$ be n copies of G, where n is the number of vertices of G. For each i, add a set I_i of $i+1$ isolated vertices, and an edge between each of these vertices and any vertex of G_i. We ask whether, in each $G_i \cup I_i$, there exists a stability blocker containing only one edge. If $i \geq \alpha(G)$, then such an edge can be used to decrease the stability number of I_i (and hence of $G_i \cup I_i$) by 1, and hence the answer is *yes*. If $i = \alpha(G) - 1$, then we need to add one edge to decrease the stability number of I_i and at least one edge to decrease the stability number of G_i, and hence the answer is *no*, concluding the proof.

Because of these **NP**-hardness results, we then studied the problems in relevant special cases. We first considered the case of *split* graphs, i.e., graphs whose vertex set can be partitioned into a stable set S and a clique K:

Theorem 4.9 ([BBPR15]). *A minimum-size chromatic blocker (resp. stability blocker) can be determined in polynomial time in split graphs.*

For the chromatic blocker, we can assume without loss of generality that the chromatic number is the size of K. If K has at least 4 vertices, then removing two disjoint edges is enough: the chromatic number of K decreases by 2, and we use an additional color for S if needed. Otherwise, K is a triangle, and removing its 3 edges is enough to obtain a bipartite graph. In both cases, we obtain chromatic blockers of size at most 3, and a minimum-size chromatic blocker can thus be determined by brute-force enumeration.

For the stability blocker, we can assume without loss of generality that the stability number is the size of S. We choose 3 vertices in S, and add three edges between them to obtain a triangle. The stability number of S decreases by 2, and we can select at most one additional vertex from K in any stable set. This stability blocker has size 3, and thus a minimum-size stability blocker can be determined by brute-force enumeration.

The second special case that was considered in [BBPR15] is the one of complements of bipartite graphs. Computing a minimum-size stability blocker is trivial in this case (as $\alpha(G) = 2$ for such a G), so we are interested in chromatic blockers. We provided in [BBPR15] a full characterization of the value taken by the minimum size of a chromatic blocker in such graphs, based on a rather long case analysis involving seven conditions. However, proving that the problem is polynomial-time solvable is much easier:

Theorem 4.10 ([BBPR15]). *A minimum-size chromatic blocker can be determined in polynomial time in the complements of bipartite graphs.*

Properly coloring the vertices of a graph G always consists in covering them by stable sets. However, in this special case, the only possible stable sets are non-edges and isolated vertices. If G contains at least 2 disjoint stable sets of size 2 (note that there necessarily exists a proper optimal vertex coloring that uses them), then we can merge them into a single stable set of size 4 by removing at most 4 edges. Otherwise, either the graph has at most 3 vertices or there exist at least 2 stable sets of size 1, and we can merge two such stable sets into a stable set of size 2 by removing one edge. In both cases, we obtain chromatic blockers of size at most 4, and a minimum-size chromatic blocker can thus be determined by brute-force enumeration.

The last special case that was considered in [BBPR15] is the one of bipartite graphs. Computing a minimum-size chromatic blocker is trivial in this case, so we are interested in stability blockers. We provided in [BBPR15] the following characterization for determining the minimum size of a stability blocker in such graphs, based on the partition described in Section 4.3:

Theorem 4.11 ([BBPR15]). *Let G be a bipartite graph, and let $G_1, \ldots, G_h$ be the (unique) partition of the free vertices of G, i.e., G_i is the ith connected component of the graph obtained by removing, from the subgraph of G induced by the free vertices, the edges associated with the forbidden arcs of the directed graph G'. The minimum size of a stability blocker in G is equal to:*

- 1 *if and only if G contains at least two forced vertices,*
- 2 *if and only if G contains at most one forced vertex, and:*
  - (a) *either G contains exactly one forced vertex,*
  - (b) *or there is an i such that G_i contains at least four vertices,*
  - (c) *or G contains the graph in Figure 4.3 as a subgraph,*
  - (d) *or G contains the graph in Figure 4.4 as a subgraph.*
- 3 *if and only if we are not in the above cases and there exist two integers i and j such that $i \neq j$ and G contains an edge having one endpoint in G_i and one endpoint in G_j,*
- 4 *otherwise.*

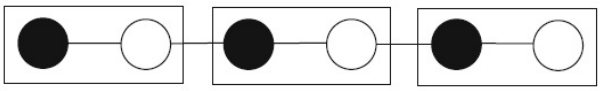

Figure 4.3: A graph containing three vertices from L (black vertices) and three vertices from R (white vertices). The three rectangles represent three different graphs G_i of the partition of G (each rectangle is such a graph).

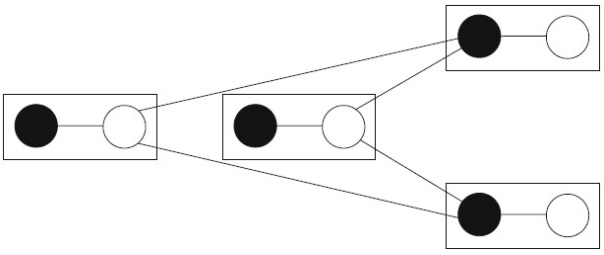

Figure 4.4: A graph containing four vertices from L (black vertices) and four vertices from R (white vertices). The four rectangles represent four different graphs G_i of the partition of G (each rectangle is such a graph).

Chapter 5

Steiner trees & related variants

5.1 Introduction

In this chapter, we review results about several variants of Steiner tree problems. Let T be a set of K terminal vertices, or simply *terminals*, in a connected undirected graph G where each edge e has a cost (or length) $c(e) \in \mathbb{Q}^+$. A *Steiner tree* in G is a tree covering the K terminals: as G is assumed to be connected, such a tree always exists (take any spanning tree, for instance). Moreover, the vertices in G that do not belong to T are called *Steiner vertices*. We shall also consider the case of directed graphs: in this case, we are interested in *rooted* Steiner trees. Let r be a root vertex (or simply root) in a graph G: there must be a (directed) path from r to any vertex in G. A rooted Steiner tree in G with respect to r is a Steiner tree rooted at r in G; we shall omit "with respect to r" except when necessary. In undirected graphs, a rooted Steiner tree can be viewed as a Steiner tree covering both T and r, which can be ensured by adding r to T. Hence, in this case, there is no need to consider rooted Steiner trees explicitly.

The minimum (rooted) Steiner tree problem (MinSteinerTree) is to determine a tree S in G, rooted at r, spanning all the terminals and having minimum total length (or cost). As any Steiner tree must cover pre-specified vertices (the terminals), this is clearly a covering problem. The case where G is undirected has been widely studied: it has many applications, as shown in [17, 50], and the associated decision problem was one of R. Karp's 21 **NP**-complete problems [41]. Also note that Steiner trees are actually the basic routing components used for multicast routing in networks.

When all vertices are terminals, MinSteinerTree becomes the minimum spanning tree problem, which is well-known to be solvable in polynomial time, using for instance greedy algorithms, such as the ones of Kruskal or Prim. In general, however, MinSteinerTree is **APX**-hard, although it can be solved in **FPT** time with respect to the number of terminals [31], and admits constant-factor approximation algorithms [50, 63].

Note that, in this problem, requiring that any feasible solution is a Steiner tree is actually not explicitly needed: it is sufficient to require that we can route 1 unit of flow from r to each terminal (i.e., any feasible solution covers all the terminals), and that, if an edge is used by at least 1 unit of flow, then its full cost must be paid. In this way, any *optimal* solution has to be a tree; otherwise, it can easily be transformed (by rerouting flow units if needed) into a feasible solution that is a Steiner tree, without increasing the total cost. This may not be the case when additional constraints are considered.

There are less results about the case where the graph is directed, which is a generalization of the undirected case; only non constant ratio approximation algorithms are known [10]. Such a case occurs, for instance, when considering multicast routing in asymmetric networks [58]. Note that, unlike in undirected graphs, this case also generalizes the well-known set cover problem (SetCover), even in directed acyclic graphs. In SetCover, we are given a set of elements E, and a set X of subsets $X_1, \ldots, X_\nu$ of elements of E, and we want to determine the minimum number of subsets from X needed to cover all the elements of E (an element e of E is covered by X_i if e belongs to X_i). The decision version of the problem is to decide whether all the elements of E can be covered by a given number of subsets from X.

On the one hand, a logarithmic inapproximability bound is known for SetCover [35], and the best lower bound known for the approximability of MinSteinerTree in directed graphs is actually based on this result and on the fact that SetCover polynomially reduces to MinSteinerTree in directed acyclic graphs. On the other hand, the best upper bound known for the approximability of MinSteinerTree in directed graphs is polynomial in K: namely, there exists a $O(K^\epsilon)$-approximation algorithm for this problem, for any $\epsilon > 0$ [10]. Moreover, no better approximation algorithm is known for the special case of MinSteinerTree in directed acyclic graphs either.

5.2 Steiner trees with a bound on the number of branching or diffusing nodes

In this section, we consider two generalizations of MinSteinerTree, studied during the PhD thesis that D. Watel did under the supervision of D. Barth, M.-A. Weisser and myself. Given a Steiner tree S rooted at a vertex r, a *branching* vertex is a vertex having at least two children in S. The first generalization, which consists in finding a minimum-cost Steiner tree having at most a given number p of branching vertices, was introduced in [WWBB13] and motivated by a routing problem in optical networks. Actually, installing a device that enables a vertex to duplicate the data it receives may be costly in such networks, and therefore, in practice, the number of vertices that can duplicate data is limited by the available budget. Hence, if we route the data along a Steiner tree, as it is commonly the case, we obtain the above problem.

Note that, if the number of branching vertices must be at most $p = K-1$, then the problem is exactly MinSteinerTree. However, when $p \leq K-2$, the problem is hard even with costs 0:

Theorem 5.1 ([WWBB13]). *For any $p \leq K-2$, deciding whether there exists a Steiner tree with at most p branching vertices is **NP**-complete:*

- *in directed graphs, for any fixed $K \geq 2$.*
- *in planar graphs (directed or undirected).*

To prove the first part of this theorem, take any directed graph G and two source-sink pairs (s_1, s_1') and (s_2, s_2'). Add a root r and $K = 2$ terminals t_1 and t_2, as well as four arcs $(r, s_1), (s_1', t_1), (t_1, s_2), (s_2', t_2)$: then, finding a tree rooted at r that covers t_1 and t_2 with $p = 0$ amounts to finding in G two vertex-disjoint paths linking s_1 to s_1' and s_2 to s_2' respectively, which is an **NP**-complete problem [38]. This reduction can be generalized to any value of p by adding a gadget linked to s_2', in order to ensure that there can be no branching vertex inside G. The second part is proved by reduction from the **NP**-complete hamiltonian path problem, which consists in deciding whether there exists a path that goes through each vertex once and only once.

By adapting the above reductions, we can also show:

Theorem 5.2 ([WWBB13]). *For any $p \leq K-2$, the problem of finding the minimum cost of a Steiner tree with at most p branching vertices is **NP**-hard to approximate within a ratio $O(n^{1-\epsilon})$ for any $\epsilon > 0$, even if a feasible solution always exists and is given in the input, in the following cases:*

- *in directed graphs with n vertices and unit costs, for any fixed $K \geq 2$.*
- *in planar graphs (directed or undirected) with n vertices and unit costs.*

Note that this lower bound is best possible when all costs are 1, as taking *any* feasible solution yields a $O(n)$-approximation algorithm in this case.

In directed acyclic graphs, we have the two following results:

Theorem 5.3 ([WWBB13]). *When $p = O(1)$, the problem of finding the minimum cost of a Steiner tree with at most p branching vertices can be solved in polynomial time in directed acyclic graphs. Moreover, the problem is **W[2]**-hard with respect to p in directed acyclic graphs.*

The first part of the theorem is proved by enumerating all the possible sets of at most p branching vertices, and then solving for each such set an instance of the minimum-cost flow problem. The second part is proved by an **FPT**-reduction from the SetCover problem parameterized by the number of sets that can be in the solution, and shows that the first part is essentially best possible, as the problem is unlikely to be **FPT** with respect to p.

Given a Steiner tree S rooted at vertex r, we define the *pattern* of S as the tree obtained from S by iteratively contracting any edge having at least one endpoint which is both different from r and neither a terminal nor a branching vertex. In the pattern of S, the only vertices are then r, terminals, and branching vertices. By enumerating all the possible patterns of a (minimum-cost) solution, and then extending every pattern to a solution (whenever this is possible) by computing vertex-disjoint paths (of minimum total cost) between the endpoints of each edge of the pattern, we can show:

Theorem 5.4 ([WWBB13]). *When $K = O(1)$, the problem of finding a minimum-cost Steiner tree with at most p branching vertices is:*

- *polynomial-time solvable in planar directed graphs if all costs are 0,*
- *polynomial-time solvable in undirected graphs if all costs are 0,*
- *polynomial-time solvable in undirected graphs of bounded tree-width.*

Note that the complexity of the problem with arbitrary costs remains open in the first and second cases of Theorem 5.4.

To define the second generalization of the (rooted) Steiner tree problem that we shall consider, we need to introduce several notions. Given a graph G, the *shortest paths graph of G* (or *metric completion of G*) is the graph whose vertices are the ones of G, and in which there is an edge between any two vertices u and v, whose cost is the distance between u and v in G.

Given a Steiner tree S rooted at a vertex r in the shortest paths graph of a graph G, a *diffusing* vertex in G is a branching vertex of S in the shortest paths graph of G. In this setting, *any* vertex is actually assumed to be able to transmit a data to several successors, *provided that it receives the data enough times*. If the same data is sent several times to a vertex, then, for instance, the cost of sending it twice is twice the cost of sending it once. On the contrary, a diffusing vertex is able to transmit a data to several successors, *even if it receives the data only once*. Hence, any vertex can be a branching vertex, but a diffusing vertex shall be a branching vertex (otherwise, making it a diffusing vertex is useless). In the remainder of this section, we consider the problem of finding an optimal routing in a graph G having at most d *diffusing* vertices, i.e., a minimum-cost Steiner tree in the shortest paths graph of G rooted at r and that has at most a given number d of *branching* vertices. Note that such a Steiner tree always exists (for instance, take any hamiltonian path starting at r in this complete graph).

In [WWBB15], we showed the following positive result regarding the parameterized complexity of the problem with respect to K:

Theorem 5.5 ([WWBB15]). *A minimum-cost routing with at most d diffusing vertices can be computed by an algorithm that is **FPT** with respect to K, while using only polynomial space. In particular, when $d = K - 1$, the algorithm returns a minimum-cost Steiner tree.*

This algorithm is obtained by combining the notion of pattern mentioned above (but where the non-leaf vertices of the pattern are not labeled, and hence the number of patterns that need to be considered in this case is **FPT** with respect to K) and a dynamic programming approach, which computes the best solution associated with each pattern. Note that it was known for a long time that MinSteinerTree is **FPT** with respect to K using dynamic programming only [31], but the only known algorithms that use polynomial space are not **FPT**. For instance, the algorithm in [36] runs in time $O(5.96^K n^{O(\log(K))})$ in graphs with n vertices, while the algorithm in [60] is **FPT** with respect to K only when all costs are $O(1)$ (when costs are arbitrary, a pseudo-polynomial factor appears in its running time).

In [WWBB14, WWBB16], we obtained the following results regarding the parameterized complexity of the problem with respect to d:

Theorem 5.6 ([WWBB14, WWBB16]). *When $d = O(1)$, the problem of finding the minimum cost of a Steiner tree with at most d branching vertices in a shortest paths graph is polynomial-time solvable. Moreover, the problem is **W[2]**-hard with respect to d.*

The first part of the theorem is proved by enumerating all the possible sets of at most d diffusing vertices, and then finding a minimum-weight spanning tree. As in Theorem 5.3, the second part is proved by an **FPT**-reduction from the SetCover problem parameterized by the number of sets that can be in the solution, and shows that the first part is essentially best possible, as the problem is unlikely to be **FPT** with respect to d. We also studied in [WWBB14, WWBB16] another parametrization of the problem:

Theorem 5.7 ([WWBB14, WWBB16]). *When $K-d = O(1)$, the problem of finding the minimum cost of a Steiner tree with at most d branching vertices in a shortest paths graph is **NP**-hard.*

Finally, an approximation-preserving reduction from MinSteinerTree to the case where there are at most d diffusing nodes implied two results, given in [WWBB14, WWBB16]. The first one deals with MinSteinerTree:

Theorem 5.8 ([WWBB14, WWBB16]). *There is an $\lceil \frac{K-1}{d} \rceil$-approximation algorithm for* MinSteinerTree *that runs in polynomial time if $d = O(1)$.*

The second one is a negative one for the case with at most d diffusing vertices, and is based on a complexity assumption different from **P**≠**NP**:

Theorem 5.9 ([WWBB14, WWBB16]). *There is no $O\left(\frac{K}{d}\right)$-approximation algorithm for computing the minimum cost of a Steiner tree with at most d branching vertices in a shortest paths graph unless **NP**⊆**ZTIME**$[n^{log^{O(1)}n}]$.*

5.3 Steiner trees with edge capacities

In this section, we consider another generalization of the (rooted) Steiner tree problem. Assume each edge e has a *capacity* $capa(e)$, i.e., an upper bound on the number of paths containing e and linking r to terminals. In other words, for every arc $e = (u, v)$ in a Steiner tree S rooted at r, the subtree of S rooted at v must contain at most $capa(e)$ terminals (with $capa(e) \leq K$, without loss of generality), which adds a packing constraint to MinSteinerTree, a covering problem. The minimum-length capacitated (rooted) Steiner tree problem (MinCapSteinerTree) is defined as follows:

Minimum-length (rooted) Capacitated Steiner Tree Problem
Input. A connected graph $G = (V, E)$ with $|V| = n$ vertices; a set $T = \{t_1, ..., t_K\} \subset V$ of $K \geq 2$ terminals; a root vertex $r \in V \setminus T$; two functions on E: a nonnegative length function c and a positive capacity function $capa$.
Goal. Determine, if it exists, a minimum-length directed tree S rooted at r in G, spanning all the vertices of T and not violating the capacity constraints.

If G is undirected and $e = (u, v)$ is an arc of S, then $[u, v]$ must be an edge of G. Note that MinSteinerTree is the special case of MinCapSteinerTree where $capa(e) = K$ for all $e \in E$ (in this case, a feasible solution always exists). MinCapSteinerTree appears naturally in several contexts, for example in the design of wind farm collection networks [48] or of telecommunication networks [17], or in power distribution system optimization [32]. When $c(e) = 0$ for all $e \in E$, MinCapSteinerTree turns into a decision problem: determine whether there exists or not a tree rooted at r, spanning all the terminals, and not violating the capacity constraints.

When $K = n - 1$, i.e., any feasible solution is a spanning tree, MinCapSteinerTree is solvable in polynomial time if $capa(e) = 2$ for all $e \in E$, while it is **NP**-hard if $capa(e) = 3$ for all $e \in E$ [41]. In [2, 52], the authors provide approximation algorithms for a variant of MinCapSteinerTree where the capacities are uniform and there always exists a feasible solution, since it is assumed that the input graph is a metric completion.

Our main result in [BCH16] is a complete characterization of the complexity of MinCapSteinerTree, which enables us to distinguish between easy and hard cases of the problem for digraphs, directed acyclic graphs and undirected graphs. In particular, we show that there are strong links between MinCapSteinerTree and another well-known graph optimization problem, namely finding vertex-disjoint paths of minimum total length between source-sink pairs. Moreover, whenever MinCapSteinerTree with arbitrary lengths is intractable while the case with lengths 0 is not, we provide approximation results for MinCapSteinerTree nearly as good as the best ones for MinSteinerTree. Therefore, any inapproximability result we obtain is in fact an intractability result for the case with lengths 0.

Note that any undirected instance of MinCapSteinerTree can be transformed into a directed one by replacing each edge by two opposite arcs with the same length and capacity. Hence, any positive result (existence of a polynomial-time algorithm or approximation result) for directed graphs also holds for undirected graphs, while any negative result for undirected graphs (**NP**-hardness or inapproximability result) also holds for directed graphs.

Apart from the assumption on the graph itself (undirected, directed with or without circuits), several parameters are considered. Namely, the number K of terminals, the minimum and maximum edge capacities, and the edge lengths: K can be $O(1)$ or not; the minimum and maximum edge capacities can be non depending on K, they can be greater than or equal to $K - \kappa$ ($1 \leq \kappa \leq K - 1$), and they can be equal (uniform capacity) or not; the edge lengths can be all equal to 0, all equal to a positive value (i.e., uniform), or non uniform. In [BCH16], we settle all cases except one, namely the undirected case with uniform capacity and fixed $K \geq 3$, but we prove that MinCapSteinerTree is then *equivalent* to finding, in an undirected graph, vertex-disjoint paths of minimum total length linking $O(1)$ source-sink pairs, a problem whose complexity is a long-standing open question [56].

The three trees drawn in Figure 5.1 provide a picture of the possible cases for the three types of graphs (undirected or directed graphs, acyclic or not). The numbers assigned to the leaves of these trees refer to the theorems that were used to prove the associated results, but we will not detail them further here: two leaves where the same number appears simply correspond to results that are proved by the same theorem. The values of the three parameters appear on the branches and each branching node corresponds to a partition of the possible cases: the value on a branch excludes the values on the branches to the left. For instance, in undirected graphs, the capacities can be either all equal to 1, or at least $K - 1$, or uniform of value at least 2 and at most $K - 2$, or, finally, any other values not yet considered.

Moreover, if a leaf corresponds to a branch where the values of some of the parameters are unspecified, then this means that the associated result holds even in the most general case (if it is a positive, i.e., tractability result) or in the most specific case (if it is a negative, i.e., hardness result) with regard to the unspecified values. For instance, the **NP**-hardness result associated with Leaf 7 holds even if K is fixed (i.e., if $K = O(1)$) and if all lengths are 0 (since neither the value of K nor the lengths appear on this branch), and the result associated with Leaf 11 holds for any lengths and any capacities (since only the assumption on K being fixed appears on this branch).

Therefore, for digraphs, the branch "any capacity" includes the case of uniform capacities between 2 and $K-2$ for K fixed (or not). Also note that, if the capacity is uniform and $K = O(1)$, then there exists some constant κ such that all capacities are equal to $K - \kappa$: hence, in the tree associated with undirected graphs in Figure 5.1, the branch "any capacity", which leads to Leaf 8, excludes the case where K is fixed (contained in Leaves 9 and 10).

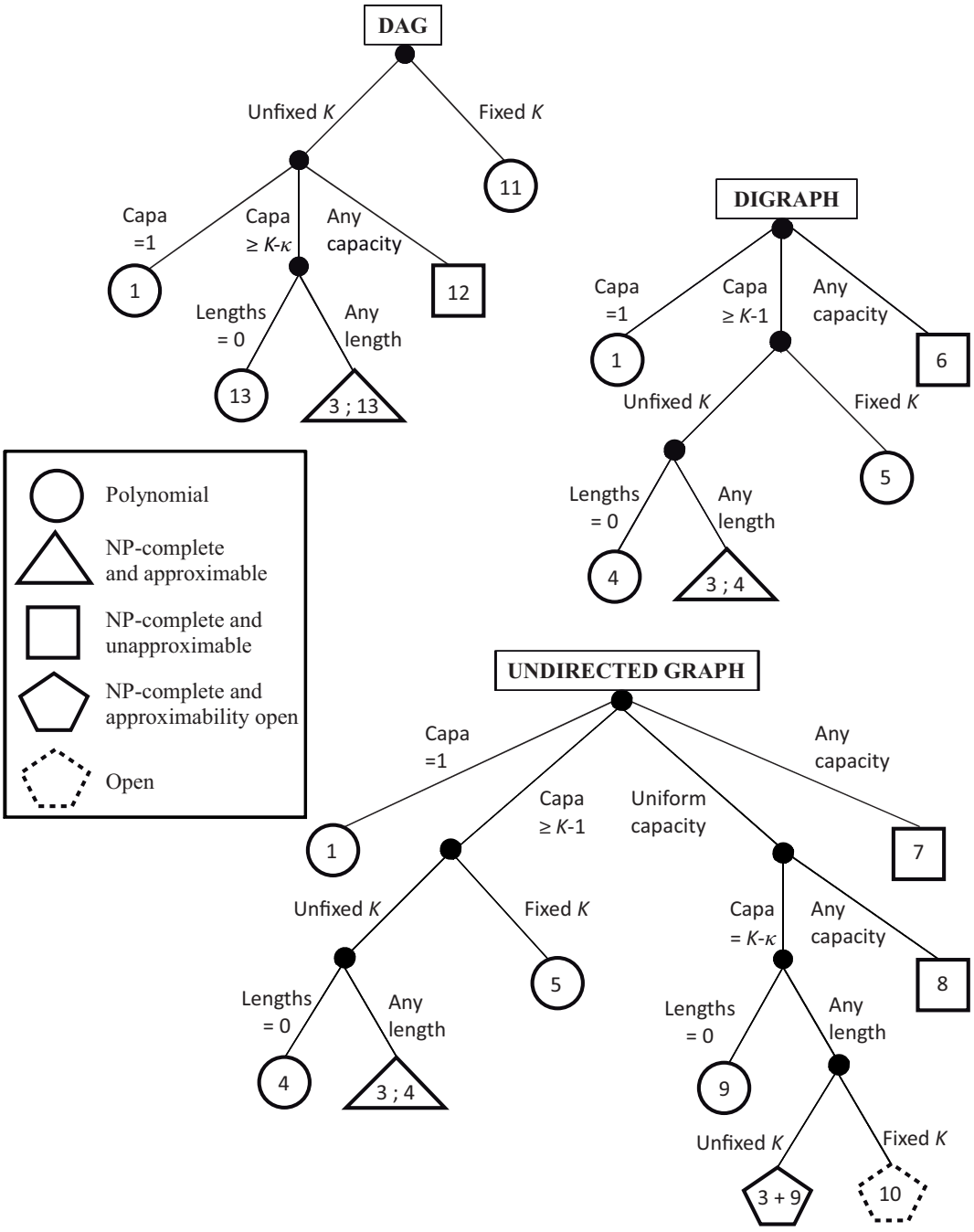

Figure 5.1: An overview of the results giving a complete characterization of the complexity of finding Steiner trees satisfying given edge capacities. (Note that *DAG* stands for *directed acyclic graph*.)

5.4 Routing along Steiner trees or networks when edges can fail

In this last section, we consider two kinds of problems related to routing data along one or several Steiner trees or networks, in particular when one of the edges in the graph can fail.

The first set of results has been obtained during the PhD thesis that T. Lefebvre did under the supervision of S. Elloumi, E. Gourdin (research engineer at Orange Labs) and myself. The purpose of his thesis was to provide answers to the following question: given a telecommunication network that supports multicast routing, is there some benefit (in terms of cost, or in terms of achievable throughput) to also use *network coding*?

In order to support multicast routing (see also Sections 5.1 and 5.2), a network must allow any of its nodes to copy the data it receives, and then emit it towards *all* its successors (even if the data was received only once). While a classical routing (or flow) can be viewed as an assignment of values to a set of paths, linking a given source to a given sink (see Section 2.1), a multicast routing (or *Steiner flow*) can be viewed as an assignment of values to a set of Steiner trees, linking a given source to a given set of terminals. This also means that a multicast routing can be viewed as a fractional packing of Steiner trees (i.e., of structures covering both the source and the terminals).

Roughly speaking, using network coding can be viewed as combining (with the help of a so-called *coding scheme*) several data received by a node into a single data, which will be emitted towards its successors. It can be proved that, if the network is represented by a directed graph, then a routing using both multicast and network coding can be viewed as the superposition of several maximum flows (i.e., any one of them can use *all* the capacity available on each arc), namely one from the source to each terminal.

In both cases, we want to define a *static* routing, i.e., one that cannot be modified even if some edge failure occurs in the network. For such a routing, its residual throughput is the throughput that is still achieved by this routing, even if any edge can fail (this corresponds to the worst case).

The initial question was actually studied in several settings [58]:

- when each edge has a convex cost function, and the goal is to minimize the total cost while ensuring a given throughput,
- when there are actually several sources, each with its own set of terminals (*multi-commodity* case) and an associated weight, and the goal is either to maximize the total weighted throughput, or to maximize the common throughput of all commodities (fairness requirement),
- when there can be one edge failure in the network, and the goal is either to minimize the total cost for a given residual throughput (each edge having an integral cost) or to maximize the residual throughput.

Moreover, in each setting, the network (graph) can be either undirected, bi-directed, or directed. Each setting has an associated *gain*, defined as the ratio between the optimal throughput of a multicast routing using network coding and the optimal throughput of one not using it (reverse the ratio if costs are considered). Obviously, such a ratio is at least 1, and, in each setting, theoretical upper bounds on it are provided [58, BEGL14, BEGL15].

This gain is also computed on real-life instances, which requires in particular to compute the optimal value of the multicast routing without network coding (for the multicast routing with network coding, this is generally much easier, as it amounts to solving a linear program or a convex problem). In the case of convex cost functions, this is essentially done by using Frank-Wolfe algorithm and differential calculus, and solving a combinatorial Steiner tree problem at each step. In all other cases, this is done by using column generation, and solving a combinatorial Steiner tree problem (or some variant of it) to generate columns at each step.

The second set of results was obtained during the Master's internship, and then the PhD thesis, that T. Ridremont did under the supervision of M.-C. Costa, one other researcher (D. Porumbel during the Master's internship and A. Hertz during the PhD thesis), and myself.

The problem studied during the Master's internship of T. Ridremont can be viewed as a reliable Steiner tree problem, and consists in computing an optimal Steiner tree rooted at some given vertex r, when all vertices, except r, are terminals (so any Steiner tree is in fact a spanning tree), and we want to minimize, in the worst case, the number of terminals (i.e., vertices) disconnected from r when we delete some edge from the graph. This is a reliable Steiner *tree* problem in the sense that we want to minimize the biggest impact that one edge failure can cause in the network.

We proved the following hardness result:

Theorem 5.10 ([62]). *Determining a spanning tree, rooted at a given root vertex r, and minimizing the maximum number of vertices disconnected from r when any one edge is removed, is strongly **NP**-hard.*

This is proved via a reduction from the problem 3-PARTITION, a variant of the problem PARTITION defined in Section 3.4, which is strongly **NP**-hard.

Several ILP formulations for this problem and some of its variants were also proposed and tested on real-life instances.

The basic problem studied during the PhD thesis of T. Ridremont can be viewed as a reliable Steiner *network* problem. A (rooted) Steiner network in a graph G is a connected subgraph of G that covers all the terminals (and the root), but is not required to be a tree. In particular, if we add edge capacities (as in Section 5.3) or if we want the network to be resilient to one edge failure, or both, then optimal solutions will not (always) be trees.

Given a graph with a root vertex r, edge costs, and edge capacities, the problem that we consider thus consists in determining a minimum-cost Steiner network rooted at r such that, for any edge e, there exists a flow in this network that does not use e and simultaneously conveys 1 unit of flow from r to any terminal, while respecting the edge capacities.

This problem is **NP**-hard, and quite hard to solve efficiently in practice. In order to solve it exactly, three formulations were developed in [BCPR17].

The first one is an ILP based on flow variables with two indices, and has a polynomial (but big) number of variables and constraints. However, the optimal value of its continuous relaxation is not good enough, and hence it cannot be used in practice to solve the problem when the input graph has more than forty or fifty vertices, even with a commercial solver like CPLEX.

The second one is an ILP based on a cutset formulation, and has few variables but an exponential number of constraints. It can be solved by column generation, and provides better results than the previous formulation.

The third and last one is a bi-level program, whose second level is a min–max problem, the max part being actually a maximum flow formulation. Since this linear programming formulation is well-known to have a totally unimodular constraint matrix, one can "forget" the integrality constraints, replace this linear program by its dual one, and obtain a min–min problem, which is easily seen to be equivalent to a simple min problem. Then, this new bi-level program can be solved by considering the convex hull corresponding to the feasible integral solutions of the problem at the second level, and progressively generating (by using a constraint generation procedure) the constraints associated with the extreme points of this convex hull. The subproblem used to generate these constraints is hard but far smaller than the original one, and in the end this formulation reveals to be the most efficient one. Moreover, it is almost oblivious to the number of edges that can fail, and hence, unlike the two previous ones, it can be used to solve the problem efficiently even when several edges can fail (instead of one only).

Chapter 6

Conclusions and perspectives

Throughout this report, several partial conclusions on some general questions about families of packing and covering problems in graphs have been reached and highlighted. Some are specific to one or two of these families, others are common to all of them. For instance, given some "basic" problem in one of the families of covering problems, one of these major general questions is: *what kind of packing constraints are we allowed to add to the basic problem in order to ensure that we stay sufficiently close to it*? Here, "staying close" would simply mean "keeping most of its good properties". Another related one would be to add packing constraints to the basic problem in order to be sufficiently "far" from it.

Concerning integral multiflows and multicuts, the number of elements (i.e., paths) to be packed or covered can be exponential in the size of the input graph. Even restricting the number of source-sink pairs is not sufficient to change this fact, although this sometimes does make MinMC easier, and sometimes not (see Section 2.3). Similarly, adding constraints such as the ones considered in Section 2.4 (cardinality constraints, or requiring only a partial covering) does not always make the problem harder, or easier. The results from these two sections are thus a first step towards understanding which types of constraints lead to harder problems, and which do not.

Another relevant question is: *can we find necessary and/or sufficient conditions for the ratio between the optimal values of* MinMC *and* MaxIMF *to be* $O(1)$*, or at least small enough*? This question has been studied in Section 2.2, but, again, only partial answers are known. Quite recently, two very nice papers involving Julia Chuzhoy and David Kim led, on the one hand, to a slightly improved upper bound for this ratio in planar graphs, and, on the other hand, to an improved inapproximability bound for MaxIMF, also in planar graphs (the best lower bound previously known was **APX**-hardness) [21, 22]. However, the gap between these two bounds remains significant, meaning that there is still a lot of room for improvement.

More details on nine other problems in this field of research, which were mentioned as open and particularly worth studying in [4], and which have not all been settled, can be found in Appendix A.

Concerning coloring problems, and their links to discrete tomography, the complexity of the generalization of the basic problem described by Ryser that uses three colors instead of two remained open for a long time, before Dürr *et al.* were finally able to settle it [33] (see Section 3.1). However, this case represents a kind of boundary case: with two colors, the problem is easy; with three or more colors, it becomes hard. So, the general question we really want to fully answer is: *when there are three colors, can we characterize the kind of structures that we can impose on the solutions we want to obtain, in order to make the problem easy*? Sections 3.2 and 3.3 provide partial answers (even for proper colorings), but we are still far from a characterization.

Concerning d-transversal and blocker problems, the number of elements (i.e., optimal solutions to a graph optimization problem) to be covered can also be exponential in the size of the input graph. This is one of the reasons why proving that such a problem belongs to **NP** or not may be a hard task, depending on the graph parameter that is considered. Actually, the fact that the nature of a d-transversal problem is heavily influenced by the graph parameter it is related to also implies that there is not really a "basic" problem in this case. Therefore, it is hard to identify general properties or ideas common to all d-transversal problems (apart from the hardness of relating them to **NP**). In Sections 4.2 to 4.4, structural results are provided for several special cases, illustrating for instance the fact that switching from regular bipartite graphs to *nearly* regular bipartite graphs (such as the rectilinear grids) can strongly impact the structure of optimal solutions.

A natural way of looking for general ideas, which may be good candidates for being common to all d-transversal problems, would be to consider solving these problems with approximation (instead of exact) algorithms. This may still be hard, especially when problems are not known to be in **NP**, but this is worth trying as this seems like a promising lead that, to the best of our knowledge, was not studied yet (although some papers did study the approximability of related, but easier, problems [69]).

Finally, concerning problems related to Steiner trees and networks, the ones that have been studied in Chapter 5 either stay close to the classical Steiner tree problem (for instance, when limiting the number of diffusing nodes) or not (for instance, when limiting the number of branching nodes, or when adding capacities, except in very special cases). Indeed, in some of these variants, it becomes necessary to require that the feasible solutions are trees (otherwise, even optimal solutions may not be trees), and sometimes there does not even exist feasible solutions.

Again, the initial question was: *can one exhibit natural sufficient (either necessary or not) conditions on the packing constraints to be added in order to ensure that the problem obtained stays close enough to the classical Steiner tree problem*? The results presented in Sections 5.2 and 5.3 provide some partial answers, and should be completed in this perspective.

Another aspect that has been studied (see Section 5.4) was the impact of a failure on a static multicast routing, i.e., on a set of Steiner trees, or on a Steiner network. But what about the case where, whenever one edge fails, one wants to ensure, at minimum cost, that there still exists a routing using a *single* Steiner tree, while not violating any capacity constraint?

This is different both from using a Steiner network (in which the data that must reach each terminal can be routed along several paths) and from considering a static routing on a set of Steiner trees, and this setting also generalizes the one considered (and fully characterized) in Section 5.3, which would correspond to the case where no edge can fail.

Moreover, this can make things considerably easier in some situations. For instance, if the graph represents a network used for collecting or distributing electrical power, taking into account the physics equations corresponding to the way the energy is routed inside the network amounts to adding non linear constraints (known as "load-flow equations", or "power-flow equations") to the model. However, if the energy is assumed to be routed *along a Steiner tree*, then it can be shown that these non linear equations simply turn into classical flow conservation and capacity constraints, which are linear constraints. This is also a particularly relevant and challenging way of combining covering problems with reliability issues, as well as relating covering problems with additional capacity (i.e., packing) constraints to the design of power distribution (or power collection) networks.

The last two open problems related to Steiner trees that should once again be mentioned are actually long-standing open questions. First, can we design better approximation algorithms for the minimum rooted Steiner tree problem in directed graphs? Indeed, lower and upper bounds are known for the approximability of this problem, but the gap between them remains significant. Second, can we solve in polynomial time the only case left as open in Section 5.3, or is it **NP**-hard? In other words, what is the complexity of the problem where one wants to determine vertex-disjoint paths of minimum total length linking $O(1)$ source-sink pairs in an undirected graph? And what happens when the graph is a planar directed graph?

Bibliography

[1] U. Adamy, C. Ambuehl, R. Sai Anand, and T. Erlebach. Call Control in Rings. Proceedings ICALP, LNCS 2380 (2002) 788–799.

[2] E. Arkin, N. Guttmann-Beck, and R. Hassin. The (K, k)-Capacitated Spanning Tree Problem. Discrete Optimization 9 (2012) 258–266.

[3] C. Bazgan, S. Toubaline, and Z. Tuza. The most vital nodes with respect to independent set and vertex cover. Discrete Applied Mathematics 159 (2011) 1933–1946.

[4] C. Bentz. Résolution exacte et approchée de problèmes de multiflot entier et de multicoupe : algorithmes et complexité (in French). PhD thesis, CEDRIC-CNAM (2006).

[5] D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, and J. Obdrzalek. The DAG-Width of Directed Graphs. Journal of Combinatorial Theory, Series B 102 (2012) 900–923.

[6] H.L. Bodlaender and F.V. Fomin. Equitable colorings of bounded treewidth graphs. Theoretical Computer Science 349 (2005) 22–30.

[7] F. Bonomo, S. Mattia, and G. Oriolo. Bounded coloring of co-comparability graphs and the pickup and delivery tour combination problem. Theoretical Computer Science 412 (2011) 6261–6268.

[8] G. Călinescu, C.G. Fernandes, and B. Reed. Multicuts in unweighted graphs and digraphs with bounded degree and bounded tree-width. Journal of Algorithms 48 (2003) 333–359.

[9] D. Chalu. Résolution efficace de problèmes de multicoupes partielles (in French). MPRI Master's thesis, LRI (2011).

[10] M. Charikar, C. Chekuri, T.-Y. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation Algorithms for Directed Steiner Problems. Journal of Algorithms 33 (1999) 73–91.

[11] S. Chawla, R. Krauthgamer, R. Kumar, Y. Rabani, and D. Sivakumar. On the Hardness of Approximating Multicut and Sparsest-Cut. Computational Complexity 15 (2006) 94–114.

[12] C. Chekuri, G. Naves, and B. Shepherd. Maximum Edge-Disjoint Paths in k-Sums of Graphs. Proceedings ICALP (2013) 328–339.

[13] C. Chekuri, S. Khanna, and B. Shepherd. Edge-Disjoint Paths in Planar Graphs. Proceedings FOCS (2004) 71–80.

[14] C. Chekuri, S. Khanna, and B. Shepherd. Multicommodity flow, well-linked terminals, and routing problems. Proc. STOC (2005) 183–192.

[15] C. Chekuri, S. Khanna, and B. Shepherd. A Note on Multiflows and Treewidth. Algorithmica 54 (2009) 400–412.

[16] C. Chekuri, S. Khanna, and B. Shepherd. Edge-Disjoint Paths in Planar Graphs with Constant Congestion. SIAM J. Comput. 39 (2009) 281–301.

[17] X. Cheng, Y. Li, D.-Z. Du, and H.Q. Ngo. Steiner Trees in Industry. Chapter in *Handbook of Combinatorial Optimization*, pp. 193–216. D.-Z. Du and P.M. Pardalos (eds.), Springer (2005).

[18] K. Cheung and K. Harvey. Revisiting a simple algorithm for the planar multiterminal cut problem. Oper. Res. Lett. 38 (2010) 334–336.

[19] M. Chrobak and C. Dürr. Reconstructing polyatomic structures from discrete X-rays: **NP**-completeness proof for three atoms. Theoretical Computer Science 259 (2001) 81–98.

[20] J. Chuzhoy and S. Khanna. Hardness of cut problems in directed graphs. Proceedings STOC (2006) 527–536.

[21] J. Chuzhoy, D. Kim, and S. Li. Improved approximation for node-disjoint paths in planar graphs. Proceedings STOC (2016) 556–569.

[22] J. Chuzhoy, D. Kim, and R. Nimavat. New hardness results for routing on disjoint paths. Proceedings STOC (2017) 86–99.

[23] J. Chuzhoy and S. Li. A Polylogarithmic Approximation Algorithm for Edge-Disjoint Paths with Congestion 2. J. of the ACM 63 (2016) 1–51.

[24] É. Colin de Verdière. Multicuts in Planar and Bounded-Genus Graphs with Bounded Number of Terminals. Algorithmica 78 (2017) 1206–1224.

[25] M.-C. Costa, D. de Werra, and C. Picouleau. Using graphs for some discrete tomography problems. Disc. Appl. Math. 154 (2006) 35–46.

[26] M.-C. Costa, D. de Werra, and C. Picouleau. Minimum d-blockers and d-transversals in graphs. J. Comb. Optim. 22 (2011) 857–872.

[27] M.-C. Costa, L. Létocart, and F. Roupin. Minimal multicut and maximal integer multiflow: A survey. European Journal of Operational Research 162 (2005) 55–69.

[28] E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. SIAM Journal on Computing 23 (1994) 864–894.

[29] N. Derhy. Résolution pratique de problèmes de multicoupes dans les graphes non orientés (in French). Master's thesis, CEDRIC-CNAM (2005).

[30] R. Downey and M. Fellows. *Parameterized Complexity*. Springer, New York (1999).

[31] S. Dreyfus and R. Wagner. The Steiner problem in graphs. Networks 1 (1971) 195–207.

[32] G. Duan and Y. Yu. Distribution System Optimization by an Algorithm for Capacitated Steiner Tree Problems with Complex flows and Arbitrary Cost Functions. International Journal of Electrical Power and Energy Systems 25 (2003) 515–523.

[33] C. Dürr, F. Guiñez, and M. Matamala. Reconstructing 3-Colored Grids from Horizontal and Vertical Projections is **NP**-Hard: A Solution to the 2-Atom Problem in Discrete Tomography. SIAM Journal on Discrete Mathematics 26 (2012) 330–352.

[34] T. Erlebach. Approximation algorithms and complexity results for path problems in trees of rings. Proc. MFCS, LNCS 2136 (2001) 351–362.

[35] U. Feige. A threshold of $\ln n$ for approximating set cover. Journal of the ACM 45 (1998) 634–652.

[36] F. Fomin, F. Grandoni, D. Kratsch, D. Lokshtanov, and S. Saurabh. Computing optimal steiner trees in polynomial space. Algorithmica 65 (2013) 584–604.

[37] M. Formann, D. Wagner, and F. Wagner. Routing through a dense channel with minimum total wire length. Journal of Algorithms 15 (1993) 267–283.

[38] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. Theoretical Computer Science 10 (1980) 111–121.

[39] A. Frank. Disjoint paths in a rectilinear grid. Combinatorica 2 (1982) 361–371.

[40] A. Frank. Edge-disjoint paths in planar graphs. Journal of Combinatorial Theory, Series B 39 (1985) 164–178.

[41] M. Garey and D. Johnson. *Computers and Intractability: a Guide to the Theory of* ***NP****-completeness*. Freeman & Co., San Fransisco (1979).

[42] N. Garg, V. Vazirani, and M. Yannakakis. Multiway Cuts in Directed and Node Weighted Graphs. Proceedings ICALP (1994) 487–498.

[43] N. Garg, V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. Algorithmica 18 (1997) 3–20.

[44] G. Gottlob and S. Tien Lee. A logical approach to multicut problems. Information Processing Letters 103 (2007) 136–141.

[45] S. Gravier, D. Kobler, and W. Kubiak. Complexity of list coloring problems with a fixed total number of colors. Discrete Applied Mathematics 117 (2002) 65–79.

[46] J. Guo, F. Hüffner, E. Kenar, R. Niedermeier, and J. Uhlmann. Complexity and exact algorithms for vertex multicut in interval and bounded treewidth graphs. Europ. J. of Oper. Res. 186 (2008) 542–553.

[47] D. Hartvigsen. The planar multiterminal cut problem. Discrete Applied Mathematics 85 (1998) 203–222.

[48] A. Hertz, O. Marcotte, A. Mdimagh, M. Carreau, and F. Welt. Optimizing the Design of a Wind Farm Collection Network. Information Systems and Operational Research 50 (2012) 95–104.

[49] P. Hunter and S. Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. Theoret. Comput. Sci. 399 (2008) 206–219.

[50] F. Hwang, D. Richards, and P. Winter. *The Steiner Tree Problem*. Annals of Discrete Mathematics 53, Springer (1992).

[51] T. Johnson, N. Robertson, P. Seymour, and R. Thomas. Directed Treewidth. Journal of Combinatorial Theory, Series B 82 (2001) 138–154.

[52] R. Jothi and B. Raghavachari. Approximation Algorithms for the Capacitated Minimum Spanning Tree Problem and Its Variants in Network Design. ACM Transactions on Algorithms 1–2 (2005) 265–282.

[53] M. Kaufmann and K. Mehlhorn. Routing problems in grid graphs. In B. Korte, L. Lovász, H. Prömel, A. Schrijver (eds.): Algorithms and Combinatorics, Vol. 9, *Paths, Flows and VLSI-Layout*. Springer (1990).

[54] J. Kleinberg. Approximation algorithms for disjoint paths problems. PhD thesis, MIT (1996).

[55] T. Kloks. Treewidth, Computations and Approximations. Lecture Notes in Computer Science 842 (1994).

[56] Y. Kobayashi and C. Sommer. On Shortest Disjoint Paths in Planar Graphs. Discrete Optimization 7 (2010) 234–245.

[57] P. Le Bodic. Variantes non standard de problèmes d'optimisation combinatoire (in French). PhD thesis, LRI-Univ. Paris-Sud (2012).

[58] T. Lefebvre. Optimization of information flows in telecommunication networks. PhD thesis, CEDRIC-CNAM (2016).

[59] D. Marx. Eulerian disjoint paths problem in grid graphs is **NP**-complete. Discrete Applied Mathematics 143 (2004) 336–341.

[60] J. Nederlof. Fast polynomial-space algorithms using inclusion-exclusion. Algorithmica 65 (2013) 868–884.

[61] H. Okamura and P. Seymour. Multicommodity flows in planar graphs. Journal of Combinatorial Theory, Series B 31 (1981) 75–81.

[62] T. Ridremont. Optimisation robuste du câblage dans les parcs d'énergie renouvelable (in French). Master's thesis, CEDRIC-CNAM (2015).

[63] G. Robins and A. Zelikovsky. Improved Steiner tree approximation in graphs. Proceedings SODA (2000) 770–779.

[64] H.J. Ryser. Combinatorial properties of matrices of zeros and ones. Canadian J. Math. 9 (1957) 371–377.

[65] L. Seguin-Charbonneau and B. Shepherd. Maximum Edge-Disjoint Paths in Planar Graphs with Congestion 2. Proceedings FOCS (2011) 200–209.

[66] V. Vazirani. *Approximation algorithms*. Springer (2001).

[67] M. Yannakakis, P. Kanellakis, S. Cosmadakis, and C. Papadimitriou. Cutting and partitioning a graph after a fixed pattern. Proceedings ICALP, Lecture Notes in Computer Science 154 (1983) 712–722.

[68] W.-C. Yeh. A simple algorithm for the planar multiway cut problem. Journal of Algorithms 39 (2001) 68–77.

[69] R. Zenklusen. Matching interdiction. Discrete Applied Mathematics 158 (2010) 1676–1690.

Personal publications

[BB12] Cédric Bentz and Pierre Le Bodic. On the complexity of partial colored multiterminal cut problems. In *Proceedings 2nd International Symposium on Combinatorial Optimization, ISCO 2012, Athens, Greece, April 17–21*, 2012.

[BBPR15] Cristina Bazgan, Cédric Bentz, Christophe Picouleau, and Bernard Ries. Blockers for the stability number and the chromatic number. *Graphs and Combinatorics*, 31(1):73–90, 2015.

[BCDR09] Cédric Bentz, Marie-Christine Costa, Nicolas Derhy, and Frédéric Roupin. Cardinality constrained and multicriteria (multi)cut problems. *J. Discrete Algorithms*, 7(1):102–111, 2009.

[BCdW+08] Cédric Bentz, Marie-Christine Costa, Dominique de Werra, Christophe Picouleau, and Bernard Ries. On a graph coloring problem arising from discrete tomography. *Networks*, 51(4):256–267, 2008.

[BCdW+11] Cédric Bentz, Marie-Christine Costa, Dominique de Werra, Christophe Picouleau, and Bernard Ries. Minimum d-transversals of maximum-weight stable sets in trees. *Electronic Notes in Discrete Mathematics*, 38:129–134, 2011.

[BCH16] Cédric Bentz, Marie-Christine Costa, and Alain Hertz. On the edge capacitated steiner tree problem. *CoRR*, abs/1607.07082, 2016.

[BCLR09] Cédric Bentz, Marie-Christine Costa, Lucas Létocart, and Frédéric Roupin. Multicuts and integral multiflows in rings. *European Journal of Operational Research*, 196(3):1251–1254, 2009.

[BCP+09] Cédric Bentz, Marie-Christine Costa, Christophe Picouleau, Bernard Ries, and Dominique de Werra. Degree-constrained edge partitioning in graphs arising from discrete tomography. *J. Graph Algorithms Appl.*, 13(2):99–118, 2009.

[BCP+12] Cédric Bentz, Marie-Christine Costa, Christophe Picouleau, Bernard Ries, and Dominique de Werra. d-transversals of stable sets and vertex covers in weighted bipartite graphs. *J. Discrete Algorithms*, 17:95–102, 2012.

[BCPR17] Cédric Bentz, Marie-Christine Costa, Pierre-Louis Poirion, and Thomas Ridremont. Formulations for designing robust networks. an application to wind power collection. In *Proceedings 8th International Network Optimization Conference, INOC 2017, Lisboa, Portugal, February 26–28,*, page (to appear in Electronic Notes in Discrete Mathematics), 2017.

[BCPZ07] Cédric Bentz, Marie-Christine Costa, Christophe Picouleau, and Maria Zrikem. The shortest multipaths problem in a capacitated dense channel. *European Journal of Operational Research*, 178(3):926–931, 2007.

[BCR] Cédric Bentz, Marie-Christine Costa, and Frédéric Roupin. Erratum to: "C. Bentz, M.-C. Costa, F. Roupin. Maximum integer multiflow and minimum multicut problems in two-sided uniform grid graphs [Journal of Discrete Algorithms 5 (2007) 36-54]". Technical Report CEDRIC-07-4068, CEDRIC laboratory, CNAM-Paris, France (url: https://cedric.cnam.fr/index.php/publis/article/BCR07a).

[BCR07] Cédric Bentz, Marie-Christine Costa, and Frédéric Roupin. Maximum integer multiflow and minimum multicut problems in two-sided uniform grid graphs. *J. Discrete Algorithms*, 5(1):36–54, 2007.

[BCR13] Cédric Bentz, Denis Cornaz, and Bernard Ries. Packing and covering with linear programming: A survey. *European Journal of Operational Research*, 227(3):409–422, 2013.

[BEGL14] Cédric Bentz, Sourour Elloumi, Éric Gourdin, and Thibaut Lefebvre. Network coding for survivable multicast video streaming networks. In *Proceedings 6th International Workshop on Reliable Networks Design and Modeling, RNDM 2014, Barcelona, Spain, November 17–19,*, 2014.

[BEGL15] Cédric Bentz, Sourour Elloumi, Éric Gourdin, and Thibaut Lefebvre. On the minimum convex cost multicast flow problem. In *Proceedings 7th International Network Optimization Conference, INOC 2015, Warsaw, Poland, May 18–20,*, pages 1–8, 2015.

[Ben05] Cédric Bentz. Edge disjoint paths and max integral multiflow/min multicut theorems in planar graphs. *Electronic Notes in Discrete Mathematics*, 22:55–60, 2005.

[Ben06] Cédric Bentz. The maximum integer multiterminal flow problem. In Marina L. Gavrilova, Osvaldo Gervasi, Vipin Kumar, Chih Jeng Kenneth Tan, David Taniar, Antonio Laganà, Youngsong Mun, and Hyunseung Choo, editors, *Computational Science and Its Applications - ICCSA 2006, International Conference, Glasgow, UK, May 8-11, 2006, Proceedings, Part III*, volume 3982 of *Lecture Notes in Computer Science*, pages 738–747. Springer, 2006.

[Ben07] Cédric Bentz. The maximum integer multiterminal flow problem in directed graphs. *Oper. Res. Lett.*, 35(2):195–200, 2007.

[Ben08] Cédric Bentz. On the complexity of the multicut problem in bounded tree-width graphs and digraphs. *Discrete Applied Mathematics*, 156(10):1908–1917, 2008.

[Ben09a] Cédric Bentz. Disjoint paths in sparse graphs. *Discrete Applied Mathematics*, 157(17):3558–3568, 2009.

[Ben09b] Cédric Bentz. New results on planar and directed multicuts. *Electronic Notes in Discrete Mathematics*, 34:207–211, 2009.

[Ben09c] Cédric Bentz. A simple algorithm for multicuts in planar graphs with outer terminals. *Discrete Applied Mathematics*, 157(8):1959–1964, 2009.

[Ben11] Cédric Bentz. On the hardness of finding near-optimal multicuts in directed acyclic graphs. *Theor. Comput. Sci.*, 412(39):5325–5332, 2011.

[Ben12] Cédric Bentz. A polynomial-time algorithm for planar multicuts with few source-sink pairs. In Dimitrios M. Thilikos and Gerhard J. Woeginger, editors, *Parameterized and Exact Computation - 7th International Symposium, IPEC 2012, Ljubljana, Slovenia, September 12-14, 2012. Proceedings*, volume 7535 of *Lecture Notes in Computer Science*, pages 109–119. Springer, 2012.

[Ben17a] Cédric Bentz. An FPT algorithm for planar multicuts with sources and sinks on the outer face. *CoRR*, abs/1708.05903, 2017.

[Ben17b] Cédric Bentz. Weighted and locally bounded list-colorings in split graphs, cographs, and partial k-trees. *CoRR*, abs/1709.05000, 2017.

[BP09] Cédric Bentz and Christophe Picouleau. Locally bounded k-colorings of trees. *RAIRO - Operations Research*, 43(1):27–33, 2009.

[RBP+10] Bernard Ries, Cédric Bentz, Christophe Picouleau, Dominique de Werra, Marie-Christine Costa, and Rico Zenklusen. Blockers and transversals in some subclasses of bipartite graphs: When caterpillars are dancing on a grid. *Discrete Mathematics*, 310(1):132–146, 2010.

[WWBB13] Dimitri Watel, Marc-Antoine Weisser, Cédric Bentz, and Dominique Barth. Steiner problems with limited number of branching nodes. In Thomas Moscibroda and Adele A. Rescigno, editors, *Structural Information and Communication Complexity - 20th International Colloquium, SIROCCO 2013, Ischia, Italy, July 1-3, 2013, Revised Selected Papers*, volume 8179 of *Lecture Notes in Computer Science*, pages 310–321. Springer, 2013.

[WWBB14] Dimitri Watel, Marc-Antoine Weisser, Cédric Bentz, and Dominique Barth. Directed steiner tree with branching constraint. In Zhipeng Cai, Alex Zelikovsky, and Anu G. Bourgeois, editors, *Computing and Combinatorics - 20th International Conference, COCOON 2014, Atlanta, GA, USA, August 4-6, 2014. Proceedings*, volume 8591 of *Lecture Notes in Computer Science*, pages 263–275. Springer, 2014.

[WWBB15] Dimitri Watel, Marc-Antoine Weisser, Cédric Bentz, and Dominique Barth. An FPT algorithm in polynomial space for the directed steiner tree problem with limited number of diffusing nodes. *Inf. Process. Lett.*, 115(2):275–279, 2015.

[WWBB16] Dimitri Watel, Marc-Antoine Weisser, Cédric Bentz, and Dominique Barth. Directed steiner trees with diffusion costs. *J. Comb. Optim.*, 32(4):1089–1106, 2016.

[ZRP+09] Rico Zenklusen, Bernard Ries, Christophe Picouleau, Dominique de Werra, Marie-Christine Costa, and Cédric Bentz. Blockers and transversals. *Discrete Mathematics*, 309(13):4306–4314, 2009.

Appendix A: Open problems related to integral multiflow and multicut problems

In my PhD thesis [4], defended in 2006 and written in French, I provided a list of ten open problems related to integral multiflows and multicuts, which I considered as major ones in this research field. For the sake of completeness, I shall include them again in this report, in the same order as in [4]:

1. What is the complexity of MaxEDP in (undirected or directed) planar graphs, when the number of source-sink pairs is $O(1)$?
2. What is the complexity of MaxEDP in graphs of bounded tree-width, when the number of source-sink pairs is $O(1)$?
3. What is the complexity of MaxEDP in graphs of bounded tree-width, when the maximum degree is $O(1)$?
4. Does there exist an $O(1)$-approximation algorithm for MaxEDP in graphs of bounded tree-width?
5. Can we find better approximation algorithms for MaxEDP in undirected graphs? And in undirected planar graphs?
6. Does there exist an $O(1)$-approximation algorithm for the multiterminal variant of MaxIMF in directed graphs?
7. What is the complexity of the multiterminal variant of MinMC in planar graphs, when the terminals lie on the boundary of $O(1)$ faces?
8. Does there exist a **PTAS** for the multiterminal variant of MinMC in planar graphs?
9. Does the multiterminal variant of MinMC admit an algorithm that is **FPT** with respect to the number of terminals in planar graphs?
10. What is the complexity of MinMC with $O(1)$ source-sink pairs: (i) in directed acyclic graphs, and (ii) in undirected planar graphs?

Problem 1 also appeared as Problem GT04-3 in [Open problems. Chapter in *Graph Theory in Paris, Proceedings of a Conference in Memory of Claude Berge*, pp. 381–389. A. Bondy, J. Fonlupt, J.-L. Fouquet, J.-C. Fournier, and J.R. Alfonsin (eds.), Birkhäuser (2007)].

Since 2006, most of these problems have indeed been studied, and settled.

Problem 1 was proved to be **NP**-hard in [G. Naves. The hardness of routing two pairs on one face. Mathematical Programming 131 (2012) 49–69], even with only two sources and two sinks, all lying on the outer face.

Problem 4 was dealt with in [12], and the answer is yes.

Problem 5 was addressed in several ways, and, as mentioned in Chapter 6, Chuzhoy *et al.* recently managed to slightly improve the best upper and lower bounds known for approximating MaxEDP in undirected planar graphs [21, 22], but the gap between these bounds is far from being closed.

Problem 8 was settled in [M. Bateni, M. Hajiaghayi, P. Klein, and C. Mathieu. A polynomial-time approximation scheme for planar multiway cut. Proceedings SODA (2012) 639–655], and the answer is yes.

Problem 9 was studied in [D. Marx. A Tight Lower Bound for Planar Multiway Cut with Fixed Number of Terminals. Proceedings ICALP (2012) 677–688], and the answer is no (under the Exponential Time Hypothesis).

Finally, Problem 10(i) is **APX**-hard even with two source-sink pairs, as shown in [Ben11], while Problem 10(ii) is tractable, as shown in [24].

However, as far as I know, Problems 2, 3, 6 and 7 are still open.

Appendix B: List of supervised students

This appendix provides the list of all the PhD, post-doc, and Master students that I (co-)supervised since 2005, as well as the associated publications.

PhD students:

1. Co-advisor, with Marie-Christine Costa (ENSTA) and Alain Hertz (GERAD laboratory, Montreal), of Thomas Ridremont. This PhD started in September 2015, and focuses on the study of variants of the Steiner tree problem, where edges (or arcs) have capacity and/or can fail. (*Associated publication*: [BCPR17].)

2. Co-advisor, with Sourour Elloumi (ENSIIE) and Éric Gourdin (Orange Labs), of Thibaut Lefebvre. This PhD took place from 2012 to 2016, and was funded by a CIFRE PhD scholarship, in collaboration with Orange Labs. It focused on the study of the impact, on several routing strategies (Multicast, Network Coding), of link failures in a telecommunication network. (*Associated publications*: [BEGL14, BEGL15].)

3. Co-advisor, with Dominique Barth (UVSQ/PRiSM) and Marc-Antoine Weisser (Supélec) of Dimitri Watel. This PhD took place from 2011 to 2014, and was funded by the DIGITEO project APPAS. It focused on the design of efficient algorithms (exact or not) for solving variants of the Steiner tree problem in directed graphs (in particular when the number of nodes that can duplicate the data is limited). (*Associated publications*: [WWBB13, WWBB14, WWBB15, WWBB16].)

4. Co-advisor, with Abdel Lisser (LRI), of Pierre Le Bodic. His PhD thesis, whose topic was *"Non standard variants of discrete optimization problems"*, was defended on September 2012, and focused in particular on the study of partial cut problems, such as partial multicuts and partial multiterminal cuts. (*Associated publication*: [BB12].)

Post-doctoral student:

- Supervisor of Benoît Robillard, which was a post-doctoral student at the LRI during one year, in 2011-2012. Under my supervision, he worked on partial cut problems and d-blocker problems.

Master students:

1. Co-supervisor, with Daniel Porumbel (CNAM) and Marie-Christine Costa (ENSTA), of the Master 2 internship of Thomas Ridremont, which took place from March to September 2015, on the topic: *"Robust Steiner trees"*.

2. Co-supervisor, with Christophe Picouleau (CNAM), of the Master 2 internship of Kpotissan Adjetey-Bahun (a student from the MPRO – *Master Parisien de Recherche Opérationnelle*), which took place from March to August 2012, on the topic: *"A study of d-blockers of disjoint paths in graphs"*.

3. Supervisor of the Master 2 internship of Didier Chalu (a student from the MPRI – *Master Parisien de Recherche en Informatique*), which took place from March to July 2011, on the topic: *"Efficient algorithms for partial multicut problems"*.

4. Co-supervisor, with Frédéric Roupin (CNAM) and Marie-Christine Costa (CNAM), of the Master 2 internship of Nicolas Derhy, which took place from March to September 2005, on the topic: *"Multicuts with cardinality constraints"*. (*Associated publication*: [BCDR09].)

Abstract

Title: *Exact and approximation algorithms for some packing and covering problems in graphs.*

In this HDR thesis, we consider several families of packing and covering problems in graphs, namely:

- Multicuts and integral multiflows, as well as some variants.
- Vertex and edge colorings with cardinality constraints, most of them being linked to a basic problem in discrete tomography.
- Blockers and d-transversals for some classical graphs parameters: the matching number, the stability number, and the chromatic number.
- Steiner trees and networks, as well as variants or generalizations (which are obtained, for instance, by adding edge or arc capacities, or by assuming that some edges or arcs can fail).

We study the computational complexity of all these problems, design approximation algorithms for some hard cases (or prove that such algorithms cannot exist), and exhibit several relevant parameters that allow to prove the fixed-parameter tractability, with respect to such parameters, of some special cases (while other special cases are proved to be fixed-parameter intractable).