Technische Universität München

Fakultät für Mathematik

Lehrstuhl für Operations Research

Combinatorial Algorithms

for Covering and Scheduling Problems

Marilena Susan Leichter

Vollständiger Abdruck der von der Fakultät für Mathematik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 30.03.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Mathematik am 10.06.2021 angenommen.

Abstract

We study combinatorial optimization problems related to covering and scheduling problems. The first chapter considers the Minimum Hitting Set of Bundles problem, a natural generalization of the Hitting Set problem, that provides an abstract framework for several scheduling problems. We present a combinatorial approximation algorithm for the general problem, study the computational complexity of several special cases, and present polynomial-time algorithms. The second chapter is related to matroids, which are essential structures in combinatorial optimization. We show how to reduce a gammoid, a particular type of matroid, efficiently to its underlying structure and address the implications of our results on related covering and coloring problems on the intersection of matroids.

The third chapter deals with the Generalized Min Sum Set Cover problem, which was initially introduced as a theoretical framework for re-ranking web search results. We study the problem on special types of hypergraphs as well as with different choices of covering constraints, providing approximation and exact algorithms. In the fourth chapter, we focus on the Bipartite Flow Scheduling problem, which is closely related to Coflow Scheduling and models data transfers in processors. We explain the connection of Bipartite Flow Scheduling to other scheduling and covering problems and give an overview of existing results.

Zusammenfassung

Wir untersuchen kombinatorische Optimierungsprobleme, welche im Zusammenhang zu Covering- und Schedulingproblemen stehen. Das erste Kapitel betrachtet das Minimum Hitting Set of Bundles Problem, eine natürliche Verallgemeinerung des Hitting Set Problems, das ein abstraktes Konzept für mehrere Schedulingprobleme bietet. Wir stellen einen kombinatorischen Approximationsalgorithmus für das allgemeine Problem vor, untersuchen die Komplexität einiger Spezialfälle und präsentieren polynomielle Algorithmen. Das zweite Kapitel bezieht sich auf Matroide, die ein wichtiges Konzept in der kombinatorischen Optimierung darstellen. Wir zeigen wie man einen Gammoid, eine bestimmte Art Matroid, effizient auf seine zugrundeliegende Struktur reduzieren kann und gehen auf die Bedeutung unserer Ergebnisse für verwandte Covering- und Färbungsprobleme auf dem Schnitt von Matroiden ein.

Das dritte Kapitel beschäftigt sich mit dem Generalized Min Sum Set Cover Problem, das ursprünglich als theoretisches Framework für das Re-Ranking von Suchergebnissen eingeführt wurde. Wir untersuchen das Problem auf speziellen Hypergraphen und mit verschiedenen Coveringbedingungen und präsentieren Approximationsalgorithmen und exakte Algorithmen. Im vierten Kapitel konzentrieren wir uns auf das Bipartite Flow Scheduling Problem, das eng mit dem Coflow Scheduling Problem verwandt ist und den Datentransfer in Prozessoren modelliert. Wir erklären den Zusammenhang zwischen Bipartite Flow Scheduling und anderen Scheduling- und Coveringproblemen und geben einen Überblick über bestehende Forschungsergebnisse.

Acknowledgments

My Ph.D. would not have been the same without the numerous wonderful people who have accompanied and supported me throughout this journey.

First of all, I would like to thank my supervisor, Andreas S. Schulz, for his support and encouragement. Thanks for giving me the freedom to work on topics that I am passionate about, for our insightful discussions, and for giving me the opportunity to travel to various summer schools and conferences. Further, I would like to thank Silke Rolles, Benjamin Moseley, and Britta Peis for being members of my Ph.D. committee.

Research is even more fun when you can share your passion with others. This is a big thank you to my co-authors Alexander Eckl [36], Marinus Gottschau [49], Felix Happach [54], Anja Kirschbaum [36], Benjamin Moseley [76], Kirk Pruhs [76], and Kevin Schewior [36]. Thanks for all the fun whiteboard, Zoom, and Jamboard discussions, for cheering each other up during frustrating times, and for celebrating our results together!

I was lucky to be part of a wonderful group that made me feel welcome in our office at Karlstraße from day one. Thanks to Alexander, Alexandros, Anne, Benedikt, Clara, Daniel, Diogo, Felix, Haakon, Jannik, Katie, Kiruna, Marcus, Marinus, Matthias, Nicola, Simon, Themis, Ulf, and Yiannis, all of whom were or are still part of the Operations Research Group. I have always enjoyed our sushi Fridays, beergarden evenings, joint runs, ice cream breaks, and ski trips. In particular, thanks to Katie for being the best office mate I could have wished for. Thanks to Marinus for hours of chatting during our morning runs, hikes, and biking trips. Thanks to Clara for all the fun table tennis matches and for sweetening our time at the office with 'Lerchen' and homemade cakes. Thanks to Anne for all the help and support, to Felix for our morning coffees, and to Ulf for the chats and endless Kinderriegel-supply.

As a member of the interdisciplinary research training group AdONE, I had the chance to meet great researchers from the Department of Mathematics, the School of Management, and the Department of Computer Science. Thanks to everyone who is or has been involved in AdONE! Furthermore, I would like to thank everyone from ProLehre for offering fantastic courses on university teaching. Thanks for all the inspiring discussions on how to teach mathematics. Thanks to Eva, Felix G., Felix H., Franzi, Lilli, and Marinus for proofreading parts of this thesis, for catching typos, and for providing valuable feedback.

Finally, I would like to thank my partner, family, and friends for all the support throughout the last years, for providing a welcome distraction from work, and for always believing in me. Felix, thanks for encouraging me to pursue a Ph.D., for cheering me up, for keep telling me to be less hard on myself, and for listening to my endless stories about mathematical problems!

Marilena

Contents

Chapter 1

Introduction

Optimization is an essential part of our everyday lives. Nowadays, algorithms help us to find the fastest bus connection or the shortest bike route to our desired destination. They match us with the closest available Uber driver and rank Google search results for us in a meaningful way. We can use them to schedule MRI and CT appointments efficiently, optimize the route of freight transportation, and minimize the running time of data transfers in a processor.

For most of these optimization problems, we can easily compute feasible solutions. These could be potential bike routes or bus connections to take, possible pairs of drivers and customers, or ways to order a list of search results. However, just obtaining any feasible solution is often not satisfactory since we prefer some solutions over others. We might be interested in the fastest bus connection, the one with the fewest transitions, or maybe in one that involves the shortest walking distances. An objective function can model these preferences by mapping solutions to values, thus making them comparable. This gives rise to the notion of optimality - a feasible solution of maximum (or minimum) objective value. But can we always compute an optimal solution efficiently?

Optimization problems become challenging whenever there are more feasible solutions than a computer can test in a reasonable amount of time. However, in some of these cases, we are still able to determine an optimal solution efficiently, using a more sophisticated approach that takes advantage of the problem's structure. In the 1960s, Jack Edmonds showed that for some problems, there exist so-called polynomial-time algorithms that solve them efficiently. A well-known example is the Matching problem.

Unfortunately, this might not be true for all combinatorial optimization problems, as Stephen Cook and Richard Karp observed. The large majority of problems, the most famous among them being the Traveling Salesperson problem, seem to be significantly harder to solve. Whether polynomial-time algorithms for these so-called NP-hard problems exist is one of the field's big open problems.

Given an NP-hard optimization problem, we have two choices on how to proceed. We can solve it to optimality requiring exponential running time, which is only a feasible approach for solving very small, concrete examples. Alternatively, we can develop algorithms to solve them in polynomial-time, but not to optimality. In this case, we would like to have a worst-case guarantee on the quality of our solution. Polynomial-time algorithms that find such near-optimal solutions are called approximation algorithms.

This thesis discusses hardness results, polynomial-time algorithms, and approximation algorithms for covering and scheduling problems. Before giving a high-level idea of the topics considered, we briefly introduce the terms Set Cover and Scheduling.

Set Cover. In the canonical Set Cover problem we are given a set of n elements together with a collection of subsets. The goal is to choose as few sets as possible to cover all elements. In Figure 1.1, the elements are visualized by points, and we are given seven sets. In this example, a cover is given by the green, yellow and red set since their union covers all elements. It is optimal, since there exists no cover of size two.

Set Cover has many applications. For example, it can be used to model the following problem. Assume you need to form a team for checking and refueling an airplane between two flights. Each worker has individual skills and is trained for a specific set of tasks. Your goal is to choose the smallest group of workers that is skilled to execute all assignments. In this application, the points of Figure 1.1 represent the collection of tasks, and every worker corresponds to a set containing all tasks that he is trained for. An optimal team of size three is formed by the workers corresponding to the green, the yellow, and the red set.

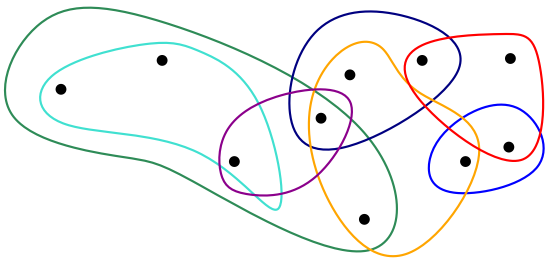


Fig. 1.1: Example of an instance of Set Cover. A cover of minimum cardinality is given by the green, yellow and red set.

Scheduling. To get some intuition for Scheduling problems, we consider a simple Single Machine Scheduling environment. We are given a set of n jobs that need to be executed on a single machine. Each job j has a processing time p_j, which means that it must be executed on the machine for p_j time units. Some jobs might be more urgent than others or simply more important, which we model by non-negative weights w_j. A schedule is, in this case, given by an ordering of the jobs in which they should be processed on the machine. A job's completion time in a specific schedule is denoted by C_j. Our goal is to find a schedule that minimizes the sum of weighted completion times.

In Figure 1.2, we are given four jobs with the respective processing times and weights, and we would like to find an ordering of the jobs that minimizes the sum of weighted completion times. Two exemplary schedules with corresponding objective values are depicted on the right. A simple approach, visualized in the first schedule, is to order the jobs with respect to increasing processing times. Job j_4 completes after time 1, job j_2 after time 3, job j_1 after time 6 and job j_3 after time 10. The sum of weighted completion times is, hence, given by $\sum w_j C_j = 1 \cdot 1 + 4 \cdot 3 + 1 \cdot 6 + 5 \cdot 10 = 69$.

Ordering jobs just by their processing times completely neglects their weights. A more advanced strategy, displayed in the second schedule, is to order the jobs with respect to decreasing ratios $\frac{w_j}{p_j}$. This leads to a strictly smaller objective value of 55. It turns out that 55 is the optimal value of this example, because the second schedule was constructed using a scheduling policy called Smith's rule, which is known to always construct an optimal schedule in this setting.

Scheduling problems arise in many applications with different settings, making it a broad class of problems with a wide variety of distinct characteristics. For example, when building a car, the dashboard has to be installed before the steering wheel, so not every ordering of the jobs is feasible. This can be modeled by precedence constraints. Furthermore, one may allow jobs to be preempted and continued later, or not all jobs might be available at time 0, which can be modeled by release dates. Moreover, we might have multiple machines or consider a different objective function.

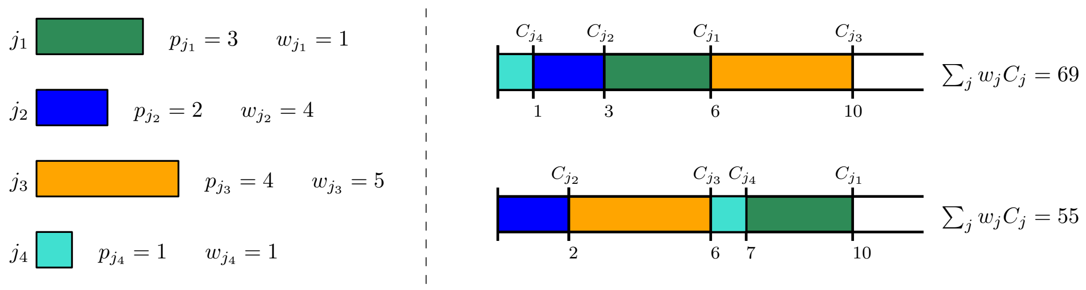


Fig. 1.2: Example of Single Machine Scheduling with 4 jobs. The sum of weighted completion times of the first schedule is given by 69 and of the second schedule by 55.

This thesis deals with four different topics. In the following, we give a high-level idea of the topics and motivate them from a practical perspective. While most of the problems are inspired by real-world applications, we focus in this thesis exclusively on theoretical results.

Minimum Hitting Set of Bundles Problem

Consider a Railway Corridor that is regularly used by freight and passenger trains. All trains follow a schedule that is adapted once a year, so in particular known far in advance. Regular track maintenance is essential to prevent train service disruptions due to technical failures. However, trains cannot operate on the affected section of the track during ongoing construction work. One advantage of disruptions by maintenance work is that, unlike unforeseeable disruptions caused by severe weather, they can be planned well in advance. Our goal is to schedule all maintenance jobs such that the impact on the train traffic is as small as possible.

The Minimum Hitting Set of Bundles problem (MHSB) mathematically abstracts this challenge. It is a generalization of the classical Set Cover problem and provides a framework not only for railway maintenance scheduling but also for several other scheduling problems. Chapter 2 of this thesis is dedicated to studying the Minimum Hitting Set of Bundles problem. We give a simple combinatorial algorithm for the general problem. Motivated by the observation that most applications have a common underlying structure, we study MHSB on so-called interval and 2-dimensional interval bundles.

Reduction of Gammoids

Many combinatorial optimization problems, amongst them the famous Traveling Salesperson problem, can be formulated as Matroid Intersection problems, making matroids one of the field's essential structures. A gammoid is a particular type of matroid, defined on a directed graph.

Chapter 3 shows how to reduce a gammoid efficiently to its underlying structure. The motivation to study reductions is based on the observation that optimization problems on simpler structures can be solved more efficiently. If the reduction maintains certain invariants, the obtained solution is also reasonable for the original problem. We also address these implications by elaborating on the impact of partition reductions for the Matroid Intersection Cover problem, which is a special case of the Set Cover problem.

Laminar Generalized Min Sum Set Cover Problem

Search engines like Google or Yahoo! have become an indispensable part of our modern life by making information instantly available almost everywhere. The number of search results for a particular keyword can be immense, such that we have to rely on a good ranking by the search engine. The main challenge is that even people searching for the exact same keyword might have very different priorities amongst the search results. This is not only caused by the fact that some words are ambivalent but also by our interest in different aspects of a keyword.

Additionally, the purpose of a search can vary. In an informational search, a user is interested in a wide range of search results, whereas, in a navigational search, one is only interested in finding specific information as quickly as possible.

The challenge of finding a good ranking can be mathematically modeled by the Generalized Min Sum Set Cover problem (Gmssc). In Chapter 4, we study Gmssc on special families and provide approximation and polynomial-time algorithms for certain choices of covering constraints.

Bipartite Flow Scheduling

As a result of the technological developments and breakthroughs of the past decades, large and complex datasets are available to us today. In order to benefit from these big datasets, for example when predicting natural disasters or investigating new medical treatment methods, we face a number of challenges. Amongst them is the question of how to store and transfer data in general. MapReduce, Spark and Dyrad are frameworks developed for the efficient processing of such large datasets. MapReduce, for example, does so by distributing the data to multiple systems, processing the tasks in parallel and aggregating the results afterwards.

This problem can be mathematically modeled by the Coflow Scheduling problem. In Chapter 5 of this thesis, we consider a special case of Coflow Scheduling that has been studied in the field under various different names. We summarize these results and elaborate on the connection of this problem to other optimization problems.

1.1 Preliminaries

We assume that the reader has profound knowledge of combinatorial optimization, particularly of graph theory, approximation algorithms, scheduling, and complexity theory. This section's primary purpose is to fix basic notation and highlight concepts used throughout this thesis. To make the individual chapters as self-contained as possible, we deferred most of the definitions that are only used locally to the corresponding sections.

1.1.1 Basic Definitions

We denote the non-negative integers, non-negative rational numbers, real numbers and non-negative real numbers by $\mathbb{N}$, $\mathbb{Q}_+$, $\mathbb{R}$, and $\mathbb{R}_+$. For some integer $n \in \mathbb{N}$, we use the notation $[n] := \{1, 2, \ldots, n\}$. The n-th **harmonic number** is denoted by $H(n) := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} = \sum_{k\in[n]} \frac{1}{k}$. Given a ground set Ω of elements, we denote the **power set** of Ω by $2^\Omega := \{X \mid X \subseteq \Omega\}$, and the cardinality of a set X by $|X|$. We write $X \subseteq Y$ if X is a subset of Y. Strict subsets are denoted by $X \subset Y$. Two subsets X, Y are disjoint if $X \cap Y = \emptyset$. We write $X \cup Y$ for the union of two sets and $X \cap Y$ for their intersection. The **symmetric difference** of two sets X and Y is denoted by $X \triangle Y := (X \cup Y) \setminus (X \cap Y)$.

The set family $\mathcal{X} := \{X_1, X_2, \ldots, X_l\}$ with $l \in \mathbb{N}$ is called a **partition** of Ω, if the sets in $\mathcal{X}$ are pairwise disjoint and $\Omega = \bigcup_{i\in[l]} X_i$. We refer to the sets of a partition as **parts**. We say that two sets $X, Y \subseteq \Omega$ **overlap** if, $X \cap Y \neq \emptyset$ and neither $X \subseteq Y$ nor $Y \subseteq X$. A set family $\mathcal{F} \subseteq 2^\Omega$ is **laminar** if and only if it is overlap-free, that is, no two sets in $\mathcal{F}$ overlap. Two sets X, Y **cross**, if they overlap and $X \cup Y \neq \Omega$. A cross-free family is a set family in which no two sets cross.

A function $f : 2^\Omega \to \mathbb{R}$ is called **submodular** if for all $X, Y \subseteq \Omega$

$$f(X \cup Y) + f(X \cap Y) \leq f(X) + f(Y).$$

1.1.2 Complexity and Algorithms

The complexity class **P** denotes all decision problems that can be solved in polynomial time. By **NP** we denote the complexity class of decision problems that can be solved in non-deterministic polynomial time. We say that a decision problem is **NP-complete** if it is in NP and all other problems in NP polynomially reduce to it. An optimization problem is **NP-hard** if its decision version is NP-complete.

Cook [31] showed that the Boolean Satisfiability problem is NP-complete. Karp [66] built upon this result by presenting his famous list of *Karp's 21 NP-complete problems*. Three of them play a role in this thesis, namely the Set Cover problem (Hitting Set problem), the Vertex Cover problem, and the Graph Coloring problem. The question of whether P=NP is one of the major open questions of the field and one of the seven

Millennium Problems by the Clay Mathematics Institute [21]. Assuming P$\neq$NP, there is no hope of finding a polynomial-time algorithm that solves an NP-hard problem to optimality. Instead, we focus on polynomial-time approximation algorithms. We say that an algorithm is an **α-approximation algorithm** with $\alpha \geq 1$ for an optimization problem $\mathcal{P}$ with non-negative weights if for all instances $I \in \mathcal{P}$

$$\frac{1}{\alpha}\; OPT(I) \leq ALG(I) \leq \alpha\; OPT(I),$$

where $OPT(I)$ denotes the optimal objective value of instance I, and $ALG(I)$ denotes the objective value of the solution obtained by the algorithm. Hence, a polynomial-time exact algorithm can also be seen as a 1-approximation algorithm.

The **Unique Games Conjecture** (UGC) was proposed by Khot [68]. If P$\neq$NP and the Unique Games Conjecture holds, then this has implications, for example, on the inapproximability of Set Cover.

Conjecture 1.1 (Unique Games Conjecture [68])
For any small constants ζ, δ, there exists a constant $c = c(\zeta, \delta)$ such that determining whether a unique 2-prover game with answers from a domain of size c has value at least $1 - \zeta$ or at most δ is NP-hard.

1.1.3 Graph Theory

We denote by $G = (V, E)$ a graph with vertex set V and edge set E. We might also refer to the vertex set or edge set of a graph G as $V(G)$ or $V(E)$, respectively. If not explicitly mentioned otherwise, G is a **simple graph**, that is, a graph without loops and parallel edges. A **multigraph** is a graph that may have multiple edges between the same pair of vertices. The degree of a vertex v in G is denoted by $\deg_G(v)$. To simplify notation, we omit the subscript whenever it is clear which graph we refer to. The **maximum degree** of a graph G is denoted by $\Delta(G) := \max_{v \in V(G)} \deg_G(v)$.

A graph $H = (V', E')$ is called subgraph of G if $V' \subseteq V$ and $E' \subseteq E$. For some $V' \subseteq V$, we denote the **induced subgraph** by $G[V']$. The union of two graphs $G = (V, E)$ and $H = (V, E')$ on the same vertex set is defined by $G \cup H := (V, E \cup E')$. Note that if the edge uv appears in both graphs, $G \cup H$ contains two copies of uv. This implies in particular that $G \cup H$ is not necessarily simple anymore. Whenever we are given a graph with non-negative **vertex weights**, this refers to a weight function $w : V \to \mathbb{R}_+$. Accordingly, we define non-negative **edge weights** by a weight function $w : E \to \mathbb{R}_+$.

Throughout this thesis, we consider various well-known graph classes. A **tree** is a connected graph without cycles and a **forest** is a collection of trees. We denote a **bipartite graph** with partitions A and B by $G = (A \cup B, E)$. A graph is bipartite if and only if it does not contain an odd cycle. We say that a graph G is **layered**, if there exists a partition of its vertices $V_1, \dots, V_l$ such that G only contains edges between subsequent layers V_i and V_{i+1}. The **line graph** of a graph G is denoted by $L(G)$. The vertex set of $L(G)$ is given by $E(G)$, and two vertices in $L(G)$ are adjacent whenever the corresponding edges in G are adjacent. See Figure 1.3 for an example. Line graphs have the important property that a so-called matching in a graph corresponds to an independent set in its line graph. This connection will be particularly useful in Chapter 5.

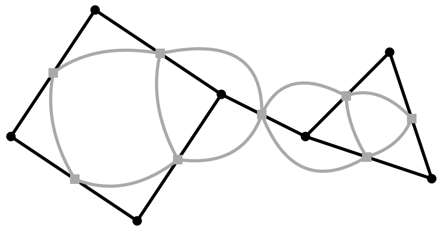


Fig. 1.3: Example of a graph (black) and its corresponding line graph (gray).

A **digraph** $D = (V, E)$ is a directed graph in which the edges are ordered pairs. The pair (v, w) corresponds to an edge from v to w. We disinguish between the in-degree and out-degree of a vertex. The **in-degree** of a vertex is denoted by $\deg^-(v)$ and the **out-degree** is denoted by $\deg^+(v)$. An **in-tree** is a directed tree with a designated sink t, such that there exists a directed path from every vertex to t. A collection of in-trees is called **in-forest**. In Chapter 4, we refer to a scheduling problem whose precedence graph is given by an in-forest. General digraphs play an important role in Chapter 3, where we study gammoids, which are matroids defined on digraphs.

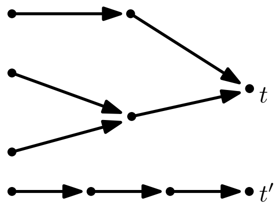


Fig. 1.4: Visualization of an in-forest, consisting of two in-trees with sinks t and t'. Note that $deg^+(v) = 1$ for all vertices besides the sinks.

A **hypergraph** $\mathcal{H} = (V, E)$ is a graph defined on the vertex set V. Every edge of a hypergraph is a subset of V, that is, $E \subseteq 2^V$. A hypergraph is called r**-uniform** if every edge has cardinality r. Note that simple graphs can also be interpreted as 2-uniform hypergraphs. A hypergraph is d**-regular** if every vertex is contained in precisely d edges. See Figure 1.5 for an example of a 3-uniform and a 2-regular hypergraph. A r**-partite hypergraph** is a hypergraph whose vertex set V can be partitioned into parts $V_1, \dots V_l$ such that every edge contains exactly one vertex from each part. Hence, every r-partite hypergraph is also r-uniform. A hypergraph is laminar if its edge set forms a laminar family and cross-free if its edge set forms a cross-free family.

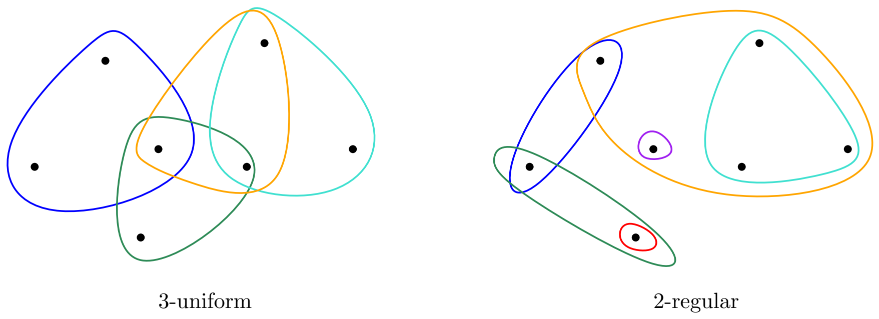


Fig. 1.5: Example of a 3-uniform and a 2-regular hypergraph.

Coloring

Given a graph $G = (V, E)$, a feasible (vertex) coloring with k colors is a function $c : V \to [k]$, such that no two adjacent vertices are colored with the same color, that is, $(u, v) \in E$ implies $c(u) \neq c(v)$. A graph is **k-colorable** if a feasible coloring with k colors exists. The smallest integer k such that G is k-colorable is called the **chromatic number** $\chi(G)$. A subset V' of vertices forms an **independent set** if no two vertices in V' are adjacent. By definition, the color classes of a feasible coloring form independent sets. Hence, an alternative way to think about a coloring is to interpret it as a partition of the vertex set into as few independent sets $C_1, \ldots, C_k$ as possible. This interpretation will be particularly useful in Chapter 3 when explaining the equivalence of Matroid Covering and Matroid Coloring. Determining the chromatic number of a graph is NP-hard. In fact, Graph Coloring is among Karp's 21 NP-complete problems [66].

In this thesis, we consider **Min Sum Coloring**. In the Min Sum Coloring problem, the goal is to find a feasible coloring, but instead of minimizing the number of colors, the task is to minimize the sum of colors, that is, minimize $\sum_{v \in V} c(v)$. Note that an optimal solution of Min Sum Coloring may require more colors than an optimal vertex coloring. A classic example, where an additional color decreases the sum of colors, is given in Figure 1.6.

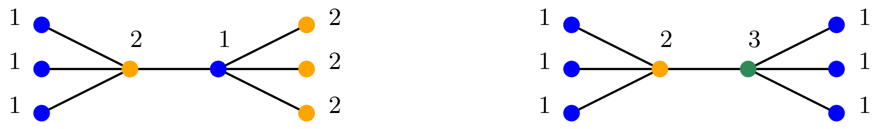


Fig. 1.6: Example of a 2-colorable graph. The Min Sum Coloring on the left has value 12, whereas the coloring with an additional color on the right has value 11.

In addition to vertex coloring, we also consider edge coloring. Given a graph $G = (V, E)$, a feasible edge coloring with k colors is a function $c : E \to [k]$, such that no two adjacent edges are colored with the same color. In other words, if $c(e) = c(e')$, then $e \cap e' = \emptyset$. A graph is **k-edge-colorable** if a feasible edge coloring with k colors exists. The smallest integer k such that G is k-edge-colorable is called the **chromatic index** $\chi'(G)$. Note that an edge coloring of a graph G corresponds to a vertex coloring of its line graph $L(G)$. The Min Sum Edge Coloring problem can be defined equivalently to the Min Sum Coloring problem above. We define the two problems in more detail in Chapter 5 and explain their connection to the considered scheduling problem.

Matchings

A **matching** M is a subset of edges, such that no two edges are adjacent to the same vertex. We say that a vertex v is **covered** by a matching M if there exists an edge in M that is incident to v. A matching is maximal if it cannot be extended to a larger matching by adding another edge. A **maximum** matching is the largest maximal matching of a graph. A **perfect** matching is a matching that covers all vertices of the graph. A path is called **M-alternating** if its edges are alternately in M and not in M. An M-alternating path of odd length whose end vertices are uncovered is called an **M-augmenting** path. Proposition 1.2 connects M-augmenting paths to the property of M being maximum.

Proposition 1.2 ([19, 87])
Let G be a graph and let M be a matching in G. M is maximum if and only if there is no M-augmenting path.

In this thesis, we only consider matchings in bipartite graphs. Matchings in bipartite graphs are closely related to the Maximum Flow problem. A maximum cardinality matching in a bipartite graph G can be found in $O(mn)$, where n denotes the number of vertices and m denotes the number of edges in G. For example, this can be done by applying the Ford-Fulkerson-Algorithm to the following graph. Direct all edges in G from A to B and add a source s and sink t to the graph. Introduce directed edges from

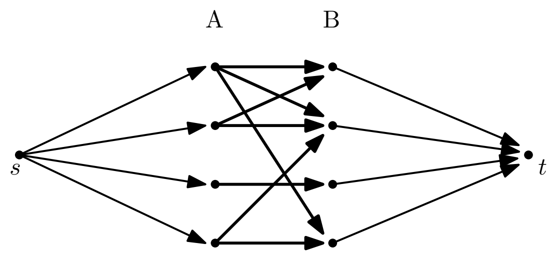


Fig. 1.7: Example to visualize the construction of the corresponding Maximum Flow problem.

s to all vertices in A and from all vertices in B to t. All edges have unit capacities. Now, any optimal maximum integral s-t-flow corresponds to a maximum cardinality matching. See Figure 1.7 for an example. In the weighted case, finding a maximum weight matching in a bipartite graph is equivalent to finding a minimum weight perfect matching. The latter can be solved in polynomial time by solving a Minimum Cost Flow problem on the corresponding flow graph.

Next, we consider the Vertex Cover Problem, another one of Karp's 21 NP-complete problems. It is defined as follows.

Definition 1.3 (Vertex Cover)
Consider a graph $G = (V, E)$ and a weight function $w : V \to \mathbb{Q}_+$. A vertex cover C is a subset of vertices such that every edge in E is incident to at least one vertex in C. The task is to find a vertex cover of minimum weight, that is, to minmize $\sum_{v \in C} w(v)$.

On bipartite graphs, it is closely related to maximum matchings and can be solved in polynomial time as the following proposition implies.

Proposition 1.4 (Kőnig's Theorem [92])
If G is bipartite, then the cardinality of a maximum matching equals the size of a minimum vertex cover.

1.1.4 Set Cover

Next, we formally introduce the Set Cover problem and the equivalent Hitting Set problem, which are both among Karp's 21 NP-complete problems [66]. We visualize why the two problems are, indeed, equivalent and briefly discuss two standard approximation algorithms.

Definition 1.5 (Set Cover)
Let Ω be the ground set of n elements and let $\mathcal{X} \subseteq 2^\Omega$ be a collection of subsets. A cover is a collection of subsets $\mathcal{C} \subseteq \mathcal{X}$, whose union equals Ω. The goal is to find a cover $\mathcal{C}$ of minimum cardinality.

The equivalent Hitting Set problem is defined as follows.

Definition 1.6 (Hitting Set)
Let Ω be the ground set of n elements and let $\mathcal{X} \subseteq 2^\Omega$ be a collection of subsets. A set $S \subseteq \Omega$ is called a hitting set if it contains at least one element of every set in $\mathcal{X}$. The goal is to find a hitting set S of minimum cardinality.

Consider an instance of Set Cover on n elements, given by a collection of subsets $\mathcal{X}$. The easiest way to see that the two problems are equivalent is to consider the following bipartite graph $G = (A \cup B, E)$. The vertex set A contains a vertex v_k for every element of the instance, and B contains a vertex v_X for every set of the family.

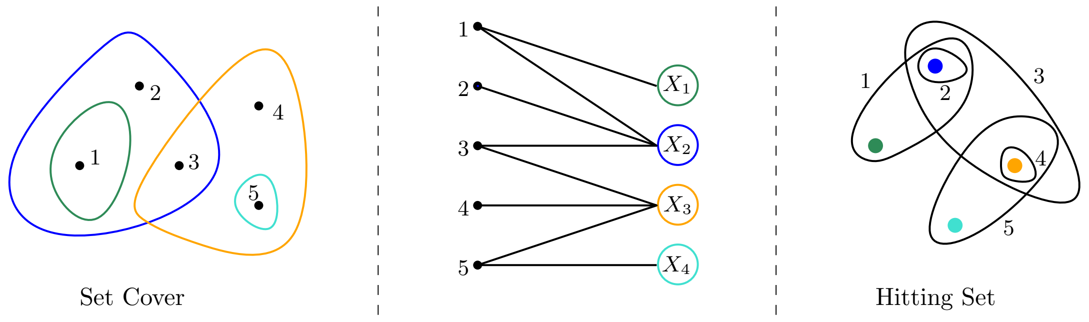


Fig. 1.8: Example of an instance of Set Cover on the left, the bipartite representation in the middle, and the equivalent instance of Hitting Set on the right.

There is an edge (v_i, v_S) in E whenever element i is contained in the set S. See Figure 1.8 for an example. We obtain the corresponding instance of Hitting Set by changing sets and elements in the following way. We introduce a set for every vertex v_k in A and an element for every vertex v_X in B. An element is contained in the corresponding set whenever there exists an edge in G. Hence, sets in the Set Cover instance correspond to elements in the Hitting Set instance and vice versa. In Figure 1.8, an instance of Set Cover is displayed on the left, the corresponding bipartite graph in the middle, and the equivalent instance of Hitting Set on the right. Note that the blue and the yellow set form a minimum set cover on the left. Equivalently, a minimum hitting set on the right is obtained by choosing the blue and the yellow element.

In the following, we present two standard approximation algorithms for Set Cover. Let n be the number of elements of an instance and let $f_{\max}$ be the maximum number of sets an element appears in. There exists a $H(n)$-approximation algorithm and a $f_{\max}$-approximation algorithm for Set Cover.

We start by discussing the Greedy algorithm with an approximation ratio of $H(n)$. In the unweighted version of Set Cover, Greedy chooses the set which covers the largest number of uncovered elements in each round. Johnson [65] and Lovász [79] analyzed Greedy for the unweighted version, and Chvátal [30] generalized the algorithm for weighted instances. In the weighted Set Cover problem, we are additionally given a non-negative weight function $w : \mathcal{X} \to \mathbb{R}_+$ and the task is to find a cover of minimum weight. The generalization of the Greedy algorithm for weighted instances works as follows. Choose the set that minimizes the ratio of weight over the number of newly covered elements. Greedy has an approximation ratio $H(n) \approx \ln n$, where $H(n)$ denotes the n-th harmonic number. Dinur and Steurer [34] proved that the approximation ratio of Greedy is indeed best possible, by showing that no polynomial-time $(1 - o(1)) \ln(n)$-approximation algorithm exists unless P=NP.

An algorithm for Set Cover with approximation ratio $f_{\max}$ can be obtained through rounding an optimal solution of the following natural Set Cover LP [92].

$$\begin{array}{lll} \text{minimize} & \sum w_S x_S & \\ \text{subject to} & \sum_{S\,:\,i\in S} x_S \geq 1 & \text{for all } i \in \Omega \\ & x_S \geq 0 & \text{for all } S \in \mathcal{X} \end{array}$$

Consider an optimal solution of the LP given above. Choose all sets whose value is bigger or equal to $\frac{1}{f_{\max}}$ to be in the cover. To see that the solution obtained is indeed a cover, note that any element is contained in at most $f_{\max}$ sets. Hence, at least one of them has to be picked with a fraction of at least $\frac{1}{f_{\max}}$. Since we increase the value of the solution by at most a factor of $f_{\max}$, we obtain the desired approximation ratio.

Assuming the Unique Games Conjecture holds, the approximation factor $f_{\max}$ is best possible due to a result by Bansal and Khot [12]. For more detailed information on the Set Cover problem, see for example [101].

The Min Sum Set Cover problem (MSSC) was originally introduced in [41]. In Chapter 4, we study MSSC on laminar families and with more general covering constraints.

Definition 1.7 (Min Sum Set Cover)
Let $\mathcal{H} = (V, E)$ be a hypergraph on n vertices. For a linear ordering $\pi : V \to [n]$ we define the **cover time** of an edge to be $\pi(e) \coloneqq \min_{v\in e} \pi(v)$. The task is to find a linear ordering π that minimizes $\sum_{e\in E} \pi(e)$.

Feige et al. [41] showed that a simple Greedy algorithm yields a 4-approximation for MSSC. The Greedy algorithm chooses in every step a vertex of maximum degree. See Figure 1.9. The hypergraph is then updated by removing the vertex and all edges that contain the vertex. Note that it is NP-hard to approximate MSSC with a ratio better than 4 [42].

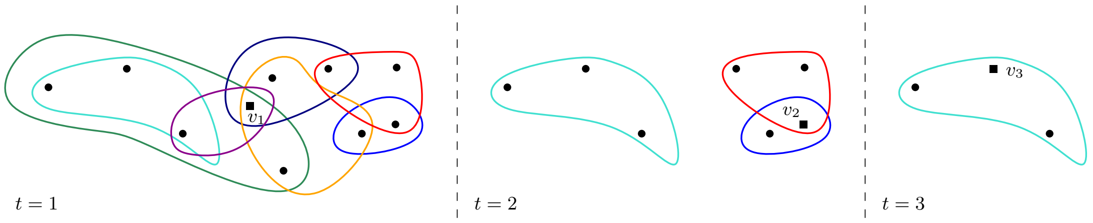


Fig. 1.9: Example to visualize the Greedy algorithm for Min Sum Set Cover. The chosen vertex is marked by a square and the graph is updated in every step.

1.1.5 Scheduling

Let J be a set of n jobs and let m be the number of machines. If $m = 1$, we call the scheduling problem a single-machine scheduling problem. Every job j comes together with a **processing time** $p_j \geq 0$ and a **non-negative weight** w_j. Additionally, we might be given **release dates** $r_j \geq 0$ for every job j. A **schedule** is an assignment of jobs to machines, such that each job is scheduled for exactly p_j units on a machine, and on every machine, there is at most one job scheduled at a time. In a non-preemptive setting, a job has to be scheduled in subsequent time intervals on the same machine. In the setting of arbitrary release dates, a job is only allowed to be scheduled on a machine after its release date. C_j denotes the completion time of a job. We consider two objective functions in this thesis. The main focus is on minimizing the **sum of weighted completion times**, that is, minimizing $\sum w_j C_j$. The **makespan** $C_{\max}$ is the completion time of the job that finishes the latest. In other words, $C_{\max} := \max\{C_1, \ldots, C_n\}$.

Scheduling problems are typically denoted by the **three-field notation** $\alpha \mid \beta \mid \gamma$, which was introduced by Graham [50]. The first entry α denotes the machine environment, such as

$$\begin{aligned} 1 &: \text{single machine} \\ PD &: \text{concurrent open shop.} \end{aligned}$$

The second entry β provides the characteristics of the scheduling problem, for example,

$$\begin{aligned} prec &: \text{precedence constraints} \\ pmtn &: \text{preemption} \\ p_j = 1 &: \text{unit processing times} \\ r_j &: \text{non-trivial release times.} \end{aligned}$$

The final entry γ represents the objective function, such as

$$\begin{aligned} C_{\max} &: \text{minimum makespan} \\ \sum w_j C_j &: \text{sum of weighted completion times.} \end{aligned}$$

In the following, we briefly discuss two concepts for single machine scheduling. We are given a set of jobs J together with processing times p_j and weights w_j for all $j \in J$. The goal is to minimize the sum of weighted completion times. In the three-field notation explained above, this problem is denoted by $1 \mid\mid \sum w_j C_j$. A well-known result by Smith [100] states that scheduling jobs in non-increasing order of $\rho(j) := \frac{w_j}{p_j}$ yields an optimal schedule. The ratio $\rho(j)$ is often referred to as Smith's Ratio, and the scheduling policy described above is known as **Smith's Rule** or the weighted shortest processing time first (WSPT) rule.

Before explaining the concept of a Sidney Decomposition, we introduce the necessary definitions. Consider a set of jobs J and a precedence relation. A subset $S \subseteq J$ of jobs is called **ideal** with respect to the precedence relation, if for every job $j \in J$, all of its predecessors are also in S. Next, we extend Smith's Ratio $\rho(j)$ of a job to a set of jobs. That is, we define for a subset $I \subseteq J$

$$\rho(S) \coloneqq \frac{w(S)}{p(S)} = \frac{\sum_{j \in S} w_j}{\sum_{j \in S} p_j}.$$

A set S^* is called **ρ-maximizing ideal** if S^* is an ideal with maximum ratio $\rho(S^*)$ among all other ideals, that is, $\rho(S^*) = \max_{S \subseteq J} \rho(S)$. A decomposition of J into $S_1 \dot\cup S_2 \dot\cup \ldots \dot\cup S_l$ is called a **Sidney Decomposition**, if every S_i is a ρ-maximizing ideal of the instance restricted to the remaining jobs $J \setminus (\bigcup_{k=1}^{i-1} S_k)$. Sidney [97] showed that there always exists an optimal schedule in which all jobs in S_i are scheduled before all jobs in S_k for all $i < k$. Chekuri and Motwani [25], and Margot et al. [81] independently proved that any Sidney Decomposition is a 2-approximation for $1 \mid prec \mid \sum w_j C_j$. Various other 2-approximation algorithms for $1 \mid prec \mid \sum w_j C_j$ are known [29, 51, 93]. Correa and Schulz [32] showed that all known 2-approximation algorithms [25, 29, 51, 81, 93] follow Sidney's decomposition.

The concept of finding ρ-maximizing ideals can be applied to scheduling problems with more general precedence constraints. In Chapter 4, we consider a similar approach for the Generalized Min Sum Set Cover problem by Happach et al. [53].

Chapter 2

Minimum Hitting Set of Bundles Problem

The Minimum Hitting Set of Bundles Problem (Mhsb) is a natural generalization of the Hitting Set problem, where instead of hitting single elements, bundles of elements are hit. More specifically, we are given a ground set of elements and a family of sets. Every set in this family contains bundles of elements which are subsets of the ground set. The task is to find a collection of elements of minimum size such that at least one bundle of every set in the family is hit.

In this chapter, we present a combinatorial approximation algorithm for Mhsb. Its approximation guarantee depends on the maximum number of bundles an element appears in and on the size of the family. We show that its approximation ratio is tight (up to constant factors) and give an improved analysis for laminar instances.

Mhsb has many applications. In particular, it provides an abstract framework for several scheduling problems. Motivated by these applications, we consider Mhsb restricted to interval and 2-dimensional interval bundles. We study the computational complexity and give polynomial-time algorithms for several classes of instances with these specially structured bundles.

This is joint work with Marinus Gottschau. Parts of this chapter correspond to or are identical to [49].

The chapter is structured as follows. We introduce the Minimum Hitting Set of Bundles problem formally and highlight two particular applications of Mhsb in Section 2.1. Previous work on Mhsb and related problems are summarized in Section 2.1.1. In Section 2.1.2 we give an overview of our results on the general Minimum Hitting Set of Bundles problem and two special classes of instances with interval and 2-dimensional interval bundles. Section 2.2 is dedicated to an approximation algorithm for Mhsb. We show that our analysis is tight and improve our results on laminar instances. Motivated by the application of Mhsb to busy time minimization of processors, we study the Minimum Hitting Set problem with interval bundles in Section 2.3. Finally, we analyze Mhsb on 2-dimensional interval bundles in Section 2.4. This bundle structure is mainly motivated by the application of Mhsb to railway maintenance scheduling. We conclude the chapter by presenting future research directions and open problems.

2.1 Introduction

The Minimum Hitting Set of Bundles Problem (Mhsb) was introduced by Angel et al. [6] and is defined as follows. Let Ω be a finite set of elements and let $\mathcal{F}$ be a family of sets. Every $F \in \mathcal{F}$ is a set of bundles, where a bundle U is a subset $U \subseteq \Omega$. A bundle U is **covered** by a set of elements $S \subseteq \Omega$ if $U \subseteq S$. We say that a set F is **hit** if at least one bundle in F is covered. We want to find a collection of elements $S \subseteq \Omega$ of minimum size such that every set $F \in \mathcal{F}$ is hit. We refer to S as a **hitting set of bundles**. In the following, let $\mathcal{U} := \bigcup_{F \in \mathcal{F}} F$ be the set of all bundles of an instance and denote by $F_{\max}$ the maximum number of bundles any set contains. To familiarize ourselves with the definitions above, we consider an example.

Example 2.1
We are given an instance of Mhsb on 9 elements, visualized in Figure 2.1. The family $\mathcal{F}$ consists of three sets F_1, F_2 and F_3. Each of them contains bundles, more specifically, $F_1 = \{U_1, U_3\}$, $F_2 = \{U_2, U_4, U_6\}$ and $F_3 = \{U_5, U_7\}$. A hitting set of bundles S is given by the three elements represented by squares in Figure 2.1. To see that S is indeed a hitting set of bundles, note that S covers the bundles U_1, U_6, and U_7, and, hence, the sets F_1, F_2 and F_3 are hit.

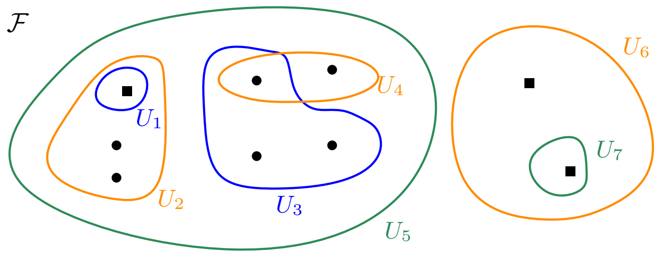
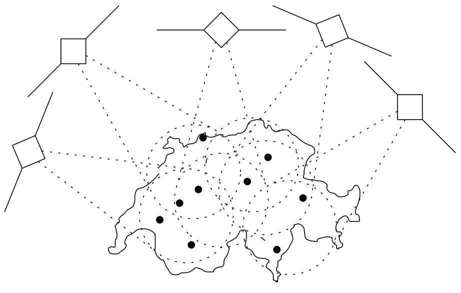


Fig. 2.1: An instance of Mhsb with sets $F_1 = \{U_1, U_3\}$ (blue), $F_2 = \{U_2, U_4, U_6\}$ (yellow) and $F_3 = \{U_5, U_7\}$ (green). A hitting set S is given by the elements represented by squares.

The Hitting Set problem is the special case of Mhsb, where every bundle contains exactly one element [6]. It corresponds to the optimization version of one of Karp's 21 NP-complete problems, the Set Cover problem [66]. This immediately implies NP-hardness of Mhsb. The Minimum Hitting Set of Bundles problem provides a mathematical framework for a variety of applications, such as the positioning of geosynchronous satellites (see Figure 2.2a), railway maintenance scheduling, and minimizing the active time of processors (see Figure 2.2b).

In this chapter, we want to highlight two of them. The first one is scheduling jobs (non-preemptively) on a single machine with the objective of minimizing active time. It can be framed in the following way. Here, Ω is a set of time slots, and $\mathcal{F}$ represents a set of jobs. The bundles of a job refer to its feasible operation times. The goal is to schedule all jobs such that the number of occupied time slots is minimized. If the bundles arise from possible execution times determined by job-specific release dates, deadlines, and processing times, the problem is referred to as Busy Time Minimization problem with unlimited capacities [23, 67].

The second application of MHSB we want to point out is in the area of Railway Maintenance Scheduling. Consider a railway corridor with bidirectional traffic and maintenance jobs that need to be carried out. Here, a train path is a movement over time along the railway track. Whenever such a train path interferes with a particular maintenance job, the train needs to be canceled. To minimize the impact of the mandatory maintenance on rail traffic, the goal is to schedule all maintenance jobs such that the number of canceled trains is as small as possible. Here, Ω represents the set of train paths, and $\mathcal{F}$ is the family of maintenance jobs. The bundles of a maintenance job refer to the corresponding sets of train paths that interfere with the feasible execution times of a job. Eskandarzadeh et al. [39] studied a variant of this problem in which bundles are determined by job-specific release dates, deadlines, and processing times. For unidirectional train traffic, this agrees with Busy Time Minimization with unlimited capacities. In this chapter, we also consider the Railway Maintenance Scheduling problem with bidirectional traffic.


(a) Positioning geosynchronous satellites.

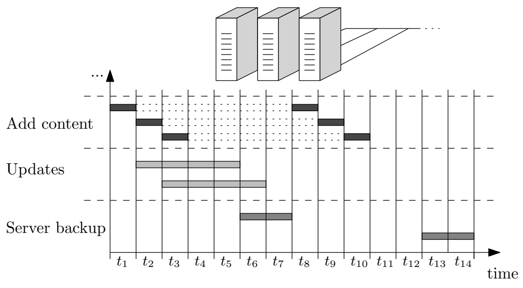

Add content
Updates
Server backup
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14}
time
(b) Minimizing active time of processors.

Fig. 2.2: Visualization of two exemplary applications of MHSB. In (a) the universe Ω is given by a set of satellites. Every set in the family $\mathcal{F}$ corresponds to an area that has to be covered by a collection of satellites and the goal is to minimize the total number of satellites. In (b) the universe Ω is given by a collection of time slots. A set corresponds to a job that has to be run on a processor and the bundles represent its feasible time slots. The goal is to minimize the active time of the processor.

MHSB allows more general bundle structures than those studied in the context of Busy Time Minimization or Railway Maintenance Scheduling. Still, we often use the scheduling terminology to give an intuition and a better understanding of the particular cases of MHSB that we consider in this chapter.

2.1.1 Related Work

Set Cover/Hitting Set Problem. As mentioned before, if every bundle contains only one element, Mhsb corresponds to the Hitting Set problem and its counterpart, the Set Cover problem. Both problems have been studied extensively. For Hitting Set, there exists a polynomial-time $f_{\max}$-approximation algorithm, where $f_{\max}$ refers to the maximum number of elements (in Mhsb this is the maximum number of bundles) a set may contain. Assuming the Unique Games Conjecture, the approximation factor $f_{\max}$ is best possible, due to a result by Bansal and Khot [12]. Additionally, there exists a polynomial-time $H(n)$-approximation algorithm, where n denotes the number of sets of the Hitting Set instance. Dinur and Steurer [34] proved that no polynomial-time $(1-o(1))\ln(n)$-approximation algorithm exists for Set Cover, unless P=NP. For more details on Hitting Set and Set Cover, see Section 1.1.4.

Minimum Hitting Set of Bundles Problem. The Minimum Hitting Set of Bundles problem was introduced by Angel et al. [6]. They presented a polynomial-time $F_{\max}$-factor approximation algorithm, where $F_{\max}$ refers to the maximum number of bundles a set may contain. This approximation guarantee is achieved by considering an LP relaxation of an integer linear programming (ILP) formulation and a simple rounding strategy. By using randomized rounding, they were able to improve the approximation guarantee to $F_{\max}(1-(1-\frac{1}{F_{\max}})^M)$. Here, M denotes the maximum number of bundles an element is contained in. Note that if the same bundle is contained in different sets, it is accounted for several times. Angel et al. [6] highlighted two applications of Mhsb, the Multiple-Query Optimization problem in database systems [94] and the Min k-Sat problem [20]. Being a generalization of the Minimum Hitting Set problem, Mhsb is $W[2]$-hard parameterized by the solution size $|S|$. Damaschke [33] proved that Mhsb, parameterized by $|\mathcal{F}|$ and the solution size $|S|$, is $W[1]$-complete.

Submodular Cover Problem. Wan et al. [103] studied the Minimum Submodular Cover problem with submodular weights. In this problem, we are given a submodular, increasing function $f : 2^{\Omega} \to \mathbb{R}$. A set $S \subseteq \Omega$ is a submodular cover if $f(S) = f(\Omega)$. The objective is to find a submodular cover of minimum weight with respect to a submodular, increasing weight function w. Mhsb can be formulated as a submodular cover problem with submodular weights in the following way. Choose the ground set to be the set of bundles $\mathcal{U}$. Define f to be the function that maps a collection of bundles on the number of sets that are hit by at least one bundle of the collection. The weight of a collection of bundles is simply the cardinality of their union. The main result in [103] implies a $H(\gamma_{\max})$-approximation for Mhsb, where $H(n)$ denotes the n-th harmonic number and $\gamma_{\max}$ is the maximum number of sets a bundle U in $\mathcal{U}$ hits. Iwata and Nagano [62] studied the Set Cover problem with submodular weights. They derived a polynomial-time $\alpha_{\max}$-approximation algorithm, where $\alpha_{\max}$ is the maximum number of sets an element appears in.

Busy Time Minimization Problem. The Busy Time Minimization problem with capacity B was introduced by Chang et al. [22]. It is a scheduling problem with job-specific release dates, deadlines, and processing times and a bound B on the number of jobs that may be executed simultaneously. Chang et al. [22] gave a polynomial-time algorithm for $B = 2$ and proved that the problem is NP-hard for $B = 3$. More closely related to our problem is the version with unlimited capacity ($B = \infty$), studied, for example, by Fong et al. [44]. They presented a polynomial-time algorithm for agreeable deadlines, i.e., instances where any job's deadline is prior to every other job's deadline that has a later release time. Fang et al. [40] studied the problem in the context of wireless sensoring and presented a polynomial-time 2-approximation algorithm for this special case. Online variants of the Busy Time Minimization problem also exist and have been studied, for example, by Koehler and Khuller in [71].

Maintenance Scheduling Problem. Eskandarzadeh et al. [39] studied the Maintenance Scheduling in a Railway Corridor problem, which is an application of Mhsb. They gave a polynomial-time algorithm for Active Time Minimization with unlimited capacities if all jobs have the same processing time. More specifically, they presented an ILP formulation, tailored to their restricted set of instances, for which they proved total unimodularity of the constraint matrix. In their computational experiments, they compared different ILP formulations for the bidirectional version.

Maximum Coverage Problem with Group Budgets. Finally, Chekuri and Kumar [24] introduced a maximization variant of Mhsb, the Maximum Coverage problem with group budgets. In this setting, we are given costs on bundles and a budget per set. The goal is to choose bundles, respecting the budget constraints, such that the size of the union of chosen bundles is maximized. Chekuri and Kumar [24] presented a polynomial-time constant-factor approximation algorithm.

2.1.2 Our Results

We study the general Minimum Hitting Set of Bundles problem and two special classes of instances with interval bundles and 2-dimensional interval bundles.

Minimum Hitting Set of Bundles Problem. In Section 2.2, we present a polynomial-time $\omega_{\max} H(|\mathcal{F}|)$-approximation algorithm for the general Mhsb, where $\omega_{\max}$ is the maximum number of bundles an element appears in and $H(n)$ denotes the n-th Harmonic number. Note that $\omega_{\max}$ may be strictly smaller than M as defined by Angel et al. in [6]. If the same bundle is contained in multiple sets, the constant M accounts for it multiple times, whereas $\omega_{\max}$ counts it just once. The algorithm is fully combinatorial and is based on the construction of a so-called cover graph. For laminar families of Mhsb we improve the approximation guarantee by presenting a polynomial-time $H(|\mathcal{F}|)$-approximation algorithm.

Minimum Hitting Set of Interval Bundles Problem. The connection to the Active Time Minimization problem leads to a number of applications that can be modeled by MHSB. In Section 2.3 we take advantage of the structure that many of these applications have in common in order to obtain polynomial-time algorithms. Assuming that jobs have to be executed without preemption, we obtain bundles of consecutive elements. We refer to these bundles as interval bundles and call this special case of MHSB the Minimum Hitting Set of Interval Bundles Problem (MHSIB). Motivated by applications, we define the following properties of special cases of MHSIB. We say that $\mathcal{F}$ is convex if for every job $F \in \mathcal{F}$ the union of all possible operating times forms an interval. We say that $\mathcal{F}$ is a-simple for some $a \in \mathbb{N}$ if all bundles have size a. In the scheduling terminology, this corresponds to equal processing times. In applications, it also seems reasonable to assume that, for example, the number of starting times or the time horizon of feasible operating times for a job is bounded.

We present polynomial-time algorithms for several classes of interval bundle instances. These algorithms use a graph construction and solve the problem by computing a shortest path. However, the Minimum Hitting Set of Interval Bundles problem is NP-hard in general. We explore the boundary of polynomial-time solvable instances and NP-hardness that arise from the aforementioned properties and parameters.

Minimum Hitting Set of 2-dimensional Interval Bundles Problem. In Section 2.4, motivated by the application of MHSB to railway maintenance scheduling, we study another special class of instances of MHSB. In the Maintenance Scheduling in a Railway Corridor problem presented by Eskandarzadeh et al. [39], Ω is the disjoint union of sets of train paths in opposite directions on a single railway track. More generally, we can think of Ω as being the disjoint union of two totally ordered sets. Assuming that every job has to be executed without preemption implies that every bundle is a set of consecutive elements in each of the two totally ordered sets. We call this special structure 2-dimensional interval and denote the special case of MHSB, where all bundles are 2-dimensional interval, by Minimum Hitting Set of 2-dimensional Interval Bundles Problem (2-DIM MHSIB). We show that 2-DIM MHSIB remains NP-hard on convex and 1-simple instances with $F_{\max} = 2$, where $F_{\max}$ refers to the maximum number of bundles a set may contain. Furthermore, we present a polynomial-time approximation algorithm for all 1-simple instances with $F_{\max} = 2$ using a result by Hochbaum [58] with an approximation guarantee slightly better than 2. For another restricted class of instances, we give a polynomial-time algorithm by making use of a problem decomposition.

Remark 2.2
The introduction of a weight function $w : \Omega \to \mathbb{R}_+$ on the elements may be of interest. For the ease of presentation, we only consider the unweighted MHSB. Techniques used throughout the chapter also work for weighted instances, thus all of our results can be easily transferred to weighted MHSB. These results can either be transferred directly

with minor modifications of inequalities, or by simple modifications of the constructed graph's edge weights.

Remark 2.3
Due to the structure of the problem, we assume, w.l.o.g., that

i) all bundles are non-trivial, that is, for all $F \in \mathcal{F}$ we have that $\emptyset \notin F$;
ii) bundles of a set are not subsets of each other, that is, for $U \in F$ if $U' \subset U$ we have that $U' \notin F$; and
iii) for all distinct $F, F' \in \mathcal{F}$ there exists a bundle $U \in F$ such that for all $U' \in F'$ we have $U' \not\subseteq U$. If there was no such bundle $U \in F$, any set of elements hitting F also hits F'.

2.2 Approximation Algorithms

In the following, we present a polynomial-time approximation algorithm for MHSB. Its analysis is rather straightforward and uses techniques similar to existing set cover approximation algorithms. We use a generic graph representation, the so called cover graph, to transform the problem into an instance of Set Cover with submodular weights. Note that Theorem 2.4 also follows from a result in [103] on Submodular Set Cover with submodular weights.

First, we introduce the concept of a bipartite cover graph. Given an instance $(\Omega, \mathcal{F})$ of MHSB, the corresponding **bipartite cover graph** $G_{\mathcal{F}} := (V_{\mathcal{F}}, E_{\mathcal{F}})$ is given by:

$$V_{\mathcal{F}} := \{\, v_U \;:\; U \in \mathcal{U} \,\} \cup \{\, v_F \;:\; F \in \mathcal{F} \,\},$$
$$E_{\mathcal{F}} := \{(v_U, v_F) \mid U \in \mathcal{U}, F \in \mathcal{F} : \exists\, U' \subseteq U \text{ with } U' \in F\}.$$

An instance of MHSB and its corresponding cover graph is visualized in Figure 2.3.

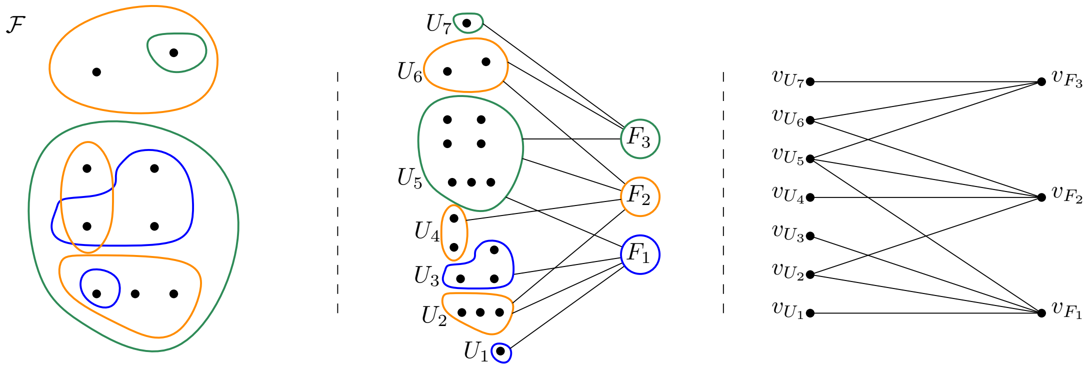


Fig. 2.3: An instance $(\Omega, \mathcal{F})$ of MHSB (left), its disjoint bundle representation (middle) and the corresponding bipartite cover graph (right).

Theorem 2.4
Let $(\Omega, \mathcal{F})$ be an instance of Mhsb. *There exists a polynomial-time $\omega_{\max} H(|\mathcal{F}|)$-approximation algorithm with*

$$\omega_{\max} := \max_{\omega \in \Omega} \{|\{U \in \mathcal{U} : \omega \in U\}|\} .$$

Proof. Given an instance $(\Omega, \mathcal{F})$ of Mhsb, we consider the corresponding cover graph $G_{\mathcal{F}} = (V_{\mathcal{F}}, E_{\mathcal{F}})$. This cover graph can be interpreted as an instance of the set cover problem, where each vertex v_U corresponds to a set and each vertex v_F corresponds to an element. Such an element can be covered by all sets U, for which there exists an edge between v_F and v_U. Let the weight of a set corresponding to v_U be $|U|$ (see Figure 2.3). Observe that we overestimate weights, e.g. if two overlapping bundles are chosen in the set cover instance, we overcount the number of elements.

Any hitting set of bundles $S \subseteq \Omega$ for $\mathcal{F}$ is the union of bundles $U \in \mathcal{U}$ that hit all respective sets in $\mathcal{F}$. The corresponding vertex set $V_S := \{v_U \mid U \in \mathcal{U} : U \subseteq S\}$ then induces a set cover C for the corresponding set cover instance. Additionally, any set cover C for this instance gives rise to a hitting set of bundles S. Let now S^* be a minimum hitting set of bundles, and let C^* be a minimum set cover for the cover graph instance. Then

$$|S^*| \leq \sum_{U \in C^*} |U| \leq \omega_{\max} |S^*|, \tag{2.1}$$

since elements in S^* are accounted for once, whereas in C^* elements count more than once. Furthermore, every element is contained in at most $\omega_{\max}$ sets. Therefore, a set cover of size $\omega_{\max}|S^*|$ always exists, from which the second inequality follows.

We next make use of the existing polynomial-time $H(|\mathcal{F}|)$-approximation algorithm for weighted set cover [101]. The algorithm greedily computes the most weight-efficient set, i.e. the set that has a best weight-over-newly-covered elements ratio. Let C be the solution found by the Greedy algorithm for the set cover instance. Then,

$$\sum_{U \in C} |U| \leq H(|\mathcal{F}|) \sum_{U \in C^*} |U|.$$

Finally, using inequalities from (2.1), we can construct a hitting set of bundles S from C such that

$$|S| \leq \sum_{U \in C} |U| \leq H(|\mathcal{F}|) \sum_{U \in C^*} |U| \leq \omega_{\max} H(|\mathcal{F}|) |S^*|.$$

This concludes the proof of the approximation guarantee. $\square$

Furthermore, the analysis of the algorithm is tight up to a constant factor. In the following, we construct an instance of Mhsb as on which the Greedy algorithm described above has an approximation guarantee of at most $\frac{1}{4}\omega_{\max}\log(|\mathcal{F}|)$.

Let the set of elements $\Omega := (\Omega' \dot\cup \Omega'')$, where $\Omega' := \{\omega_1, \omega_1', \dots, \omega_d, \omega_d'\}$. Now, for every $U \in \mathcal{P}(\Omega')$ that contains exactly one of the elements ω_i or ω_i' for all $i \in [d]$, we introduce $d-1$ sets $(F_U^l)_{l \in [d-1]} \in \mathcal{F}$ all of which contain this bundle. In addition to U, the set F_U^l contains a bundle of l other elements from Ω'' all of which are only contained in this bundle and nowhere else. Observe that by construction $|\mathcal{F}| = 2^d(d-1)$ and, due to symmetry of the bundles on $\{\omega_1, \omega_1', \dots, \omega_d, \omega_d'\}$, we have that $\omega_{\max} = \frac{|\mathcal{F}|}{2(d-1)}$.

The Greedy algorithm always chooses the bundle of l elements in F_U^l over U for all $l \in [d-1]$. Therefore, the solution of the Greedy algorithm has size

$$2^d \cdot \sum_{l=1}^{d-1} l = 2^{d-1}(d-1)(d-2) = \frac{1}{2}|\mathcal{F}|(d-2).$$

In contrast, the optimal solution is of size $2d$. Thus, the Greedy approximation has an approximation factor of at most

$$\begin{aligned}\frac{d-2}{4d}|\mathcal{F}| &= \frac{1}{2}\left(1-\frac{2}{d}\right)\frac{1}{2}|\mathcal{F}| = \frac{1}{2}\left(1-\frac{2}{d}\right)(d-1)\omega_{\max}\\ &= \frac{1}{2}\left(d+\frac{2}{d}-3\right)\omega_{\max} \geq \frac{1}{4}\omega_{\max}\log(|\mathcal{F}|).\end{aligned}$$

Here, we made use of the fact that $\log(|\mathcal{F}|) = \log(2^d(d-1)) = d + \log(d-1) \leq 2(d+\frac{2}{d}-3)$ for $d \geq 8$ and that $H(n) \in \mathcal{O}(\log n)$.

2.2.1 Approximation Algorithms for Laminar Instances

In the proof of Theorem 2.4, we constructed an instance of Mhsb to show that the approximation guarantee for the algorithm is tight (up to a constant factor). However, the bundle structure in this example is very nested, and the algorithm itself does not consider any of the combinations of bundles that would have led to an optimal solution for that particular example. In general, it is not clear to what extent taking certain combinations of bundles into account can improve the approximation guarantee. Additionally, one would have to overcome the exponential number of possible combinations of bundles.

The question arises of how the approximation guarantee improves when we restrict our algorithm to instances in which the nonempty intersection of bundles is always a bundle itself. We call the family $\mathcal{F}$ **laminar** if $\mathcal{U}$ is laminar, i.e., if for all $U, U' \in \mathcal{U}$ with $U \cap U' \neq \emptyset$, we have $U \subseteq U'$ or $U' \subseteq U$. We present a polynomial-time approximation algorithm for laminar instances of Mhsb with an improved approximation guarantee.

Theorem 2.5
Let $(\Omega, \mathcal{F})$ be an instance of Mhsb. *If $\mathcal{F}$ is a laminar family, then there exists a polynomial-time $H(|\mathcal{F}|)$-approximation algorithm.*

Proof. Due to the laminar structure, we can represent any hitting set of bundles as a union of disjoint bundles. Thus, there is no overcounting and a minimum hitting set of bundles gives rise to a set cover of the same weight. □

2.3 Minimum Hitting Set of Interval Bundles

In this section, we focus on bundle structures that arise in many applications. As mentioned in the introduction, Ω often corresponds to time slots if viewed as a scheduling problem. In this context, one may assume that jobs have to be executed without preemption. In the following, we make use of this particular structure and identify polynomial-time solvable special cases of Mhsb.

Throughout this section we look at instances in which we are given an ordering $\prec$ of Ω. To simplify notation, we may assume that $\Omega = [n]$ for some $n \in \mathbb{N}$ with the natural ordering. In this context, we consider a special case of Mhsb, the Minimum Hitting Set of Interval Bundles Problem (Mhsib), where every bundle corresponds to an interval. More specifically, if $i, j \in U$, then $k \in U$ for all $i \leq k \leq j$. In addition to the interval property, we define two other properties that a family $\mathcal{F}$ may have.

Definition 2.6
Let $([n], \mathcal{F})$ be an instance of Mhsib. We call the family $\mathcal{F}$

i) **convex**, if all $F \in \mathcal{F}$ are convex, i.e. the following holds. Let $i \in U', j \in U''$, for some $U', U'' \in F$. For all $k \in [n]$ with $i \leq k \leq j$ there exists a bundle U in F such that $k \in U$. In other words, the union of all interval bundles U in F is again an interval.

ii) ***a*-simple** for some $a \in \mathbb{N}$ if all bundles are a-simple, i.e. $|U| = a$ for all $U \in \mathcal{U}$, where $\mathcal{U} := \bigcup_{F \in \mathcal{F}} F$.

Figure 2.4 is an example of an instance of the general Mhsb problem and visualizes the properties we defined above. The next theorem shows that Mhsib remains NP-hard even on restricted instances in terms of Definition 2.6.

Theorem 2.7
Let $([n], \mathcal{F})$ be an instance of Mhsib *and let $F_{\max}$ be the maximum number of bundles a set may contain. The problem remains NP-hard, if $\mathcal{F}$*

i) is 1-simple and $F_{\max} = 2$; or
ii) is convex and $F_{\max} = 3$; or
iii) is convex and a-simple for some a, where a is some function in n.

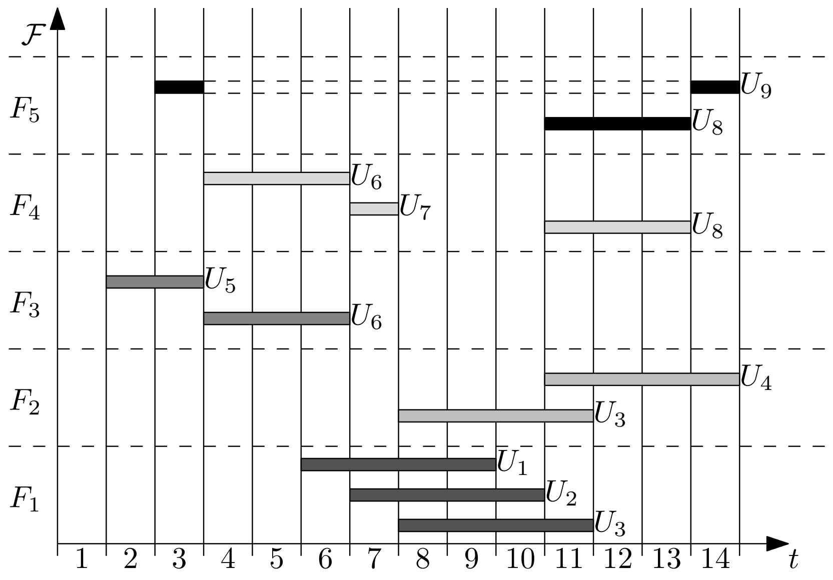


Fig. 2.4: Example of a family $\mathcal{F}$ to visualize the different sets and bundles properties: All bundles but U_9 have the property that they are interval. The sets F_1, F_2 and F_3 are convex, and F_1 and F_2 are in addition to that a-simple with $a = 4$. The bundle structure of the set F_1 is determined by the release date 6, deadline 11 and processing time 4. This overlapping bundle structure occurs for example in Busy Time Minimization.

Proof. i) An easy reduction from Vertex Cover[1] implies the statement. Given a graph $G = (V, E)$, we let $\Omega := V$ and $\mathcal{F} := \{\{\{u\}, \{v\}\} : (u, v) \in E\}$. Observe that a minimum vertex cover is an optimal solution to MHSIB and vice versa.

ii) The statement follows again from a reduction of Vertex Cover to MHSIB. Let $G = (V, E)$ be a graph with an arbitrary ordering $\prec_V$ of the vertices. The main idea is to use the same construction as in i), i.e. introducing a set for every edge (u, v) of the graph. The challenge is to guarantee that our constructed family is convex, while respecting the structure assumed in Remark 2.3. We do so by constructing sets that additionally contain a large dummy bundle. Because of its size, the dummy bundle is never entirely covered by an optimal solution. A large number of dummy elements serves to ensure a one-to-one correspondence between a minimum hitting set of bundles and a minimum vertex cover.

We begin by constructing the ground set of the corresponding instance of MHSIB. We introduce d elements for each vertex of the graph, that is, $\Omega_v := \{\omega_i^v : i \in [d]\}$ for all $v \in V$. A vertex $v \in V$ is represented by the element ω_1^v and we refer to the $d - 1$ elements $\omega_2^v \dots \omega_d^v$ as dummy elements. The ground set is now given by $\Omega := \bigcup_{v \in V} \Omega_v$.

[1]For a definition of the Vertex Cover problem see Def. 1.3.

We use the ordering $\prec_V$ of V to obtain an ordering $\prec_\Omega$ of Ω in a natural way by

$$\omega_i^u \prec_\Omega \omega_j^v \Leftrightarrow \begin{cases} u \prec_V v; & \text{or} \\ u = v \quad \wedge & i < j. \end{cases}$$

Next, we define bundles and sets of the instance. As in the construction of i) for every edge $(u, v) \in E$ with $u \prec_V v$ we are given two bundles $\{\omega_1^u\}$ and $\{\omega_1^v\}$. In addition, we define a dummy bundle $U_{(u,v)}$ containing all elements in Ω that are between ω_1^u and ω_1^v. Formally, we define

$$U_{(u,v)} := \Omega_u \setminus \{\omega_1^u\} \cup \bigcup_{u \prec_V w \prec_V v} \Omega_w.$$

Note that all constructed bundles are indeed interval with respect to the ordering of Ω induced by $\prec_\Omega$. The sets are then formally given by

$$\mathcal{F} = \left\{ \left\{ \{\omega_1^u\}, \{\omega_1^v\}, U_{(u,v)} \right\} \ : \ (u,v) \in E \right\}.$$

The dummy bundle $U_{(u,v)}$ contains a large number of elements, i.e. at least d and, hence, is never entirely hit in an optimal solution. Its only purpose is to ensure convexity of the respective family.

Any vertex cover C corresponds to a solution of the Mhsib instance of the same size. This simply follows from the fact that the vertices in C cover every edge and therefore, all sets $F \in \mathcal{F}$ contain at least one covered bundle. Additionally, a solution S of Mhsib of minimum size only contains elements that correspond to vertices of the graph, i.e. $S \subseteq \{\omega_1^v : v \in V\}$. This follows from the fact that if S covers any bundle $U_{(u,v)}$, this immediately implies $|S| \geq d - 1 = |V| + 1$. Also, any element in $\Omega \setminus \{\omega_1^v : v \in V\}$ can be omitted unless it was needed to cover some $U_{(u,v)}$.

Finally, we have that S hits all sets $F \in \mathcal{F}$ and covers the bundle $\{\omega_1^u\}$ or $\{\omega_1^v\}$ for all $(u,v) \in E$, with either ω_1^u or ω_1^v. Therefore, the set of vertices represented by elements in S form a vertex cover in the graph.

iii) Again, we prove the statement by a reduction of Vertex Cover to Mhsib. Let $G = (V, E)$ be a graph and let $\prec_V$ be an arbitrary ordering of the vertices. This time we have to ensure that the constructed family is convex and a-simple. For every edge (u, v) we construct a corresponding set $F_{(u,v)}$ in the family. The set contains multiple a-simple bundles, amongst them the bundles U_u and U_v which correspond to the vertices u and v. Instead of a single large dummy bundle as in ii), the sets contain a collection of dummy bundles, each of size a. To ensure that the optimal solution to Mhsib does indeed hit every set by covering a bundle corresponding to a vertex in V, we introduce additional dummy sets F_v for every $v \in V$. The set F_v contains a single bundle that overlaps with the bundle U_v in all but one element. Since there is only one bundle in F_v, it has to be in every feasible solution of the instance of Mhsib. This ensures that hitting a set $F_{(u,v)}$ by a bundle U_v or U_u only requires adding one additional element to the solution, whereas hitting $F_{(u,v)}$ by any dummy bundle immediately increases the solution size significantly (by at least $\frac{a-1}{2}$).

Again, we start by constructing the ground set of the corresponding instance of Mhsib. Formally, let $a := 4|V| + 1$ and let v_0 be a dummy vertex with $v_0 \prec_V v$ for all $v \in V$. For every vertex in V and the dummy vertex v_0 we introduce $3a$ elements, that is, $\Omega_v := \{\omega_i^v : i \in [3a]\}$ for all $v \in V \cup \{v_0\}$. Intuitively, the element ω_1^v corresponds to the vertex $v \in V$ and all other elements in Ω_v are dummy elements. The ground set of elements is then given by $\Omega := \bigcup_{v \in V \cup \{v_0\}} \Omega_v$. As in ii) we extend the ordering $\prec_V$ of the vertices in a natural way to an ordering $\prec_\Omega$ of the elements in Ω.

We continue by constructing the bundles and the sets of the family $\mathcal{F}$ of the corresponding instance of Mhsib. For a vertex $v \in V$ let U_v be the bundle that contains the element ω_1^v and the $a - 1$ previous elements with respect to $\prec_\Omega$. Formally,

$$U_v := \{\omega_{2a+2}^{v'}, \omega_{2a+3}^{v'}, \ldots, \omega_{3a-1}^{v'}, \omega_{3a}^{v'}, \omega_1^v\}$$

where v' is the vertex preceding v in the natural ordering. Note that all bundles U_v are indeed a-simple.

Next, we define a collection B of dummy bundles, which is a partition of $\Omega \setminus \{\omega_1^{v_0}, \omega_2^{v_0}, \ldots, \omega_{\frac{5a+1}{2}}^{v_0}\}$ in a-simple, disjoint bundles. Later, B is used to ensure convexity of the family $\mathcal{F}$. More explicitly B is given by

$$B := \bigcup_{v \in V} \Big\{ \underbrace{\{\omega_{\frac{5a+3}{2}}^{v'}, \ldots, \omega_{3a}^{v'}, \omega_1^v, \ldots, \omega_{\frac{a+1}{2}}^v\}}_{a}, \underbrace{\{\omega_{\frac{a+3}{2}}^v, \ldots, \omega_{\frac{3a+1}{2}}^v\}}_{a}, \underbrace{\{\omega_{\frac{3a+3}{2}}^v, \ldots, \omega_{\frac{5a+1}{2}}^v\}}_{a} \Big\},$$

where, again, v' is the vertex preceding v in the natural ordering. Note that the bundles in B are well-defined, since we chose a to be odd. We continue by constructing the sets of the family $\mathcal{F}$. For every edge $(u, v) \in E$ the family $\mathcal{F}$ contains a set

$$F_{(u,v)} = \{U_u, U_v\} \cup B.$$

In addition to that, for every vertex $v \in V$ we add a dummy set containing a single bundle with the a elements preceding ω_1^v to $\mathcal{F}$. More explicitly,

$$F_v := \{\{\omega_{2a+1}^{v'}, \omega_{2a-2}^{v'}, \ldots, \omega_{3a-1}^{v'}, \omega_{3a}^{v'}\}\}$$

where, again, v' is the vertex preceding v in the natural ordering. This concludes the construction of the family. A visualization can be seen in Figure 2.5.

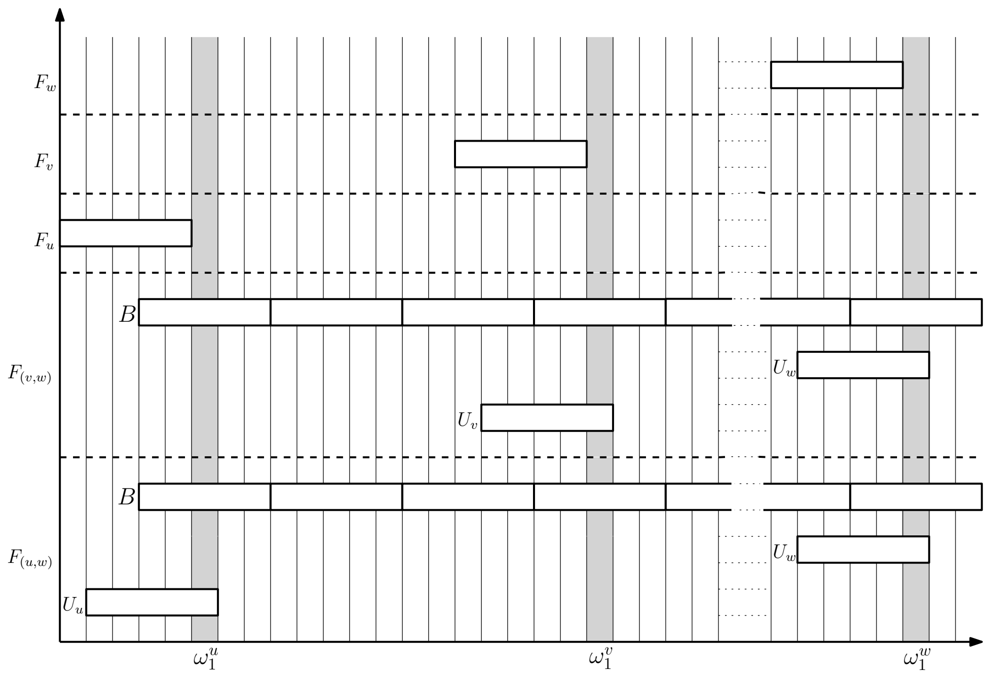


Fig. 2.5: Illustration of a constructed Mhsib instance in the proof of Theorem 2.7 iii).

Let C be a minimum vertex cover in G. We obtain the corresponding hitting set of bundles in the following way. As already mentioned, every bundle of a dummy set F_v has to be part of a feasible solution. Next, we ensure that every $F_{(u,v)}$ is hit. Adding the element ω_1^v for every $v \in C$ to the hitting set of bundles, we cover the bundle U_v and, hence, hit the corresponding set $F_{(u,v)}$. All in all, we obtain a hitting set of bundles of size $a \cdot |V| + |C|$. Note, every set $F_{(u,v)}$ is hit by one of the bundles U_u or U_v as either u or v must be contained in C. We claim that there is no solution to Mhsib of smaller cardinality. Observe, that an optimal solution to Mhsib has to hit every set F_v, each containing only one bundle of size a. Since all these sets are pairwise disjoint, a solution must be of size at least $a \cdot |V|$. Any bundle from the collection B contains at least $a/2 > |V|$ elements not contained in the bundles of the dummy sets F_v that are covered by the solution. Again, if such a bundle from B is covered, the solution is of size at least $a \cdot |V| + |V| + 1$, which cannot be optimal by the previous argument, that there always exists a solution of size at most $a \cdot |V| + |V|$. □

Theorem 2.7 states that Mhsib remains NP-hard for convex families $\mathcal{F}$ with $F_{\max} \leq 3$ for all $F \in \mathcal{F}$. However, the following theorem shows that the computational complexity changes for convex instances if we reduce $F_{\max}$ by one.

Theorem 2.8
Let $([n], \mathcal{F})$ be an instance of MHSIB. *If $\mathcal{F}$ is convex and $F_{\max} = 2$, then it is solvable in polynomial time.*

Proof. We describe a reduction to a shortest path computation. The high-level idea is to construct a layered graph, such that the vertices contained in an s-t-path describe an interval decomposition of the solution.

Before explaining the construction in more details, let us take a closer look at the bundle structure of our instance to give some intuition. Since every set contains at most two bundles, w.l.o.g., we may assume that for all $F \in \mathcal{F}$, if $U, U' \in F$ then $U \cap U' = \emptyset$, as elements in the intersection have to be contained in any feasible solution. Given some $F \in \mathcal{F}$, since $\mathcal{F}$ is convex, there exist l, i and $u \in [n]$ such that $F = \{[l,i], [i+1,u]\}$. To make sure that the bundle is hit, we have to guarantee that either the set of elements $\{l, \dots i\}$ or $\{i+1, \dots, u\}$ is contained in the minimum hitting set of bundles.

Let $([n], \mathcal{F})$ be an instance of MHSIB. We define a graph, with layers $V_1, \dots V_n$ corresponding to the elements $1, \dots n$. A set V_i contains a vertex for every interval in $[1, n]$ containing i and, additionally, a vertex $v^i_{\varnothing}$ representing the empty set. Formally, for every $i \in [n]$ we define $V_i := \{v^i_{[a,b]} \mid 1 \le a \le i \le b \le n\} \cup \{v^i_{\varnothing}\}$. Note that $a, b \le n$ and therefore, the number of vertices introduced for each $i \in [n]$ is polynomial in the size of the input. Let $V_0 := \{s\}$ and $V_{n+1} := \{t\}$. Then the vertex set of the layered graph is given by $V := \bigcup_{i=0}^{n+1} V_i$.

The graph being layered, means that there only exist edges from layer V_i to the subsequent layer V_{i+1}. In particular, every shortest s-t-path contains exactly one vertex v^i from every layer V_i. Intuitively, if $v^i = v^i_{\varnothing}$, then the element i is not part of the corresponding minimum hitting set of bundles. If $v^i = v^i_{[a,b]}$, we add i to the solution. In particular, the solution contains all elements of the interval $[a, b]$. To ensure this, we define the edges and edge weights of the layered graph in the following way. There are four feasible types of edges:

(1) $(v^i_{\varnothing}, v^{i+1}_{\varnothing})$ of weight 0,

(2) $(v^i_{\varnothing}, v^{i+1}_{[i+1,b]})$ of weight 1,

(3) $(v^i_{[a,b]}, v^{i+1}_{[a,b']})$ for some $b' \ge b$ of weight 1, and

(4) $(v^i_{[a,i]}, v^{i+1}_{\varnothing})$ of weight 0.

Additionally, we add edges (s, v^1) for all $v^1 \in V_1$ of weight 0, if $v^1 = v^1_{\emptyset}$ and of weight 1 in all remaining cases. Finally, we add edges (v^n, t) for all $v^n \in V_n$, all of weight 0. We only have positive weights of 1 on edges of type $(s, v^1_{[1,b]})$ and $(v^i_{\varnothing}, v^{i+1}_{[i+1,b]})$ and $(v^i_{[a,b]}, v^{i+1}_{[a,b']})$. These correspond to adding the element 1 and $i+1$ to the hitting set of bundles, respectively.

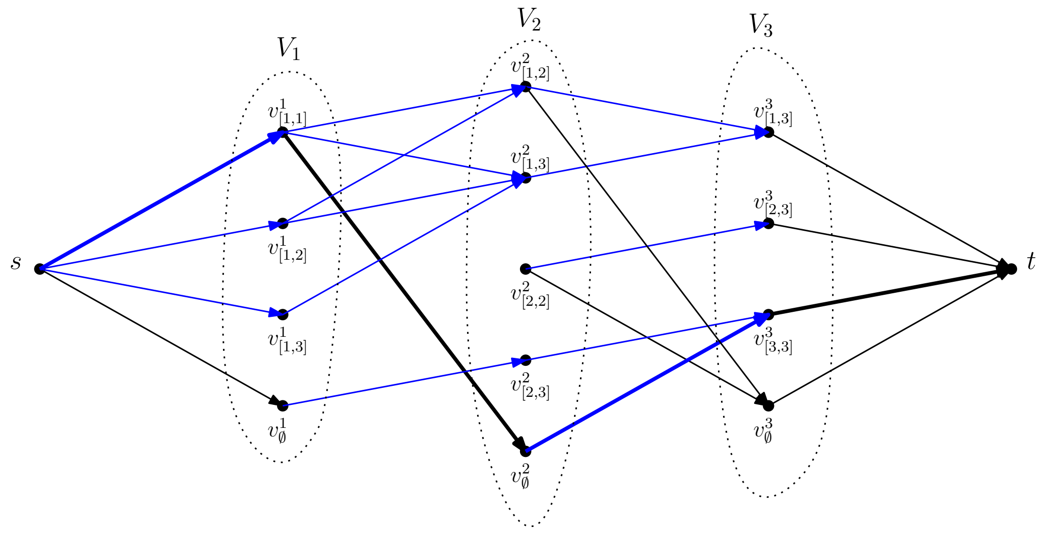


Fig. 2.6: Illustration of the constructed graph on an instance of 3 elements $\Omega = \{1, 2, 3\}$ with set family $\mathcal{F} = \{F_1, F_2\}$ with $F_1 = \{\{1\}, \{2, 3\}\}$ and $F_2 = \{\{2\}, \{3\}\}$. Blue edges have weight 1, black edges weight 0. A shortest s-t-path is highlighted in bold. It corresponds to the minimum hitting set of bundles containing elements $\{1\}$ and $\{3\}$.

In a next step, we tailor the graph to the set structure of our specific instance by deleting certain edges. This step ensures, that every set is hit. Consider a set $F \in \mathcal{F}$ with $F = \{[l, i], [i + 1, u]\}$. If not all elements $\{l, l + 1, \ldots, i\}$ are part of the solution, we have to make sure that all elements in $\{i + 1, \ldots, u\}$ are added to the solution. To do so, for every $F \in \mathcal{F}$ with $F = \{[l, i], [i + 1, u]\}$, we delete all edges of the following types:

(1) the edge $(v^i_{\varnothing}, v^{i+1}_{\varnothing})$,

(2) edges $(v^i_{\varnothing}, v^{i+1}_{[i+1,b']})$ for which $b' < u$,

(3) edges $(v^i_{[a,b]}, v^{i+1}_{[a,b']})$ for which $a > l$ and $b' < u$, and

(4) edges $(v^i_{[a,i]}, v^{i+1}_{\varnothing})$ for which $a > l$.

As argued above, doing so for all sets in $\mathcal{F}$, any s-t-path gives rise to a hitting set of bundles. Additionally, any hitting set of bundles in its interval decomposition corresponds to an s-t-path. I.e. let $S \subset [n]$ be the set that contains element i if and only if the s-t-path does not contain the vertex $v^i_{\varnothing}$. The weight of the s-t-path equals the cardinality of the corresponding set S. A depiction of the constructed graph for a small instance can be seen in Figure 2.6.

We now compute a shortest s-t-path in the graph. Correctness of the algorithm follows from the one-to-one correspondence of s-t-paths and the respective interval decomposition of a hitting set of bundles and the fact that the weight of any path equals the sum of the cardinality of the sets (intervals). $\square$

Theorem 2.7 also states that Mhsib remains NP-hard if $\mathcal{F}$ is a-simple and convex. The following theorem considers two different types of convex instances with an additional property. In scheduling terminology the first type of instances corresponds to the case where the difference of any job's release date and deadline is bounded by k. In ii) the processing time of every job is bounded by k and the number jobs intersection with time slot i is bounded by c, for some $c, k \in \mathbb{R}^+$.

Theorem 2.9
Let $([n], \mathcal{F})$ be an instance of Mhsib. *For any constants $c, k \in \mathbb{R}^+$ the problem is solvable*

i) in $O(2^{2k}n)$ if $|i - j| \leq k$ for all $i, j \in \bigcup_{U \in F} U$ and all $F \in \mathcal{F}$; or

ii) in $O(2^{2(k+c)}n)$ if $|U| \leq k$ for all $U \in \mathcal{U}$, $\mathcal{F}$ is convex and $|\mathcal{F}_{|_i}| \leq c$ for all $i \in [n]$, where $\mathcal{F}_{|_i} := \{F \in \mathcal{F} \mid \exists\, U \in F : i \in U\}$.

Proof. i) The key idea is to construct a weighted graph, whose shortest s-t-path gives rise to a decomposition of a solution of the corresponding Mhsib instance. Here, we make use of the fact that the size of every set is bounded by k and, thus, all subsets of elements that intersect and hit a set F can be represented by a bounded number of sets. Note that we say that an element i intersects a set F if there exists a bundle $U \in F$ with $i \in U$.

We start by giving a formal construction of the graph $G = (V, E)$. First, define the vertex set $V := V_0 \cup \ldots \cup V_{n+1}$ with $V_i := \{v_S^i : S \subseteq \{i - \min\{i, k\}, \ldots, i\}\}$, $V_0 := \{s\}$ and $V_{n+1} := \{t\}$. Note that $|V_i| \leq 2^k$. In the following, for all $i \in [n]$ we let $\mathcal{F}_i$ be the family of sets with largest element i, i.e. $i = \max\{j \in \bigcup_{U \in F} U\}$ for all $F \in \mathcal{F}_i$. The set of edges E is then defined as follows. Our graph G only has edges between subsequent layers, that is, there only exist edges from V_i to V_{i+1}. More specifically, edges from a vertex $v_S^i \in V_i$ to all other vertices in V_{i+1} exist if S hits all sets in $\mathcal{F}_i$. If $\mathcal{F}_i = \emptyset$, we include all edges in $V_i \times V_{i+1}$. Finally, all edges from s to V_1 are contained in the edge set. The weight of each edge $(v_S^i, v_{S'}^{i+1})$ is given by the number of elements in $S' \setminus S$. Edges (s, v_S^1) have weight $|S|$, and the exiting edges (v_S^n, t) have weight 0. This concludes the construction of the graph G. For a schematic picture of such a constructed graph see Figure 2.7.

Since the size of every set is bounded by k, i.e. $|i - j| \leq k$ for all $i, j \in \bigcup_{U \in F} U$, every subset of elements that hits all sets in F_i is a subset of $\{i - \min\{i, k\}, \ldots, i\}\}$. Observe that any shortest s-t-path P passes through exactly one vertex of every V_i.

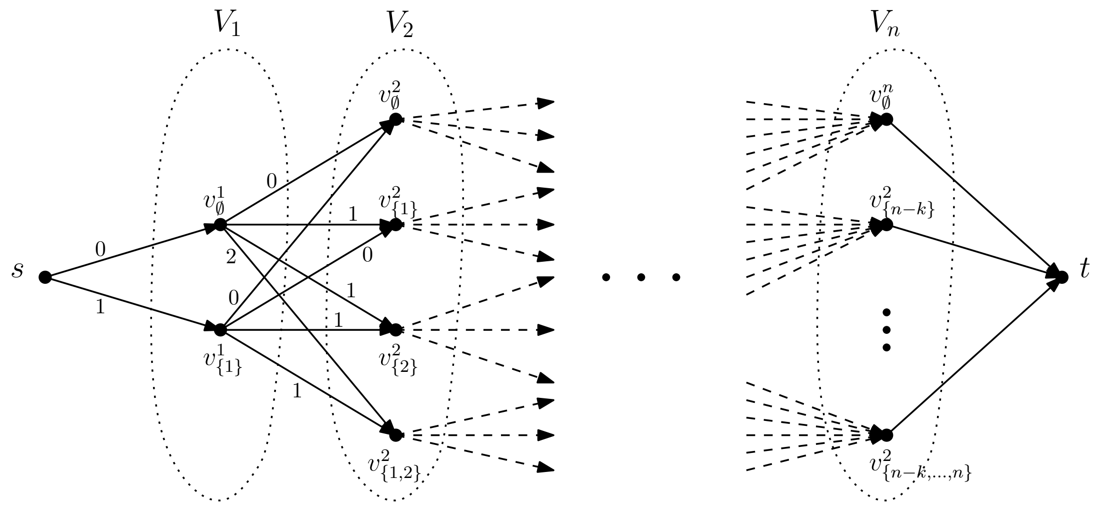


Fig. 2.7: Illustration of the constructed graph with exemplary edge weights.

On that path, each vertex v^i_S represents a set of elements S. Let $S^* := \bigcup_{S:v^i_S \in P} S$. We claim that S^* is an optimal solution to the corresponding Mhsib instance. Every set F has a largest element i, and only vertices v^i_S where S hits all sets in $\mathcal{F}_i$ have outgoing edges. Thus, S^* hits every set in $\mathcal{F}$ and, hence, S^* is feasible. Observe, that by the choice of the weight function the length of any shortest s-t-path P equals $|S^*|$.

On the other hand, every hitting set of bundles S^* gives rise to a s-t-path of same cost in the corresponding graph G. The path is obtained by traversing the respective vertices of the sets $\{i - \min\{i,k\}, \dots, i\} \cap S^*$ for all $i \in [n]$. Note that the size of the constructed graph is in $O(2^{2k}n)$, following by the bounds on $|V_i|$ and the fact that we only have edges between subsequent layers V_i and V_{i+1}. A simple breadth-first search (BFS) finds a shortest path in linear time of the size of the graph.

ii) Again we make use of a graph $G = (V, E)$ defined on $V := V_0 \cup \ldots \cup V_{n+1}$. Here,

$$V_i := \mathcal{P}(\{i - \min\{i,k\}, \dots, i\}) \times \mathcal{P}(\{F \in \mathcal{F} \mid \exists U \in F : i \in U\}),$$

where $\mathcal{P}$ denotes the power set. $V_0 := \{s\}$ and $V_{n+1} := \{t\}$. In other words, every vertex in V_i for $i \in [n]$ corresponds to a tuple $(S^\Omega, S^\mathcal{F})$ with $S^\Omega \subseteq \{i - \min\{i,k\}, \dots, i\}$ and $S^\mathcal{F} \subseteq \{F \in \mathcal{F} \mid \exists\, U \in F : i \in U\}$. The set $S^\mathcal{F}$ shall be used to encode which of the sets containing element i already have been hit. Note that by assumption the number of vertices in each set V_i is bounded by the number of tuples, which is at most $2^k \cdot 2^c$.

Given the vertex set as defined above, we only allow edges between subsequent layers, that is, between vertices in V_i and vertices in V_{i+1}. More specifically, there exists an edge from $(S^\Omega, S^\mathcal{F}) \in V_i$ towards $(\bar{S}^\Omega, \bar{S}^\mathcal{F}) \in V_{i+1}$ if the two following conditions hold:

a) for all $F \in \bar{S}^{\mathcal{F}} \setminus S^{\mathcal{F}}$ there exists a bundle U in F such that $U \subseteq \bar{S}^{\Omega}$. That is, $F \in \bar{S}^{\mathcal{F}} \setminus S^{\mathcal{F}}$ if $\bar{S}^{\Omega}$ covers one bundle in F; and

b) if $F \in \mathcal{F}_i$, then $F \in S^{\mathcal{F}}$. This ensures that any s-t-path corresponds to a feasible solution of the respective instance of MHSIB.

All edges in $\{s\} \times \{(S^{\Omega}, S^{\mathcal{F}}) \in V_1 : S^{\mathcal{F}} = \emptyset\}$ are contained in E. Let the weight function $w : E \to \mathbb{N}$ be defined as

$$w((S^{\Omega}, S^{\mathcal{F}}), (\bar{S}^{\Omega}, \bar{S}^{\mathcal{F}})) = |\bar{S}^{\Omega} \setminus S^{\Omega}|.$$

We claim that an optimal solution to MHSIB can be obtained by computing a shortest s-t-path in the corresponding graph G. A solution to the MHSIB instance is, again, obtained by taking the union of all sets S^{Ω} of elements represented by the traversed vertices. Feasibility of the hitting set of bundles follows directly from the construction of the edges in b). Moreover, the size of the solution equals the length of the path.

Similarly to i) the choice of weights implies that any solution to the corresponding instance of MHSIB gives rise to an s-t-path of same length. At this point we make use of the fact that the family is convex, which implies that the set of elements that is contained in a set forms an interval. Without this condition, there might exist layers i,j with $i < j - 1$, and a set F such that i and j intersect F but $j - 1$ does not. Due to condition b), this forces the set F to be hit by the elements from $\{j, \dots, n\}$, even though in an optimal solution the set F might only be hit by elements from $\{1, \dots, j-1\}$. Note that the size of the constructed graph is in $O(2^{2(k+c)}n)$, following by the bounds on $|V_i|$ and the fact that we only have edges between subsequent layers V_i and V_{i+1}. A simple BFS finds a shortest path in linear time of the size of the graph. □

2.4 Minimum Hitting Set of 2-dimensional Interval Bundles

So far, we focused on instances with a given total ordering of Ω. In this section, we consider instances with $\Omega := N_1 \dot{\cup} N_2$ where we are given a total ordering $\prec_1$ of the elements in N_1 and a total ordering $\prec_2$ of the elements in N_2. Throughout this section we may assume, w.l.o.g., that $N_1 := \{1, \dots, n_1\}$ as well as $N_2 := \{n_1 + 1, \dots, n\}$ with cardinalities $n_i := |N_i|$ for $i \in \{1, 2\}$.

As mentioned in the introduction, this setting is also motivated by an application to railway maintenance ([39] and see Example 2.11). Interpret N_1 and N_2 as two sets of train paths in opposite direction on a railway track and let $\Omega := N_1 \dot{\cup} N_2$. If we assume that maintenance jobs are executed without preemption, we obtain bundles with a very specific structure, so-called 2-dimensional interval bundles. More specifically, a set of bundles is **2-dimensional interval** if for all $U \in \mathcal{U}$, if $i_1, i_2 \in U$ with $i_1, i_2 \in N_1$, then for all $i' \in N_1$ with $i_1 \leq i' \leq i_2$ we also have that $i' \in U$. Analogously, if $j_1, j_2 \in U$ with $j_1, j_2 \in N_2$, then for all $j' \in N_2$ with $j_1 \leq j' \leq j_2$ we also have that $j' \in U$.

We refer to this problem as the Minimum Hitting Set of 2-dimensional Interval Bundles Problem (2-dim Mhsib) and define the properties of being 2-dim-convex and a-simple as follows.

Definition 2.10
Let $(N_1 \dot\cup N_2, \mathcal{F})$ be an instance of 2-dim Mhsib. For $i \in \{1, 2\}$, let $\mathcal{F}_{|N_i}$ be the family of sets restricted to N_i, that is, $\mathcal{F}_{|N_i} := \{(\bigcup_{U \in F}\{U \cap N_i\}) \setminus \{\emptyset\} : F \in \mathcal{F}\}$. We call the family $\mathcal{F}$

i) **2-dim-convex** if $\mathcal{F}_{|N_1}$ and $\mathcal{F}_{|N_2}$ are convex.

ii) ***a*-simple** for some $a \in \mathbb{N}$ if $|U| = a$ for all $U \in \mathcal{U}$.

The following example provides a visualization of the bundle structures in the context of railway maintenance scheduling.

Example 2.11
We are given a railway corridor between location A and location B with bidirectional traffic. The parallel lines in Figure 2.8 represent train paths. More specifically, the paths $\{1, \dots n_1\}$ correspond to trains from A to B and the paths $\{n_1 + 1, \dots n\}$ correspond to trains from B to A. The maintenance jobs are represented by the dashed boxes. In particular, the height of a box corresponds to the section of the railway corridor that requires maintenance work and the length corresponds to the time window in which a job has to be carried out. The length of a solid box represents the processing time of a job. This example of Railway Maintenance Scheduling can be formulated as an

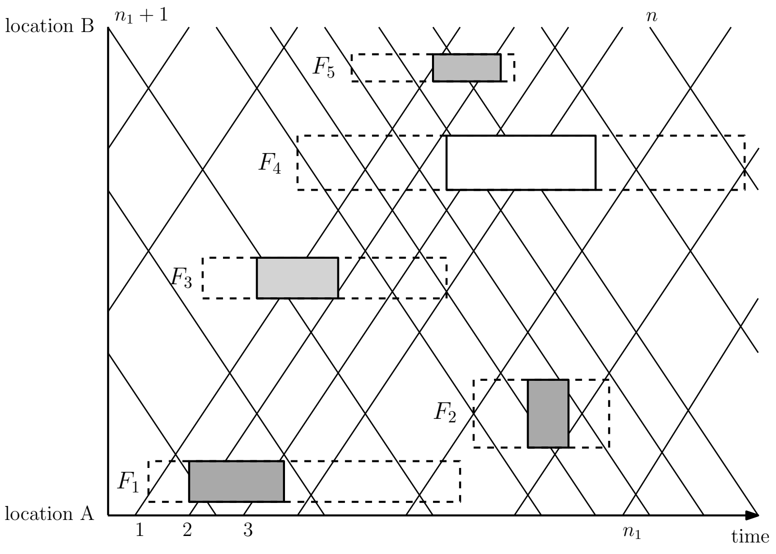


Fig. 2.8: An example of an instance of railway maintenance scheduling to obtain some intuition.

instance of 2-DIM MHSIB in the following way. The ground set of elements is given by the set of train paths. For every dashed box we introduce a set F. The bundles in F are determined by the sets of train path that interfere with the solid box, given a certain position within the dashed box.

To obtain some intuition, we interpret the properties of Definition 2.10 in the context of railway maintenance scheduling. The property of $\mathcal{F}$ being 2-dim-convex refers to a setting where the starting time of a job can be anywhere in a given time window. a-simple is a natural property in a regular train schedule setting where, independently of the starting time, a job always interferes with the same number of train paths.

In [39], it was shown that 2-DIM MHSIB remains NP-hard even on 2-dim-convex families. We are able to prove a slightly stronger result.

Theorem 2.12
The 2-DIM MHSIB *problem remains NP-hard if $\mathcal{F}$ is 2-dim-convex, 1-simple, and $F_{\max} = 2$.*

Proof. We use the following graph construction to reduce Vertex Cover to an instance with the required properties. Given a graph $G = (V, E)$, we subdivide every edge $e = (u, v) \in E$ into three edges (u, e_u), (e_u, e_v), (e_v, v) and denote the resulting graph by $G' = (V', E')$. Note that any vertex cover S in G yields a vertex cover in G' if we add exactly one vertex e_u, e_v for every edge $e = (u, v) \in E$ to the cover. This new cover S' has size $S + |E|$.

Also, w.l.o.g. a minimum vertex cover S' in G' contains exactly one vertex e_u, e_v for every edge $e = (u, v) \in E$ (if not, it is either not a cover or we can add u and remove e_u from the cover without changing its cardinality). We claim that $S' \cap V$ is a vertex cover in G. Assume the contrary, i.e. there is an edge $e = (u, v)$ that is not covered. Then, since S' was a cover in G', the edges (e_u, e_v) must be covered by S' by either e_u or e_v. Assume it is covered by e_u, then it immediately follows by assumption that $v \in S'$. A contradiction. Note that the cover $S = S' \cap V$ has cardinality $|S'| - |E|$.

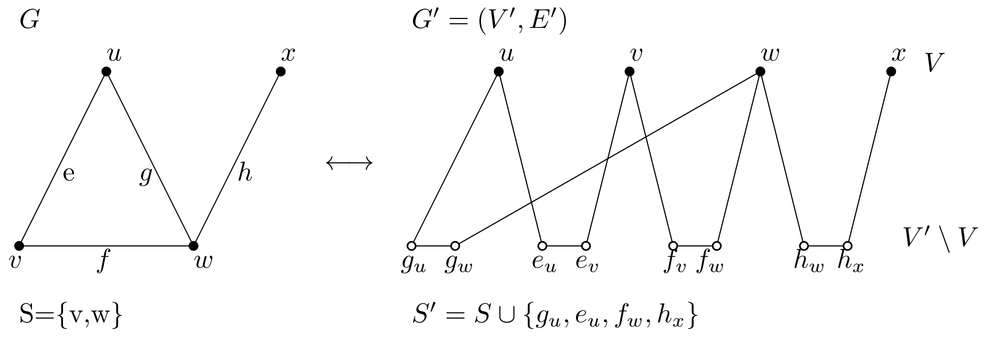


Fig. 2.9: Construction of G' from G and corresponding vertex covers S and S' in G and G'.

Now, it is easy to see that $\mathcal{F} := \{\{\{u\},\{v\}\} : (u,v) \in E'\}$ is an instance which is 1-simple, $F_{\max} = 2$ and 2-dim convex. The latter follows from the fact that we do not have edges in V and edges (e_u, e_v) only. Thus, all sets are convex using an arbitrary ordering on V and an arbitrary ordering on the set of edge-vertices $V' \setminus V$ as long as e_v, e_u follow one after another for all $e \in E$. Figure 2.9 shows the construction and indicates the partition of elements into V and $V' \setminus V$. □

In the following, we outline a polynomial-time approximation algorithm for 2-dim Mhsib where the family $\mathcal{F}$ is 1-simple and $|F| \leq 2$ for all $F \in \mathcal{F}$. Since $F_{\max} \leq 2$ in this setting, there already exists a 2-factor approximation algorithm [6]. We improve the approximation guarantee slightly by using techniques from approximation algorithms for the Vertex Cover problem.

Theorem 2.13
Let $(N_1 \dot\cup N_2, \mathcal{F})$ be an instance of 2-dim Mhsib. If $\mathcal{F}$ is 1-simple and $F_{\max} = 2$, then there exists a polynomial-time $(2 - \frac{1}{k+1})$-factor approximation algorithm with

$$k := \max\Big\{|i-j| : \{\{i\},\{j\}\} \in \mathcal{F} \text{ with either } i,j \in N_1 \text{ or } i,j \in N_2\Big\}.$$

Proof. The proof of the theorem is based on a result by Hochbaum [58] for which we are going to convert our instance of 2-dim Mhsib into a Vertex Cover instance. The theorem is stated below.

Proposition 2.14 (Hochbaum [58])
Let G be a k-colorable graph. There exists a polynomial-time algorithm that finds a vertex cover with a size of at most $(2 - \frac{2}{k})$ times the size of an optimal vertex cover.

Given a 1-simple instance $(N_1 \dot\cup N_2, \mathcal{F})$ of 2-dim Mhsib with $F_{\max} = 2$, we construct a graph $G = (V, E)$ with $V := N_1 \dot\cup N_2$ and $E := \{(i,j) : \{\{i\},\{j\}\} \in \mathcal{F}\}$. Then, an optimal solution of 2-dim Mhsib corresponds to a minimum vertex cover in the graph G and vice versa.

Consider the subgraphs of $G[N_1]$ and $G[N_2]$ and observe that by the construction of G and the definition of k it holds that $\Delta(G[N_1]) \leq k$ and $\Delta(G[N_2]) \leq k$. Therefore, by Brooks' Theorem both induced subgraphs admit a $(k+1)$-coloring. These colorings can be extended to a $(2k+2)$-coloring of G. Applying Proposition 2.14, a chromatic number of at most $2k+2$ immediately guarantees a polynomial-time algorithm that computes a vertex cover whose size is at most $(2 - \frac{1}{k+1})$ times the size of an optimal vertex cover. □

Finally, we present a first polynomial-time algorithm for a special case of 2-dim Mhsib. It is based on a decomposition of the problem.

Theorem 2.15
Let $(N_1 \dot{\cup} N_2, \mathcal{F})$ be a 2-dim-convex instance of 2-DIM MHSIB *with* $F_{\max} \leq 2$. *The problem is solvable in polynomial time if for all $F \in \mathcal{F}$ and $U, U' \in F$, the symmetric difference only contains two elements, i.e. $|U \triangle U'| = 2$ and either*

i) those elements belong to N_1 or N_2, i.e. $U \triangle U' \subset N_1$ or $U \triangle U' \subset N_2$; or
ii) the symmetric difference always contains one element from N_1 and one element from N_2, i.e. $U \triangle U' \cap N_1 \neq \emptyset$ and $U \triangle U' \cap N_2 \neq \emptyset$.

Proof. First, note that for any $F = \{U, U'\}$ all elements in $U \cap U'$ must be contained in any hitting set of bundles. Let $(N_1' \dot{\cup} N_2', \mathcal{F}')$ be the instance after removing all elements in the respective intersections. Note that there is a one-to-one correspondence between a minimum hitting set of bundles of $\mathcal{F}'$ and a minimum hitting set of bundles of $\mathcal{F}$ by adding back or removing the aforementioned elements of the intersections. Due to 2-dim-convexity and a 2-dimensional interval family $\mathcal{F}$, we have that the reduced family $\mathcal{F}'$ on the reduced set of elements $N_1' \dot{\cup} N_2'$ is again 2-dim-convex and 2-dimensional interval. Furthermore, w.l.o.g., $|F| = 2$ for all $F \in \mathcal{F}'$. (If $|F| = 1$, all elements in $U \in F$ are contained in any hitting set of bundles and can be removed to reduce the instance even further.) Additionally, by assumption the family $\mathcal{F}'$ is 1-simple.

Observe, the graph on $N_1' \dot{\cup} N_2'$ with edges corresponding to the sets in $\mathcal{F}'$ is bipartite. This follows from the fact that on the one hand, if $U \triangle U' \subset N_1$ or $U \triangle U' \subset N_2$ for all $\{U, U'\} \in \mathcal{F}$, due to convexity, the graph consists of a collection of paths since each element in $N_1' \dot{\cup} N_2'$ can neighbor at most two others and cycles cannot occur. On the other hand, if $U \triangle U' \cap N_1 \neq \emptyset$ and $U \triangle U' \cap N_2 \neq \emptyset$ for all $\{U, U'\} \in \mathcal{F}$, edges only occur between vertices representing the sets N_1' and N_2', but not within the sets N_1' and N_2', respectively. Now, any vertex cover in the constructed graph corresponds to a solution of 2-DIM MHSIB and vice versa. Since the graph is bipartite, we can find a vertex cover in polynomial time, which follows from Kőnig's theorem and any polynomial time maximum matching algorithm (see Proposition 1.4). Finally, a solution to $\mathcal{F}'$ can easily be lifted to a solution to $\mathcal{F}$. □

2.5 Open Problems

In this chapter, we studied the Minimum Hitting Set of Bundles problem on general set families and restricted to interval and 2-dimensional interval bundles. In the following, we provide an outlook on future research directions.

In Theorem 2.7, we showed that MHSIB remains NP-hard even if the family $\mathcal{F}$ is convex and a-simple, where a is some function in n. It is open how the complexity changes if we require a to be constant.

Problem 2.16
Consider an instance of MHSIB. Does the problem remain NP-hard if the family $\mathcal{F}$ is convex and a-simple for constant a?

The following application of MHSB gives rise to several interesting questions for future research. The universe is given by a collection of courses. Every set represents a student, and the bundles of the set correspond to a feasible subset of courses the student could take. In this case, the bundle constraints do not only express time overlaps but can also be used to encode that some courses might only be taken in combination (e.g., lectures and corresponding exercise classes, lab experiments).

In this setting, introducing an upper bound on the number of students per course seems natural. This corresponds to introducing an upper bound on the number of sets that are hit by a bundle containing a specific element. If the bundles are interval, this problem is equivalent to Busy Time Minimization with capacity B.

Problem 2.17 (Constrained MHSB)
Consider an instance of MHSB together with an upper bound $b(\omega)$ for all $\omega \in \Omega$. A feasible solution is given by a collection of bundles $\mathcal{C}$ such that

(i) $\mathcal{C}$ contains at least one bundle from every $F \in \mathcal{F}$, and
(ii) every element $\omega \in \Omega$ is contained in at most $b(\omega)$ bundles in $\mathcal{C}$.

The objective is to find a feasible solution that minimizes the number if distinct elements in $\mathcal{C}$, in other words, to minimize $|\bigcup_{u \in \mathcal{C}} U|$.

Chapter 3

Partition Reductions of Gammoids

The main contribution of this chapter is a polynomial-time algorithm to reduce a k-colorable gammoid to a $(2k-2)$-colorable partition matroid. Gammoids are combinatorial matroids that generalize partition, transversal and laminar matroids. Intuitively, this type of reduction reduces a gammoid to its simple underlying structure. It is known that there are gammoids that cannot be reduced to any $(2k-3)$-colorable partition matroid, so this result is tight. Our motivation to study partition reductions is based on their connection to certain covering and coloring problems on the intersection of matroids. Parts of this chapter are dedicated to explaining the implication of our results on these types of problems.

This is joint work with Benjamin Moseley and Kirk Pruhs. The chapter corresponds to or is in parts identical to [76].

The chapter is structured as follows. After introducing the necessary terms and notation, Section 3.1.1 gives an overview of the matroid classes that play a role throughout this chapter and addresses the relation between the different classes. Next, we highlight the connection of partition reductions to two combinatorial optimization problems, the Matroid Intersection Cover problem (Section 3.1.2) and List Coloring of Matroids (Section 3.1.3). An overview of related work is given in Section 3.1.4. After this, we summarize our main results and explain their implications on related coloring and list coloring problems in Section 3.1.5. This is followed by an overview of our techniques in Section 3.1.6. Section 3.2 is dedicated to the description of the partition reduction Algorithm. The analysis is deferred to Section 3.3. Finally, we restate an example, showing that our algorithm is tight and conclude with open problems.

3.1 Introduction

A set system is a pair $M = (S, \mathcal{I})$ where S is a universe of n elements and $\mathcal{I} \subseteq 2^S$ is a collection of subsets of S. The set system is hereditary (or downward-closed) if $A \subseteq B \subseteq S$ and $B \in \mathcal{I}$ imply $A \in \mathcal{I}$. A **matroid** is a downward-closed set system with the additional properties that $\emptyset \in \mathcal{I}$ and if $A \in \mathcal{I}$, $B \in \mathcal{I}$, and $|A| < |B|$ then there exists an element $s \in B$ such that $A \cup \{s\} \in \mathcal{I}$. Sets in $\mathcal{I}$ are called **independent** and the **rank** r is the maximum cardinality of a set in $\mathcal{I}$. A partition $C_1, C_2, \ldots C_k$ of S into independent sets is a k-coloring of M. The **coloring number** of M is the smallest k such that a k-coloring exists.

If $R \subseteq S$, then the **restriction** of M to R, denoted by $M|R$, is a set system where the universe is $S \cap R$ and where a set $I \subseteq S$ is independent if and only if $I \subseteq R$ and $I \in \mathcal{I}$. The intersection of matroids $(S, \mathcal{I}_1) \ldots (S, \mathcal{I}_\ell)$ on a common universe is a downward-closed set system with universe S where a set $I \subseteq S$ is independent if and only if for all $i \in [\ell]$ it is the case that $I \in \mathcal{I}_i$.

A **gammoid** is a matroid that has a graphical representation $(D = (V, E), S, Z)$, where $D = (V, E)$ is a directed graph, $S \subset V$ is a collection of source vertices and $Z \subset V$ is a collection of sink vertices. In the gammoid, which is represented by D, a set $I \subseteq S$ is in $\mathcal{I}$ if and only if there exist $|I|$ vertex-disjoint paths from the vertices in I to some subcollection of vertices in Z. A **partition matroid** is a matroid that can be represented by a partition $\mathcal{X}$ of S. In the matroid that is represented by the partition a set $Y \subseteq S$ is in $\mathcal{I}$ if and only if $|Y \cap X| \leq 1$ for all $X \in \mathcal{X}$. [1]

A matroid N is a **reduction** (also called weak map) of a matroid M with the same universe if and only if every independent set in N is also an independent set in M. If N is a partition matroid, then we say that there exists a **partition reduction** from M to N. Im et al. [60] defined the following decomposability concept, which generalizes partition reduction.

Definition 3.1
A k-colorable matroid $M = (S, \mathcal{I})$ is (b, c)-**decomposable** if S can be partitioned into sets $X_1, X_2, .., X_\ell$ such that:

- For all $i \in [\ell]$ it is the case that $|X_i| \leq c \cdot k$.
- For a set $Y = \{v_1, \ldots, v_\ell\}$, consisting of one representative element v_i from each X_i, the matroid $M \mid Y$ is b-colorable.

If $b = 1$, then $X_1, X_2, .., X_\ell$ represent a partition matroid. Thus, $(1, c)$-decomposability means there exists a partition reduction, where the coloring number increases by at

[1]One can generalize this to let there be a separate upper bound $a(X)$ for each $X \in \mathcal{X}$ on the number of elements Y can obtain from X, but throughout this thesis the term partition matroid refers to a matroid where the bound is one. We refer to the generalization as general partition matroid.

most a factor of c. Note that if we do not require a bound on the coloring number of the corresponding partition matroid, a partition decomposition of any matroid is trivial. Simply, put all elements of the universe into a single partition. This corresponds to a $(1, \frac{n}{k})$-decomposition and implies a trivial upper bound on c. A simple lower bound on c is given by 1, since the coloring number of the corresponding partition matroid cannot be smaller than the coloring number of the original matroid.

3.1.1 Matroid Classes

We already introduced gammoids and partition matroids. This subsection gives an overview about the most important combinatorial matroids and addresses the connection between the different matroid classes.

A **uniform** matroid is a matroid in which a set I is independent if and only if $|I| \leq a$ for some $a \in \mathbb{N}$. By definition, uniform matroids have rank a and all circuits in M have exactly a elements. A matroid is called **paving** if all circuits have more than $a - 1$ elements. Hence, every uniform matroid is also paving.

Another family of matroids that generalizes uniform matroids are **general partition matroids**. We already introduced partition matroids, where each independent set may intersect with each part of the partition in at most one element. In a general partition matroid, we are given in addition to the partition $\mathcal{X}$ positive integers $a(X)$ for all $X \in \mathcal{X}$. A set Y is independent if and only if $|Y \cap X| \leq a(X)$ for all $X \in \mathcal{X}$.

Given a laminar family $\mathcal{F}$ and positive integers $a(X)$ for all $X \in \mathcal{X}$, we define the corresponding **laminar matroid** in the following way. A set Y is independent in the laminar matroid if and only if $|Y \cap X| \leq a(X)$ for all $X \in \mathcal{X}$. Since every partition is in particular laminar, every general partition matroid is also a laminar matroid.

Another class of matroids that generalizes partition matroids are **transversal matroids**. Let $X_1, X_2, \ldots, X_l$ be a collection of subsets of S. A set Y is independent

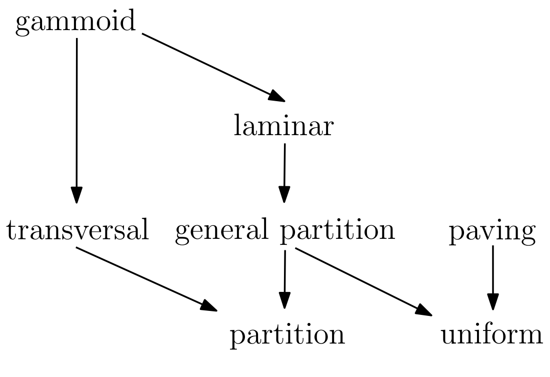


Fig. 3.1: Connection between matroid classes. An arc from class A to B in the diagram implies that B is a subclass of A.

in the corresponding transversal matroid if and only if $|Y \cap X_i| \leq 1$ for all $i \in [l]$. Gammoids generalize laminar and transversal matroids. For a visualization of the connection between the matroid classes defined above see Figure 3.1.

A **graphic matroid** is a matroid that can be represented by a multigraph $G = (V, E)$. The universe is given by the set E and a subset $Y \subseteq E$ is independent in the graphic matroid if and only if E induces a forest in G. An example of a uniform matroid that is not graphic is given by the uniform matroid on 4 elements with rank 2 [86].

3.1.2 Connection to the Matroid Intersection Cover Problem

In the following, we introduce the Matroid Intersection Cover problem and explain its connection to partition reductions. The Matroid Intersection Cover problem is the special case of Set Cover, where the underlying set system is given by the intersection of l matroids. Note that the intersection of l matroids is a downward-closed set system and, hence, the coloring number and the covering number are essentially the same. This follows from the subsequent observation. Given a cover $\mathcal{C} = C_1, C_2, \dots, C_r$ of a downward-closed set system, we consider an element s of the universe that is contained in multiple sets $C_{i_1}, \dots, C_{i_l}$ of the covering. Simply replace these sets by $C_{i_1}, C_{i_2} \setminus \{s\}, \dots, C_{i_l} \setminus \{s\}$, to ensure that s is only contained in one of them. The fact that the set system is downward-closed guarantees that the sets $C_{i_2} \setminus \{s\}, \dots, C_{i_l} \setminus \{s\}$ are also independent. Therefore, the Matroid Intersection Cover problem is sometimes also referred to as Matroid Intersection Coloring problem.

It is formally defined as follows.

Definition 3.2 (Matroid Intersection Cover Problem)
Given l matroids $M_1 = (S, \mathcal{I}_1), \dots, M_l = (S, \mathcal{I}_l)$ on a common universe S, find a collection of subsets $\mathcal{C}$, such that $\mathcal{C}$ covers S and every set in $\mathcal{C}$ is independent in all l matroids, that is, $\mathcal{C} \subseteq \bigcap_{i \in [l]} \mathcal{I}_i$. The objective is to minimize $|\mathcal{C}|$.

Im et al. [60] showed that given any set of l matroids that are $(1, c)$-decomposable, there exists a polynomial-time algorithm to construct a $(c \cdot l)$-approximate set cover for the intersection of their set systems. The main idea is to consider the partition reductions of the l matroids. For a matroid M_i let $\mathcal{X}_i$ be the partition that represents the corresponding partition matroid. Im et al. [60] constructed an l-partite hypergraph in the following way. The vertex set is given by $V_1, V_2 \dots, V_l$ such that every V_i contains a vertex for every part in $\mathcal{X}_i$. For every element s in S, we introduce a hyperedge e_s that contains the vertex from every V_i that corresponds to the part in $\mathcal{X}_i$ that contains s. Figure 3.2 displays the vertex set of the hypergraph and visualizes the structure of a single hyperedge e_s.

Consider a coloring of the hyperedges such that if two hyperedges e and e' are assigned the same color, then $e \cap e' = \emptyset$. Then every color class corresponds to a set C of elements in S that contains at most one vertex from a part of any partition $\mathcal{X}_i$. In

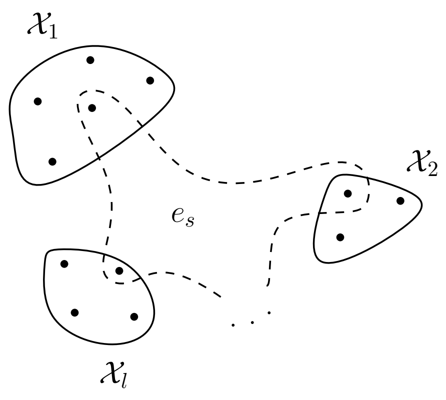


Fig. 3.2: Vertex set of the hypergraph together with a visualization of a hyperedge e_s.

other words, C is independent in the partition matroids represented by $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_l$ and, hence, C is also independent in the original matroids. A simple Greedy algorithm is used in [60] to obtain a coloring of the hypergraph. Overall, the main algorithmic result from [60] is:

Proposition 3.3 (Im et al. [60])
Consider matroids $M_1, M_2, \ldots M_\ell$ defined over a common universe, where matroid M_i has coloring number k_i. There is a polynomial-time algorithm that, given a (b_i, c_i)-decomposition of each matroid M_i, computes a coloring of the intersection of $M_1, M_2, \ldots M_\ell$ using at most $\left(\prod_{i\in[l]} b_i\right) \cdot \left(\sum_{i\in[l]} c_i\right) k^*$ colors, where $k^* = \max_{i\in[l]} k_i$.

3.1.3 Connection to List Coloring of Matroids

For graphs the concept of coloring can be extended to the concept of list coloring, where we are given a set or so-called list of colors $L(v)$ for every vertex of the graph. A list coloring of a graph is a coloring that additionally has the property that each vertex is colored with a color contained in its list. The list coloring number of the graph is the smallest k such that if $|L(v)| \geq k$ for all vertices v, we can guarantee the existence of a feasible list coloring.

In a similar way, one can extend the concept of coloring on matroids to **list coloring** on matroids. If each element $s \in S$ is additionally given an associated list $L(s)$ of allowable colors, then a list coloring is a coloring with the additional property that every element is colored with a color from its list. The **list coloring number** of M is the smallest k that guarantees that if for $s \in S$ it is the case that $|L(s)| \geq k$ then a list coloring exists. For a single matroid, Seymour [95] showed that the list coloring number equals its coloring number. The following question arises in a natural way: How about the list coloring number of the intersection of two matroids?

Definition 3.4 (List Coloring Problem for two Matroids)
Given two matroids $M_1 = (S, \mathcal{I}_1)$ and $M_2 = (S, \mathcal{I}_2)$ on a common universe S, find the smallest integer k, such that if $|L(s)| \geq k$ for all $s \in S$, there always exists a list coloring of the intersection of the two matroids.

The type of result one would hope for is an upper bound on the list coloring number of the intersection of two matroids in terms of the list coloring number (or equally the coloring number) of the two single matroids. If the two matroids M_1 and M_2 are partition matroids, a result by Galvin [45] implies that the list coloring number of the intersection of M_1 and M_2 equals $\max\{k_1, k_2\}$, where k_i is the coloring number (or equally the list coloring number) of M_i.

An immediate consequence of [45], which was also observed in [18], is that the existence of partition reductions for two matroids implies an upper bound on the list coloring number of their intersection. Formally, one has the following proposition.

Proposition 3.5 ([18, 45])
Let M_1, M_2 be two matroids with coloring numbers k_1, k_2, respectively. If each matroid is $(1, c_i)$-decomposable, then the list coloring number of the intersection of the two matroids is at most $\max\{c_1 k_1, c_2 k_2\}$.

3.1.4 Related Work

Partition Reductions. There are two prior, independent papers in the literature that are directly relevant to our results. Bérczi et al. [18] showed that any gammoid M admits a $(1, (2 - \frac{2}{k}))$-decomposition. This proof is constructive and can be converted into an algorithm. The resulting algorithm is essentially a local search algorithm that selects a neighboring solution in the dual matroid such that an auxiliary potential function always decreases. There seems to be little hope of obtaining a polynomial-time algorithm using techniques from [18] since the potential can be exponentially large. Bérczi et al. [18] showed that no better bound is achievable. Independently, Im et al. [60] gave a polynomial-time algorithm to construct a $(18, 1)$-decomposition of a gammoid. The reduction was shown by leveraging prior work on unsplittable flows [70]. Combining Proposition 3.3 with the decompositions in [18, 60] one obtains $O(1)$-approximation algorithms for problems that can be expressed as coloring problems on the intersection of $O(1)$ common combinatorial matroids. Several natural examples of such problems are given in [60].

Both papers [18, 60] also observed that partition reductions are relatively easily obtainable for other common types of combinatorial matroids. In particular, transversal matroids are $(1, 1)$-decomposable [60], graphic matroids are $(1, 2)$-decomposable [18, 60] and paving matroids are $(1, \lceil \frac{r}{r-1} \rceil)$-decomposable if they are of rank r [18].

Set Cover Problem. We already highlighted the connection of partition reductions to the Matroid Intersection Cover problem, which is essentially a special case of Set

Cover. Set Cover has been studied extensively in the field of approximation algorithms. See Section 1.1.4 for more information.

There is considerable interest in discovering special instances of Set Cover where this approximation ratio can be improved. Despite the interest, there are few cases where the approximation can be improved. One case is given by geometric covering problems. Examples of this include covering points in the plane using discs [84] for which a polynomial-time approximation scheme is known. Another case is given by restricting the number of sets an element can appear in. For example, in the vertex cover problem, each set can have a size of at most two. A 2-approximation is known in this case [101, 106]. More generally, an $f_{\max}$-approximation is known for vertex covering hypergraphs with edges that include at most $f_{\max}$ vertices.

Matroid Intersection Cover Problem. Several results are known for the special case of $l = 2$. Kőnig's [72] famous line coloring theorem can be interpreted in terms of matroid intersection covers. Given two k-colorable partition matroids, the intersection is also k-colorable. The statement does not hold for matroids in general. For an example see [92, Section 42.6c]. An upper bound for two general matroids was proven by Aharoni and Berger [2]. They showed that given two matroids M_1 and M_2 with color numbers k_1, k_2 respectively the color number of the intersection of the two matroids is at most $2\max\{k_1, k_2\}$. Matroid Intersection Cover is NP-hard for $l \geq 3$. This follows from the fact that it is NP-hard to decide, whether a 3-partite, uniform hypergraph of degree 3 is 3-edge colorable [85].

Im et al. [60] studied the Matroid Intersection Cover problem on a collection of l matroids. Given a (b_i, c_i)-decomposition of M_i for all $i \in [l]$, they present a polynomial-time algorithm that computes a cover of the intersection of size at most $\left(\prod_{i\in[l]} b_i\right) \cdot \left(\sum_{i\in[l]} c_i\right)k^*$, where $k^* = \max_{i\in[l]} k_i$. See Section 3.1.2 for more details. As already mentioned above this implies constant-factor approximation algorithms if each of the constant number of matroids is either a partition matroid, a transversal matroid, a graphic matroid, a laminar matroid or a gammoid.

A related problem is the existence of a partition into common bases. Given two matroids M_1, M_2, can we partition the ground set into k sets such that all of them are bases in both matroids? Since the intersection of two matroids forms a downward closed family, a cover and a partition of the ground set is essentially the same. The crucial difference is the additional property that we require the sets of the partition to be bases. Bérczi and Schwarcz [17] show that it is already NP-hard to decide whether there exists a partition into two common bases if one matroid is a partition and the other one a linear matroid.

Matroid Intersection Problem. In the literature, the term Matroid Intersection problem usually refers to the special case of $l = 2$. That is, given two matroids $M_1 = (S, \mathcal{I}_1)$ and $M_2 = (S, \mathcal{I}_2)$, find a set of maximum cardinality (or weight) that is independent in M_1 and M_2. It is well-known that Edmond's [38] Matroid Intersection

Algorithm solves this problem in polynomial-time [92].

For $l \geq 3$, Matroid Intersection is NP-hard by a reduction from Hamiltonian Path on directed graphs [105]. A result by Fisher et al. [43] implies that Greedy is a $\frac{1}{l}$-approximation algorithm for finding an independent set of maximum cardinality in the intersection of l matroids. Lee et al. [75] presented an $\frac{1}{l+\epsilon}$-approximation algorithm using local search techniques. For the special case of $l = 3$ Linhares et al. [78] gave a 2-approximation algorithm.

Matroid Cover/Coloring Problem. This is the special case of Matroid Intersection Cover for $l = 1$. Given a single matroid $M = (S, \mathcal{I})$, find a collection $\mathcal{C} \subseteq \mathcal{I}$ of minimum size that covers S. Note that this is equivalent to partitioning S in as few independent sets as possible. Therefore, the Matroid Cover problem is sometimes also referred to as Matroid Coloring problem. Matroid Cover can be solved in polynomial time, since it can be reduced to finding a maximum independent set in the intersection of two matroids. For more details on the reduction, see for example [74, Chapter 8].

Zhou [108] studied the Minimum Partitioning problem on downward-closed systems. In contrast to matroids, it is NP-complete to partition a downward-closed set system into a minimum number of independent sets. This follows directly from the fact that graph coloring is a special case. Given a graph on a vertex set V, choose V to be the ground set of elements. Let $\mathcal{I}$ be the set of independent sets of the graph, then $(V, \mathcal{I})$ forms an independent set system and a minimum partition corresponds to a coloring of G with a minimum number of colors. See [108] for an overview of connections of the Minimum Partition problem on downward-closed set systems to other combinatorial optimization problems and lower bounds.

3.1.5 Our Results

Our main result is a partition reduction of a k-colorable gammoid. This reduction ensures that the coloring number of the partition matroid is at most $2k - 2$. Previous lower bounds [18] imply that this is the best reduction possible in terms of the coloring number.

Theorem 3.6
A partition reduction from a k-colorable gammoid M to a $(2k-2)$-colorable partition matroid can be computed in polynomial time given a directed graph D that represents M as input.

Combining our main result, Theorem 3.6, with Proposition 3.3 from [60] we obtain significantly better approximation guarantees for Matroid Intersection Cover problems in which one of the matroids is a gammoid.

Corollary 3.7
Given any collection of l matroids on the same ground set S where each matroid is either a graphic matroid, paving matroid, or gammoid, let $\mathcal{I}$ be the intersection of their

independent sets. There is a polynomial-time algorithm to compute a $2l$-approximation of the Set Cover of S using sets in $\mathcal{I}$.

In the following, we give an example of a Matroid Intersection Cover problem in which one of the matroids is a gammoid. Initially, assume that the input consists of a directed graph D with a designated file server location (a sink) and a collection of clients requesting files from the server at various locations in the networks (the sources). Every time step, a collection of clients, for which there exist disjoint paths to the server, can be served. The goal is to get every client, as quickly as possible, the requested file from the server. The problem described above is a Matroid Cover problem that can be solved to optimality in polynomial-time [37].

Now additionally assume that sets of clients are employed by different companies. There is a Service Level Agreement (SLA) for each company that upper bounds how many clients from a particular company can be serviced per time unit. With the additional constraints, the problem becomes a Matroid Intersection Cover problem, where the intersecting matroids are a gammoid and a partition matroid. Using the $(18, 1)$-decomposition of a gammoid and Proposition 3.3 from [60] one obtains a polynomial-time 36-approximation algorithm. However, combining the $(1, 2 - \frac{2}{k})$-decomposition of a gammoid from Theorem 3.6 with Proposition 3.3 from [60] we now obtain a polynomial-time 3-approximation algorithm.

Another algorithmic consequence is an efficient algorithm for list coloring the intersection $M_1 \cap M_2$ of a k_1-colorable matroid M_1 and a k_2-colorable matroid M_2 if the list of allowable colors for each element has cardinality at least $2\max(k_1, k_2)$, and each of the matroids is either a graphic matroid, paving matroid, transversal matroid, or gammoid. Casting this into the context of our running file server example implies that, additionally, each client has a list of allowable times when the file transfer may be scheduled. Our partition reduction of a gammoid then yields an efficient algorithm to find a feasible schedule as long as the cardinality of allowable times for each client is at least $2\max(k_1, k_2)$, where k_1 is the time required if the network had infinite capacity (so only the Service Level Agreement constraints come into play), and k_2 is the time required if the SLA allowed infinitely many file transfers (so only the network capacity constraints come into play).

Set Cover is a canonical algorithmic problem. Hence, there is considerable interest in discovering examples of natural special types of set cover instances that allow better approximation algorithms than the well-known $H(n)$ and $f_{\max}$-approximations (see Section 1.1.4). Our results provide another example of such a natural special case, namely when the sets come from the intersection of a small number of standard combinatorial matroids.

Finally, Theorem 3.6 and its proof reveal structural properties of gammoids that would seem likely to be of use to address future research on gammoids.

3.1.6 Overview of Techniques

Given a graphic representation of a gammoid, an optimal coloring can be computed in polynomial-time [37]. By superimposing the source-sink paths for the various color classes, one can obtain a flow f from the sources to the sinks that moves at most k units of flow over any vertex. Using standard cycle-canceling techniques [5] one can then convert f to what we call an acyclic flow. A flow f is acyclic if for every undirected cycle C in D at least one edge in C either has flow k or has no flow. Thus, by deleting edges that support no flow in f, as they are unnecessary, we are left with a forest $\mathcal{T}$ of edges that have flow in the range $[1, k-1]$ and a collection of disjoint paths, which we call highways, that have flow k. See Figure 3.3.

Now each part X in the computed partition $\mathcal{X}$ will be entirely contained in one tree $T \in \mathcal{T}$, and the parts X in a tree $T \in \mathcal{T}$ are computed independently of other trees in $\mathcal{T}$. There can be four types of vertices in T: (1) sources s that have outflow 1, (2) source portals $\tilde{s}$, which are vertices that have a highway directed into them, and which have outflow k in T, (3) sink portals $\tilde{z}$, which are vertices that have a highway directed out of them, and which have inflow k in T, and (4) normal vertices.

We give a recursive partitioning algorithm for forming the parts X in a tree $T \in \mathcal{T}$. In each recursive step, our algorithm first identifies a single part X of at most $2k-2$ sources and an associated sink portal that are in some sense near each other on the edge of T. The algorithm then removes these sources and sink portal from T, and reconnects disconnected sources back into appropriate places in T. The algorithm then recurses on this new tree T.

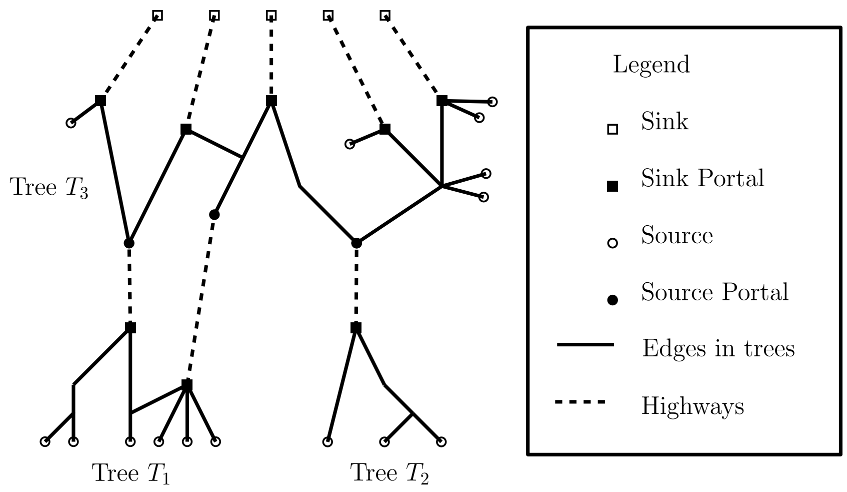


Fig. 3.3: Example of trees created. Here $k = 3$. Source portals are matched to sink portals along a path not in the trees. All sink portals have k units of flow entering them and source portals have k units leaving.

Most of the proof that our partitioning algorithm produces a $(1, 2 - \frac{2}{k})$-decomposition focuses on routing individual trees in $\mathcal{T}$. So let Y be a collection of sources such that $|Y \cap X| \leq 1$ holds for all $X \in \mathcal{X}$. The first key part is proving that as the partitioning algorithm recurses on a tree T, it is always possible to route both the flow coming into T, and the flowing emanating within T, out of T, without routing more than k units of flow through any vertex in T. Note that as the algorithm recurses, the tree T loses a sink portal (which reduces the capacity of the flow that can leave T by k) and loses up to $2k - 2$ sources (which means there is less flow emanating in T that has to be routed out).

The second key part is to prove that there is a vertex-disjoint routing from the source portals in T and the sources in $T \cap Y$ to the sink portals in T. To accomplish this, we trace our partitioning algorithm's recursion backwards. So in each step a new collection X of sources and a sink portal is added back into T. We then prove by induction that no matter how the previously considered sources in Y were routed, there is always a feasible way to route the chosen source in $Y \cap X$ to a sink portal in T. We finish by observing that unioning the routings constructed within the trees with the highways gives a feasible routing for Y.

3.2 The Partition Reduction Algorithm

This section gives the Partition Reduction algorithm. First, we define a corresponding flow graph. Using a Cycle-Canceling algorithm, we decompose the flow graph into a collection of trees. Then we algorithmically create the partitions from the local structure in these trees. The analysis of the algorithm is deferred to the next section.

Let $D = (V, E)$ be a directed graph that represents a gammoid. Let $S \subseteq V$ be a set of sources, and $Z \subseteq V$ be the collection of sinks. We may assume without loss of generality that:

- Each vertex $v \in V$ has either out-degree 1 or in-degree 1.
- Each source $s \in S$ has in-degree 0 and out-degree 1.
- Each sink $z \in Z$ has in-degree 1 and out-degree 0 and $|Z| = r$.
- If (u, v) is an edge in E, then (v, u) is not an edge in E.

We assume without loss of generality that all color classes have full rank, that is $|S| = rk$. This can be assumed by adding dummy sources to M.

Definition 3.8

- A **feasible flow** in a digraph D from a collection $S' \subset S$ is a collection of paths $\{p^s \mid s \in S'\}$ such that (1) p^s is a simple path from s to some sink, and (2) no vertex or edge in D has more than k such paths passing through it.

- A **feasible routing** in a digraph D from a collection $S' \subset S$ is a collection of paths $\{p^s \mid s \in S'\}$ such that (1) p^s is a simple path from s to some sink, and (2) no vertex or edge in D has more than one such path passing through it.

3.2.1 Defining the Flow Graph

Given the digraph D, we can compute a minimum k such that M is k-colorable in polynomial time using a polynomial-time algorithm for Matroid Intersection [86]. Further we can compute the collection of resulting color classes $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$. So $\mathcal{C}$ is a partition of the sources S, and for each $C_i \in \mathcal{C}$ there exist r vertex-disjoint paths $p_i^1, \dots p_i^r$ in the digraph D from the r sources C_i to Z. We create an f where the flow $f(u,v)$ on each edge (u,v) is initialized to the number of paths p_i^j that traverse (u,v), that is

$$f(u,v) = \sum_{i=1}^{k} \sum_{j=1}^{r} \mathbb{1}_{(u,v) \in p_i^j}.$$

A flow f is acyclic if for every undirected cycle C in D at least one edge in C either has flow k or has no flow in f. An arbitrary flow can be converted acyclic by finding cycles in a residual network D^r. This is standard [5], but for completeness we define it here. For every directed edge (u,v) with $f(u,v) < k$ there exists a forward directed edge (u,v) in D^r with capacity $c_r(u,v) := k - f(u,v)$. For every directed edge (u,v) with $f(u,v) > 0$ there exists a backward directed edge (v,u) in D^r with capacity $c_r(v,u) := f(u,v)$. An augmenting cycle in D^r is a simple directed cycle with strictly more than two edges.

Cycle-Canceling Algorithm. While there exists an augmenting cycle C do the following:

- Let $c := \min_{(u,v) \in C} c_r(u,v)$ be the minimum capacity of an edge in C.
- For each forward edge $(u,v) \in C$, increase $f(u,v)$ by c.
- For each backward edge $(u,v) \in C$, decrease $f(u,v)$ by c.

As every iteration increases the number of edges that have flow k in f or that have no flow in f by one, the Cycle-Canceling algorithm terminates after at most $|E|$ iterations. The following observations are straight-forward.

Observation 3.9
The following properties hold when the Cycle-Canceling algorithm terminates:

- *f is a feasible flow of kr units of flow from all the sources.*
- *Every undirected cycle C in D contains at least one edge with flow k in f or one edge with no flow in f.*

- *The collection of edges in D that has flow strictly between 0 and k in f forms a forest.*
- *The collection of edges in D with flow k in f are a disjoint union of directed paths, which we call* ***highways****.*

3.2.2 Properties of the Acyclic Flows

We now give several definitions and straightforward observations about our acyclic flow f that will be useful in our algorithm design and analysis.

Definition 3.10

- A vertex v is a source portal if its in-degree in D is 1, and it has k units of flow passing through it in f.
- A vertex v is a sink portal if its out-degree in D is 1, and it has k units of flow passing through it in f.
- Let $\mathcal{T}$ be the forest consisting of edges in D that have flow in f strictly between 0 and k.
- For a tree $T \in \mathcal{T}$ and a vertex $v \in T$ define T_v to be the forest that results from deleting the vertex v from T.

Observation 3.11
Each sink $z \in Z$ is in a tree $T \in \mathcal{T}$ that consists solely of z.

Proof. By assumption, the sink z has in-degree 1 in D and all color classes $\mathcal{C}$ have full rank. Hence, k units of flow are entering z through a unique edge. □

As our Partition Reduction algorithm partitions each tree $T \in \mathcal{T}$ independently, it is notationally more convenient to fix an arbitrary tree $T \in \mathcal{T}$, and make some definitions relative to this fixed T, and make some observations that must hold for any such T. To a large extent these observations are intended to show that the Figure 3.4 is accurate.

Definition 3.12

- Let $\tilde{S}$ be the collection of source portals in tree T.
- Let $\tilde{Z}$ be the collection of sink portals in tree T.
- A normal vertex is a vertex that is none of a source, a sink, a source portal, nor a sink portal.

Definition 3.13

- A **feasible flow** in T from a collection $S' \subset S$ is a collection of paths, one path p^s for each $s \in S'$ and k paths $p_1^{\tilde{s}}, \dots p_k^{\tilde{s}}$ for each source portal $\tilde{s} \in \tilde{S}$ such that (1) p^s is a simple path from s to some sink portal, (2) each $p_i^{\tilde{s}}$ is a simple path from $\tilde{s}$ to a sink portal, and (3) no vertex or edge in T has more than k such paths passing through it.

- A **feasible routing** in T from a collection $S' \subset S$ is a collection of paths, one path p^s for each $s \in S'$ and one path $p^{\tilde{s}}$ for each source portal $\tilde{s} \in \tilde{S}$ such that (1) p^s is a simple path from s to some sink portal, (2) $p^{\tilde{s}}$ is a simple path from $\tilde{s}$ to a sink portal, and (3) no vertex or edge in T has more than one such paths passing through it.

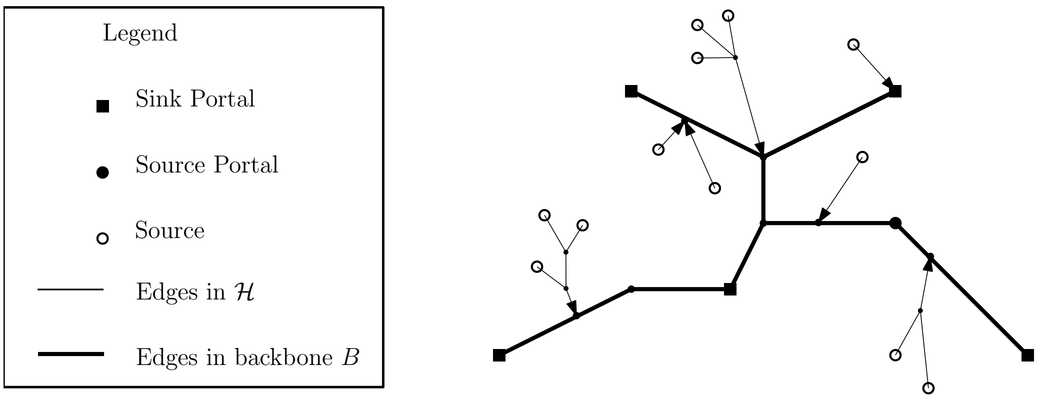


Fig. 3.4: Backbone of a tree.

The following observation holds for trees in $\mathcal{T}$ initially and gives intuition for the structure of $\mathcal{T}$. We remark that this observation may not hold for all trees throughout the execution of our algorithm.

Observation 3.14
The number of sources in T is an integer multiple of k.

Proof. This follows from the fact that each source portal $\tilde{s} \in T$ has exactly k units of flow coming into T via $\tilde{s}$ in the flow f and each sink portal $\tilde{z} \in T$ has exactly k units of flow leaving T via $\tilde{z}$ in f. □

Definition 3.15
- For two vertices $u, v \in T$, let $P(u, v)$ be the unique undirected path from u to v in T.
- The **backbone** B of T is the subgraph of T consisting of the union of all paths in between pairs of sink portals in T, that is $B = \bigcup_{\tilde{y} \in \tilde{Z}} \bigcup_{\tilde{z} \in \tilde{Z}} P(\tilde{y}, \tilde{z})$.
- For the backbone B, let B_v be the induced forest that results from deleting v from B.

- A vertex v in a backbone B is a **branching vertex** if either:
  - v is not a sink portal and the forest B_v contains at least two trees that each contain exactly one sink portal, or
  - v is a sink portal and the forest B_v contains at least one tree that contains exactly one sink portal.
- Let $\mathcal{H}$ be the forest that results from deleting the edges in B from the tree T.
- For two vertices $u, v \in B$, let $S(P(u, v))$ be the sources $s \in S$ such that there exists a tree $H \in \mathcal{H}$ such that $s \in H$ and such that H contains a vertex $w \in P(u, v)$. Intuitively these are the sources in trees in $\mathcal{H}$ hanging off vertices of the path $P(u, v)$.
- Let $S(v)$ denote $S(P(v, v))$.

Observation 3.16
If $\tilde{s} \in \tilde{S}$ is a source portal in T then $\tilde{s}$ is in the backbone B and $\deg_B^+(\tilde{s}) \geq 2$, that is $\tilde{s}$ has out-degree at least 2 in B.

Proof. By definition, $\tilde{s}$ has a unique incoming edge which is saturated in f, and at least one outgoing edge in T that is not saturated in f. Hence, $\deg_B^+(\tilde{s}) \geq 2$. By flow conservation, there have to be at least two directed paths from $\tilde{s}$ to two different sink portals in T. This implies that $\tilde{s}$ is in the backbone B. □

Observation 3.17
If B contains at least two sink portals, then B contains a branching vertex v.

Proof. Consider an arbitrary vertex $v \in B$. If v is not a branching vertex, then there must be a subtree $T' \in B_v$ that contains two sink portals. One can then recurse on T' to find a branching vertex. □

Observation 3.18
If $s \in S$ is a source in T then s is not in the backbone B.

Proof. Since every source s has out-degree 1 in D, it cannot be on a path between a pair of sink portals in T. □

Observation 3.19
For each tree $H \in \mathcal{H}$ it must be the case that all edges in H are directed towards the unique vertex w in H that is also in B.

Proof. This follows from the fact that $H \setminus \{w\}$ cannot contain a sink portal. □

Observation 3.20
Assume that T has at least two sink portals. Let v be a branching vertex. Let T' be a tree in the forest B_v that contains exactly one sink portal $\tilde{z}$. Then the following must hold:

- $T' = P(v, \tilde{z}) \setminus \{v\}$.
- *If T' contains a source portal $\tilde{s}$, then* $\deg_B^+(\tilde{s}) = 2$.
- *The path T' contains at most one vertex y such that* $\deg_B^+(y) = 2$.

Proof. The first statement follows from the definition of B and the fact that T' only contains one sink portal. The second statement follows since every vertex on a path other than its endpoints has degree two. For the last statement assume, to reach a contradiction, that there were two such y's, y_1 and y_2 with y_1 being closer to v in B. Then the flow leaving y_1 toward $\tilde{z}$ could not be feasibly routed through y_2. □

3.2.3 Description of the Partition Reduction Algorithm

Given the collection of trees $\mathcal{T}$, our Partition Reduction algorithm returns a partition $\mathcal{X}$ of the sources in S. The algorithm iterates through the trees T in $\mathcal{T}$ and partitions the sources in T based on their locality in T. So let us consider a particular tree $T \in \mathcal{T}$.

The algorithm performs the first listed case below that applies, with the base cases being checked before the other cases. In the non-base cases the tree T is modified, and the algorithm called tail-recursively on the modified tree. After the algorithm description, we show that the algorithm maintains the invariant that there is a feasible flow on the tree T throughout the recursion.

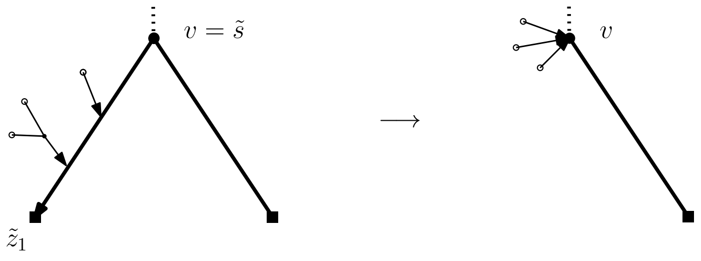
Base Case A: If T contains no sources then the recursion terminates, and the algorithm moves to the next tree in $\mathcal{T}$.
Base Case B: Otherwise, if T contains at most $2k - 2$ sources and no source portal then these sources are added as a part X in $\mathcal{X}$. The recursion terminates, and the algorithm then moves to the next tree in $\mathcal{T}$.

We perform the following recursively on T if neither base case holds. Let v be an arbitrary branching vertex in B. We show the existence of such a vertex in Observation 3.21.

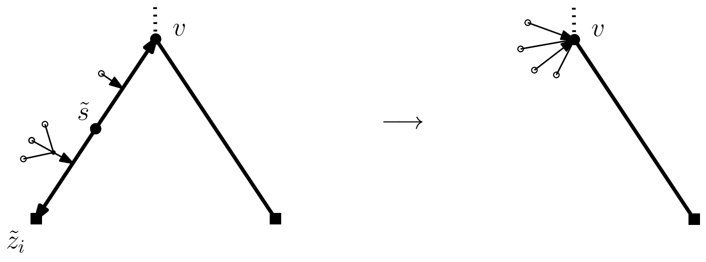
Let $\tilde{z}_1$ be a sink portal in some tree T_1 in T_v that only contains one sink portal. If v is not a sink portal, let $\tilde{z}_2$ be a sink portal in some tree T_2, where $T_1 \neq T_2$, in T_v that only contains one sink portal. If v is a sink portal let $\tilde{z}_2 = v$.

The algorithm's cases are broken up as follows. Case 1 is executed when there is a source portal at v or in T_1 or in T_2. Case 2 is executed when there is a vertex of out-degree 2 in T_1 or T_2 and there is no source portal. Case 3 is everything else.

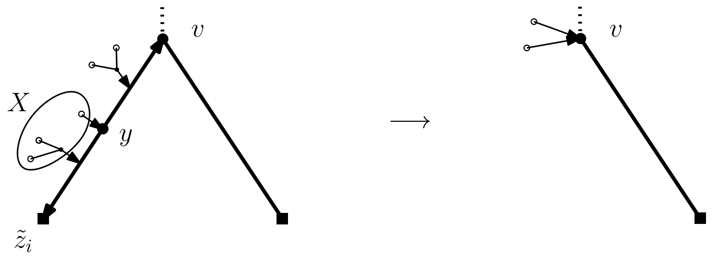
Recursive Case 1a: The vertex v is a source portal. In this case T is modified as follows: (1) for each source $s \in T_1$ a directed edge (s, v) is added to T, (2) v is converted into a normal vertex, and (3) all the nonsources in T_1 are deleted from T. The algorithm then recurses on this new T.



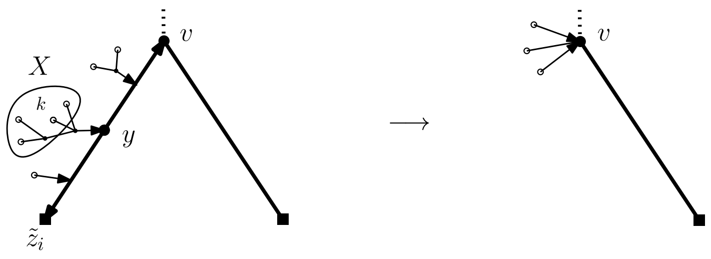
Recursive Case 1b: In this case for some $i \in \{1, 2\}$ the path $P(v, \tilde{z}_i) \setminus \{v\}$ contains a source portal. In this case T is modified as follows: (1) for each source $s \in T_i$ a directed edge (s, v) is added to T, and (2) all the nonsources in T_i are deleted from T. The algorithm then recurses on this new T.



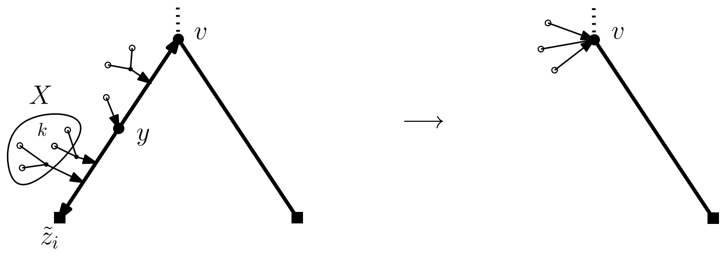
Recursive Case 2a: In this case for some $i \in \{1, 2\}$, the path $P(v, \tilde{z}_i) \setminus \{v\}$ contains a vertex y with $\deg^+_B(y) = 2$ and $|S(P(y, \tilde{z}_i))| \leq 2k - 2$. Add the sources in $S(P(y, \tilde{z}_i))$ as a part X to $\mathcal{X}$. The tree T is then modified as follows: (1) for each source $s \in T_i - X$ a directed edge (s, v) is added to T, and (2) the sources in X and all the nonsources in T_i are deleted from T. The algorithm then recurses on this new T.



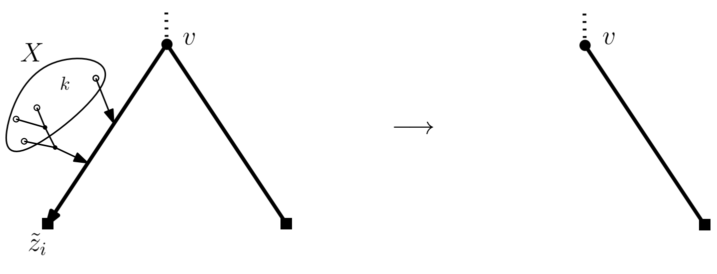
Recursive Case 2b: In this case for some $i \in \{1,2\}$, the path $P(v, \tilde{z}_i) \setminus \{v\}$ contains a vertex y with $\deg_B^+(y) = 2$ and $|S(y)| = k$. In this case the algorithm adds the k sources in $S(y)$ as a part X to $\mathcal{X}$. The tree T is then modified as follows: (1) for each source $s \in T_i - X$ a directed edge (s, v) is added to T, and (2) the sources in X and all the nonsources in T_i are deleted from T. The algorithm then recurses on this new T.



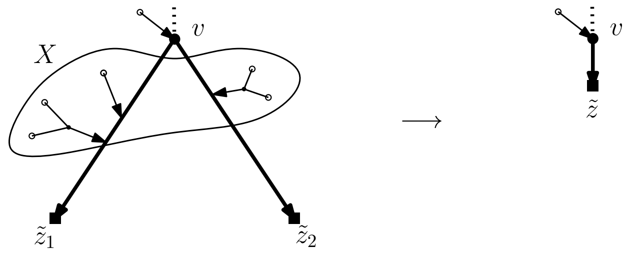
Recursive Case 2c: In this case for some $i \in \{1,2\}$, the path $P(v, \tilde{z}_i) \setminus \{v\}$ contains a vertex y with $\deg_B^+(y) = 2$ and $|S(P(y, \tilde{z}_i) \setminus \{y\})| = k$. In this case the algorithm adds the k sources in $S(P(y, \tilde{z}_i) \setminus \{y\})$ as a part X to $\mathcal{X}$. The tree T is then modified as follows: (1) for each source $s \in T_i - X$ a directed edge (s, v) is added to T, and (2) the sources in X and all the nonsources in T_i are deleted from T. The algorithm then recurses on this new T.



Recursive Case 3a: In this case for some $i \in \{1,2\}$ T_i contains exactly k sources. Add the sources in T_i as a part X to $\mathcal{X}$. The tree T is modified by deleting T_i. The algorithm then recurses on this new T.



Recursive Case 3b: The set of sources in $T_1 \cup T_2$ are added as a part X in $\mathcal{X}$. The tree T is modified by deleting the vertices in T_1 and T_2. Add a new sink portal $\tilde{z}$ together with a directed edge $(\tilde{z}, v)$ to T. The algorithm then recurses on this new T.



3.3 Analysis of the Partition Reduction Algorithm

Our goal is to show that the partition matroid, represented by the partition constructed from the trees, indeed corresponds to a feasible partition reduction from the gammoid. The analysis has the following key components.

- Every tree T has a corresponding feasible flow throughout the algorithm.
- Every part X of the partition has size at most $2k-2$ and all sources are in some part.
- Any collection of sources Y such that $|Y \cap X| \leq 1$ for all $X \in \mathcal{X}$ is in $\mathcal{I}$ and, therefore, can each route a unit of flow to the sink in D.

3.3.1 Trees Always have a Feasible Flow

This section's goal is to show that each tree has a feasible flow as defined in Definition 3.12 throughout the execution of the algorithm. Later, we use this property to prove that our partition indeed represents a partition matroid that corresponds to a feasible partition reduction in the following section.

We begin by showing various invariants hold for each tree when a feasible flow exists. In particular, this implies that a branching vertex exists if any of the recursive cases are executed. Moreover, arriving at Cases (3a) and (3b) ensure the existence of T_1 and T_2. All together, this with the fact that each tree has a feasible flow establishes that the algorithm always has a case to execute if $\mathcal{T}$ is non-empty. The following observation shows that a branching vertex exists if neither base case holds.

Observation 3.21
Fix a tree $T \in \mathcal{T}$ during the execution of the algorithm and say T supports a feasible flow as defined in Definition 3.12. If neither of the base cases apply, then T contains at least two sink portals. Moreover, a branching vertex must exist in T in this case.

Proof. This observation holds because T must contain either more than $2k-2$ source portals or a source portal along with at least one source. In either case, we require two sink portals to support the strictly more than k units of flow from these sources and source portal. A branching vertex must then exist by Observation 3.17. □

Observation 3.22
Fix a tree $T \in \mathcal{T}$ during execution of the algorithm and say T supports a feasible flow as defined in Definition 3.12. Say that T has a branching vertex v with a tree T_i containing exactly one sink $\tilde{z}_i$. Moreover, say that there is no vertex with out-degree 2 in $P(v, \tilde{z}_i)$. It is the case that $P(v, \tilde{z}_i)$ is a directed path from v to $\tilde{z}_i$.

Proof. No vertex with out-degree 2 is in $P(v, \tilde{z}_i)$. Thus, $P(v, \tilde{z}_i)$ is either a path from v to $\tilde{z}_i$ or from $\tilde{z}_i$ to v. Sink portals always have out-degree 0, so the observation follows. We note that, sink portals have out-degree 0 initially and are never given outgoing edges by the algorithm. □

The next observation shows that a branching vertex is not a sink portal when Cases (3a) or (3b) are executed.

Observation 3.23
Fix a tree $T \in \mathcal{T}$ during execution of the algorithm and say T supports a feasible flow as defined in Definition 3.12. Say that T has a branching vertex v and a corresponding tree T_i with exactly one sink. If $P(v, \tilde{z}_i) \setminus \{v\}$ does not contain a vertex of out-degree 2 in B, then v is not a sink portal.

Proof. Observation 3.22 implies that $P(v, \tilde{z}_i)$ is a directed path form v to $\tilde{z}_i$. Sink portals always have out-degree 0, so the observation follows. We note that, sink portals have out-degree 0 initially and are never given outgoing edges by the algorithm. □

The previous observations guarantee the algorithm always has a case to execute if a feasible flow exists in all trees. The next lemma guarantees the existence of a feasible flow.

Lemma 3.24
Fix any tree T during the execution of the algorithm. There must be a feasible flow in T as described in Definition 3.12.

Proof. The statement trivially holds at the beginning of the algorithm. In particular, each edge of D supports at most k units of flow. Initially, every vertex in each tree T has out-degree or in-degree 1, ensuring that no vertex supports more than k units of flow. We remark that during the execution of the algorithm it can happen that there are vertices with both out- and in-degree more than one in some trees.

We inductively show that it holds after a single iteration. Let T^b be the tree at the beginning of the iteration and T^e the tree at the end. Let f^b be the initial flow on T^b

that is feasible and we use this to construct a flow f^e on T^e. We break the proof into the case that gets executed.

Consider Recursive Case (1a). Let S' be the set of sources from the tree T_1 which have newly created edges into v. There must be at least $|S'|$ units of flow in f^b departing v that (1) do not enter T_1 and (2) originate from sources in T_1 or the source portal v. This is because T_1 has one sink $\tilde{z}_1$ that can support at most k units of flow. Let $F = \{f_1, f_2 \ldots\}$ denote these units of flow from v to sinks. Set f^e to include all flow paths in f^b except those in F. Note these paths never enter the deleted vertices or edges. Next observe that the sink $\tilde{z}_1$ is removed and v is no longer a source portal. Thus, we simply need to find a way to send flow for the sources in S'. These vertices each route one unit of flow to their new edge directly to v and then choose a unique flow path in F and add these paths to f^e. Notice that no vertex or edge capacity is violated by definition of f^b and F.

Consider Recursive Cases (1b), (2b) and (2c). Again, let S' be the set of sources which have newly created edges into v. Let T_i be the tree that initially contained the sources. In this case there must be exactly $|S'|$ units of flow in f^b entering v from vertices in T_i. This is because in Case (1b) T_i contains one source portal, one sink portal and there are $|S'|$ sources in T_i. In Case (2b), T_i contains one sink portal, $|S(y)| = k$ and there are $|S'|$ other sources in T_i. In Case (2c), T_i contains one sink portal, $|S(P(y, \tilde{z}_i) \setminus \{y\})| = k$ and there are $|S'|$ other sources in T_i. Where this flow goes in f^b will exactly correspond to where the flow in f^e goes for the sources that now connect to v. That is, f^e has the same flow on every edge as f^b for the edges shared by T^b and T^e. Additionally, there is one unit of flow from each of the $|S'|$ sources on the new edge that connects to v.

Consider Recursive Case (2a). In this case the vertex y has out-degree two and the sources $S(P(v, y) \setminus \{v \cup y\})$ have a new direct edge into v. Notice that any vertex in $S(P(v, y) \setminus \{v \cup y\})$ cannot reach $\tilde{z}_i$ (since they can't route through y to $\tilde{z}_i$). Thus, these sources route to a sink through v. In this case, f^e is the same as f^b except these sources now directly send their flow to v and then follow their remaining path to a sink as in f^b.

Consider Recursive Case (3a). Let T_i be the tree with exactly k sources. There is one sink portal in T_i. No flow in f^b can enter T_i via v. This is because then v has an outgoing edge in T to a vertex in T_i by Observation 3.22. Then all k sources in T_i and at least one unit of flow from v must route to $\tilde{z}_1$, contradicting that $\tilde{z}_1$ receives at most k units of flow in f^b. Set f^e to be the same as f^b for all shared edges between T^b and T^e, except remove any flow sent through v from sources in T_i in f^b.

Consider Recursive Case (3b). We know v is not a sink portal by Observation 3.23. Therefore, T_1 and T_2 exist, each with a unique sink portal. At most k units of flow in f^b can route through v by definition of f^b. The flow f^e is set to the same as f^b on the shared edges between T^b and T^e. Further, the flow from v to sinks $\tilde{z}_1$ and $\tilde{z}_2$ in f^b now go directly to the new sink added that is adjacent to v in T^e. This sink must receive at most k units of flow. □

3.3.2 Bounding the Size of the Parts in the Partition

This section shows that every source is in some part X in $\mathcal{X}$ and that every $X \in \mathcal{X}$ has size at most $2k-2$. Thus, we have a valid partition with each part having the desired size.

Lemma 3.25
It is the case that $|X| \leq 2k-2$ for all $X \in \mathcal{X}$. Moreover, every source in S is in some set in $\mathcal{X}$.

Proof. It is easy to see that every source in S is in some set in $\mathcal{X}$. This is because sources are always contained in some tree of $\mathcal{T}$ until they are added to a set placed in $\mathcal{X}$ and the algorithm stops once there is no tree in $\mathcal{T}$.

Now we show how to bound the size of sets in $\mathcal{X}$. Cases (1a) and (1b) do not add a set to $\mathcal{X}$. Cases (2b), (2c) and (3a) add a set to $\mathcal{X}$ of size k by definition. Case (2a) adds a set of size $2k-2$.

Case (3b) is more interesting. Consider the execution of this case on a tree T with branching vertex v. Let $\tilde{z}_1$ and $\tilde{z}_2$ be the corresponding sinks. We know v is not a sink portal by Observation 3.23 and therefore T_1 and T_2 both exist. By Observation 3.22, the paths from v to $\tilde{z}_1$ and $\tilde{z}_2$ are directed paths from v to $\tilde{z}_1$ and from v to $\tilde{z}_2$. Hence, neither T_1 nor T_2 contains more than k sources. Since case (3a) does not hold, T_1 and T_2 has strictly less than k sources. Thus, there are at most $2k-2$ sources in the set added to $\mathcal{X}$. □

3.3.3 Routing Sources within a Tree

In this section, we show that the algorithm is indeed a partition reduction from a k-colorable gammoid to a $(2k-2)$-colorable partition matroid. That is, we show that every set Y, that is independent in the partition matroid represented by $\mathcal{X}$ is also independent in the gammoid. More specifically, for any set Y where $|Y \cap X| \leq 1$ for all $X \in \mathcal{X}$ it is the case that $Y \in \mathcal{I}$ or, equivalently, there is a feasible routing in the digraph D from the sources in Y. The key to this is Lemma 3.26 which essentially states that there exists a feasible routing in each tree $T \in \mathcal{T}$.

Lemma 3.26
Let T be an arbitrary tree in $\mathcal{T}$. Let $\mathcal{X}_T$ be the parts in $\mathcal{X}$ that are also in T. For a set Y with $|Y \cap X| \leq 1$ for all $X \in \mathcal{X}$, let Y_T be the subset of sources in Y that are also in T. Then there is a feasible routing R in T from Y_T to a subset of $\tilde{Z}$.

Proof. For convenience we consider one fixed tree T and drop T for the notation. The proof is by reverse induction on the recursion in our partitioning algorithm. So the base case of the induction will be the base cases of the partitioning algorithm.

Consider Base Case (A). In this case T does not contain any sources. By Lemma 3.24 there exists a feasible flow. Scaling by a factor of $\frac{1}{k}$ implies that we can route one unit of flow from every source portal in $\tilde{S}$ to the collection of sink portals, such that no more than one unit of flow is routed through any edge or vertex. Then there must also exist an integer flow, which equals a routing in T.

Consider Base Case (B). In this case $\mathcal{X}$ consists of a single part X and there are no source portals in T. Let s be the unique source in $X \cap Y$. Then s can be routed in T as s is routed in the flow f that is guaranteed to exist by Lemma 3.24.

For the recursive cases we construct a routing R^b for the state T^b of the tree T before the recursive call from the routing R^e that inductively exists for the state T^e after the recursive calls.

Consider Recursive Case (1a). Let S' be the collection of sources in T_1. Every source in S' has a directed edge into v in T^e. There can be at most one source $s' \in S' \cap Y$ since by the induction hypothesis R^e is a routing and at most one path can pass through v. If there is no $s' \in S' \cap Y$ then in the routing R^b, v is routed along $P(v, \tilde{z}_1)$ to $\tilde{z}_1$. Otherwise let s' be the unique source in $S' \cap Y$. Let $H' \in \mathcal{H}$ be the tree that contains s' and let w' be the unique vertex in $H' \cap P(v, \tilde{z}_1)$. Then in R^b, v follows the path of s' in R^e and s' is routed through H' to w' and from w' along the path $P(w', \tilde{z}_1)$ to $\tilde{z}_1$. We have to argue that there exists a directed path from v to $\tilde{z}_1$ and from s' to $\tilde{z}_1$ in T_1. Since v is a source portal it has in-degree 0 and, hence, $P(v, \tilde{z}_1)$ does not contain a vertex with out-degree 2. By Observation 3.22, $P(v, \tilde{z}_1)$ is a directed path from v to $\tilde{z}_1$. Observation 3.19 states that there also exists a directed path from s' to w'. We also have to argue that no vertex or edge in T^b has more than one path passing through it. This clearly holds for the edges and vertices shared by T^b and T^e. Since the paths in T_1 are disjoint, it also holds for the edges and vertices that are in T^b and not in T^e.

Consider Recursive Case (1b). Let S' be the collection of sources in T_i. Every source in S' has a directed edge into v in T^e. There can be at most one $s' \in S' \cap Y$, since at most one path can pass through v in R^e. If there is no $s' \in S' \cap Y$, then in the routing R^b the source portal $\tilde{s}$ is routed through $P(\tilde{s}, \tilde{z}_i)$ to $\tilde{z}_i$. Otherwise let s' be the unique source in $S' \cap Y$. Let $H' \in \mathcal{H}$ be the tree that contains s' and let w' be the unique vertex in $H' \cap P(v, \tilde{z}_i)$.

If $w' \in P(\tilde{s}, v)$ then in R^b s' is routed through H' to w' and from w' along the path $P(w', v)$ to v, where it follows the path of s' in R^e. The source portal $\tilde{s}$ is routed through $P(\tilde{s}, \tilde{z}_i)$ to $\tilde{z}_i$. If $w' \in P(\tilde{s}, \tilde{z}_i)$ then in R^b s' is routed through H' to w' and from w' along the path $P(w', \tilde{z}_i)$ to $\tilde{z}_i$. The source portal $\tilde{s}$ is routed through $P(\tilde{s}, v)$ to v, where it follows the path of s' in R^e.

By Observation 3.20, $\tilde{s}$ is the only vertex with out-degree 2 in T_i. Hence, the path $P(\tilde{s}, v)$ is a directed path from $\tilde{s}$ to v and the path $P(\tilde{s}, \tilde{z}_i)$ is a directed path from $\tilde{s}$ to $\tilde{z}_i$. Together with Observation 3.19, which implies that there exists a directed path from s' to w', it follows that the newly created paths above are directed paths from s' and $\tilde{s}$ to $\tilde{Z}$. On the edges and vertices in T^b that are also in T^e the routing R^b equals the routing R^e and in both cases, the new paths in T_i are disjoint. Hence, R^b fulfills the property that no more than one path passes through every vertex or edge in T^b.

Consider Recursive Case (2a). Let $H \in \mathcal{H}$ be the tree that contains the selected source $s \in X$ and let w be the unique vertex in $H \cap P(v, \tilde{z}_i)$. Then in R^b, s is routed through H to w and from w along the path $P(w, \tilde{z}_i)$ to $\tilde{z}_i$. Let S' be the collection of sources in $S(P(v, y) \setminus \{v \cup y\})$. Every source in S' has a directed edge into v in T^e. There can be at most one source $s' \in S' \cap Y$ since at most one route can pass through v in R^e.

If there is such a source $s' \in S' \cap Y$, let $H' \in \mathcal{H}$ be the tree that contains the source s' and let w' be the unique vertex in $H' \cap P(y, v)$. Then in R^b s' is routed through H to w, from w along $P(w, v)$ to v where it follows the path of s' in R^e.

Consider Recursive Case (2b). Let $H \in \mathcal{H}$ be the tree that contains the unique source in $s \in X \cap Y$. Let S' be the collection of sources in $S(P(v, \tilde{z}_i) \setminus \{y\})$. Every source in S' has a directed edge into v in T^e. There can be at most one source $s' \in S' \cap Y$ since at most one route can pass through v in R^e. If there is no $s' \in S' \cap Y$ then in R^b s is routed through H to y and from y along the path $P(y, \tilde{z}_i)$ to $\tilde{z}_i$. Otherwise, let $H' \in \mathcal{H}$ be the tree that contains the unique source $s' \in S' \cap Y$ and let w' be the unique vertex in $H' \cap P(v, \tilde{z}_i)$. If $w' \in P(y, v) \setminus \{y\}$, then in R^b s' is routed through H' to w', from w' along $P(w', v)$ to v where it follows the path of s' in R^e. Then s is again routed through H to y and from y along the path $P(y, \tilde{z}_i)$ to $\tilde{z}_i$.

If $w' \in P(y, \tilde{z}_i) \setminus \{y\}$, then in R^b s is routed through H to y, from y along $P(y, v)$ to v where it follows the path of s' in R^e. The source s' is routed through H' to w' and from w' along $P(w', \tilde{z}_i)$ to $\tilde{z}_i$.

Consider Recursive Case (2c). Let $H \in \mathcal{H}$ be the tree that contains the unique source in $s \in X \cap Y$ and let w be the unique vertex in $H \cap P(y, \tilde{z}_i)$. Let S' be the collection of sources in $S(P(v, y))$. Every source in S' has a directed edge into v in T^e. There can be at most one source $s' \in S' \cap Y$ since at most one route can pass through v in R^e. If there is no $s' \in S' \cap Y$ then in R^b s is routed through H to w and from w along the path $P(w, \tilde{z}_i)$ to $\tilde{z}_i$. Otherwise, let $H' \in \mathcal{H}$ be the tree that contains the unique source $s' \in S' \cap Y$ and let w' be the unique vertex in $H' \cap P(v, y)$. Then in R^b s' is routed through H' to w', from w' along $P(w', v)$ to v where it follows the path of s' in R^e. s is still routed to $\tilde{z}_i$.

In the Recursive Cases (2a), (2b) and (2c), by Observation 3.20, y is the only vertex with out-degree 2 in T_i. Hence, the path $P(y, v)$ is a directed path from y to v and the path $P(y, \tilde{z}_i)$ is a directed path from y to $\tilde{z}_i$. Together with Observation 3.19, which implies that there exists a directed path from s to w and from s' to w', it follows that the newly created paths above are directed paths leaving at s and s'. In all three cases, the routing R^b on the edges and vertices in T^b that are also in T^e equals the routing R^e. In every case, the new paths in T_i are disjoint. Hence in the Recursive Cases (2a), (2b) and (2c), R^b fulfills the property that no more than one path passes through every vertex or edge in T^b.

Consider Recursive Case (3a). Let $H \in \mathcal{H}$ be the tree that contains the selected source $s \in X \cap Y$ and let w be the unique vertex in $H \cap P(v, \tilde{z}_i)$. Then in R^b s is routed through H to w and from w along the path $P(w, \tilde{z}_i)$ to $\tilde{z}_i$. By Observation 3.22, the path $P(v, \tilde{z}_i)$ is a directed path from v to $\tilde{z}_i$. Together with Observation 3.19, which implies that there exists a directed path from s to w, it follows, that the path from s to $\tilde{z}_i$ is a directed path. Since the routing R^b equals the routing of R^e on all edges and vertices shared by T^e and T^b and the route from s to $\tilde{z}_i$ only uses edges and vertices in T_i, also R^b fulfills the property that no more than one path passes through every vertex or edge in T^b.

Consider Recursive Case (3b). Let $H \in \mathcal{H}$ be the tree that contains the selected source $s \in X \cap Y$ and let w be the unique vertex in $H \cap P(v, \tilde{z}_i)$. Then in R^b s is routed through H to w and from w along the path $P(w, \tilde{z}_i)$ to $\tilde{z}_i$. Let $j \in \{1, 2\} \setminus \{i\}$. If there is a source s' routed to $\tilde{z}$ in R^e, then in R^b the route from s' to v equals the route from s' to v in R^e. Instead of going to $\tilde{z}$, we continue along the path $P(v, \tilde{z}_j)$ to $\tilde{z}_j$. By Observation 3.22, for $i \in \{1, 2\}$ the paths $P(v, \tilde{z}_i)$ are directed paths from v to $\tilde{z}_i$. Together with Observation 3.19, which implies that there exists a directed path from s to w, it follows that the path from s to $\tilde{z}_i$ and from v to $\tilde{z}_j$ are directed paths. Since the routing R^b equals the routing of R^e on all edges and vertices shared by T^e and T^b and the paths from s to $\tilde{z}_i$ and from v to $\tilde{z}_j$ are vertex-disjoint and in $T_1 \cup T_2$, also R^b fulfills the property that no more than one path passes through every vertex or edge in T^b. □

Lemma 3.27
Let $Y \subset S$ such that for all $X \in \mathcal{X}$ it is the case that $|X \cap Y| \leq 1$. Then there exists a feasible routing from Y in the digraph D.

Proof. This follows from Lemma 3.26 and the fact there is a unique highway into each source portal in each tree and a unique highway leaving each sink portal in every tree. □

3.3.4 Tightness

For the sake of completeness, we present the example by Bérczi et al. [18] for which no reduction to a $(2k-3)$-colorable partition matroid exists. The layered digraph is visualized in Figure 3.5. All edges are directed downwards. The universe S is given by $k(k-1)$ sources that can be partitioned into sets $S_1, S_2, \ldots, S_k$, each of cardinality $k-1$, such that all sources in S_i are connected via an outgoing edge to the vertex v_i of the next layer. The layer $v_1, \ldots, v_k$ is followed by a layer that consists of $k-1$ sinks, connected to the previous layer by a complete bipartite graph.

Consider a minimal partition of S. Clearly, all sources of a set S_i have to be contained in the same part of the partition, since subset of sources in S_i can be routed simultaneously in the digraph. On the other hand, we can have at most $k-1$ parts in the partition, since there are only $k-1$ sinks. Hence, there exist at least two sets S_i and S_j that have to be in the same part of a partition. Which implies that $|X| \geq 2k-2$ for some $X \in \mathcal{X}$. Note that the gammoid of the example is, in fact, also a laminar matroid.

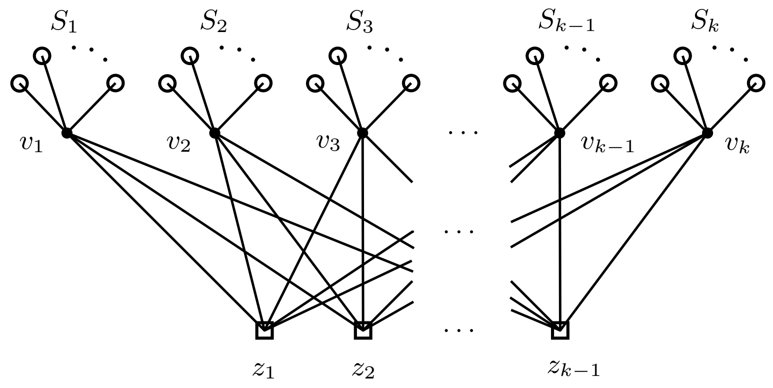


Fig. 3.5: Digraph representing a k-colorable gammoid that cannot be reduced to a $(2k-3)$-colorable partition matroid [18].

3.4 Open Problems

In this chapter, we showed that there exists a polynomial-time algorithm to compute a $(1,2)$-decomposition of a gammoid. This is also true for other standard combinatorial matroids such as graphic matroids, paving matroids and laminar matroids. A natural question to ask is whether representable matroids or matroids in general are $(1,2)$-decomposable.

Problem 3.28
Is every matroid M $(1, O(1))$-decomposable? And if yes, can we compute a $(1, O(1))$-decomposition in polynomial-time?

In general, it is not easy to detect if a given partition reduction is feasible. The challenge is that every set Y in the above description needs to be feasible and there are exponentially many such sets. Finding a polynomial-time algorithm to check if a partition has the property that the partition matroid represented by it corresponds a feasible partition reduction is an interesting open question.

Problem 3.29
Let M be a k-colorable matroid M and let $\mathcal{X}$ be a partition of S, where the maximum size of any part is ck. Is there a polynomial-time algorithm to check if a given partition is a feasible $(1, c)$-decomposition of the k-colorable matroid M?

Chapter 4

Laminar Generalized Min Sum Set Cover

The Generalized Min Sum Set Cover Problem (Gmssc) was introduced as a theoretical framework for re-ranking search results in a web search and is defined as follows. We are given a hypergraph and covering constraints $\kappa(e)$ for every edge e in the hypergraph. For a linear ordering of the hypergraph vertices, we define the cover time of an edge to be the smallest t such that the first t vertices in the linear ordering contain at least $\kappa(e)$-many vertices of the edge e. The task is to find a linear ordering that minimizes the sum of edge cover times.

We study Gmssc on laminar and cross-free hypergraphs. For laminar Gmssc with arbitrary covering constraints, we give a polynomial-time 2-approximation algorithm. For special choices of covering constraints, we provide exact polynomial-time algorithms.

This is joint work with Felix Happach. Parts of this chapter correspond to or are identical to [54].

The chapter is structured as follows. We start by introducing the Generalized Min Sum Set Cover problem and explaining its application to re-ranking search results in a web search. In Section 4.1.2 we summarize previous work on Gmssc and related problems, such as Min Sum Vertex Cover, Min Latency Set Cover, and related single machine scheduling problems. An overview of our results is given in Section 4.1.3.

Section 4.2 is dedicated to the connection of Gmssc to the Ratio problem. We show that the Ratio problem remains NP-complete, even when restricted to a particular choice of weights and processing times. In Section 4.3 we study Min Sum Set Cover on laminar and cross-free families and provide polynomial-time algorithms for both cases. This is followed by a 2-approximation for laminar Gmssc. Finally, we present another special case of laminar Gmssc, which can be solved in polynomial-time.

4.1 Introduction

Azar et al. [8] introduced the Generalized Min Sum Set Cover Problem (Gmssc), which is defined in the following way. We are given a hypergraph $\mathcal{H} = (V, E)$ on n vertices and **covering constraints** $\kappa(e) \in \{1, \dots, |e|\}$ for all $e \in E$. For a linear ordering $\pi : V \to [n]$ we define the **cover time** of an edge with respect to $\kappa(e)$ as

$$\pi_\kappa(e) := \min\{k \in [n] : |\{\pi^{-1}(1), \pi^{-1}(2), \dots, \pi^{-1}(k)\} \cap e| = \kappa(e)\}. \tag{4.1}$$

The task is to find a linear ordering π that minimizes $\sum_{e \in E} \pi_\kappa(e)$. Gmssc generalizes several well-studied optimization problems. The special case of Gmssc with unit covering constraints coincides with the Min Sum Set Cover problem (Mssc). If all covering constraints equal the cardinality of the edge, we obtain the Minimum Latency Set Cover problem (Mlsc). See Section 4.1.2 for more details on these two problems. We start by giving an example of an instance of Gmssc.

Example 4.1
We are given a hypergraph $\mathcal{H} = (V, E)$ with vertex set $V = \{v_1, \dots, v_{10}\}$ and edge set $E = \{e_1, \dots, e_5\}$. The covering constraints of the edges are given by $\kappa(e_1) = 1, \kappa(e_2) = 3, \kappa(e_3) = 2, \kappa(e_4) = 3$, and $\kappa(e_5) = 1$. The instance is displayed in Figure 4.1.

Consider a linear ordering π with $\pi(v_9) = 1$, $\pi(v_1) = 2$, $\pi(v_7) = 3$, $\pi(v_8) = 4$, $\pi(v_6) = 5$, $\pi(v_2) = 6$ and $\pi(v_3) = 7$. Now, the cover time of edge e_5 is given by $\pi_\kappa(e_5) = 1$, since e_5 has covering constraint $\kappa(e_5) = 1$ and $v_9 \in e_5$ with $\pi(v_9) = 1$. Similarly, we obtain the following cover times for the other edges

$$\kappa(e_1) = 2, \kappa(e_2) = 7, \kappa(e_3) = 5, \kappa(e_4) = 4, \kappa(e_5) = 1.$$

Overall, we obtain an objective value of $\sum_{e \in E} \pi_\kappa(e) = 19$.

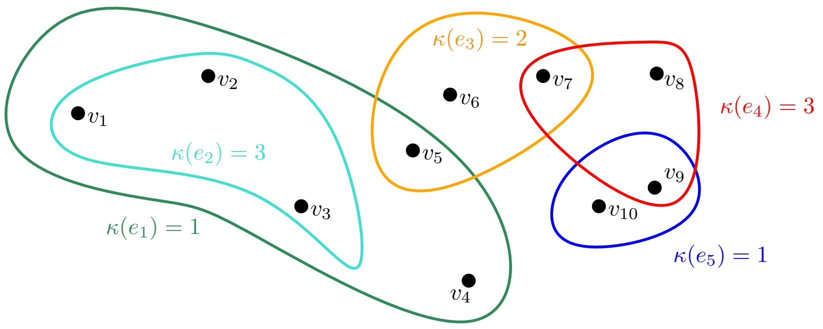


Fig. 4.1: Example of an instance of Gmssc given by a hypergraph $\mathcal{H} = (V, E)$ with vertex set $V = \{v_1, \dots, v_{10}\}$ and edge set $E = \{e_1, \dots, e_5\}$. The cover times are given by $\kappa(e_1) = 1, \kappa(e_2) = 3, \kappa(e_3) = 2, \kappa(e_4) = 3$, and $\kappa(e_5) = 1$.

Gmssc provides a theoretical framework for re-ranking search results in web searches. Search engines like Google, Yahoo! Search, Bing, and Ecosia display their search results for a specific keyword in an ordered list. Their aim is to meet the needs of their users by displaying the most relevant search results first in the ranking. Since relevance is individually characterized, different users might not necessarily be interested in the same subset of search results. Instead, we can define user types, where all users of the same type are interested in the same subset of search results. In addition, one may distinguish between navigational and informational searches. In a navigational search, a user is just interested in one of the relevant search results, whereas in an informational search, a user is interested in all of the relevant search results.

This setting is captured by Gmssc in the following way. The vertex set corresponds to a set of search results. Every hyperedge represents a user type, and a vertex is contained in a hyperedge, if and only if the user of this type is interested in the particular search result. The covering constraints represent the number of search results each user type is interested in. The extreme case, where all covering constraints equal 1, models a navigational search, whereas the other extreme case, where the covering constraints equal the cardinality of an edge, model an informational search. An optimal linear ordering corresponds to a ranking of the search results, such that the average time of finding the desired search results is minimized.

4.1.1 Bipartite Graph Representation of GMSSC

Min Sum Set Cover and Generalized Min Sum Set Cover can be interpreted as special instances of single machine scheduling with bipartite precedence constraints to minimize the sum of weighted completion times. In the following, we describe the single machine setting with bipartite precedence constraints and introduce a bipartite representation of Gmssc and Mssc.

Let $J = A \cup B$ be a set of jobs, together with processing times p_j and weights w_j for all $j \in J$. The precedence relations are given by a bipartite graph $G = (A \cup B, E)$. By $N(b) := \{a \in A \mid (a, b) \in E\}$ we denote the set of **predecessors** of job $b \in B$. In the classical single machine scheduling problem with bipartite precedence constraints, a job b in B can only be scheduled after all its predecessors in $N(b)$ have been scheduled. Note that w.l.o.g. we may assume $N(b) \neq \emptyset$, since any job $b \in B$ without predecessors can simply be moved to A. We refer to the special case where all jobs $a \in A$ have processing time $p_a = 1$ and all jobs $b \in B$ have processing time $p_b = 0$ as 1/0-processing times. In line with this, 0/1-weights imply that all jobs $a \in A$ have weight $w_a = 0$ and all jobs $b \in B$ have weight $w_b = 1$.

In addition to the classical single machine scheduling problem with bipartite precedence constraints, we are given covering constraints $\kappa(b) \in \{1, \ldots, |N(b)|\}$ for all $b \in B$, that is, b can only be scheduled after $\kappa(b)$ of its predecessors in $N(b)$ have been scheduled. If we set $\kappa(b) = |N(b)|$ for all $b \in B$, this agrees with the classical single machine scheduling problem with bipartite precedence constraints, which is sometimes also referred to as AND-scheduling.

It is easy to see that GMSSC corresponds to minimizing the sum of weighted completion times on a single machine with bipartite precedence constraints, 0/1-weights, 1/0-processing times and covering constraints κ. Given an instance of GMSSC, that is, given a hypergraph $\mathcal{H} = (V, E)$ and covering constraints κ, we introduce for every vertex $v \in V$ a job a_v in A with $p(a_v) = 1$ and $w(a_v) = 0$, and for every edge $e \in E$ a job b_e in B with $p(b_e) = 0$ and $w(b_e) = 1$. There exists a directed edge (a_v, b_e) in the bipartite precedence graph, whenever $v \in e$, and the covering constraint of a vertex $b_e \in B$ is simply given by $\kappa(e)$. An example of an instance of GMSSC together with its bipartite representation is given in Fig. 4.2.

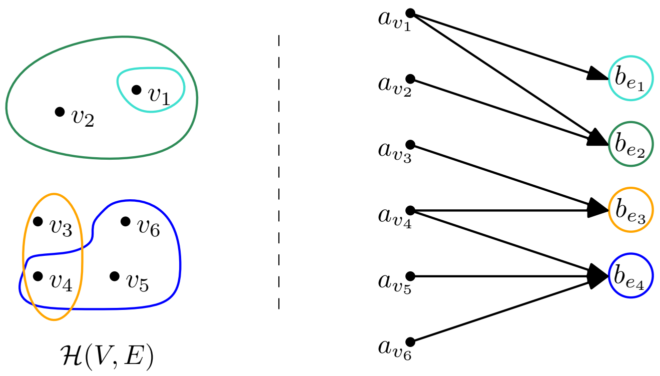


Fig. 4.2: Example of an instance of GMSSC given by the hypergraph $\mathcal{H} = (V, E)$ on the left, and its bipartite representation on the right.

At this point, we would briefly like to mention the connection to OR-scheduling. In the setting of OR-precedence constraints, every job $b \in B$ can already be scheduled after only one of its predecessors in $N(b)$ has been scheduled. In our setting, this corresponds to the case where $\kappa(b) = 1$ for all $b \in B$. In particular, MSSC can be interpreted as a single machine scheduling problem with bipartite OR-precedence constraints to minimize the sum of weighted completion times. See e.g. [64] for more information on OR-scheduling.

The bipartite representation of GMSSC will be useful when explaining the connection to the Ratio problem in Section 4.2.

4.1.2 Related Work

Generalized Min Sum Set Cover Problem (GMSSC). Azar et al. [8] introduced GMSSC as Multiple Intents Re-ranking problem, using a slightly different, yet equivalent definition. Instead of a covering constraint $\kappa(e)$, we are given a weight vector, or so-called profile vector, $w(e) := (w_1(e), \dots, w_{|e|}(e))$ for every hyperedge e. The task is to find a linear ordering π of the search results that minimizes $\sum_{e\in E}\sum_{i=1}^{|e|} w_i(e)\pi_i(e)$, where $\pi_i(e)$ is the cover time of edge e with respect to i. Note that covering constraint $\kappa(e) = k$ corresponds to a profile vector of $w(e) = (0, \dots, 0, 1, 0, \dots, 0)$ whose k-th entry is 1 and all other entries are 0. Hence, by introducing $w_i(e)$-many copies of e with covering constraint $\kappa = i$, we can reformulate any instance in terms of the definition used in this chapter.

Azar et al. [8] presented an $O(\log r)$-approximation algorithm, where r denotes the maximum size of a hyperedge in $\mathcal{H}(V, E)$. Their algorithm is based on a concept called harmonic interpolation. Bansal et al. [10] later gave an example for which the approximation ratio of the algorithm of Azar et al. [8] is $\Omega(\sqrt{\log n})$. For the special case where the entries of each weight vector are monotonically non-increasing, Azar et al. [8] presented a 4-approximation based on techniques by Feige et al. [41, 42]. Furthermore, they gave a 2-approximation algorithm for the special case where the entries of each weight vector are non-decreasing.

The first constant-factor approximation algorithm for Gmssc was presented by Bansal et al. [10]. More specifically, they obtained a randomized 485-approximation algorithm by adding knapsack-cover inequalities to the standard LP relaxation. They showed that even though the LP relaxation has exponentially many constraints, a polynomial-time separation algorithm exists. Using randomized rounding on increasing intervals, they obtained their final integral solution. Additionally, they proved that the integrality gap of the standard LP relaxation can get arbitrarily large without the additional knapsack inequalities. Skutella and Williamson [99] improved the analysis of Bansal et al. [10]. Using α-points[1] instead of the randomized rounding in $O(\log n)$ stages as proposed in [10], Skutella and Williamson [99] obtained an approximation ratio of 28.

Im et al. [61] studied the preemptive version of Gmssc. In the preemptive version, vertices can be scheduled fractionally, and an edge e is covered at the first point in time when the fractions of covered vertices of the edge sum up to at least $\kappa(e)$. They obtained a 2-approximation for preemptive Gmssc by solving a so-called configuration LP. Additionally, they showed that any α-approximation for preemptive Gmssc can be transformed to a solution of Gmssc loosing a factor of 6.2, which immediately implies a 12.4-approximation for non-preemptive Gmssc. Finally, Im et al. [61] conjectured that there exists a 4-approximation for non-preemptive Gmssc.

Happach et al. [53] presented a general framework for approximating Min Sum Ordering problems, including Gmssc. In terms of Gmssc their main result states, that any α-approximation for the Ratio problem[2] implies a 4α-approximation for Gmssc.

Recently, Bansal et al. [9] presented a 4.642-approximation algorithm for Gmssc. Again, their algorithm is based on solving the standard LP relaxation with additional

[1]For more information on α-point scheduling, see [98] and references therein.

[2]See Section 4.2 for a definition of the Ratio problem.

knapsack inequalities. The crucial difference is the rounding technique. They used a linear transformation (referred to as kernel) and α-point rounding. This Kernel α-point Rounding framework can also be used to obtain an improved approximation ratio for the Min Sum Vertex Cover problem, as mentioned below.

The special case of Gmssc, so-called All-But-One Min Sum Set Cover, where every edge has covering constraint $\kappa(e) = \max\{|e| - 1, 1\}$ was introduced by Happach and Schulz [55]. They gave a 4-approximation, which is based on a time-indexed linear program combined with α-point scheduling.

Min Sum Set Cover (MSSC). Feige et al. [41] introduced the Min Sum Set Cover problem, which is the special case of Gmssc when $\kappa \equiv 1$. Feige et al. [41] showed that a simple Greedy algorithm yields a 4-approximation for Mssc. This result was already implicitly shown by Bar-Noy et al. [13], but Feige et al. gave a simpler, histogram-based proof. Bar-Noy et al. [13] studied the closely related Min Sum Coloring problem, where the goal is to find a linear ordering of the independent sets of a graph $G = (V, E)$, or equivalently a coloring $c : V \to \mathbb{N}$ that minimizes $\sum_{v \in V} c(v)$. Mssc captures the Min Sum Coloring problem in the following way. For a graph $G = (V, E)$ we obtain a hypergraph $\mathcal{H} = (V_{\mathcal{H}}, E_{\mathcal{H}})$ by introducing a vertex in $V_{\mathcal{H}}$ for every independent set in G and for every vertex v in V we construct an edge e in $E_{\mathcal{H}}$ that contains all vertices that correspond to independent sets containing v in G.

In the journal version [42], Feige et al. showed that Greedy is indeed best possible since it is NP-hard to approximate Mssc with a ratio strictly better than 4, even when restricted to uniform hypergraphs. If an r-uniform hypergraph is in addition d-regular, Greedy is a $\frac{2r}{r+1}$-approximation. They also proved that it is NP-hard to approximate Mssc better than $2 - \epsilon$ on r-uniform, d-regular graphs.

Min Sum Vertex Cover Problem (MSVC). Min Sum Set Cover on graphs (restricted to 2-uniform hypergraphs) is called the Min Sum Vertex Cover problem. The result by Feige et al. [41, 42] for Mssc on r-uniform, d-regular hypergraphs mentioned above immediately implies that Greedy is a $\frac{4}{3}$-approximation for Msvc on d-regular graphs. For graphs with arbitrary degrees, Feige et al. additionally presented a randomized 2-approximation for Msvc. Barenholz et al. [16] improved this result slightly by combining the techniques in [42] with an improved rounding scheme. Using a kernel α-point rounding framework, Bansal et al. [9] recently obtained a $\frac{16}{9} \approx 1.778$-approximation algorithm for Min Sum Vertex Cover.

Min Latency Set Cover Problem (MLSC) and related scheduling problems. The other extreme case of Gmssc, where the covering constraint of every edge e equals its cardinality, is the so-called Min Latency Set Cover problem. It was introduced by Hassin and Levin [57] as a framework for the following application. Consider a set of tools and jobs that require specific subsets of tools to be processed. The company can install one tool per time unit. Find an ordering of the tools that minimizes the sum of weighted completion times of the jobs. Hassin and Levin [57] presented an

e-approximation algorithm that is based on a framework for the Minimum Latency problem by Archer et al. [7].

MLSC is equivalent to single machine scheduling with bipartite precedence constraints, 0/1-weights and 1/0-processing times. Woeginger [107] showed that the latter is as hard to approximate as $1 \mid prec \mid \sum_j w_j C_j$, for which several 2-approximation algorithms exist, e.g. [25, 29, 51, 81, 93]. Bansal and Khot [11] showed that assuming a stronger variant of the Unique Games Conjecture, this approximation ratio is best possible.

Happach and Schulz [56] studied the problem of scheduling jobs subject to AND/OR-precedence constraints on a single machine, which is denoted by $1 \mid ao - prec = A \dot{\vee} B \mid \sum w_j C_j$. Here, the set of jobs can be divided into a set A and B. Within the sets, we are given AND-precedence constraints. That is, every job in A (or B) can only be scheduled after all of its predecessors in A (or respectively in B) have been scheduled. Between A and B, we are additionally given OR-precedence constraints that imply that at least one of the predecessors in A of a job in B has to be scheduled first. They obtained a 2Δ-approximation for $1 \mid ao - prec = A \dot{\vee} B \mid \sum w_j C_j$, where Δ denotes the maximum number of OR-predecessors of a job in B.

4.1.3 Our Results

Happach et al. [53] provided a general framework for approximation algorithms for GMSSC based on the Ratio problem. We show that the Ratio problem remains NP-complete, even when restricted to 0/1-weights and 1/0-processing times. In particular, this result implies that there is no hope to obtain an exact 4-approximation for GMSSC using their framework, unless P=NP.

The remaining part of the chapter focuses on the Generalized Min Sum Set Cover problem on laminar and cross-free hypergraphs. We start by showing that the Greedy algorithm solves weighted MSSC in polynomial-time on laminar hypergraphs and cross-free hypergraphs. Next, we study weighted, laminar GMSSC with arbitrary covering constraints. Our main result is that any preemptive solution of weighted, laminar GMSSC can be transformed into a non-preemptive solution without increasing the objective value. In combination with the result in [61], we obtain a 2-approximation for weighted, laminar GMSSC.

Finally, we consider laminar ALL-BUT-K MSSC, which is GMSSC restriced to covering constraints of the form $\kappa(e) = \max\{|e| - K, 1\}$ for some $K \in \mathbb{N}_0$. We show that laminar ALL-BUT-K MSSC can be solved in polynomial-time by transforming it to an instance of the Minimum Latency problem. An overview of our results is displayed in Table 4.1.

From an application point of view, our results imply the following. Given a navigational search, where for any two user types, the set of search results they are interested in is either a subset of one another or their search interests are disjoint, one may compute an optimal ordering in polynomial-time.

Table 4.1: Overview of our results on laminar GMSSC and a comparison to existing results for arbitrary set structures. The approximation ratios in the third column display the currently best-known approximation ratios for arbitrary set families. Our results for the corresponding problems on laminar set families are highlighted in bold.

covering constraints	$\min_{\pi} \sum_{e \in E} \pi_\kappa(e)$	**general**	**laminar**
$\kappa = 1$	MIN SUM SET COVER	4 [41]	**polynomial**
general κ	GEN. MIN SUM SET COVER	4.642 [9]	**2**
$\kappa = \max\{\lvert N(b)\rvert - K, 1\}$	ALL-BUT-K MSSC		**polynomial**
$\kappa = \max\{\lvert N(b)\rvert - 1, 1\}$	ALL-BUT-ONE MSSC	4 [55]	**polynomial**
$\kappa = \lvert N(b)\rvert$	MIN LATENCY COVER	2 (e.g.[51])	**polynomial**

4.2 Connection to the Ratio Problem

Happach et al. [53] introduced a framework for constructing approximation algorithms for Min Sum Ordering problems, including GMSSC. The framework is based on the RATIO PROBLEM, which is defined in the following way. Let $J = A \cup B$ be a set of jobs with processing times p_j and weights w_j for all $j \in J$. Additionally, we are given bipartite precedence constraints and covering constraints $\kappa(b)$ for all $b \in B$. A subset $S \subseteq J$ is an **ideal** if for all $b \in S \cap B$ at least $\kappa(b)$ of its predecessors are also in S, that is $S \cap N(b) \geq \kappa(b)$. For an ideal S we define

$$\rho(S) \coloneqq \frac{w(S)}{p(S)} = \frac{\sum_{j \in S} w_j}{\sum_{j \in S} p_j}.$$

A set S is called **ρ-maximizing ideal**, if S is an ideal with maximum ratio $\rho(S)$, among all other ideals, that is, $\rho(S) = \max_{\text{ideal } S'} \rho(S')$. The task is to find a ρ-maximizing ideal $S \subseteq J$.

The following proposition connects the Ratio problem to GMSSC.

Proposition 4.2 (Happach, Hellerstein, and Lidbetter [53])
Any polynomial-time α-approximation for the Ratio problem implies a polynomial-time 4α-approximation for GMSSC.

The algorithm in [53] is based on greedily scheduling ρ-maximizing sets and works as follows. Consider an instance of GMSSC, with its bipartite representation as introduced in Section 4.1.1. That is, we are given jobs $J = A \cup B$ with 0/1-weights and 1/0-processing times, bipartite precedence constraints and covering constraints $\kappa(b)$ for all $b \in B$. Use the given α-approximation for the Ratio problem to compute an α-approximate ρ-maximizing set S, that is a subset $S \subseteq J$ with $\rho(S) \geq \frac{1}{\alpha}\rho(S^*)$, where S^* denotes a ρ-maximizing ideal. Remove S from the instance, schedule the jobs in S at the end of the current schedule in any feasible order and recurse. Consider the ordering of the jobs in A in the final schedule to obtain the final linear ordering π for the instance of GMSSC.

In particular, Proposition 4.2 states that the existence of a polynomial-time algorithm for the Ratio problem would imply a 4-approximation for GMSSC. Unfortunately, there is no hope for a polynomial-time algorithm for the Ratio problem with 0/1-weights and 1/0-processing times, unless P=NP, as the following theorem shows.

A simple observation that we use is the subsequent one. Let $a, b, c, d \in \mathbb{N}$, then

$$\frac{a}{b} < \frac{c}{d} \Longrightarrow \frac{a}{b} < \frac{a+c}{b+d} < \frac{c}{d.} \tag{4.2}$$

The Densest k-Subgraph problem is defined as follows. Given a graph $G = (V, E)$ on n vertices and a constant $k \leq n$, find the densest subgraph of G on k vertices, that is a subgraph on k vertices with the largest numbers of edges. Note that the densest k-subgraph problem is NP-hard, since it generalizes the well-know Maximum Clique problem.

Theorem 4.3
The Ratio problem is NP-hard, even when restricted to 0/1*-weights,* 1/0*-processing times and two different values of* κ*.*

Proof. We prove NP-hardness by a reduction from the Densest k-Subgraph problem. Let $G = (V, E)$ be a graph on n vertices and let $k \leq n$ be integer. We obtain an instance of the Ratio problem in the following way. For every vertex $v \in V$ we define a job $a_v \in A$, and for every edge $e \in E$ we define a job b_e in B_E. Let D be a set of dummy vertices of cardinality $l > |E|k$ and let $B := B_E \cup D$. The set of edges is defined as $\{(a_v, b_e) \mid v \text{ is incident to } e \text{ in } G\} \cup \{(a_v, d) \mid a_v \in A \text{ and } d \in D\}$. The cover constraints are $\kappa(b_e) = 2$ for all $b_e \in B_E$ and $\kappa(d) = k$ for all $d \in D$. See Figure 4.3 for a visualization of the constructed instance of the Ratio problem.

We need to show that a ρ-maximizing ideal in the constructed instance corresponds to a densest k-subgraph in G and vice versa. Let S be a ρ-maximizing ideal in the constructed instance. Denote by $S(A) := A \cap S$ the jobs in A that are contained in the ideal. We begin by showing that S contains exactly k vertices in A, i.e. $|S(A)| = k$. Since any ρ-maximizing ideal containing D has a ratio of at least

$$\frac{|D|}{k} = \frac{l}{k} > \frac{|E| \cdot k}{k} = |E| = |B_E|,$$

which is strictly larger than the ratio of any ideal not containing D, it follows that $D \subseteq S$ and, hence, $|S(A)| \geq k$.

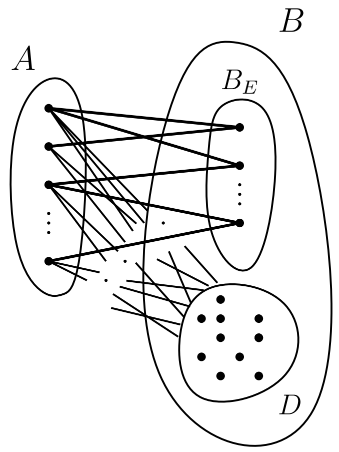


Fig. 4.3: Reduction from an instance of the densest k-subgraph problem to an instance of the Ratio problem.

Now, assume that $|S(A)| = k' > k$. Denote the subset of vertices in G whose corresponding jobs are in $S(A)$ by V_S, that is, $V_S \coloneqq \{v \in V \mid a_v \in S(A)\}$, and let $G_S \coloneqq G[V_S]$. Let $a_x \in S(A)$ be the job with the smallest number of successors in B_E, or equivalently let $x \in V_S$ be a vertex of minimum degree in G_S. Denote by S^- the ideal after removing a_x and all successors of a_x in B_E. Then,

$$\begin{aligned}\rho(S) &< \frac{|E|\cdot k + |\{b_e | e = uv \text{ with } a_u, a_v \in S(A)\}|}{k'} = \frac{|E|\cdot k + |E[G_S]|}{k'}\\ &= \frac{|E|\cdot k + |E[G_{S^-}]| + \deg(x)}{(k'-1)+1}\\ &< \frac{|E|\cdot k + |E[G_{S^-}]|}{(k'-1)} = \rho(S^-),\end{aligned}$$

where the last inequality follows with (4.2), since

$$\begin{aligned}\frac{|E|\cdot k + |E[G_{S^-}]|}{(k'-1)} &= \frac{\frac{k}{2}\sum_{v\in V}\deg(v) + \frac{1}{2}\sum_{v\in V_S\setminus\{x\}}\deg(v)}{(k'-1)}\\ &> \frac{\frac{k}{2}(k'-1)\deg(x) + \frac{1}{2}(k'-1)\deg(x)}{(k'-1)} > \frac{\deg(x)}{1}.\end{aligned}$$

The fact that $\rho(S) < \rho(S^-)$ contradicts the maximality of S. Hence, $|S(A)| = k$. Since S is a ρ-maximizing ideal, it contains the k jobs in A that have the largest number of successors in B_E, that is $S(A)$ is chosen such that the set $\{b_e | e = uv \text{ with } a_u, a_v \in S(A)\}$ is maximized. This is equivalent to choosing k vertices in G that maximize the number of edges induced by them. Hence, the k vertices form a densest k-subgraph in G.

On the other hand, every densest subgraph on k vertices corresponds to a ρ-maximizing ideal in the following way. Simply, let S be the ideal induced by the k jobs in A that correspond to the k vertices of the densest k-subgraph. Since all vertices in D have covering constraint k, D is contained in S, and since the vertices chosen in A correspond to a densest subgraph, they maximize the number of jobs in $B_E \cap S$. $\square$

4.3 Min Sum Covering Problems on Laminar Instances

In this section, we consider min sum covering problems on special structured hypergraphs. A first natural restriction would be to consider uniform or regular hypergraphs. Recall, that a hypergraph is r-uniform if every edge has cardinality r, and d-regular if every vertex is contained in d edges. Unfortunately, one may not hope for better approximation algorithms when restricting instances to uniform hypergraphs alone. Feige et al. [42] showed that it still remains NP-hard to approximate Mssc with a ratio strictly better than 4 on uniform hypergraphs.

To obtain an improvement, we additionally have to bound the number of edges a single vertex may be contained in. If we require the r-uniform hypergraph to be d-regular as well, Feige et al. [42] showed that a simple Greedy algorithm yields a $\frac{2r}{r+1}$-approximation. They also proved that it is indeed NP-hard to approximate Mssc on uniform, regular hypergraphs with a ratio strictly better than 2.

Motivated by the improvement for uniform, regular hypergraphs we study Mssc and Gmssc on other classes of hypergraphs. In the following let $\mathcal{H} = (V, E)$ be a **laminar hypergraph**. That is, the set of edges E forms a laminar family. Recall that a family is called **laminar**, if for two edges $e_1, e_2 \in E$ with $e_1 \cap e_2 \neq \emptyset$ either $e_1 \subseteq e_2$ or $e_2 \subseteq e_1$. See Figure 4.4 for an example of a laminar hypergraph.

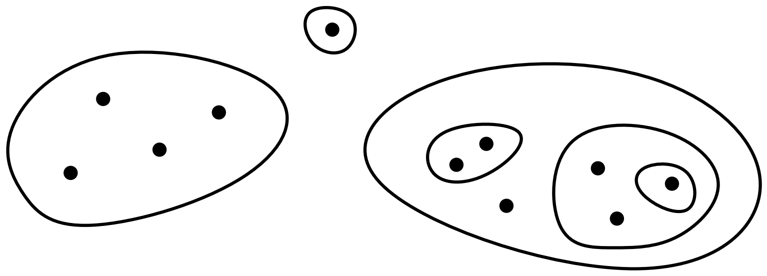


Fig. 4.4: Example of a laminar hypergraph.

4.3.1 Laminar Min Sum Set Cover

First, we show that a simple Greedy algorithm returns an optimal solution for weighted, laminar MSSC in polynomial-time. In the weighted version, every edge e has a positive weight $w(e)$ and the objective is to minimize the sum of weighted cover times. The natural Greedy algorithm chooses in every step a vertex that maximizes the weight of newly covered edges.

Theorem 4.4
The Greedy algorithm returns an optimal solution for weighted, laminar MSSC.

Proof. Let $\mathcal{H} = (V, E)$ be an instance of weighted, laminar MSSC. W.l.o.g., we may assume that E does not contain any duplicate edges. Otherwise, we could increase the weight $w_e \in \mathbb{N}$ of an edge $e \in E$ correspondingly and remove its copies.

For $v \in V$, let $E_v = \{e \in E \mid v \in e\}$ be the set of edges that contains v, and let $u \in V$ be the vertex that the Greedy algorithm picks first, i.e., $w(E_u) \geq w(E_v)$ for all $v \in V$. Consider an optimal linear ordering $\pi : V \to [n]$, and suppose that u is not the first vertex, i.e., $\pi(u) > 1$. We give a procedure of how to alter the optimal solution without increasing the objective function value such that u appears first in the linear ordering. More specifically, we find a vertex v with $\pi(v) < \pi(u)$ such that switching v and u in the linear ordering does not increase the objective function. This proves the claim, as we can iteratively repeat this argument for the other vertices until the optimal solution coincides with the Greedy solution.

Let $E_u^+ = \{e \in E_u \mid \pi(e) = \pi(u)\}$ be the set of edges that are covered by u, and let $E_u^- := E_u \setminus E_u^+$ be the set of edges that contain u but were already covered by previous vertices in π (see Figure 4.5). If $E_u^- = \emptyset$, we choose $v \in V$ to be the vertex with $\pi(v) = 1$ and swap v and u, that is $\pi'(u) = 1$ and $\pi'(v) = \pi(u)$. Since $E_u^- = \emptyset$ and $w(E_u) \geq w(E_v)$, we did not increase the objective function.

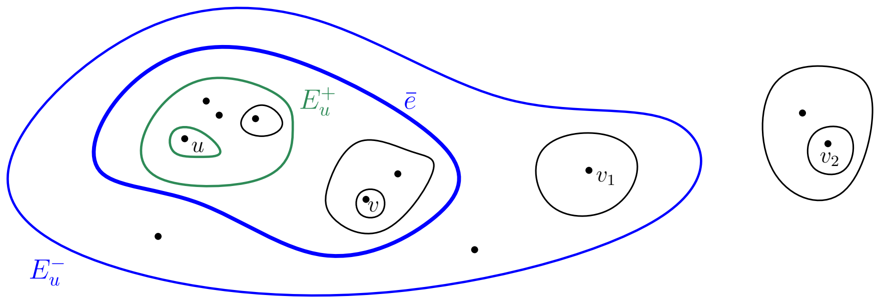


Fig. 4.5: Example to visualize the definitions of E_u^+, E_u^- and $\bar{e}$. Assume that in the optimal linear ordering π the vertices appear in the order $v_1 \to v_2 \to v \to u$. Then E_u^+ is given by the green edges, E_u^- is given by the blue edges, and the inclusion-minimal edge $\bar{e} \in E_u^-$ is highlighted in bold.

Else, since E is a laminar family and the edges in E_u^- are covered before time $\pi(u)$, we get that $f \supseteq e$ for all $f \in E_u^-$ and $e \in E_u^+$. In particular, there is an inclusion-minimal edge $\overline{e} \in E_u^-$ that was covered by some vertex $v \in V$ with $\pi(v) < \pi(u)$, i.e., $\pi(\overline{e}) = \pi(v)$. Observe that $v \in \overline{e} \subseteq f$ for all $f \in E_u^-$ implies $\pi(f) \leq \pi(v)$ for all $f \in E_u^-$ and, further, $E_u^- \subseteq E_v$. That is, all edges that contain u and where already covered by previous vertices in π, also contain v. So, v covers the last uncovered edge(s) in E_u^-, and we can write $E_v = E_u^- \,\dot\cup\, E_v^+$. Note that in contrast to the definition of E_u^+, the set E_v^+ might contain sets that are covered before time $\pi(v)$. Thus, $w(E_u) \geq w(E_v)$ is equivalent to $w(E_u^- \,\dot\cup\, E_u^+) \geq w(E_u^- \,\dot\cup\, E_v^+)$ which implies $w(E_u^+) \geq w(E_v^+)$.

Let $S := \{v' \in V \mid \pi(v) < \pi(v') < \pi(u)\}$ be the set of vertices that appear between v and u in the linear ordering. We swap v and u, i.e., we consider the linear ordering $\pi' : V \to [n]$ with $\pi'(u) = \pi(v)$, $\pi'(v) = \pi(u)$ and $\pi'(v') = \pi(v')$ for all $v' \in V \setminus \{u, v\}$.

Compared to π, the cover times in π' of all sets in E_u^+ decreases by $|S| + 1$, sets in E_v^+ are covered at most $|S| + 1$ time steps later, and the cover times of all other edges remain unchanged. The difference in the objective values of π and π' is

$$\sum_{e \in E} w_e \left(\pi'(e) - \pi(e)\right) \leq (|S| + 1) \Big(w(E_v^+) - w(E_u^+)\Big) \leq 0.$$

So, swapping u and v in the linear ordering does not increase the objective function, and u appears earlier in the new linear ordering. We can repeat this procedure iteratively until $\pi'(u) = 1$. Note that the set of edges E_u^+ strictly increases in each iteration. □

MSSC on t-laminar Hypergraphs. We just proved that Greedy solves Mssc in polynomial-time on laminar hypergraphs. On the other hand, Feige et al. [42] showed that Greedy approximates Mssc on general hypergraphs within a ratio of 4. So the natural question is to ask how well Greedy performs if we allow the edges to overlap *slightly*. A nice framework for a generalization of laminar set families, studied for example in [35], is given by the following definition.

A set family E on n elements is called **t-laminar**, if for all sets e, f in E with $|e \cap f| \geq t$ either $e \subseteq f$ or $f \subseteq e$ holds. In other words, we allow sets to overlap in up to $t - 1$ elements. A hypergraph $\mathcal{H} = (V, E)$ is called t-laminar, if E forms a t-laminar set family. Note that this equals the definition of laminar hypergraphs for $t = 1$ and for large enough t (i.e. $t \geq n$) we simply obtain general hypergraphs. A natural question to ask is how Greedy performs on 2-laminar hypergraphs and, more general, on t-laminar hypergraphs, for increasing t. More specifically, can we show that Greedy performs strictly better than 4 on t-laminar families for fixed t?

Unfortunately, this claim does not hold. An example, initially given by Bar-Noy et al. [13] and slightly modified by Feige et al. [42] immediately implies that there is no hope for a better performance of Greedy, not even on 2-laminar families. The example presented is an instance of Msvc and, hence, all sets are of size 2. So, in particular, the instance is 2-laminar. Feige et al. [42] used the example to show that the analysis of Greedy is tight. In other words, there exists a 2-laminar hypergraph on which Greedy already performs as badly as on arbitrary set families. For an explicit description of the instance, see Proposition 8 in [42].

4.3.2 Min Sum Set Cover on Cross-Free Hypergraphs

Cross-free set families are another generalization of laminar set families. Let E be a set of hyperedges on the vertex set V. We say that two edges $e, f \in E$ **overlap** if $e \cap f \neq \emptyset$ and neither $e \subseteq f$ nor $f \subseteq e$. Two edges e, f **cross**, if they overlap and $e \cup f \neq V$. A cross-free family is a set family in which no two edges cross. A hypergraph is called a **cross-free hypergraph** if its edge set E forms a cross-free family. Note that this generalizes laminar hypergraphs in the following way. A hypergraph is laminar if and only if E is overlap-free, that is, no two edges in E overlap. In a cross-free hypergraph, two edges are allowed to overlap if their union equals the ground set V. See Figure 4.6 for an example of a cross-free family.

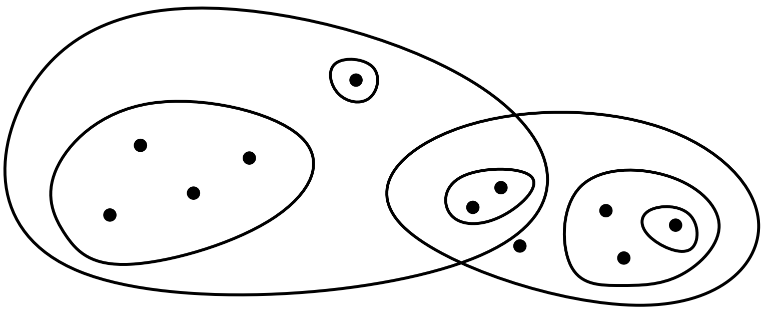


Fig. 4.6: Example of a cross-free hypergraph.

In Theorem 4.4 we showed that the Greedy algorithm is optimal for weighted, laminar Mssc. This is still the case if the instance is cross-free, as the following lemma states.

Theorem 4.5
The Greedy algorithm returns an optimal solution for weighted, cross-free Mssc.

Before preceding with the proof of Theorem 4.5, we state the following lemma on the structure of cross-free families.

Lemma 4.6
Let E be a cross-free family over a ground set V. Then for all $v \in V$, the family $E \setminus \{v\} := E \setminus \{e \in E : v \in e\}$ is laminar.

Proof. Assume there exist two edges e, f in $E \setminus \{v\}$ that overlap. Since $E \setminus \{v\} \subset E$ and E is cross-free, this implies that $e \cup f = V$. Hence, $v \in e$ or $v \in f$ holds, which contradicts the fact that e and f in $E \setminus \{v\}$. □

Proof (Theorem 4.5). The proof follows a similar framework as the proof of Theorem 4.4 and uses the same notation. The main difference is the choice of v. Let $\mathcal{H} = (V, E)$ be an instance of weighted, cross-free MSSC without duplicate edges. Again, let $u \in V$ be the vertex picked first by the Greedy algorithm, that is, $w(E_u) \geq w(E_v)$ for all $v \in V$. Let $\pi : V \to [n]$ be an optimal ordering, and suppose that $\pi(u) > 1$.

If $E_u^- = \emptyset$, we switch u with the vertex that comes first in the optimal ordering. That is, $\pi'(u) = 1$ and $\pi'(v) = \pi(u)$, where v is the vertex with $\pi(v) = 1$. Since $E_u^- = \emptyset$ and $w(E_u) \geq w(E_v)$, we did not increase the objective function.

Else, we choose v in the following way. Let $t := \max\{\tau < \pi(u) : \pi(e) = \tau \text{ and } u \in e\}$ and let v be the vertex with $\pi(v) = t$.

If $t = 1$, then all edges in E_u^- are covered by v, that is $E_u^- \subseteq E_v$. Let $E_v^+ := E_v \setminus E_u^-$. Since u is the vertex picked first by Greedy,

$$w(E_u) \geq w(E_v) \Leftrightarrow w(E_u^+ \dot\cup E_u^-) \geq w(E_v^+ \dot\cup E_u^-) \Leftrightarrow w(E_u^+) \geq w(E_v^+)$$

We have to argue that swapping v and u does not increase the objective function. Let $\pi' : V \to [n]$ be the linear ordering after swapping v and u, that is $\pi'(u) = 1$, $\pi'(v) = \pi(u)$ and $\pi'(v') = \pi(v')$ for all $v' \in V \setminus \{u, v\}$. Compared to π, all edges in E_u^+ are covered $\pi(u)$ time steps earlier, edges in E_v^+ are covered at most $\pi(u)$ time steps later, and the cover times of all other edges remain unchanged. The difference in the objective values of π and π' is

$$\sum_{e \in E} w_e \left(\pi'(e) - \pi(e)\right) \leq \pi(u) \left(w(E_v^+) - w(E_u^+)\right) \leq 0,$$

where the last inequality follows from the fact that $w(E_u^+) \geq w(E_v^+)$.

For $t > 1$ the instance at time t and the following steps is laminar, by Lemma 4.6. In this case, v is the vertex that covers the inclusion-minimal edge $\bar{e}$ and the same argument as in the proof of Theorem 4.5 applies. □

4.3.3 Laminar Generalized Min Sum Set Cover

In Theorem 4.4 we showed that Greedy is optimal for laminar MSSC. This is not necessarily the case if we consider general covering constraints, i.e. laminar GMSSC, as the following example shows.

Example 4.7
Consider the following instance of GMSSC with two disjoint edges e and f. The edge e contains two vertices, has weight $w > 2$ and covering constraint $\kappa(e) = 2$ and the edge f contains one vertex, has weight 1 and covering constraint $\kappa(f) = 1$. Note that this is in particular an instance of Laminar Minimum Latency Set Cover. See Figure 4.7 for a visualization of the instance.

An optimal solution picks the two vertices in e first and then the vertex in f, which leads to an optimal objective value of $2w + 3$. The Greedy algorithm on the other hand chooses the vertex in f first and then the two vertices in e. Hence, the objective value of the Greedy solution is given by $1 + 3w$, which is strictly worse than optimal for $w > 2$.

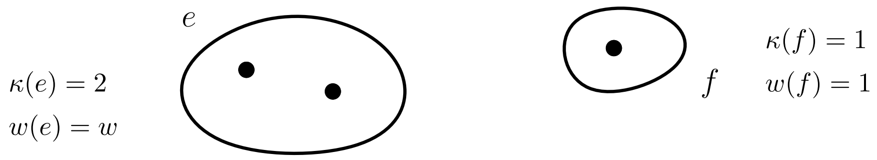


Fig. 4.7: Instance of laminar Gmssc for which Greedy is not optimal. Edge e has covering constraint $\kappa(e) = 2$ and weight $w(e) = w$, and the edge f has covering constraint $\kappa(f) = 1$ and weight $w(f) = 1$.

A simple way to improve the Greedy algorithm seems to not only consider the edges covered immediately by the next vertex but to consider all edges that contain the vertex and are not covered yet. More specifically, let E_v be the set of edges that contain v and are not covered yet. We choose the vertex v to be the vertex that maximizes $\sum_{e \in E_v} \frac{1}{\kappa(e)} w(e)$ and call this variant the Look-ahead Greedy. The following example shows that the Look-ahead Greedy is not optimal for laminar Gmssc either.

Example 4.8
The instance is given by a an edge e that consists of a single vertex with weight 2 and two edges f_1 and f_2 as displayed in Figure 4.8. Edge f_1 has weight 1 and edge f_2 has weight 4 and for all three edges the covering constraints are given by their cardinality, i.e. $\kappa(f) = |f|$ for all $f \in \{e, f_1, f_2\}$. An optimal solution covers e first, then f_1 and finally f_2, which yields an optimal objective value of $1 \cdot 2 + 2 \cdot 1 + 5 \cdot 4 = 24$. For the Look-ahead Greedy the vertex in e and the vertex in f_1 have the same weight,

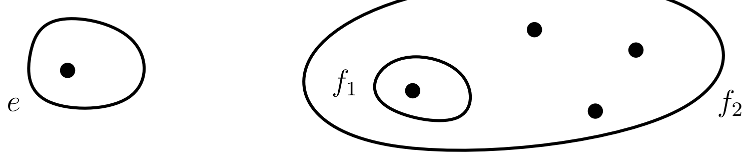


Fig. 4.8: Instance of Gmssc for which Look-ahead Greedy is not optimal. Edge e has covering constraint $\kappa(e) = 1$ and weight $w(e) = 2$, edge f_1 has covering constraint $\kappa(f_1) = 1$ and weight $w(f_1) = 1$ and the edge f_2 has covering constraint $\kappa(f_2) = 4$ and weight $w(f_1) = 4$.

i.e. $2 = 1 + \frac{1}{4} \cdot 4$. Hence, Look-ahead Greedy might choose to cover f_1 first, then e and afterwards the three remaining vertices in f_2, which yields an objective value of $1 + 2 \cdot 2 + 5 \cdot 4 = 25 > 24$.

In the following, we present a polynomial-time 2-approximation algorithm for Gmssc on laminar hypergraphs. The algorithm is based on a result by Im et al. [61]. They gave a 2-approximation for preemptive Gmssc and proved that any preemptive schedule can be transformed to a non-preemptive schedule, losing a factor of 6.2. We show that for laminar instances of Gmssc we can transform the preemptive schedule to a non-preemptive schedule without increasing the objective function. Theorem 4.9 states the main result of this section.

Theorem 4.9
There is a 2*-approximation for weighted, laminar* Gmssc*.*

We already gave a brief summary of the results by Im et al. [61] in Section 4.1.2. At this point, a more detailed discussion of preemptive Gmssc seems crucial for understanding the remainder of this section. After formally defining preemptive Gmssc and explaining the results in [61], we state and analyse our algorithm.

Recall the connection of Gmssc to single machine scheduling with bipartite precendence constraints, 0/1-weights and 1/0-processing times as described in Section 4.1.1. When introducing the preemptive version of Gmssc, this interpretation is particularly useful. So instead of a linear ordering, we are looking for a schedule that minimizes the sum of weighted completion times. When thinking about schedules instead of linear orderings, preemption can be defined in a natural way. In the preemptive version of Gmssc, vertices may be scheduled fractionally and an edge e is covered at the point in time when the sum over its fractionally scheduled vertices sums up to κ.

We denote the fraction of vertex $v \in V$ that is assigned to time $t \in [n]$ by $x_{vt} \in [0, 1]$. Certainly, each vertex needs to be completely assigned, i.e., $\sum_{t=1}^{n} x_{vt} = 1$ for all $v \in V$, and a total fraction of one is assigned to each point in time, i.e., $\sum_{v \in V} x_{vt} = 1$ for all $t \in [n]$. The cover time of an edge $e \in E$ is then defined as

$$\pi(e)_\kappa := \min \left\{ t \in [n] \,\middle|\, \sum_{v \in e} \sum_{\tau=1}^{t} x_{v\tau} \geq \kappa(e) \right\}. \tag{4.3}$$

Note that if x_{vt} is binary for all $v \in V$ and $t \in [n]$, (4.3) defines a linear ordering, that is, the definition coincides with the definition of cover times (4.1) for non-preemptive Gmssc.

Im et al. [61] presented a 2-approximation algorithm for preemptive Gmssc which is based on a so-called configuration LP. A valid configuration for an edge e, is an assignment of the vertices contained in e to time slots in $[n]$. For every edge and a corresponding configuration, one can define the completion time of e in the particular configuration. A feasible solution to the configuration LP is given by a unique configuration for every edge, such that no more than one vertex is scheduled per time slot and the configurations agree. Im et al. [61] showed that the configuration LP is a valid relaxation for preemptive GMSSC and that, even though it has exponentially many variables, it can be solved in polynomial-time. The solution of the configuration LP is rounded to a feasible integer schedule using random α-points.

In the following, we describe an algorithm that, given an instance of laminar GMSSC together with a preemptive schedule, turns it into an non-preemptive schedule without increasing the objective function.

Algorithm Description

As input, the algorithm to transform a preemptive solution for laminar GMSSC into a non-preemptive one receives the cover times of the edges in a feasible preemptive solution. The algorithm works as follows. The edges are processed in an *earliest cover time first* manner (ties are broken arbitrarily). That is, in each iteration a vertex of an uncovered edge with lowest cover time is scheduled.

If the edge e considered in the current iteration contains no uncovered edges in E that are subsets of e, then the vertex is chosen arbitrarily within e (recall that the vertices are indistinguishable).

Else, the current edge is updated to an uncovered edge e' of E with $e' \subset e$ of lowest cover time. In this way, the vertex scheduled next is always chosen from the most urgent subset(s). Finally, after a vertex is scheduled, this vertex is removed from the instance and the covering requirements are updated accordingly. The cover time of an edge is the first point in time when its remaining covering requirement is set to zero.

If an edge is covered, it is removed from the instance and its cover time is set to t. Else, the vertex is removed from the instance and the edge is updated accordingly. The complete algorithm is summarized in the following (Algorithm 1).

Note that Algorithm 1 returns a feasible (non-preemptive) solution for GMSSC, since exactly one vertex is assigned to time t (line 8) in each iteration of the outer while-loop and that, technically, one does not need to assign the cover time of an edge in line 12, as the cover time is already well-defined by the linear ordering $\pi : V \to [n]$.

Before we show that the cover times of the solution returned by Algorithm 1 satisfy $\pi(e) \leq \pi'(e)$ for all $e \in E$, we illustrate the procedure in the following example.

Algorithm 1: An algorithm to transform a feasible preemptive solution of laminar GMSSC to a feasible non-preemptive solution.

Input: Instance $(\mathcal{H} = (V, E), \kappa)$ of laminar GMSSC and a feasible preemptive solution with cover times $\pi'(e)$

Output: Feasible non-preemptive solution of $(\mathcal{H} = (V, E), \kappa)$ with cover times $\pi(e)$

```
$F \leftarrow E$;
$t \leftarrow 1$;
while $t \leq n$ do
    Choose $e \in \arg\min\{\pi'(f) \mid f \in F\}$;
    while $\exists\ f \in F$ with $f \subset e$ do
        Set $e' \leftarrow e$ and choose $e \in \arg\min\{\pi'(f) \mid f \in F$ and $f \subset e'\}$;
    end
    Choose $v \in f$ and set $\pi(v) \leftarrow t$;
    for $f \in F$ with $v \in f$ do
        $\kappa(f) \leftarrow \kappa(f) - 1$;
        if $\kappa(f) = 0$ then
            $F \leftarrow F \setminus \{f\}$ and $\pi(f) \leftarrow t$;
        else
            $F \leftarrow (F \setminus \{f\}) \cup \{f \setminus \{v\}\}$;
        end
        $t \leftarrow t + 1$;
    end
end
return $\pi$;
```

Example 4.10
Consider the following instance of laminar GMSSC. The hypergraph is defined on the vertex set $V = \{v_1, \ldots, v_8\}$ and we are given four edges $e = \{v_1, v_2, v_3\}$, $f_1 = \{v_4, v_5\}$, $f_2 = \{v_6, v_7, v_8\}$ and $f = \{v_4, v_5, v_6, v_7, v_8\}$ with covering constraints $\kappa(e) = \kappa(f) = 1$ and $\kappa(f_1) = \kappa(f_2) = 2$. See Figure 4.9 for a visualization of the instance. Additionally, we are given the cover times π' of the edges in a preemptive schedule. Here, $\pi'(e) = 1$, $\pi'(f) = 2$, $\pi'(f_1) = 4$ and $\pi'(f_2) = 5$ as displayed in the figure above.

Algorithm 1 considers the edges in order of increasing cover times π'. Hence, it chooses edge e first and schedules an arbitrary vertex in e. The edge e is now covered and removed from the list. Next, the algorithm considers the edge f. The edge f contains two edges f_1 and f_2. Since $\pi'(f_1) < \pi'(f_2)$, the algorithm schedules an arbitrary vertex in f_1 next. The edge f is now covered and removed from the list, the covering constraint of f_1 is updated. The uncovered edge with the smallest cover time

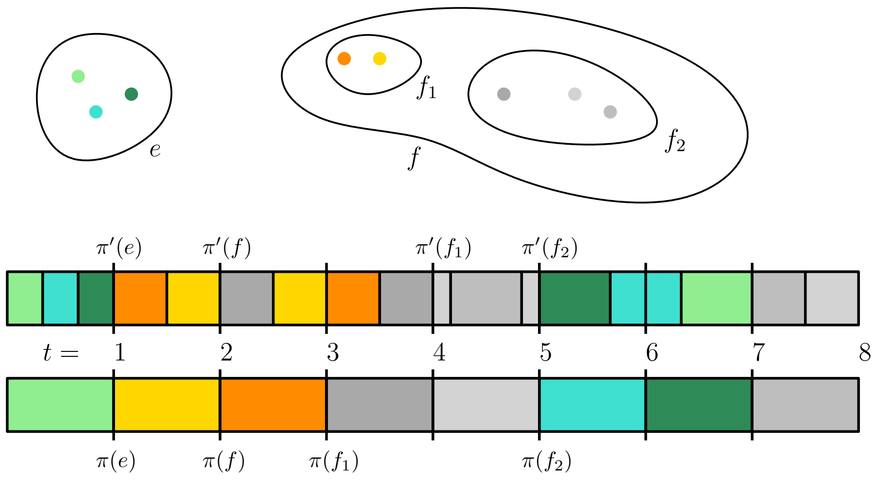


Fig. 4.9: Instance of laminar GMSSC. The covering constraints are given by $\kappa(e) = \kappa(f) = 1$ and $\kappa(f_1) = \kappa(f_2) = 2$. A feasible preemptive schedule with cover times π' and the corresponding non-preemptive schedule with cover times π, as constructed by Algorithm 1, are displayed at the bottom.

π' is now f_1. Since f_1 does not contain any other edges, the remaining vertex in f_1 is scheduled next. The edge f_1 is now covered and removed from the list. Finally, the algorithm chooses twice a vertex in f_2 to cover the remaining edge f_2. For the sake of completeness, one may schedule the remaining vertices in an arbitrary order at the end of the current schedule.

As visualized in the figure above, the completion times of the edges e, f and f_2 remained the same and the completion time of f_1 decreased. That is, we constructed a non-preemptive schedule with smaller objective value.

Algorithm Analysis

Instead of breaking ties in lines 4 and 6 arbitrarily, we now assume that Algorithm 1 processes an edge e completely in the following iterations of the outer while-loop before it continues with an edge disjoint from e. For convenience, we refrain from making this explicit in the algorithm.

Theorem 4.11
Given a feasible preemptive solution for laminar Generalized Min Sum Set Cover with cover times $\pi'_\kappa : E \to [n]$, Algorithm 1 returns a feasible non-preemptive solution with cover times $\pi_\kappa(e) \leq \pi'_\kappa(e)$ for all $e \in E$.

Before proving Theorem 4.11, we introduce the necessary notation. A vertex v is called **free** in a hyperedge e if $v \in e$ and v is not contained in any other edge $f \in E$ with

$f \subseteq e$. An edge $f \in E$ with $f \subset e$ is called **maximal in e w.r.t. E** if there is no $f' \in E$ with $f \subset f' \subset e$.

Proof. Without loss of generality, we assume that an edge $e \in E$ with $|e| > \kappa(e)$ contains no free vertices, since no *reasonable solution* for laminar GMSSC chooses a free vertex in e if there is still a vertex in some subset of e available.

Let $e \in E$ be the edge chosen by Algorithm 1 after the last iteration of the inner while-loop at time $t \in [n]$. We consider the instance reduced to the edges disjoint from e that are covered by time $\pi'(e)$ in the preemptive solution. Let $E_e^- := \{f \in E \mid \pi'(f) \leq \pi'(e), f \cap e = \emptyset\}$ be those edges. Further, let $D_e \subseteq E_e^-$ be the set of inclusion-maximal edges in E_e^- that contain free vertices. Note that E_e^- and D_e are laminar and that the edges in D_e are disjoint by construction (see Figure 4.10).

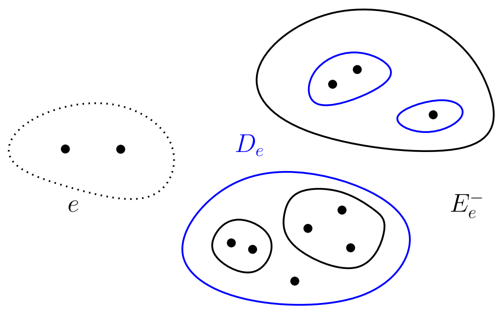


Fig. 4.10: Example to visualize the definition D_e and E_e^-. Inclusion-maximal edges are highlighted in blue.

We claim that for every $f \in E_e^-$, there is an edge $f' \in D_e$ such that $f \subseteq f'$. Assume there exists an edge $f \in E_e^-$, such that there is no edge $f' \in D_e$ with $f \subseteq f'$. If there exists a set $f' \in D_e$ such that $f \supseteq f'$, then f does not contain any free vertices by definition of D_e. Hence, e decomposes into edges that are contained in E_e^-. At least one of these edges has to be disjoint from all sets in D_e, otherwise $f \subseteq f'$. Let $\hat{f} \in E_e^-$ with $\hat{f} \subseteq f$ be an inclusion-minimal such subset that is disjoint from all sets in D_e. Since $\hat{f}$ is inclusion-minimal, it contains free vertices. So, it is contained in D_e, which is a contradiction.

We can assume that the edges in E_e^- (and, therefore, also in D_e) are covered by Algorithm 1 in previous iterations before time $\pi(e)$. Otherwise, if ties were broken in favor of e in line 4, we can remove the corresponding edges from E_e^- and D_e. So, the edges in E_e^- precede e in the non-preemptive solution, i.e., $\pi(f) < \pi(e)$ for all $f \in E_e^-$.

For the cover time of e, we obtain

$$\begin{aligned}\pi(e) &= \kappa(e) + |\{v \in f \mid f \in E_e^-, \pi(v) < \pi(e)\}| \\ &= \kappa(e) + |\{v \in f \mid f \in D_e, \pi(v) < \pi(e)\}| \\ &= \kappa(e) + \sum_{f \in D_e} \kappa(f). \end{aligned} \tag{4.4}$$

The first equality is by definition of the cover time and since the edges in E_e^- are precisely those edges that are covered before e and are disjoint from e (i.e., any vertex in such an edge does not contribute to the covering of e). The second equality holds because every $f \in E_e^-$ is contained in some $f' \in D_e$ and $D_e \subseteq E_e^-$. The last inequality is due to the fact that edges in D_e are disjoint and every edge $f \in E$ is covered as soon as $\kappa(f)$ of its vertices have appeared in the linear ordering.

On the other hand, let $\nu_v(t)$ be the fractional amount of vertex $v \in V$ that appears before time $t \in [n]$ in the preemptive solution. For the cover time of e in the preemptive solution, we have

$$\begin{aligned}\pi'(e) = \sum_{v \in V} \nu_v(\pi'(e)) &= \sum_{v \in e} \nu_v(\pi'(e)) + \sum_{v \notin e} \nu_v(\pi'(e)) \\ &\geq \kappa(e) + \sum_{v : \exists f \in E_e^- \text{ with } v \in f} \nu_v(\pi'(e)) \\ &= \kappa(e) + \sum_{f \in D_e} \sum_{v \in f} \nu_v(\pi'(e)) \\ &\geq \kappa(e) + \sum_{f \in D_e} \sum_{v \in f} \nu_v(\pi'(f)) \\ &\geq \kappa(e) + \sum_{f \in D_e} \kappa(f). \end{aligned} \tag{4.5}$$

The first inequality holds, since the first sum in the second row contains fewer summands than the last sum in the first row and $\sum_{v \in e} \nu_v(\pi'(e)) \geq \kappa(e)$ by definition of the cover time. The equality in the third row is similar to Eq. (4.4) and the second inequality follows from $\nu_v(t) \geq \nu_v(s)$ for $t \geq s$ and all $v \in V$. Hence, when comparing Eq. (4.4) and Eq. (4.5), we get $\pi(e) = \kappa(e) + \sum_{f \in D_e} \kappa(f) \leq \pi'(e)$ for any $e \in E$ that is considered explicitly by Algorithm 1 after line 6.

Finally, let $f \in E$ with $f \subseteq e$ be an edge that is never chosen explicitly by the algorithm, but is covered in line 12, because its covering requirement is set to zero in the iteration where $e \in E$ is considered. Since f has not yet been considered by the algorithm in lines 4 or 6, it satisfies $\pi'(f) \geq \pi'(e)$. By construction, $\pi(f) \leq \pi(e)$. With Eq. (4.4) and Eq. (4.5), we get $\pi'(f) \geq \pi'(e) \geq \pi(e) \geq \pi(f)$, which yields the claim. □

Now, we prove that there is a 2-approximation for weighted, laminar GMSSC.

Proof (Theorem 4.9). First, we use the algorithm of [61] to obtain a 2-approximate solution for preemptive laminar GMSSC. Note that the algorithm in [61] also works for the weighted variant. Then, we apply Algorithm 1 to transform the preemptive solution into a non-preemptive one. By Theorem 4.11, the objective value of the solution returned by Algorithm 1 is

$$\sum_{e \in E} w(e)\pi(e) \leq \sum_{e \in E} w(e)\pi'(e).$$

Since preemptive GMSSC is a relaxation of GMSSC and the algorithm of [61] is 2-approximate, we have

$$\sum_{e \in E} w(e)\pi'(e) \leq 2 \sum_{e \in E} w(e)\pi^*(e),$$

where $\pi^*(e)$ are the cover times of an optimum solution for the instance of laminar GMSSC. □

4.3.4 Laminar All-But-K Min Sum Set Cover

In the following, we provide another special case of laminar GMSSC that can be solved in polynomial-time. In Theorem 4.4 we proved that laminar GMSSC can be solved in polynomial-time for unit covering constraints. This is also the case for the more general covering constraints of the form $\kappa(e) = \max\{1, |e| - K\}$ for all $e \in E$ and some $K \in \mathbb{N}_0$. We refer to GMSSC with this particular choice of covering constraints as the LAMINAR ALL-BUT-K MIN SUM SET COVER PROBLEM. Our main result is a polynomial-time algorithm for laminar ALL-BUT-K MSSC. We actually consider the more general weighted, laminar ALL-BUT-K MSSC, where every edge $e \in E$ is associated with a positive weight $w_e \in \mathbb{N}$ and the objective is to minimize the sum of weighted cover times.

Theorem 4.12
Weighted, laminar ALL-BUT-K MSSC *can be solved in polynomial-time.*

The high-level idea of the polynomial-time algorithm for laminar ALL-BUT-K MSSC is the following. First, we construct an equivalent instance of laminar Min Latency Set Cover (Algorithm 2), and in a second step we solve this instance to optimality.

To distinguish an instance of ALL-BUT-K MSSC from its correpsonding MLSC instance, the edges in the MLSC instance are indicated with an overline, i.e., are denoted by $\overline{e}$, $\overline{f}$, etc. Similarly, we denote the collection of edges of the MLSC instance by $\overline{E}$.

Before describing the algorithm in more detail, we introduce the following notation. Observe, that since E is laminar, there is a unique inclusion-minimal superset for every

$v \in V$. For $e \in E$, let $U_e \coloneqq \{v \in e \mid \nexists f \in E \text{ with } v \in f \subset e\}$ be the vertices in e that are not contained in any subset $f \in E$ of e. The vertices in U_e are called **free** in e. An edge $f \in E$ with $f \subset e$ is called **maximal in** e **w.r.t.** E if there is no $f' \in E$ with $f \subset f' \subset e$.[3] We can partition an edge $e \in E$ into a disjoint union

$$e = U_e \ \cup \bigcup_{\substack{f \in E,\, f \subset e \\ \text{max w.r.t. } E}} f. \tag{4.6}$$

Definition 4.13 (Level of an Edge)
Let E be a laminar family, $e \in E$ and $U_e \subseteq e$ be the free vertices in e. The **level** of e is recursively defined as

1. $\operatorname{lev}(e) \coloneqq 0$ if $e = U_e$, i.e., there is no $f \in E$ with $f \subset e$, and
2. $\operatorname{lev}(e) \coloneqq 1 + \max\{\operatorname{lev}(f) \mid f \in E,\ f \subset e \text{ maximal in } e \text{ w.r.t. } E\}$.

Fig. 4.11 illustrates the definition of free vertices of an edge and the definition of a level of an edge.

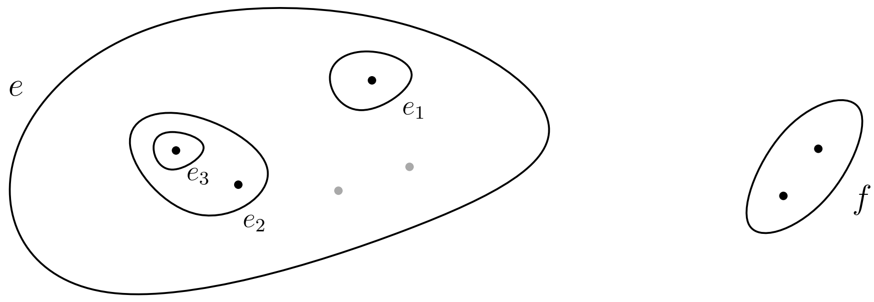


Fig. 4.11: Instance of All-But-K Mssc. The free vertices of edge e are highlighted in gray. The edges f, e_1 and e_3 are level 0 edges, the level of e_2 is 1 and the level of e is 2. As stated in (4.6) the edge e can be partitioned into a disjoint union $e = U_e \cup e_1 \cup e_2$ of maximal edges in e.

Algorithm Description

The algorithm to transform an instance of All-But-K Mssc to an equivalent instance of Mlsc works as follows. We iterate over the levels, i.e., we start with the edges in E of level 0, hence, they do not contain other edges in E. Since the vertices in V are indistinguishable, it does not matter which $\kappa(e)$ of the $|e|$ vertices cover an edge $e \in E$ of level 0. We remove $|e| - \kappa(e)$ of the (free) vertices from any edge e at level 0.

[3]Note that $f \subset e$ with $f \in E$ can be maximal in e w.r.t. E, although e is not contained in E.

Note that these vertices are not removed from the instance, but are now free in the unique inclusion-minimal superset of e. The resulting edge, which is a subset of e, is denoted by $\overline{e}$. We continue in the same manner with the edges at levels $1, \dots, n$, i.e., we remove $|e| - \kappa(e)$ of the free vertices and add them to the minimal superset. If there are not enough free vertices to be removed, we additionally remove the singleton sets $\{v\} \in \overline{E}$ with $\{v\} \subset e$ of lowest weight from e, and "place" them into the inclusion-minimal superset of e. Lemma 4.14 states that this can be done in a feasible way.

Thereby, we successively decrease the size of all edges in E until their cardinalities coincide with their covering requirements. This procedure might produce duplicate edges if we create the edge $\overline{e}$ when dealing with an edge $e \in E$, although the instance on $\overline{E}$ already contains an edge $\overline{f} = \overline{e}$ from an earlier iteration where $f \in E$ was considered. In this case, the "old edge" $\overline{f} \in \overline{E}$ is replaced by the "new edge" $\overline{e}$, i.e., we update $\overline{E}$. The weight of $\overline{e}$ is the sum of the weights of $\overline{f}$ and e. Note that this case only occurs if e is a superset of $\overline{f}$.

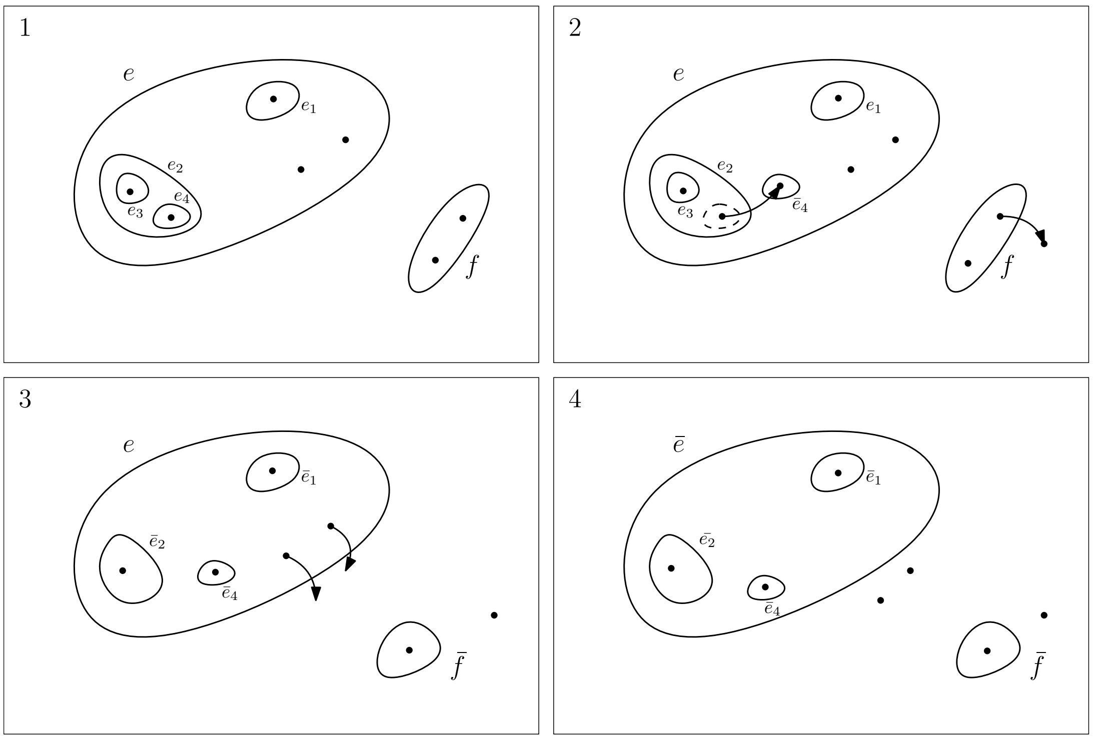


Fig. 4.12: Transformation of an instance of All-But-2 Mssc with unit weights to an instance of Mlsc. Consider edges in order of increasing level. Note that since $\overline{e}_3 = \overline{e}_2$, we delete both edges from the instance and replace it by an edge $\overline{e}_3$ of weight 2.

By updating $\overline{E}$ in this way, we ensure that $\overline{f} \subset e$ is maximal in e w.r.t. $\overline{E}$ if and only if the corresponding edge $f \in E$ with $f \subset e$ is maximal in e w.r.t. E. Also, if $\overline{e} = \{v\}$ is a singleton, then $v \in f \subset e$ for all subsets $f \in E$ of $e \in E$. See Fig. 4.12 for an example of how to transform an instance of All-But-K Mssc to an instance of Mlsc.

Algorithm 2: An algorithm to transform an instance of laminar All-But-K Mssc to an equivalent instance of laminar Mlsc.

Input: Instance $(\mathcal{H} = (V, E), \kappa)$ of weighted laminar All-But-K Mssc
Output: Instance $(\mathcal{H} = (V, \overline{E}), \overline{\kappa})$ of weighted laminar Mlsc

```
1  $\overline{E} \leftarrow \emptyset$;
2  for $\ell = 0, 1, \ldots, n$ do
3      for $e \in E$ with $\operatorname{lev}(e) = \ell$ do
4          $U \leftarrow e \setminus \bigcup_{\substack{\overline{f} \in \overline{E} \\ \overline{f} \subset e}} \overline{f}$;
5          $\mathcal{S} \leftarrow \left\{ \overline{\{v\}} \in \overline{E} \,\middle|\, \overline{\{v\}} \subset e \text{ is maximal in } e \text{ w.r.t. } \overline{E} \right\}$;
6          $q \leftarrow |e| - \kappa(e)$;
7          if $|U| \geq q$ then
8              Choose $U^- \subseteq U$ of cardinality $q$;
9          else
10             Sort $\mathcal{S}$ in non-decreasing order of the weights;
11             Let $\mathcal{S}^- \subseteq \mathcal{S}$ be the first $q - |U|$ vertices of $\mathcal{S}$;
12             $U^- \leftarrow U \cup \{v \mid \overline{\{v\}} \in \mathcal{S}^-\}$;
13         end
14         $\overline{e} \leftarrow e \setminus U^-$;
15         if $\exists\, \overline{S} \in \overline{E}$ with $\overline{f} = \overline{e}$ then
16             $w_{\overline{e}} \leftarrow w_{\overline{f}} + w_e$;
17             $\overline{E} \leftarrow (\overline{E} \setminus \{\overline{f}\}) \cup \{\overline{e}\}$;
18         else
19             $\overline{E} \leftarrow \overline{E} \cup \{\overline{e}\}$ and $w_{\overline{e}} \leftarrow w_e$;
20         end
21     end
22 end
23 return instance $(\mathcal{H} = (V, \overline{E}), \overline{\kappa})$ with requirements $\overline{\kappa} : \overline{E} \to \mathbb{N}$, $\overline{\kappa}(\overline{e}) = |\overline{e}|$;
```

The algorithm is summarized in Algorithm 2. The free vertices and singleton edges in the current edge e are defined in lines 4 and 5, respectively. The if-clause in line 7 checks whether there are "enough" free vertices to be removed. If not, the algorithm chooses all free vertices and the singleton edges of lowest weight to be removed in line 12. Finally, the if-clause in line 15 checks whether an edge is duplicated, and if so, the "old edge" is replaced by the "new edge" and the weights are updated accordingly.

Algorithm Analysis

Note that Algorithm 2 runs in polynomial-time, since every edge in E is considered exactly once. The following lemma shows that there are always enough free vertices and singletons to be removed, and that the output of Algorithm 2 is an instance of weighted laminar Mlsc.

Lemma 4.14
Algorithm 2 works correctly, i.e., for every $e \in E$, we have $|e| \leq |U| + |\mathcal{S}| + \kappa(e)$, where U and $\mathcal{S}$ are the corresponding set of free vertices and the set of singletons in the iteration of e. Further, $\overline{E}$ is laminar.

Proof. We first turn to the second part of the statement, and assume, for the moment, that Algorithm 2 terminates correctly. For $\ell \in \{0, 1, \dots, n\}$, let $\overline{E}_\ell$ be the current collection of subsets at the end of the for-loop for ℓ. Clearly, $\overline{E}_0$ is laminar, since it comprises of disjoint edges. Now, assume that $\overline{E}_{\ell-1}$ is laminar for some $\ell \in [n]$. In the next iteration of the outer for-loop, the set $\overline{e}$ is derived from $e \in E$ with $\operatorname{lev}(e) = \ell$ by removing free vertices and singletons. Note that $\overline{e} \in \overline{E}_\ell$ is a superset of all non-singleton edges $\overline{f} \in \overline{E}_{\ell-1}$ with $f \subset e$, $f \in E$. Also, two disjoint edges $e, e' \in E$ with $\operatorname{lev}(e) = \operatorname{lev}(e') = \ell$ do not intersect by Definition 4.13. Hence, the resulting edges $\overline{e}, \overline{e}' \in \overline{E}_\ell$ do not intersect. So, $\overline{E}_\ell$ is laminar by induction, and $\overline{E} = \overline{E}_n$ is laminar.

We prove the first part of the statement by induction on the level. If $\operatorname{lev}(e) = 0$ then $e = U$, i.e., $|e| \leq |U| + \kappa(e)$ by non-negativity of $\kappa(e)$. Let $\ell \in [n]$ and consider an edge with $\operatorname{lev}(e) = \ell$. By induction hypothesis, all edges $f \in E$ with $\operatorname{lev}(f) \leq \ell - 1$ were already processed by the algorithm in previous iterations. Note that $|\overline{f}| = \kappa(f) \leq |f|$ for $f \in E$ and its corresponding set $\overline{f} \in \overline{E}$. Similar to (4.6), we get

$$e = U \cup \bigcup_{\substack{\overline{f} \in \overline{E}, \overline{f} \subset e \\ \text{max w.r.t. } \overline{E}}} \overline{f} = U \cup \bigcup_{\{v\} \in \mathcal{S}} \{v\} \cup \bigcup_{\substack{\overline{f} \in \overline{E}, \overline{f} \subset e \\ \text{max w.r.t. } \overline{E} \\ |\overline{f}| \geq 2}} \overline{f}. \tag{4.7}$$

If e contains no maximal subsets w.r.t. $\overline{E}$ of cardinality ≥ 2, then $|e| = |U| + |\mathcal{S}| \leq |U| + |\mathcal{S}| + \kappa(e)$ by non-negativity of $\kappa(e)$. Also, $\kappa(e) = 1$ implies $\kappa(f) = 1$ for all $f \subset e$ and $f \in E$. In that case, all edges $\overline{f} \in \overline{E}$ with $\overline{f} \subset e$ are singletons. So, assume there is at least one maximal subset w.r.t. $\overline{E}$ of cardinality ≥ 2 and $\kappa(e) = |e| - K > 1$. With (4.7), we obtain

$$\begin{aligned}
|e| &= |U| + |\mathcal{S}| + \sum_{\substack{\overline{f}\in\overline{E},\,\overline{f}\subset e\\ \text{max w.r.t. } \overline{E}\\ |\overline{f}|\geq 2}} |\overline{f}| = |U| + |\mathcal{S}| + \sum_{\substack{f\in E,\,f\subset e\\ \text{max w.r.t. } E\\ |\overline{f}|\geq 2}} \kappa(f) \\
&= |U| + |\mathcal{S}| + \sum_{\substack{f\in E,\,f\subset e\\ \text{max w.r.t. } E\\ |\overline{f}|\geq 2}} (|f| - K) \leq |U| + |\mathcal{S}| + |e| - K \qquad (4.8)\\
&= |U| + |\mathcal{S}| + \kappa(e).
\end{aligned}$$

This proves the claim. □

Lemma 4.15
Let $\mathcal{I}$ be an instance of weighted laminar All-But-K Mssc*, and let $\overline{\mathcal{I}}$ be the corresponding instance of weighted laminar* Mlsc *that is returned by Algorithm 2. Then any optimal solution for $\overline{\mathcal{I}}$ is also optimal for $\mathcal{I}$.*

Proof. A feasible linear ordering for $\overline{\mathcal{I}}$ is also feasible for $\mathcal{I}$. Consider a linear ordering $\pi : V \to [n]$ that is optimal for $\mathcal{I}$. We show that we can alter π without increasing its objective value such that it is feasible for $\overline{\mathcal{I}}$. So, any optimal solution for $\overline{\mathcal{I}}$ has lower total weighted covering cost than π, i.e., it is optimal for $\mathcal{I}$, which proves the claim.

For $e \in E$, let U_e and $\mathcal{S}_e$ be the free vertices and singletons in the corresponding iterations of Algorithm 2, respectively. Let $\mathcal{S}'_e = \{f \in E \mid f \subset e,\, \kappa(f) = 1\}$ be the edges that correspond to the singletons in $\mathcal{S}_e$ or are contained in such an edge. That is, $\mathcal{S}'_e$ is the set of "pre-images" of the singletons in $\mathcal{S}_e$. Recall that every edge e is covered as soon as $\kappa(e)$ vertices appeared in the linear ordering. In particular, $\pi(v) \geq \pi(f')$ for all $\overline{\{v\}} \in \mathcal{S}_e$ and $f' \in \mathcal{S}'_e$ with $v \in f'$. We show by induction on the level that, w.l.o.g., the following holds for all $e \in E$:

1. $\pi(f) \leq \pi(e)$ for all $f \subseteq e$, $f \in E$ with $\kappa(f) > 1$,
2. $\pi(v) > \pi(e)$ for all $v \in U_e \setminus \overline{e}$, and
3. if $\pi(u) \leq \pi(e) < \pi(v)$ for $u, v \in e$ with $\overline{\{u\}}, \overline{\{v\}} \in \mathcal{S}_e$, then $w_{\overline{\{u\}}} \geq w_{\overline{\{v\}}}$.

Observe that π is feasible for the Mlsc instance $\overline{\mathcal{I}}$ if it satisfies these three conditions. The first condition yields that every edge is preceded by its non-singleton subsets. Conditions 2 and 3 state that an edge is not preceded by the vertices and singletons that are removed by Algorithm 2. It remains to be shown that we can alter the optimal solution π such that it satisfies 1, 2 and 3.

If $\text{lev}(e) = 0$, then 1 and 3 are trivially true. As for 2, suppose there is $v \in U_e \setminus \overline{e}$ with $\pi(v) \leq \pi(e)$. The set e is covered as soon as $\kappa(e)$ vertices appeared, so there is

$u \in \overline{e}$ with $\pi(e) < \pi(u)$. Recall that $\text{lev}(e) = 0$ implies that E does not contain any subsets of e. Hence, u and v are contained in the same edges in E. So, swapping u and v does not destroy feasibility of the linear ordering nor does the objective value increase. This establishes the base case. Now, suppose $\text{lev}(e) \geq 1$, i.e., E contains at least one subset of e.

1. For $\kappa(e) = 1$, the statement is trivially true because $\kappa(f) = 1$ holds for all $f \subset e$. So, assume $\kappa(e) = |e| - K$ and suppose that an edge $f \subset e$ with $\kappa(f) = |f| - K$ is covered after e, i.e., $\pi(f) > \pi(e)$. That is, K vertices of f appear strictly after time $\pi(f)$. Since f is covered as soon as $\kappa(f)$ vertices of f appeared, there is a vertex in f that appears at time $\pi(f) > \pi(e)$. Hence, at least $K + 1$ vertices of e appear strictly after time $\pi(e)$. This is a contradiction to the feasibility of the solution because, then, at most $|e| - (K + 1) < \kappa(e)$ vertices appear before e.

2. Suppose there is a vertex $v \in U_e \setminus \overline{e}$ with $\pi(v) \leq \pi(e)$. Similar to the base case, there is $u \in \overline{e}$ with $\pi(e) < \pi(u)$. If there is an edge $f \subset e$, $f \in E$ with $v \in f$, then $v \in U_e$ implies that v was also part of the free vertices in the iteration of f, i.e., $v \in U_f \setminus \overline{f}$. By induction $\text{lev}(f) < \text{lev}(e)$, we get that $\pi(v) > \pi(f)$. So, v does not contribute to any subset of e, but only to supersets $e' \supseteq e$ with $e' \in E$. But then $u \in e \subseteq e'$ implies that we can swap v and u without loosing feasibility or increasing the objective value (some edges that contain u might be even covered earlier).

3. Suppose there are $u, v \in e$ with $\overline{\{v\}}, \overline{\{u\}} \in \mathcal{S}_e$ and $\pi(u) \leq \pi(e) < \pi(v)$ such that $w_{\overline{\{u\}}} < w_{\overline{\{v\}}}$. The edges in $\mathcal{S}'_e$ that contain u or v have a lower level than e. By induction and 2, all sets in $\mathcal{S}'_e$ that contain v or u are covered at time $\pi(v)$ or $\pi(u)$, respectively. Also, since $\overline{\{v\}}, \overline{\{u\}} \in \mathcal{S}_e$ are maximal in e w.r.t. $\overline{E}$, the corresponding edges $f_v, f_u \in E$ with $\overline{f_v} = \overline{\{v\}}$ and $\overline{f_u} = \overline{\{u\}}$ are maximal in e w.r.t. E. Hence, there is no $f' \subset e$, $f' \in E$ with $f_v \subset f' \subset e$ or $f_u \subset f' \subset e$. That is, v and u do not contribute to any subset of e other than the subsets of f_v and f_u, respectively. Similar to 2, if we swap v and u (together with the corresponding edges in $\mathcal{S}'_e$), we do not loose feasibility and obtain solution of better objective value, since $w_{\overline{\{u\}}} < w_{\overline{\{v\}}}$. This contradicts to the optimality of π.

Thus, we can, w.l.o.g., assume that the optimal solution $\pi : V \to [n]$ satisfies 1, 2 and 3. That is, π is feasible for the Mlsc instance $\overline{\mathcal{I}}$. Hence, any optimal solution for $\overline{\mathcal{I}}$ is also optimal for the initial instance $\mathcal{I}$ of All-But-K Mssc. □

Lemma 4.14 shows that the instance returned by Algorithm 2 is indeed an instance of laminar Mlsc, and Lemma 4.15 states that solving this instance is equivalent to solving the initial instance of laminar All-But-K Mssc. We are now in the position to prove that weighted, laminar All-but-K Mssc can be solved in polynomial-time.

Proof (Theorem 4.12). Consider an instance $\mathcal{I}$ of weighted laminar All-But-K Mssc. First, we apply Algorithm 2 to obtain the corresponding instance $\overline{\mathcal{I}}$ of weighted laminar Mlsc. By Lemma 4.15, any optimal solution for $\overline{\mathcal{I}}$ is also optimal for $\mathcal{I}$. It remains to be shown that we can solve weighted laminar Mlsc in polynomial-time.

Let $\overline{\mathcal{I}} = (\mathcal{H} = (V, \overline{E}), \overline{\kappa})$ be an instance of weighted laminar Mlsc. We construct an equivalent AND-scheduling instance by using the in-forest representation of $\overline{\mathcal{I}}$. That is, we introduce a job j_v for every vertex $v \in V$ and a job $j_{\overline{e}}$ for every edge $\overline{e} \in \overline{E}$. The processing times and weights of the jobs are $p_{j_v} = 1$, $w_{j_v} = 0$ for all $v \in V$ and $p_{j_{\overline{e}}} = 0$, $w_{j_{\overline{e}}} = w_{\overline{e}}$ for all $\overline{e} \in \overline{E}$. The precedence constraints are given by the graph $G_{prec} = (J, \vec{A})$, where $J = \{j_v \,|\, v \in V\} \cup \{j_{\overline{e}} \,|\, \overline{e} \in \overline{E}\}$. The jobs $\{j_v \,|\, v \in V\}$ do not have any predecessors, and, for $\overline{e} \in \overline{E}$, the predecessors of $j_{\overline{e}}$ are the jobs corresponding to the maximal subsets and the free vertices in $\overline{e}$. That is, we introduce an edge $(j_{\overline{f}}, j_{\overline{e}}) \in \vec{A}$ for all $\overline{e}, \overline{f} \in \overline{E}$ such that $\overline{f} \subset \overline{e}$ is maximal in $\overline{e}$ w.r.t. $\overline{E}$ and an edge $(j_v, j_{\overline{e}}) \in \vec{A}$ for every $\overline{e} \in \overline{E}$ and free vertex $v \in U_{\overline{e}}$. Clearly, there is a one-to-one correspondence between a linear ordering of the vertices in V and a feasible single-machine schedule of the jobs in $\{j_v \,|\, v \in V\}$. Also, finding a linear ordering that minimizes the sum of weighted cover times is equivalent to finding a schedule that minimizes the sum of weighted completion times.

Since $\overline{E}$ is laminar, every edge in $\overline{E}$ is maximal in at most one edge and every vertex is free in at most one edge. So, every job of the scheduling instance described above has at most one successor, i.e., the constructed precedence graph $G_{prec} = (J, \vec{A})$ is an in-forest. Hence, we can solve weighted laminar MLSC by solving the corresponding scheduling problem $1 \,|\, prec \,|\, \sum w_j C_j$ with a an in-forest precedence graph, which can be done in polynomial time [59]. □

4.4 Open Problems

In this section, we studied the generalized Min Sum Set Cover problem on laminar families. For special choices of covering constraints such as Mssc, All-But-K Mssc, and Mlsc we provided polynomial-time algorithms for laminar instances. All of these algorithms rely heavily on the specific choice of covering constraints. For arbitrary covering constraints, the complexity remains open.

Problem 4.16
Is there a polynomial-time algorithm for laminar Gmssc?

Now consider Gmssc on general hypergraphs. Im et al. [61] showed that any solution to preemptive Gmssc can be converted to a solution of Gmssc, loosing a factor of at most 6.2. For laminar instances, we showed that one can convert any preemptive schedule to a non-preemptive schedule without increasing the cover times. For general hypergraphs, Bansal et al. [9] gave an example, which shows that the gap between the preemptive and non-preemptive schedule is at least 4.

Problem 4.17
Is there a polynomial-time algorithm that converts any preemptive schedule to a non-preemptive schedule for GMSSC, loosing a factor of at most 4?

Recently, Bansal et al. [9] improved the best-known approximation ratio for GMSSC from 12.4 to 4.642. Still, new methods seem to be necessary to close the gap and obtain a 4-approximation algorithm.

Problem 4.18 ([61])
Is there a 4-approximation algorithm for GMSSC?

Chapter 5

Bipartite Flow Scheduling

In this chapter, we consider a special case of the Coflow Scheduling problem. We are given an $m \times m$ switch, consisting of m input and m output ports. A coflow is a collection of flow demand, so-called tasks, between pairs of in- and output ports. Each port can send or receive at most one unit of flow in every time step. The completion time of a coflow is the earliest point in time when all tasks of the coflow have been completed. The goal is to determine a schedule that minimizes the sum of weighted completion times over all coflows.

Coflow Scheduling was introduced as a framework for data-parallel computing. In comparison to fundamental scheduling problems, it incorporates two significant challenges. First, the grouping constraints, that is, the requirement that each coflow only completes after all its tasks are completed, and secondly, the conflicts, that is no pair of tasks with the same input or output port can be scheduled at the same time. So-called Concurrent Open Shop Scheduling can be interpreted as the special case of Coflow Scheduling in which we neglect the conflict on the output ports. The scheduling problem considered in this chapter is the special case of Coflow Scheduling, in which we relax the grouping constraints. In other words, every coflow consists only of a single task. We refer to this problem as Bipartite Flow Scheduling.

In contrast to Concurrent Open Shop, for which there exists a clear line of research, Bipartite Flow Scheduling has been studied in the past under various names. In this chapter, we explain the connection of Bipartite Flow Scheduling to other scheduling and covering problems. We give an overview of existing results and propose further research directions.

The chapter is structured as follows. In Section 5.1, we introduce Coflow Scheduling formally and explain its application in the context of parallel computing. An overview of related work is given in Section 5.1.5. The remainder of this chapter deals with the Bipartite Flow Scheduling problem. We explain the connection of the problem to Min Sum Edge Coloring, Min Sum (Multi-)Coloring, Min Sum Set Cover, Data Migration, and Generalized Open Shop and summarize existing hardness results (Section 5.2.1) and approximation algorithms (Section 5.2.2).

5.1 Introduction

The Coflow Scheduling Problem is defined in the following way. We are given a switch that consists of m input and m output ports. A **coflow** j consists of a collection of **tasks** and can be described by an integer matrix $D_j := (d^j_{i,o})_{i,o\in[m]}$, where $d^j_{i,o}$ represents the amount of flow that needs to be transferred from input port i to output port o in order to process coflow j. Due to capacity restrictions each input and output port can only receive or send one unit of flow at a time. In particular, this implies that we schedule a matching in every time step (see Section 5.1.1). Consider a set of coflows J. For every coflow $j \in J$ we are given a **release time** r_j and a **weight** w_j. The **completion time** C_j of a coflow j is defined as the earliest point in time when all of its tasks have been scheduled. The goal is to minimize the sum of weighted completion times, i.e. $\min \sum_{j\in J} w_j C_j$.

Coflow Scheduling was initially introduced in the context of distributed computing. MapReduce, Spark, and Dyrad are frameworks developed for efficiently processing large data sets. MapReduce, for example, does so by distributing the data to multiple systems. The tasks are then processed in parallel, and the final results aggregated again. Compared to classical network flow problems, we are given a collection of tasks in this setting, and a coflow completes only after all its tasks have been scheduled. Coflow Scheduling has also been studied extensively from an application point of view, and various heuristics are known. However, in this chapter, we exclusively focus on theoretical results.

The currently best-known approximation algorithms for Coflow Scheduling are a 4-approximation in case all release times are zero and a 5-approximation for the case of arbitrary release times [4, 96]. Up until now, it is unclear whether Coflow Scheduling can be approximated within a factor strictly smaller than 4. The best known lower bound is $2-\epsilon$ [91]. Hence, we are particularly interested in the existence of a 2-approximation algorithm for Coflow Scheduling. To get closer to answering this question, a deep understanding of the special cases of Coflow Scheduling may be essential.

Coflow Scheduling combines two challenges, namely the grouping constraints and the conflicts. Concurrent Open Shop can be seen as the special case of Coflow Scheduling that incorporates the grouping constraints but neglects conflicts. For more information on the Concurrent Open Shop problem, see Section 5.1.4. Whereas the Concurrent Open Shop problem has received a lot of attention and ideas have been used to generate approximation algorithms for Coflow Scheduling, the other special case, the so-called Bipartite Flow Scheduling Problem, has only received little attention in the context of Coflow Scheduling.

In this chapter, we focus on the Bipartite Flow Scheduling problem, which is the special case of Coflow Scheduling that still incorporates the conflicts but neglects the grouping constraints. In particular, this is the special case of Coflow Scheduling, where every coflow consists of a single edge or equivalently, the corresponding matrix D_j

contains precisely one non-zero entry. A detailed definition of the problem is given in Section 5.2. Bipartite Flow Scheduling is closely related to Min Sum Edge Coloring, Min Sum (Multi-)Coloring, Min Sum Set Cover, and Generalized Open Shop Scheduling. We explain their connection and summarize existing approximation algorithms and hardness results for Bipartite Flow Scheduling.

5.1.1 Bipartite Graph Representation of Coflow Scheduling

Besides the matrix representation, given above, it is often convenient to represent a coflow by a bipartite graph. Let j be a coflow defined by the integer matrix $D^j := (d^j_{i,o})_{i,o\in[m]}$. The corresponding bipartite graph is given by $G_j = (I \cup O, E_j)$, where I and O represent the set of m input and output ports, respectively. The set of edges is defined as $E_j := \{e^j_{i,o} = (i,o) \mid d^j_{i,o} > 0\}$ with weights $w(e^j_{i,o}) := d^j_{i,o}$. That is, there exists an edge from input port i to output port o of weight $d_{i,o}$ in the graph G_j, whenever coflow j requires $d_{i,o}$ units of flow to be transferred from i to o. The following example illustrates the Coflow Scheduling problem and visualizes both, the matrix and the bipartite graph representation of a coflow.

Example 5.1
We are given the following instance of Coflow Scheduling on three input and three output ports, displayed in Figure 5.1. The set of coflows is given by $J = \{j_1, j_2, j_3\}$

$$D_{j_1} := \begin{bmatrix} 2 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \qquad w(j_1) = 10$$

$$D_{j_2} := \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad w(j_2) := 5$$

$$D_{j_3} := \begin{bmatrix} 0 & 0 & 1 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad w(j_3) = 1$$

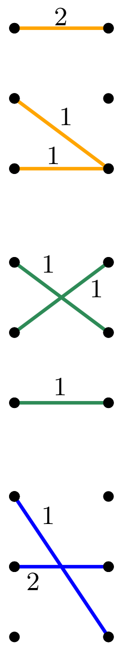


Fig. 5.1: Example of three coflows in matrix and bipartite graph representation.

with corresponding weights $w(j_1) = 10$, $w(j_2) = 5$ and $w(j_3) = 1$. All release times are equal to zero. A feasible schedule is given in Figure 5.2. Note that the yellow edges correspond to coflow j_1, the green edges to coflow j_2, and the blue edges to coflow j_3. Coflow j_1 completes at time 2, coflow j_2 completes at time 3, and the last task of coflow j_3 is scheduled at time 4. Hence, the sum of weighted completion times of the given schedule is $\sum w_j C_j = 10 \cdot 2 + 5 \cdot 3 + 1 \cdot 4 = 39$.

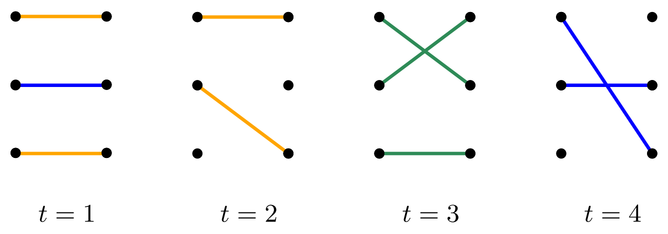


Fig. 5.2: A feasible schedule of the instance displayed in Figure 5.1. Yellow edges correspond to coflow j_1, green edges to coflow j_2 and blue edges to coflow j_3.

5.1.2 Preemption in the Context of Coflow Scheduling

At this point, it makes sense to briefly discuss the meaning of preemption in the context of Coflow Scheduling. The (non-preemptive) Coflow Scheduling problem is preemptive by default since we neither require the different tasks of a coflow nor a task itself to be scheduled in subsequent times steps (see Figure 5.2). We refer to this type of preemption as **integer preemption**. Whenever we use the term **preemptive** Coflow Scheduling, we refer to non-integer preemption. That is, in the preemptive Coflow Scheduling problem, we do not necessarily schedule a matching in every time step. Instead, we are also allowed to schedule fractional matchings.

5.1.3 Coflow Scheduling to Minimize Makespan

Scheduling Coflows with the objective of minimizing the makespan $C_{\max}$ can be solved in polynomial time. One way to observe this is the following. Given a set of coflows J, consider their bipartite graph representation and let $G_J := \bigcup_{j \in J} G_j$ be the cumulative multigraph that contains an edge for every task of a coflow in J. Let Δ be the maximum weighted degree of the graph G_J. Clearly, we need at least Δ time steps to schedule all coflows, so $C_{\max} \geq \Delta$. Qiu et al. [89] showed that there always exists a so-called *Birkhoff-von Neumann Decomposition* of G_J into Δ matchings. The main idea is to replace every edge (i, o) of a coflow j by $d^j_{i,o}$ copies, add dummy edges to G_J until all edges have degree Δ and decompose the resulting graph into Δ maximum matchings. See Figure 5.3 for an example. Hence, there always exists a schedule with

$C_{\max} = \Delta$. Moreover, a Birkhoff-von Neumann Decomposition can be computed in polynomial-time.

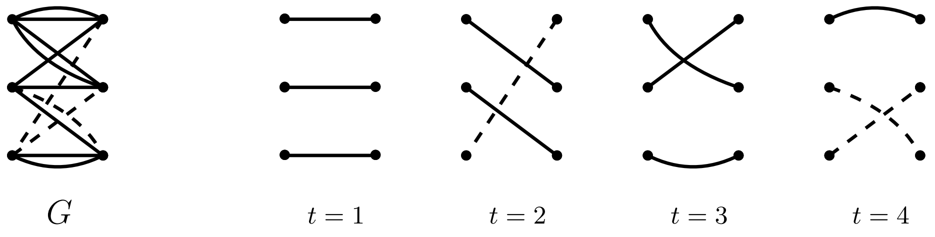


Fig. 5.3: Example of a Birkhoff-von Neumann Decomposition of a graph G. Dummy edges are displayed as dashed edges.

Another way to observe that Scheduling Coflows to minimize makespan can be solved in polynomial-time is by reformulating it as Matroid Intersection Cover problem on two partition matroids. This can be solved in polynomial-time using Edmond's Matroid Intersection algorithm [38] since it equals the Matroid Intersection problem on two partition matroids. See Chapter 3 for more information on Matroid Covering problems. Throughout this chapter, Coflow Scheduling will always refer to the objective of minimizing the sum of weighted completion times.

5.1.4 Connection to Concurrent Open Shop

We already mentioned the connection of Coflow Scheduling to Concurrent Open Shop Scheduling in the introduction. In the following, we introduce the problem formally and explain its connection to Coflow Scheduling in more detail.

The Concurrent Open Shop Scheduling Problem is defined as follows. Given a set of m machines and jobs j that consist of tasks, every task can only be executed by a particular machine, and the processing time of a task that belongs to job j on machine i is p_{ij}. The completion time of a job is the first point in time when all its tasks are completed. In contrast to the Open Shop Scheduling problem, different tasks of the same job can be executed in parallel on different machines [88]. Additionally, every job has a corresponding weight w_j, and the goal is to minimize the sum of weighted completion times. Following the three-field notation by Graham [50], the Concurrent Open Shop Scheduling problem is in the literature often denoted by PD $|| \sum w_j C_j$. Concurrent Open Shop Scheduling has many applications in the context of maintenance work and manufacturing and was studied extensively in both the theory and application community. An overview of the theoretical results for Concurrent Open Shop is given in Section 5.1.5.

Concurrent Open Shop relates to Coflow Scheduling in the following way. Concurrent Open Shop can be seen as the special case of Coflow Scheduling, where the integer matrices D^j of all coflows j are given by diagonal matrices. In other words, the only non-zero entries in D^j are on the diagonal. Figure 5.4 visualizes why Concurrent Open Shop can be seen as Coflow Scheduling without conflicts. We only require flow to be transferred from input port i to output port i. In particular, having no conflict between two tasks at the input port implies that there is also no conflict at the output ports, or in terms of the Concurrent Open Shop terminology, there is no conflict between two tasks on different machines.

Wagneur [102] observed that there always exists an optimal **permutation schedule** for Concurrent Open Shop, that is, a solution in which jobs are scheduled in the same order on every machine. This also implies that there is no benefit in preempting jobs in the Concurrent Open Shop setting. Note that for Coflow Scheduling, we cannot guarantee the existence of an optimal permutation schedule [28].

$$D_j := \begin{bmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

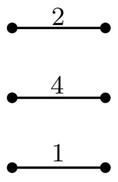


Fig. 5.4: Example of a coflow j to visualize the connection to Concurrent Open Shop. The matrix representing coflow j is diagonal and, hence, we can interpret j as a job of a Concurrent Open Shop instance. In the Concurrent Open Shop setting, we are given three machines and j corresponds to a job consisting of tasks with processing times $p_{1j} = 2, p_{2j} = 4$ and $p_{3j} = 1$ on the respective machines.

5.1.5 Related Work

Coflow Scheduling. Chowdhury and Stoica [27] introduced the Coflow Scheduling framework as an abstraction for cluster applications. In [28], Chowdhury et al. presented a heuristic for Coflow Scheduling and discussed the complexity of the problem. In addition, they pointed out the connection to Concurrent Open Shop Scheduling. In contrast to the latter, the existence of an optimal permutation schedule is not necessarily given for Coflow Scheduling. A simple counterexample was provided in [28]. The connection to Concurrent Open Shop implies that Coflow Scheduling remains NP-hard even for zero release times, unit weights, and all $d_{i,o} \in \{0, 1\}$ [48].

Qiu et al. [89] presented the first constant-factor approximation algorithm for Coflow Scheduling. It consists of two stages. First, they formulated a time-indexed linear program with relaxed matching and load constraints. To get a formulation of polynomial-size, they divided time into geometrically increasing intervals. The obtained interval-indexed linear program can be solved in polynomial-time by the interior point method. The completion times of the coflows in the optimal LP solution induce an ordering of the coflows. This part is mainly based on ideas from Wang and Cheng [104] for Concurrent Open Shop. In the second stage, the ordering of the coflows is converted into a feasible

Table 5.1: Overview of approximation guarantees of known algorithms for Coflow Scheduling with and without release times.

	Deterministic		Randomized	
	without release times	with release times	without release times	with release times
Qiu et al. [89]	$\frac{64}{3}$	$\frac{76}{3}$ [4][2]	$8 + \frac{16\sqrt{2}}{3}$	$9 + \frac{16\sqrt{2}}{3}$
Khuller, Purohit [69]	8	12	$3 + \sqrt{2}$	
Ahmadi et al. [4]	4	5		
Ghaderi, Shafiee [96]	4	5		

schedule. The coflows are grouped based on the ordering. Each group is treated as a single coflow and then decomposed into matchings using a Birkhoff-von Neumann Decomposition[1]. Overall, Qiu et al. [89] obtained a deterministic $\frac{64}{3}$-approximation algorithm and a randomized $8 + \frac{16\sqrt{2}}{3}$-approximation algorithm for the case where all release times are zero. Their algorithm for arbitrary release times, as stated in the paper, is flawed and was corrected by Ahmadi et al. [4] to a $\frac{76}{3}$-approximation.

Khuller and Purohit [69] improved the approximation guarantee further by presenting a 12-approximation for Coflow Scheduling. For the special case where all release times are equal to zero, they obtained an 8-approximation. Given an instance of Coflow Scheduling, they solved a corresponding instance of Concurrent Open Shop and used the LP solution to obtain an ordering of the coflows. Stage two of the algorithm is equivalent to the grouping algorithm presented by Qiu et al. [89].

Two algorithms with the currently best-known approximation guarantee of 5 were presented by Ahmadi et al. [4] and Shafiee and Ghaderi [96]. The algorithm by Ahmadi et al. [4] is the first purely combinatorial algorithm. In the first stage, an ordering of the coflows is obtained through a primal-dual algorithm. Simply scheduling coflows according to the obtained ordering does not necessarily yield a good solution. Instead, Ahmadi et al. [4] presented an algorithm that greedily moves edges forward, as long as they do not increase the completion time of previous coflows. For the special case when all release times are zero, this implies a 4-approximation. Independently, Shafiee

[1]See Section 5.1.3 for more information on Birkhoff-von Neumann Decompositions.

[2]The originally claimed deterministic $\frac{64}{3}$-approximation in [89] was corrected by Ahmadi et al. [4] to a $\frac{76}{3}$-approximation.

and Ghaderi [96] derived an algorithm with the same approximation guarantee, using a completely different approach. They solved a linear program in linear ordering variables to obtain an ordering of the coflows. A simple list scheduling algorithm is then used to construct a feasible schedule. For zero release times, they also obtain a 4-approximation. Another 4-approximation was presented by Agarwal et al. [1]. It is based on two stages and works as follows. First, an ordering of the coflows is obtained through a primal-dual based Greedy algorithm. They showed that for the second stage, any per-flow rate allocation mechanism that is work-conserving[3], preemptive[4] and respects the ordering can be used to receive the final schedule with the desired approximation guarantee.

See Table 5.1 for an overview on the approximation guarantees of the currently known algorithms for Coflow Scheduling with and without release times. Note that besides the line of theoretical research, there is also an immense list of experimental results and suggested heuristics for Coflow Scheduling.

Concurrent Open Shop. The special case of Coflow Scheduling, where all coflows are represented by diagonal matrices, is called the Concurrent Open Shop Scheduling problem. An explicit definition of Concurrent Open Shop and more information on its connection to Coflow Scheduling is given in Section 5.1.4.

The problem was introduced by Ahmadi and Bagchi [3] and is known to be APX-hard, even for zero release times, unit weights and processing times $p_{ij} \in \{0, 1\}$ [48]. For Concurrent Open Shop with zero release times there exist multiple 2-approximations based on LP rounding [26, 48, 77] as well as a combinatorial algorithm [83]. Sachdeva and Saket [91] showed that it is NP-hard to approximate Concurrent Open Shop within a factor of $(2 - \epsilon)$, which implies tightness of the approximations given above. Concurrent Open Shop with arbitrary release times can be approximated with a factor of 3 [4, 48, 77].

Min Sum Coloring and Scheduling with Conflicts. The Min Sum Coloring problem was introduced by Kubicka and Schwenk [73] and is known to be NP-hard even on bipartite graphs [15]. Several approximation algorithms for different graph classes are know [63]. Closely related to this chapter are the results for Min Sum Coloring on line graphs, in particular a 2-approximation by Bar-Noy et al. [13].

Another closely related problem is given by the Data Migration problem, see e.g. [47]. Note that even though the setting is similar to Bipartite Flow Scheduling, the objective is different. Instead of minimizing the sum of weighted completion times of the jobs, the objective in the Data Migration problem is to minimize the sum of weighted completion times of the vertices of the graph. Scheduling with a conflict graph has been studied

[3]Work-conserving implies that for any task (i, o) that is scheduled at time t, at least one of the ports i or o is busy in all previous times steps $t' < t$.

[4]Preemption in this context refers to integer preemption.

mainly with the objective of minimizing the makespan, see e.g. [80], which is trivial for Coflow Scheduling (Section 5.1.3).

5.2 Bipartite Flow Scheduling

In this section, we consider the special case of Coflow Scheduling that still incorporates conflicts but neglects the grouping constraints. This is the special case, where every coflow consists of a single edge or, equivalently, for all coflows j, the corresponding matrix D_j contains precisely one non-zero entry.

More specifically, the Bipartite Flow Scheduling Problem is defined as follows. We are given a bipartite multigraph $G = (I \cup O, E)$ with partitions I and O. Every edge e in E corresponds to a job j, that is $J := E$. Additionally, for every job $j \in J$ we are given a processing time p_j and a weight w_j for every job j. We say that two jobs $j = (i, o)$, $j' = (i', o')$ have a **conflict**, that is $\text{conf}(j, j') = 1$, if $i = i'$ or $o = o'$. Otherwise let $\text{conf}(j, j') = 0$. The **total number of conflicts** is denoted by $\text{conf}(J) = \sum_{j \neq j' \in J} \text{conf}(j, j')$. In a feasible schedule, no pair of jobs that has a conflict is scheduled during the same time interval. Integer preemption is allowed, that is a job does not have to be scheduled in subsequent time intervals. The **completion time** C_j of job j is the first point in time, when p_j units of the job have been scheduled. The goal is to minimize the sum of weighted completion times, i.e. $\sum_{j \in J} w_j C_j$.

In some cases, it will be useful to differentiate between conflicts at input ports and conflicts at output ports. For two jobs $j = (i, o)$ and $j' = (i', o')$ we define

$$\text{conf}_I(j, j') := \begin{cases} 1 & \text{if } i = i' \\ 0 & \text{else} \end{cases} \quad \text{and} \quad \text{conf}_O(j, j') := \begin{cases} 1 & \text{if } o = o' \\ 0 & \text{else.} \end{cases}$$

The total number of conflicts at all input and output ports is denoted by $\text{conf}_I(J)$ and $\text{conf}_O(J)$, respectively. Note that since G is a multigraph, two jobs can have a conflict at both, the input and the output port side. Hence, $\text{conf}_I(J) + \text{conf}_O(J) \geq \text{conf}(J)$.

5.2.1 Complexity of Bipartite Flow Scheduling

We consider the simplest class of Bipartite Flow Scheduling instances, that is, instances with unit weights and unit processing times. Surprisingly, this case is already NP-hard, which follows directly from a result for Min Sum Edge Coloring by Marx [82].

Connection to Min Sum Edge Coloring

The Min Sum Edge Coloring problem is defined in the following way. Given a graph $G = (V, E)$, find a proper edge coloring $c : E \to \mathbb{N}$ that minimizes $\sum_{e \in E} c(e)$.

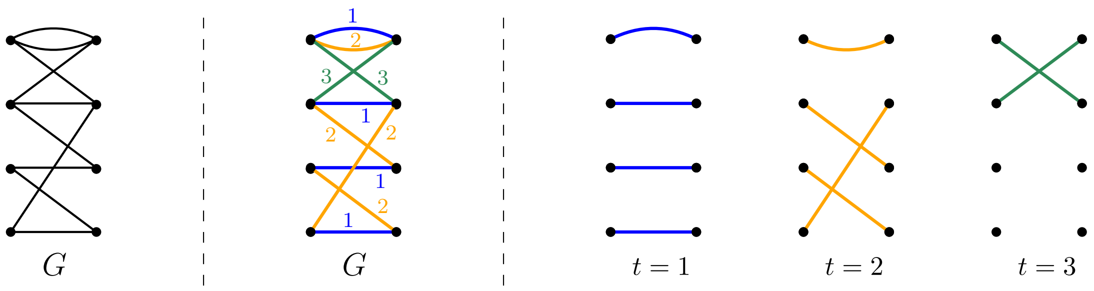


Fig. 5.5: An example to illustrate the connection between Min Sum Edge Coloring and Bipartite Flow Scheduling with unit weights and unit processing times. An optimal Min Sum Edge Coloring of G is depicted in the middle. The corresponding optimal schedule is visualized on the right.

It is easy to see that Bipartite Flow Scheduling with unit weights and unit processing times is equivalent to Min Sum Edge Coloring on bipartite multigraphs (see Figure 5.5 for a visualization).

Proposition 5.2 (Marx [82])
Bipartite Flow Scheduling remains NP-hard, even when restricted to unit weights and unit processing times.

Marx [82] showed that Min Sum Edge Coloring remains NP-hard on planar bipartite graphs with maximum degree 3, by a reduction from the Edge Precoloring Extension problem. The proposition above states the result in terms of Bipartite Flow Scheduling. Additionally, Marx [82] proved that Min Sum Edge Coloring is APX-hard on bipartite graphs.

5.2.2 Approximation Algorithms for Bipartite Flow Scheduling

Since the most trivial case of Bipartite Flow Scheduling is already NP-hard, we focus on approximation algorithms. When considering Bipartite Flow Scheduling with unit weights and unit processing times, the most natural idea is to schedule a maximal matching in every time step. In the following, we show that this simple idea yields a 2-approximation.

Lemma 5.3
Scheduling any maximal matching in every step is a 2-approximation algorithm for Bipartite Flow Scheduling with unit weights, unit processing times and zero release times.

Proof. Given an instance of Bipartite Flow Scheduling problem with unit weights, unit processing times and zero release times, let OPT be the value of an optimum

solution and denote by ALG the value of a solution obtained by the algorithm that schedules a maximal matching in every time step. We start by deriving a simple lower bound on OPT.
Let w.l.o.g. the input side be the one with more conflicts, that is $\mathrm{conf}_I(J) \geq \mathrm{conf}_O(J)$. Consider the following trivial scheduling problem. We are given a single machine for every input port in I and a partition $J_1, J_2, \dots, J_m$ of the jobs in J such that J_i is the set of jobs incident to input port i. The goal is to minimize $\sum_{i\in I}\sum_{j\in J_i} C_j$. Choose any ordering of the jobs and schedule every set of jobs J_i on machine i, respecting the ordering. The schedule obtained is optimal and let $\tilde{C}_j$ be the completion time of job j in this schedule. Clearly, an optimal solution to this trivial scheduling problem is a lower bound on OPT. Hence,

$$OPT \geq \sum_{i\in I}\sum_{j\in J_i}\tilde{C}_j = \sum_{i\in I}\sum_{j\in J_i}\Big(1 + \sum_{j'\colon\, \tilde{C}_{j'}<\tilde{C}_j} \mathrm{conf}_I(j,j')\Big) = \mathrm{conf}_I(J) + |J|. \tag{5.1}$$

On the other hand, we can bound any solution obtained by the algorithm in the following way. Denote by C_j the completion time of job j in the algorithm and assume the jobs are ordered with respect to their completion times. Choosing a maximal matching in every step implies that the solution has the following property. If a job j is scheduled at time t, in every step $t' < t$ at least one job that is in conflict with j must have been scheduled. Otherwise we could schedule j earlier which contradicts the fact that we chose a maximal matching. Hence, we can upper bound the completion time of job j by $\sum_{j'<j}\mathrm{conf}(j,j')+1$. Overall it holds that,

$$ALG = \sum_{j\in J} C_j = \sum_{j\in J}\Big(\sum_{j'\colon\, C_{j'}<C_j}\mathrm{conf}(j,j')+1\Big) = \mathrm{conf}(J) + |J| \leq 2\cdot OPT,$$

where the last inequality follows from (5.1) and the fact that $\mathrm{conf}(J) \leq 2\cdot\mathrm{conf}_I(J)$.
□

Note that an equivalent result in the context of Min Sum Coloring was already proven by Bar-Noy et al. [13].

Connection to Min Sum Coloring

The Min Sum Coloring problem is defined as follows. Given a graph $G = (V, E)$ on n vertices, find a proper vertex coloring $c : V \to \mathbb{N}$ that minimizes $\sum_{v\in V} c(v)$.

Observation 5.4
Bipartite Flow Scheduling with unit weights, unit processing times and zero release times is a special case of Min Sum Coloring on line graphs of bipartite multigraphs.

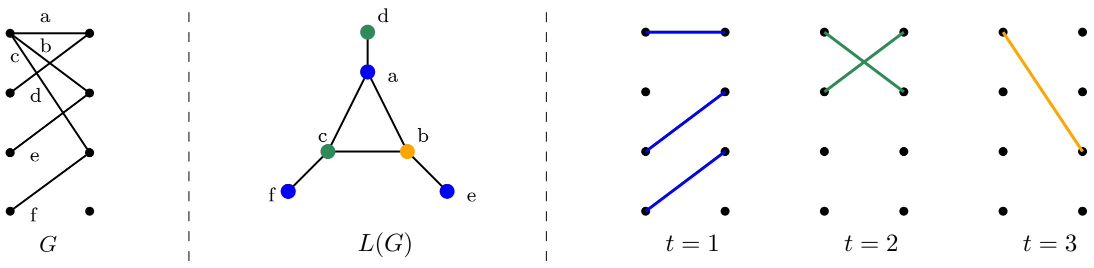


Fig. 5.6: An example to illustrate the connection to Min Sum Coloring. The instance of Bipartite Flow Scheduling is given on the left by G, and its line graph $L(G)$ is displayed in the middle together with an optimal Min Sum Coloring. Blue corresponds to color class 1, green to color class 2 and yellow to color class 3. The picture on the right illustrates the corresponding optimal schedule.

An example of an instance of Bipartite Flow Scheduling together with an optimal Min Sum Coloring of the line graph and the corresponding schedule is given in Figure 5.6. Halldórsson et al. [52] showed that for any graph on which the maximum weight k-colorable subgraph problem can be solved in polynomial-time, there exists a 1.796-approximation algorithm for weighted Min Sum Coloring. In the maximum weight k-colorable subgraph problem, we are given a graph G together with vertex weights. The goal is to find a k-colorable subgraph of maximum total weight. Gandhi et al. [46] observed that the k-colorable subgraph problem can be solved in polynomial-time on line graphs of bipartite multigraphs since it equals the weighted b-matching problem on the original bipartite multigraph. Combined with the result by Halldórsson et al. [52], this observation implies a 1.796-algorithm for the Bipartite Flow Scheduling problem with non-negative weights and unit processing times.

Proposition 5.5 (Halldórsson et al. [52])
There exists a 1.796*-approximation algorithm for Biparite Flow Scheduling with non-negative weights and unit processing times.*

Connection to Min Sum Set Cover

Bipartite Flow Scheduling can also be formulated as Min Sum Set Cover problem[5]. Consider an instance of Bipartite Flow Scheduling given by a bipartite multigraph G, where the edges correspond to the set of jobs. We construct an instance of Mssc in the following way. The vertex set of the hypergraph $\mathcal{H}$ contains a vertex for every matching in G, that is, $V(\mathcal{H}) := \{v_M \mid M \text{ matching in } G\}$. For every job j we define an edge $e_j := \{v_M \mid j = (i,o) \in M\}$, that is, the edge e_j contains all vertices that correspond to matchings containing j. Any linear ordering of the vertices corresponds

[5]For more information on Min Sum Set Cover and its generalizations, see Chapter 4.

to a schedule, in which we schedule a matching in every time step. Furthermore, the cover time of a set corresponds to the completion time of the corresponding job.

Note that the hypergraphs that correspond to instances of Bipartite Flow Scheduling have a particular structure. If a vertex corresponding to a matching M is contained in some edge e_j, all submatchings, including the edge j, are also contained in e_j. The hypergraphs may neither be regular nor uniform, which is why none of the inapproximability results in [42] hold for Bipartite Flow Scheduling.

Finally, we would like to mention that the 2-approximation algorithm of Lemma 5.3 can be interpreted in terms of MSSC. Assume that we choose a maxi*mum* matching instead of a maximal matching in every step. In terms of MSSC, this corresponds to choosing the vertex v_M that is contained in a maximum number of edges, or in other words, choosing the vertex of maximum degree. Hence, the algorithm can also be interpreted as Greedy algorithm for MSSC and Lemma 5.3 implies that Greedy is a 2-approximation algorithm for instances of MSSC that arise from Bipartite Flow Scheduling.

Connection to Data Migration

In the Data Migration problem, we are given a, not necessarily bipartite, transfer graph G. The vertices model storage devices, and every edge corresponds to a data transfer between two storage devices. All edges have unit processing time, and we are given non-negative weights for all vertices of the graph. Each vertex can process only one data transfer at a time, which implies that we schedule a matching in each time step. Instead of minimizing the sum of weighted completion time of the edges, we are interested in minimizing the sum of weighted completion times of the vertices.

Gandhi and Mestre [47] consider the variant of Data Migration, where G is bipartite, and the objective is to minimize the sum of completion times of the edges. This variant is precisely Bipartite Flow Scheduling with unit weights and unit processing times. Before explaining their result, we briefly discuss the limitations of the 2-approximation given above.

Example 5.6
Consider the following instance of Bipartite Flow Scheduling. Let $G = (I \cup O, E)$ with $I = \{1, 2, 3, 4\}, O = \{5, 6, 7, 8\}$ and coflows $j_1 = (1, 5), j_2 = (1, 6), j_3 = (1, 7), j_4 = (2, 5), j_5 = (3, 6), j_5 = (4, 7)$. Figure 5.7 visualizes the instance. An optimal schedule is, for example, given by scheduling coflows j_1, j_5 and j_6 during the first time step, coflows j_2 and j_4 during the second time step and j_3 during the third time step (see second schedule in Figure 5.7). Its value is given by $1 \cdot 3 + 2 \cdot 2 + 3 \cdot 1 = 10$. Note that in this example, picking any maximal matching that contains a coflow incident to vertex 1 in the first step, leads to an optimal schedule.

Now consider a solution obtained by the 2-approximation algorithm of Lemma 5.3. Recall, that the algorithm schedules a maximal matching in every step. Unfortunately, it

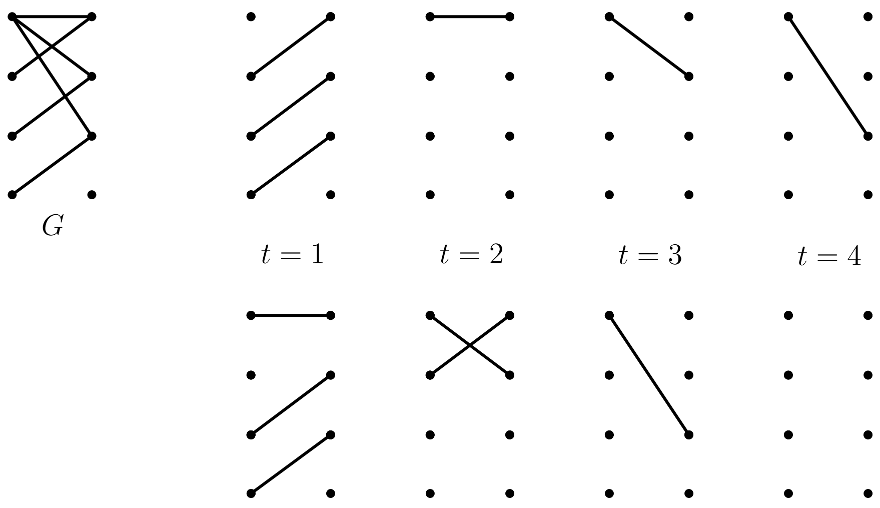


Fig. 5.7: An example to illustrate the limitations of the 2-approximation presented above. The instance is given by the cumulative graph G together with unit weights and unit processing times. The upper schedule displays a solution of value 12, obtained by iteratively scheduling maximal matchings. The schedule below is optimal with value 10.

might choose the maximal matching consisting of coflows j_4, j_5 and j_6 in step one. Since all the remaining coflows j_1, j_2, and j_3 are adjacent to vertex 1 and, hence, in conflict, the algorithm needs at least three more steps to schedule them (see upper schedule in Figure 5.7). This results in a schedule with value $1 \cdot 3 + 2 \cdot 1 + 3 \cdot 1 + 4 \cdot 1 = 12$, which is strictly greater than the optimal value. Note that choosing maximum matchings instead of maximal matchings in Example 5.6 does not lead to an improved schedule.

One way to improve the performance of the simple algorithm presented in Lemma 5.3 is to guarantee that the degree of all vertices of maximum degree is reduced by one in every time step. This can be achieved by starting at the back of the schedule and recursively choosing a matching that covers all vertices of maximum degree in each time step.

Gandhi and Mestre [47] formalized this idea. A subgraph M of G is a **b-matching**, if $\deg_M(v) \leq b$ for all $v \in V(G)$. A decomposition of G into matchings $M_1, M_2, \dots, M_\Delta$ is called **strongly minimal** if for all $b \in [\Delta]$ the b-matching $\bigcup_{i=1}^{b} M_i$ is maximal with respect to G. A schedule is called **strongly minimal schedule**, if the matchings scheduled in every time step form a strongly minimal decomposition of the instance.

One may obtain a strongly minimal schedule in the following way. We start at the back of the schedule at time $t = \Delta$ and schedule a matching that covers all vertices of degree Δ. The matching is then deleted from the graph and we continue with time step $\Delta - 1$, where we schedule a matching that covers all vertices of degree $\Delta - 1$. We continue until all edges have been scheduled.

It was shown in [47] that any strongly minimal schedule is a $\sqrt{2}$-approximation for Bipartite Flow Scheduling with unit weights and unit processing times.

Proposition 5.7 (Gandhi and Mestre [47])
The exists a $\sqrt{2}$-approximation for Bipartite Flow Scheduling with unit weights and unit processing times.

In the following, we explain the weakness of strongly minimal schedules, propose a new algorithm for Bipartite Flow Scheduling and give some intuition on its performance. The matchings scheduled in a strongly minimal schedule might be significantly smaller than a maximum matching. An example, where this is the case, is given in Figure 5.8. The upper schedule is a strongly minimal schedule and the lower schedule is optimal. Note that in the first time step, the matching scheduled in the upper schedule is significantly smaller than the maximum matching scheduled in the optimum solution. An extension of this example was used in [47] to show that strongly minimal schedules are at best a 1.375-approximation.

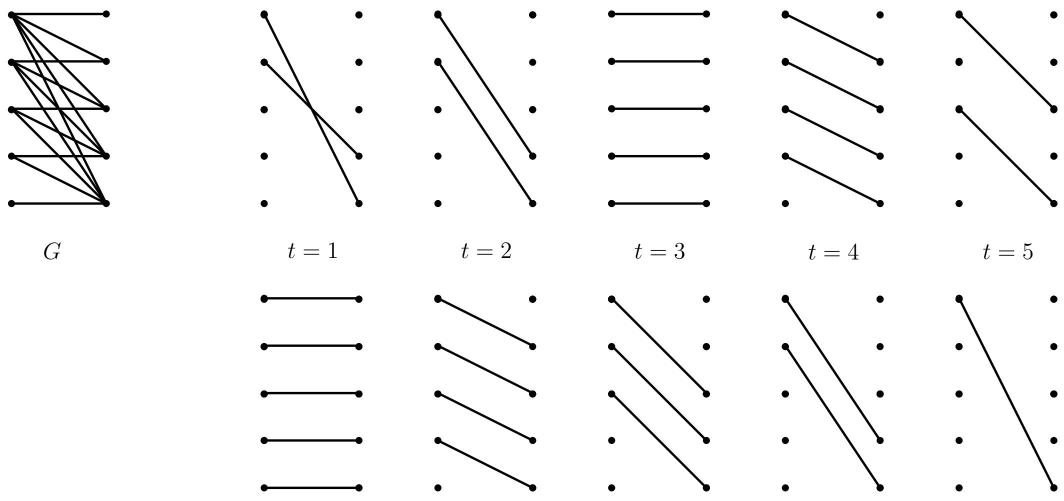


Fig. 5.8: Example in which the matchings of a strongly minimal schedule are significantly smaller than a maximum matching. The second schedule is optimal.

A first approach to solve this issue is the following algorithm that we refer to as maxGreedy: Start at time $t = 1$ and schedule in each time step a maximum matching that covers all vertices of maximum degree. Delete the matching and recurse.

The algorithm maxGreedy prioritizes maximum degree vertices and ensures that we schedule a large matching in every time step. To explain how we can improve maxGreedy further, we consider the example displayed in Figure 5.9. The second schedule is optimal and consists of maximum matchings M_1^*, M_2^* and M_3^* of size

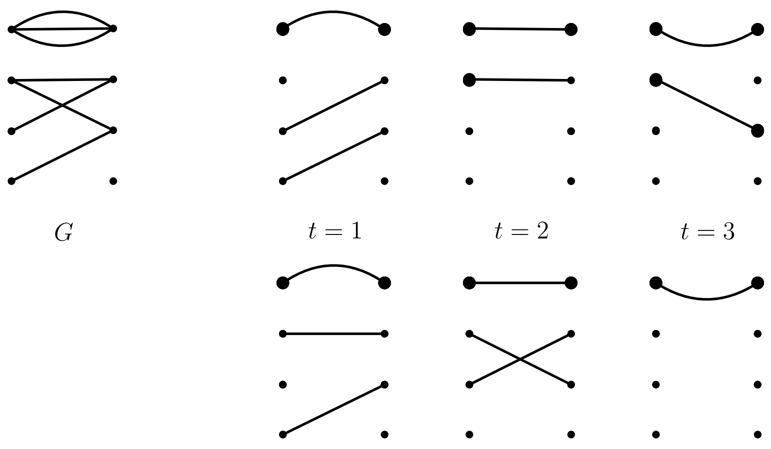


Fig. 5.9: The instance is given by the graph G. The first schedule corresponds to a schedule obtained by greedily choosing maximum matchings that reduce the degree of all maximum degree vertices. The second schedule is optimal.

$|M_1^*| = 3$, $|M_2^*| = 3$ and $|M_3^*| = 1$. The first schedule corresponds to a solution by maxGreedy. The displayed maximum matchings M_1, M_2, and M_3 are of size $M_1 = 3$, $M_2 = 2$ and $M_3 = 2$. In time step $t = 1$, after covering the vertices of maximum degree, maxGreedy has the choice between leaving input port $2, 3$ or 4 uncovered. If input port 2 remains uncovered, we obtain in the next time step a graph whose maximum matching is strictly smaller than in the optimum solution. Overall this leads to a solution whose objective value is strictly larger than the optimal value.

Note that input port 2 is the one with the largest degree among the input ports $2, 3$ and 4. Even though the algorithm prioritizes covering vertices of maximum degree, it does not prioritize large degree vertices in general. Before proposing a new algorithm for Bipartite Flow Scheduling that takes the degrees of all vertices into account, we share another observation.

Intuitively, instead of requiring the algorithm to prioritize vertices of large degree one could also require it to keep the set of maximum degree vertices as small as possible. Consider any algorithm that reduces all vertices of maximum degree in every step. Observe that the set of maximum degree vertices increases throughout the iterations. The simple reason is that a vertex of maximum degree remains a vertex of maximum degree throughout the subsequent time steps. The vertices of maximum degree in Figure 5.9 are highlighted in bold. Observe that the optimal solution keeps the set of maximum degree vertices as small as possible.

We extend the ideas above by proposing a new algorithm for Bipartite Flow Scheduling with unit weights and unit processing times that solves both of these issues. We introduce edge weights $w(e) := \deg(x) + \deg(y) - 2$ for all $e = (x, y) \in E$ and schedule a maximum weight matching M of maximum cardinality. M is then deleted from the graph and the weights are adapted accordingly. The following lemma shows the existence of such a matching M.

Lemma 5.8
Given a bipartite multigraph G with edge weights $w(e) := \deg(x) + \deg(y) - 2$ for all $e = (x, y) \in E$, let M_w be a maximum weight matching and let M_c be a maximum cardinality matching in G. There always exists a matching M in G with $|M| = |M_c|$ and $w(M) = w(M_w)$.

Proof. Consider the set of maximum weight matchings $\mathcal{M}_w$ and choose $M_w \in \mathcal{M}_w$ to be a matching with the largest cardinality. Similarly, let $\mathcal{M}_c$ be the set of matchings of maximum cardinality and choose $M \in \mathcal{M}_c$ to be a matching with maximal weight. Suppose that $w(M) < w(M_w)$. Consider the symmetric difference H of M and M_w. H is a subgraph of G whose vertices have degree at most 2. Hence, H consists of paths and cycles.

Observe that by definition of M, $|M_w| < |M|$, since otherwise we would have chosen M to be M_w. This implies, that H must contain an M_w-augmenting path.

Let $(v_1, v_2, v_3, \ldots, v_n)$ be the vertex sequence of this path and consider the weight difference between the edges in M and the edges in M_w. Then,

$$
\begin{aligned}
&w(M \cap C) - w(M_w \cap C) = \sum_{i=1}^{\frac{n}{2}} w(v_{2i-1}v_{2i}) - \sum_{i=1}^{\frac{n}{2}-1} w(v_{2i}v_{2i+1}) \\
&= \sum_{i=1}^{\frac{n}{2}} \Big(\deg_G(v_{2i-1}) - 1 + \deg_G(v_{2i}) - 1 \Big) - \sum_{i=1}^{\frac{n}{2}-1} \Big(\deg_G(v_{2i}) - 1 + \deg_G(v_{2i+1}) - 1 \Big) \\
&= \deg_G(v_1) - 1 + \deg_G(v_n) - 1 \geq 0,
\end{aligned}
$$

where $\frac{n}{2} \in \mathbb{N}$, since n is even. This implies that we can augment M_w along the path without decreasing $w(M_w)$, which contradicts the choice of M_w. □

We conjecture that greedily scheduling maximum matchings of maximum weight yields at least a $\sqrt{2}$-approximation.

So far, we only considered Bipartite Flow Scheduling with unit processing times. In the following, we consider integer processing times.

Connection to Min Sum Multicoloring

Min Sum Multicoloring is the extension of Min Sum Coloring in which every vertex requires a coloring with multiple colors. More specifically, let $G = (V, E)$ be a graph together with positive integers $k(v)$ for all $v \in V$. Let $c_{\max}(v)$ be the maximum color assigned to a vertex v, then the goal is to minimize the sum of maximum colors, that is $\min \sum_{v \in V} c_{\max}(v)$. This setting is sometimes also referred to as preemptive Min Sum

Multicoloring (pSMC). In the non-preemptive version of the problem, the set of colors assigned to a vertex additionally has to be contiguous.

Bipartite Flow Scheduling is a special case of preemptive Min Sum Multicoloring on line graphs of bipartite multigraphs, where the integers $k(v)$ correspond to processing times. Bar-Noy et al. [14] showed that a Sorted Greedy algorithm is a 2-approximation for pSMC on line graphs. It is based on the property of a line graph, that there exists a partition of the edges into cliques, such that every vertex is contained in at most two of them.

Proposition 5.9 (Bar-Noy et al. [14])
There exists a 2-approximation for Bipartite Flow Scheduling with non-negative weights and integer processing times.

Connection to Generalized Open Shop Scheduling

A variant of the Generalized Open Shop problem, considered in [90], is defined as follows. We are given a set of m machines and n jobs. Every job j consists of operations o_{ij} with processing time p_{ij} on machine i, weight w_{ij} and release date r_{ij}. In every time step, a machine can schedule at most one operation. Additionally, no pair of operations of the same job can be scheduled in parallel. Note that this is a significant difference between Open Shop and Concurrent Open Shop. The completion time $C(o_{ij})$ of an operation o_{ij} is the earliest point in time, when o_{ij} has been scheduled for p_{ij} time units. The objective is to minimize the sum of weighted operation completion times, that is $\min \sum w_{ij} C(o_{ij})$.

This is a special case of Bipartite Flow Scheduling. Given an instance of Generalized Open Shop to minimize the sum of weighted operation completion times, we can construct an instance of Bipartite Flow Scheduling in the following way. For every job $j \in J$, we create an input port v_j and for every machine $m \in M$ we create an output port v_m. For every operation o_{ij} we introduce a corresponding edge $v_j v_m$ with processing time p_{ij}, weight w_{ij} and release date r_{ij}. The conflicts at the input ports assure that no two operations of the same jobs are processed simultaneously on different machines. The conflicts at the output ports make sure that no two operations

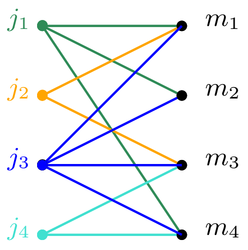


Fig. 5.10: Example to illustrate the connection to Open Shop Scheduling.

are scheduled on the same machine during the same times slot. The construction is visualized in Figure 5.10. Note that this variant of Generalized Open Shop is only a special case of Bipartite Flow Scheduling and that the two problems are not equivalent. In Bipartite Flow Scheduling, the corresponding bipartite graph may have multi-edges. This is not captured by the Generalized Open Shop setting, since a job cannot have multiple operations on the same machine.

5.3 Open Problems

In this chapter, we studied the Bipartite Flow Scheduling problem and explained its connection to related scheduling and coloring problems. We proposed the following algorithm for Bipartite Flow Scheduling with unit weights and unit processing times. Introduce edge weights $w(e) := \deg(x) + \deg(y) - 2$ for all $e = (x, y) \in E$ and schedule a maximum weight matching M of maximum cardinality in each time step, delete M from the graph and adapt the weights accordingly.

Problem 5.10
What is the approximation ratio of the proposed algorithm for Bipartite Flow Scheduling with unit weights and unit processing times?

Bar-Noy et al. [14] gave a 2-approximation for preemptive Min Sum Multicoloring on line graphs. It is built on the property that the edge set of a line graph can be partitioned into cliques, such that every vertex is contained in at most two cliques. In the context of Bipartite Flow Scheduling, we are not interested in general line graphs but in line graphs of bipartite multigraphs. If a graph G is the line graph of a bipartite multigraph, then the clique graph of G is bipartite. This observation might help derive better approximation algorithms for Bipartite Flow Scheduling with unit weights.

Problem 5.11
What is the best possible approximation ratio for Bipartite Flow Scheduling with arbitrary processing times?

To the best of the author's knowledge, the Bipartite Flow Scheduling problem has only been studied with zero release times. Because of its close connection to Generalized Open Shop Scheduling, similar techniques might help to obtain good approximation algorithms.

Problem 5.12
How well can Bipartite Flow Scheduling with arbitrary release dates be approximated? In particular, is there a 2-approximation for Bipartite Flow Scheduling with release dates?

Our primary motivation to study Bipartite Flow Scheduling is its connection to Coflow Scheduling. An inapproximability result from Concurrent Open Shop implies that it is NP-hard to approximate Coflow Scheduling within $2 - \epsilon$. The currently best-known approximation algorithm for Coflow Scheduling with zero release times is 4. Hence, an obvious open problem is the following question.

Problem 5.13
Is there a 2-approximation for Coflow Scheduling?

Many known algorithms for Coflow Scheduling are based on two stages. In the first stage, an ordering of the coflows is obtained. In the second stage, a feasible schedule respecting the ordering is constructed.

Problem 5.14
Given an optimal ordering of coflows, can one find an optimal schedule in polynomial-time?

Index

Bibliography

[1] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys, and A. Vahdat. “Sincronia: Near-optimal network design for coflows”. In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication.* (2018), pp. 16–29.

[2] R. Aharoni and E. Berger. “The intersection of a matroid and a simplicial complex”. In: *Transactions of the American Mathematical Society* 358.11 (2006), pp. 4895–4917.

[3] R. Ahmadi and U. Bagchi. “Scheduling of multi-job customer orders in multi-machine environments”. In: *ORSA/TIMS, Philadelphia* (1990).

[4] S. Ahmadi, S. Khuller, M. Purohit, and S. Yang. “On scheduling coflows”. In: *International Conference on Integer Programming and Combinatorial Optimization.* Vol. 10328. LNCS. Springer. (2017), pp. 13–24.

[5] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, (1993).

[6] E. Angel, E. Bampis, and L. Gourvès. “On the minimum hitting set of bundles problem”. In: *Theoretical Computer Science* 410.45 (2009), pp. 4534–4542.

[7] A. Archer and D. P. Williamson. “Faster approximation algorithms for the minimum latency problem”. In: *Proceedings of the 2003 Annual ACM-SIAM Symposium on Discrete Algorithms.* SIAM. (2003), pp. 88–96.

[8] Y. Azar, I. Gamzu, and X. Yin. “Multiple intents re-ranking”. In: *Proceedings of the 2009 Annual ACM Symposium on Theory of Computing.* (2009), pp. 669–678.

[9] N. Bansal, J. Batra, M. Farhadi, and P. Tetali. “Improved approximations for min sum vertex cover and generalized min sum set cover”. In: *Proceedings of the 2021 Annual ACM-SIAM Symposium on Discrete Algorithms.* SIAM. (2021), pp. 998–1005.

[10] N. Bansal, A. Gupta, and R. Krishnaswamy. “A constant factor approximation algorithm for generalized min-sum set cover”. In: *Proceedings of the 2010 Annual ACM-SIAM Symposium on Discrete Algorithms.* SIAM. (2010), pp. 1539–1545.

[11] N. Bansal and S. Khot. “Optimal long code test with one free bit”. In: *2009 Annual IEEE Symposium on Foundations of Computer Science.* (2009), pp. 453–462.

[12] N. Bansal and S. Khot. “Inapproximability of hypergraph vertex cover and applications to scheduling problems”. In: *Proceedings of the 37th International Colloquium Conference on Automata, Languages and Programming.* ICALP’10. Berlin, Heidelberg: Springer-Verlag, 2010, 250–261.

[13] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir. “On chromatic sums and distributed resource allocation”. In: *Information and Computation* 140.2 (1998), pp. 183–202.

[14] A. Bar-Noy, M. M. Halldórsson, G. Kortsarz, R. Salman, and H. Shachnai. “Sum multicoloring of graphs”. In: *Journal of Algorithms* 37.2 (2000), pp. 422–450.

[15] A. Bar-Noy and G. Kortsarz. “Minimum color sum of bipartite graphs”. In: *Journal of Algorithms* 28.2 (1998), pp. 339–365.

[16] U. Barenholz, U. Feige, and D. Peleg. “Improved approximation for min-sum vertex cover”. In: *Technical Report MCS06-07 Computer Science and Applied Mathematics* (2006).

[17] K. Bérczi and T. Schwarcz. “Complexity of packing common bases in matroids”. In: *Mathematical Programming* 188 (2020), pp. 1–18.

[18] K. Bérczi, T. Schwarcz, and Y. Yamaguchi. “List colouring of two matroids through reduction to partition matroids”. In: *arXiv preprint arXiv:1911.10485* (2019).

[19] C. Berge. “Two theorems in graph theory”. In: *Proceedings of the National Academy of Sciences of the United States of America* 43.9 (1957), p. 842.

[20] D. Bertsimas, C. Teo, and R. Vohra. “On dependent randomized rounding algorithms”. In: *Operations Research Letters* 24.3 (1999), pp. 105–114.

[21] J. A. Carlson, A. Jaffe, and A. Wiles. *The Millennium Prize Problems.* 2006.

[22] J. Chang, H. N. Gabow, and S. Khuller. “A model for minimizing active processor time”. In: *20th Annual European Symposium on Algorithms.* Springer Berlin Heidelberg, (2012), pp. 289–300.

[23] J. Chang, S. Khuller, and K. Mukherjee. “LP rounding and combinatorial algorithms for minimizing active and busy time”. In: *Journal of Scheduling* 20.6 (2017), pp. 657–680.

[24] C. Chekuri and A. Kumar. “Maximum coverage problem with group budget constraints and applications”. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques.* Vol. 3122. LNCS. Springer, (2004), pp. 72–83.

[25] C. Chekuri and R. Motwani. “Precedence constrained scheduling to minimize sum of weighted completion times on a single machine”. In: *Discrete Applied Mathematics* 98.1-2 (1999), pp. 29–38.

[26] Z.-L. Chen and N. G. Hall. “Supply chain scheduling: Conflict and cooperation in assembly systems”. In: *Operations Research* 55.6 (2007), pp. 1072–1089.

[27] M. Chowdhury and I. Stoica. “Coflow: A networking abstraction for cluster applications”. In: *Proceedings of the 11th ACM Workshop on Hot Topics in Networks.* (2012), pp. 31–36.

[28] M. Chowdhury, Y. Zhong, and I. Stoica. “Efficient coflow scheduling with varys”. In: *Proceedings of the 2014 ACM Conference on SIGCOMM.* (2014), pp. 443–454.

[29] F. A. Chudak and D. S. Hochbaum. “A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine”. In: *Operations Research Letters* 25.5 (1999), pp. 199–204.

[30] V. Chvatal. “A greedy heuristic for the set-covering problem”. In: *Mathematics of Operations Research* 4.3 (1979), pp. 233–235.

[31] S. A. Cook. “The complexity of theorem-proving procedures”. In: *Proceedings of the 1971 Annual ACM Symposium on Theory of Computing.* (1971), pp. 151–158.

[32] J. R. Correa and A. S. Schulz. “Single-machine scheduling with precedence constraints”. In: *Mathematics of Operations Research* 30.4 (2005), pp. 1005–1021.

[33] P. Damaschke. “Parameterizations of hitting set of bundles and inverse scope”. In: *Journal of Combinatorial Optimization* 29.4 (2015), pp. 847–858.

[34] I. Dinur and D. Steurer. “Analytical approach to parallel repetition”. In: *Proceedings of the 2014 Annual ACM Symposium on Theory of Computing.* New York, NY, USA: ACM, (2014), pp. 624–633.

[35] P. J. Dukes. “Generalized laminar families and certain forbidden matrices”. In: *Order* 32.3 (2015), pp. 401–408.

[36] A. Eckl, A. Kirschbaum, M. Leichter, and K. Schewior. “A stronger impossibility for fully online matching”. In: *Operations Research Letters* (To appear).

[37] J. Edmonds. “Minimum partition of a matroid into independent subsets”. In: *Journal of Research of the National Bureau of Standards* 69B (1965), 67–72.

[38] J. Edmonds. “Submodular functions, matroids, and certain polyhedra”. In: *Combinatorial Optimization—Eureka, You Shrink!* Springer, (2003), pp. 11–26.

[39] S. Eskandarzadeh, T. Kalinowski, and H. Waterer. “Maintenance scheduling in a railway corricdor”. In: *arXiv preprint arXiv:1910.10348* (2019).

[40] X. Fang, H. Gao, J. Li, and Y. Li. “Application-aware data collection in wireless sensor networks”. In: *2013 Proceedings IEEE International Conference on Computer Communications.* (2013), pp. 1645–1653.

[41] U. Feige, L. Lovász, and P. Tetali. “Approximating min-sum set cover”. In: *International Workshop on Approximation Algorithms for Combinatorial Optimization.* LNCS. Springer. (2002), pp. 94–107.

[42] U. Feige, L. Lovász, and P. Tetali. “Approximating min sum set cover”. In: *Algorithmica* 40.4 (2004), pp. 219–234.

[43] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. “An analysis of approximations for maximizing submodular set functions—II”. In: *Polyhedral Combinatorics.* Springer, (1978), pp. 73–87.

[44] K. C. K. Fong, M. Li, Y. Li, S.-H. Poon, W. Wu, and Y. Zhao. “Scheduling tasks to minimize active time on a processor with unlimited capacity”. In: *Theory and Applications of Models of Computation.* Springer International Publishing, (2017), pp. 247–259.

[45] F. Galvin. “The list chromatic index of a bipartite multigraph”. In: *Journal of Combinatorial Theory, Series B* 63.1 (1995), pp. 153–158.

[46] R. Gandhi, M. M. Halldórsson, G. Kortsarz, and H. Shachnai. “Improved results for data migration and open shop scheduling”. In: *ACM Transactions on Algorithms (TALG)* 2.1 (2006), pp. 116–129.

[47] R. Gandhi and J. Mestre. “Combinatorial algorithms for data migration to minimize average completion time”. In: *Algorithmica* 54.1 (2009), pp. 54–71.

[48] N. Garg, A. Kumar, and V. Pandit. “Order scheduling models: hardness and algorithms”. In: *International Conference on Foundations of Software Technology and Theoretical Computer Science.* Springer. (2007), pp. 96–107.

[49] M. Gottschau and M. Leichter. “Minimum hitting set of interval bundles problem: computational complexity and approximability”. Submitted for publication. (2019).

[50] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan. “Optimization and approximation in deterministic sequencing and scheduling: a survey”. In: *Annals of Discrete Mathematics.* Vol. 5. Elsevier, (1979), pp. 287–326.

[51] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. “Scheduling to minimize average completion time: Off-line and on-line approximation algorithms”. In: *Mathematics of Operations Research* 22.3 (1997), pp. 513–544.

[52] M. M. Halldórsson, G. Kortsarz, and H. Shachnai. “Sum coloring interval and k-claw free graphs with application to scheduling dependent jobs”. In: *Algorithmica* 37.3 (2003), pp. 187–209.

[53] F. Happach, L. Hellerstein, and T. Lidbetter. “A general framework for approximating min sum ordering problems”. In: *arXiv preprint arXiv:2004.05954* (2020).

[54] F. Happach and M. Leichter. “On the generalized min-sum set cover problem with laminar sets”. In preparation. (2020).

[55] F. Happach and A. S. Schulz. “Precedence-constrained scheduling and min-sum set cover”. In: *International Workshop on Approximation and Online Algorithms.* Vol. 11926. LNCS. Springer. (2019), pp. 170–187.

[56] F. Happach and A. S. Schulz. “Approximation algorithms and LP relaxations for scheduling problems related to min-sum set cover”. In: *arXiv preprint arXiv:2001.07011* (2020).

[57] R. Hassin and A. Levin. “An approximation algorithm for the minimum latency set cover problem”. In: *13th Annual European Symposium on Algorithms.* Vol. 3669. LNCS. Springer. (2005), pp. 726–733.

[58] D. S. Hochbaum. “Efficient bounds for the stable set, vertex cover and set packing problems”. In: *Discrete Applied Mathematics* 6.3 (1983), pp. 243–254.

[59] W. Horn. “Single-machine job sequencing with treelike precedence ordering and linear delay penalties”. In: *SIAM Journal on Applied Mathematics* 23.2 (1972), pp. 189–202.

[60] S. Im, B. Moseley, and K. Pruhs. “The matroid intersection cover problem”. In: *Operations Research Letters* 49.1 (2020), pp. 17–22.

[61] S. Im, M. Sviridenko, and R. Van Der Zwaan. “Preemptive and non-preemptive generalized min sum set cover”. In: *Mathematical Programming* 145.1-2 (2014), pp. 377–401.

[62] S. Iwata and K. Nagano. “Submodular function minimization under covering constraints”. In: *2009 Annual IEEE Symposium on Foundations of Computer Science.* (2009), pp. 671–680.

[63] Y. Jin, J.-P. Hamiez, and J.-K. Hao. “Algorithms for the minimum sum coloring problem: a review”. In: *Artificial Intelligence Review* 47.3 (2017), pp. 367–394.

[64] B. Johannes. “On the complexity of scheduling unit-time jobs with OR-precedence constraints”. In: *Operations Research Letters* 33.6 (2005), pp. 587–596.

[65] D. S. Johnson. “Approximation algorithms for combinatorial problems”. In: *Proceedings of the 1973 Annual ACM Symposium on Theory of Computing.* (1973), pp. 38–49.

[66] R. M. Karp. “Reducibility among combinatorial problems”. In: *Complexity of Computer Computations.* Springer, (1972), pp. 85–103.

[67] R. Khandekar, B. Schieber, H. Shachnai, and T. Tamir. “Minimizing busy time in multiple machine real-time scheduling”. In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science.* Vol. 8. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. (2010).

[68] S. Khot. “On the power of unique 2-prover 1-round games”. In: *Proceedings of the 2002 Annual ACM Symposium on Theory of Computing.* (2002), pp. 767–775.

[69] S. Khuller and M. Purohit. “Brief announcement: Improved approximation algorithms for scheduling co-flows”. In: *Proceedings of the 2016 ACM Symposium on Parallelism in Algorithms and Architectures.* (2016), pp. 239–240.

[70] J. M. Kleinberg. “Single-source unsplittable flow”. In: *Proceedings of 1996 Conference on Foundations of Computer Science.* (1996), pp. 68–77.

[71] F. Koehler and S. Khuller. “Busy time scheduling on a bounded number of machines”. In: *Workshop on Algorithms and Data Structures.* Vol. 10389. LNCS. Springer International Publishing, (2017), pp. 521–532.

[72] D. König. “Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre”. In: *Mathematische Annalen* 77.4 (1916), pp. 453–465.

[73] E. Kubicka and A. J. Schwenk. “An introduction to chromatic sums”. In: *Proceedings of the 1989 Conference on ACM Annual Computer Science Conference.* (1989), pp. 39–45.

[74] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Courier Corporation, (2001).

[75] J. Lee, M. Sviridenko, and J. Vondrák. “Submodular maximization over multiple matroids via generalized exchange properties”. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques.* Vol. 5687. LNCS. Springer, (2009), pp. 244–257.

[76] M. Leichter, B. Moseley, and K. Pruhs. “An efficient reduction of a gammoid to a partition matroid”. In: *29th Annual European Symposium on Algorithms.* Vol. 204. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, (2021), 62:1–62:13.

[77] J. Y.-T. Leung, H. Li, and M. Pinedo. “Scheduling orders for multiple product types to minimize total weighted completion time”. In: *Discrete Applied Mathematics* 155.8 (2007), pp. 945–970.

[78] A. Linhares, N. Olver, C. Swamy, and R. Zenklusen. “Approximate multimatroid intersection via iterative refinement”. In: *International Conference on Integer Programming and Combinatorial Optimization.* Springer. 2019, pp. 299–312.

[79] L. Lovász. “On the ratio of optimal integral and fractional covers”. In: *Discrete Mathematics* 13.4 (1975), pp. 383–390.

[80] A. Mallek, M. Bendraouche, and M. Boudhar. “Scheduling identical jobs on uniform machines with a conflict graph”. In: *Computers & Operations Research* 111 (2019), pp. 357–366.

[81] F. Margot, M. Queyranne, and Y. Wang. “Decompositions, network flows, and a precedence constrained single-machine scheduling problem”. In: *Operations Research* 51.6 (2003), pp. 981–992.

[82] D. Marx. “Complexity results for minimum sum edge coloring”. In: *Discrete Applied Mathematics* 157.5 (2009), pp. 1034–1045.

[83] M. Mastrolilli, M. Queyranne, A. S. Schulz, O. Svensson, and N. A. Uhan. “Minimizing the sum of weighted completion times in a concurrent open shop”. In: *Operations Research Letters* 38.5 (2010), pp. 390–395.

[84] N. H. Mustafa and S. Ray. “PTAS for geometric hitting set problems via local search”. In: *Proceedings of 2009 Annual Symposium on Computational Geometry.* (2009), pp. 17–22.

[85] P. Obszarski and A. Jastrzębski. “Edge-coloring of 3-uniform hypergraphs”. In: *Discrete Applied Mathematics* 217 (2017), pp. 48–52.

[86] J. Oxley. *Matroid Theory.* Oxford Graduate Texts in Mathematics. Oxford University Press, (2006).

[87] J. Petersen et al. “Die Theorie der regulären Graphen”. In: *Acta Mathematica* 15 (1891), pp. 193–220.

[88] M. Pinedo. *Scheduling.* Springer, (2012).

[89] Z. Qiu, C. Stein, and Y. Zhong. “Minimizing the total weighted completion time of coflows in datacenter networks”. In: *Proceedings of the 2015 ACM Symposium on Parallelism in Algorithms and Architectures.* (2015), pp. 294–303.

[90] M. Queyranne and M. Sviridenko. “A (2+ε)-approximation algorithm for the generalized preemptive open shop problem with minsum objective”. In: *Journal of Algorithms* 45.2 (2002), pp. 202–212.

[91] S. Sachdeva and R. Saket. “Optimal inapproximability for scheduling problems via structural hardness for hypergraph vertex cover”. In: *2013 IEEE Conference on Computational Complexity.* IEEE. (2013), pp. 219–229.

[92] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Vol. 24. Springer Science & Business Media, (2003).

[93] A. S. Schulz. “Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds”. In: *International conference on Integer Programming and Combinatorial Optimization.* Springer. (1996), pp. 301–315.

[94] T. K. Sellis. “Multiple-query optimization”. In: *ACM Transactions on Database Systems* 13.1 (1988), pp. 23–52.

[95] P. D. Seymour. “A note on list arboricity”. In: *Journal of Combinatorial Theory, Series B* 72.1 (1998), pp. 150–151.

[96] M. Shafiee and J. Ghaderi. “Scheduling coflows in datacenter networks: Improved bound for total weighted completion time”. In: *ACM SIGMETRICS Performance Evaluation Review* 45.1 (2017), pp. 29–30.

[97] J. B. Sidney. “Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs”. In: *Operations Research* 23.2 (1975), pp. 283–298.

[98] M. Skutella. “List scheduling in order of α-points on a single machine”. In: *Efficient Approximation and Online Algorithms.* Springer, (2006), pp. 250–291.

[99] M. Skutella and D. P. Williamson. “A note on the generalized min-sum set cover problem”. In: *Operations Research Letters* 39.6 (2011), pp. 433–436.

[100] W. E. Smith. “Various optimizers for single-stage production”. In: *Naval Research Logistics Quarterly* 3.1-2 (1956), pp. 59–66.

[101] V. Vazirani. *Approximation Algorithms.* Springer-Verlag, (2001).

[102] E. Wagneur and C. Sriskandarajah. “Openshops with jobs overlap”. In: *European Journal of Operational Research* 71.3 (1993), pp. 366–378.

[103] P.-J. Wan, D.-Z. Du, P. Pardalos, and W. Wu. “Greedy approximations for minimum submodular cover with submodular cost”. In: *Computational Optimization and Applications* 45.2 (2010), pp. 463–474.

[104] G. Wang and T. E. Cheng. “Customer order scheduling to minimize total weighted completion time”. In: *Omega* 35.5 (2007), pp. 623–626.

[105] D. J. Welsh. *Matroid Theory.* Courier Corporation, 2010.

[106] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms.* Cambridge University Press, (2011).

[107] G. J. Woeginger. “On the approximability of average completion time scheduling under precedence constraints”. In: *Discrete Applied Mathematics* 131.1 (2003), pp. 237–252.

[108] S. Zhou. “Minimum partition of an independence system into independent sets”. In: *Discrete Optimization* 6.1 (2009), pp. 125–133.