TUM School of Computation, Information and Technology
Technische Universität München

Three perspectives on integer programming: practical and theoretical applications, and the case of bounded subdeterminants

Stefan A. Kober

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz: Prof. Dr. Christina Kuttler

Prüfer*innen der Dissertation:

1. Prof. Dr. Stefan Weltge
2. Prof. Dr. Christopher Hojny

Die Dissertation wurde am 26.04.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 19.06.2023 angenommen.

Abstract

The main body of the thesis consists of three chapters, each considering a different viewpoint on integer programming.

Firstly, we consider a practical perspective in form of a real-world problem, formulated as an integer program. To be precise, we examine how to translate a problem with many impacting factors into a realistic combinatorial optimization problem. We investigate different formulations of the problem as an integer program and use a state of the art integer programming solver in order to find the best formulation and solve realistic large-scale instances efficiently. In this context, we tackle the problem of placing public charging infrastructure for electric vehicles in cities. Our approach incorporates individual demand and temporal interactions of drivers, exact positioning of charging stations, as well as various charging speeds, and realistic charging curves. We show that the resulting integer programming framework can efficiently compute optimal placements of charging stations for instances based on traffic data for cities with up to 600 000 inhabitants and future electrification rates of up to 15%.

Secondly, we consider a more theoretic application of integer programming, by giving a proof on a mathematical problem that has been found using integer programming, but can be written in a way that is easily readable and verifiable by humans. The specific problem that we investigate in this work is to determine the dimension of the voting system of the EU council (based on the 2014 population data), i.e., the minimum number of weighted games, whose intersection represents this simple game. Kurz and Napel (2015) proved that the dimension is at least 7. This set a new record for real-world voting rules and the authors posed the exact determination as a challenge. We improve this lower bound and show that the dimension is at least 8.

Finally, we consider a theoretic question regarding the efficient solvability of certain subclasses of integer programs. It is a well-known question, whether integer programs on a totally Δ-modular constraint matrix are solvable in polynomial time, i.e., the absolute value of each subdeterminant of the (integer) constraint matrix is bounded by a constant. We restrict ourselves to the case of constraint matrices that become totally unimodular after the removal of one specific row. We present partial progress on this question by giving an algorithm both for the optimization and recognition of instances where the totally unimodular matrix in addition is a transposed incidence matrix or a transposed network matrix that induces a planar, 3-connected graph. Our results rely on a strengthening of the proximity result by Cook et al. for our specific problem, Seymour's decomposition of totally unimodular matrices, and structural graph theory and graph minor results.

Zusammenfassung

Diese Arbeit besteht aus drei Kapiteln, die Ganzzahlige Programme jeweils aus einem unterschiedlichen Blickwinkel betrachten.

Zunächst nehmen wir eine angewandte Perspektive ein. Dafür formulieren wir ein reales Problem als Ganzzahliges Programm. Genauer gesagt untersuchen wir, wie ein reales Problem in ein kombinatorisches Optimierungsproblem übersetzt werden kann. Wir formulieren das Problem in verschiedenen Varianten als Ganzzahliges Programm und verwenden einen Löser, um die beste Formulierung zu finden und realistische Instanzen effizient zu lösen. Konkret beschäftigen wir uns mit dem Problem, öffentliche Ladeinfrastruktur für Elektrofahrzeuge in Städten zu platzieren. Unser Ansatz berücksichtigt individuelle Nachfrage und Interaktionen zwischen Fahrern, exakte Positionierung der Ladestationen, sowie verschiedene Ladeleistungen und realistische Ladekurven. Wir zeigen, dass unser Modell effizient optimale Platzierungen von Ladestationen für Instanzen mit Verkehrsdaten von Städten mit bis zu 600.000 Einwohnern und Elektrifizierungsraten von bis zu 15% berechnen kann.

Zweitens betrachten wir eine theoretische Anwendung von Ganzzahligen Programmen, indem wir einen mathematischen Beweis für eine Frage liefern, der mithilfe eines solchen Programms gefunden wurde, aber in einer leicht lesbaren und überprüfbaren Weise formuliert werden kann. Das spezifische Problem, das wir hier untersuchen, ist die Bestimmung der Dimension des Abstimmungssystems des Europäischen Rats (basierend auf Bevölkerungsdaten von 2014), also die minimale Anzahl von Weighted Games, deren Schnittmenge diesem Simple Game entspricht. Kurz und Napel (2015) haben bewiesen, dass die Dimension mindestens 7 beträgt (ein neuer Rekord für Abstimmungssysteme). Die Autoren stellten die Bestimmung des genauen Werts als Herausforderung. Wir verbessern diese untere Schranke und zeigen, dass die Dimension mindestens 8 beträgt.

Schließlich betrachten wir eine theoretische Fragestellung bezüglich der effizienten Lösbarkeit bestimmter Unterklassen von Ganzzahligen Programmen. Es ist eine bekannte Fragestellung, ob Ganzzahlige Programme mit einer total Δ-modularen Bedingungsmatrix in polynomieller Zeit lösbar sind, also Bedingungsmatrizen, bei denen der Betrag jeder Subdeterminante durch eine Konstante beschränkt ist. Wir beschränken uns auf den Fall von Bedingungsmatrizen, die durch Entfernen einer der Zeilen total unimodular werden. Wir präsentieren partiellen Fortschritt bezüglich dieser Fragestellung, indem wir einen Algorithmus sowohl für die Optimierung als auch für die Erkennung von Instanzen geben, bei denen die total unimodulare Matrix zusätzlich eine transponierte Inzidenzmatrix oder eine transponierte Netzwerkmatrix ist, die einen planaren, 3-zusammenhängenden Graphen induziert. Unsere Ergebnisse beruhen auf einer Verstärkung der Näherung von Cook et al. für das Problem, Seymour's Zerlegung total unimodularer Matrizen, sowie Ergebnissen der Graphentheorie und der Graphenminorentheorie.

Acknowledgements

Let me start by thanking my supervisor, Stefan Weltge. I'm very grateful for him introducing me to the academic world, a world I've grown to like a lot during the past years. He always has an open door for any kind of question, be it of academic, professional, or personal nature. Every time I called him or came in order to ask or discuss, he was happy to listen, understand, and share his insights and opinion. I admire his patience in all kinds of situations and felt continuously supported in my ideas, both to discuss them with him, or pursue them on my own, or with other collaborators. I'm very thankful for all the support, and for looking past my grades and believing in me.

I would also like to thank my (second) supervisor, Maximilian Schiffer. He always made time for a meeting, and was happy to join in on any discussion.

In my last year, Samuel Fiorini kindly hosted me in Brussels for 3 months of a research visit. I want to wholeheartedly thank him for this experience, which I feel gave me a new perspective both on research as well as the field of optimization and integer programming. I was able to learn an incredible amount of new things in my time there, also thanks to other members of the ALGO group, and always felt supported. Thank you to the whole ALGO group for being very welcoming and in particular to Eileen, Lena, and Manuel for making Brussels feel like a second home.

Further, I would like to thank Christopher Hojny for agreeing to be part of my examination committee and Christina Kuttler for being my mentor and agreeing to chair my defense. It was good to know that there was always someone to support me.

Coming to work gave me the opportunity to meet many wonderful people. As part of the research training group AdONE and the chair *M9*, this list of people is a little too long to fit the scope of this short section. I am particularly thankful to Ina for sharing my office and being a friend and always up for a chat, to Angelika, Isabel, Michael, and René for your patience and help, to Jamico for solving more riddles than I could bring, to Donghao for many crazy ideas and fun hours, to Alex and Florian for stopping by and many nice chats, to all of M9 for many amazing cake breaks, especially Anja, Ina, Jamico, and Mia brought a virtually uncountable amount, and to everyone else who shared my time here, and made coming to work more fun.

I was very lucky to be able to collaborate and discuss with a diverse set of researchers. Thanks to Donghao, Holger, Johannes, Lena, Manuel, Martin, Max, Sam, Stefan, and Stephan for many fun days of working together, and for all I could learn from them.

Finally, I want to thank my family, and friends, many of which shared my paths at TU Munich from the beginning of the Bachelor. Without them, this thesis would not have been possible. Let me just name Igor, Kathi, Lea, and Vjosa here. In addition, there are a few very amazing people who I am particularly indebted to, namely Lena, Martin, and Steffi. I hope you know what you mean to me.

Contents

1 Introduction

A linear program is a mathematical optimization problem that arises from a linear objective function that is to be optimized over a feasible region described by linear inequalities. One common way to denote linear programs is given by

$$\max c^\intercal x \text{ s.t. } Ax \leq b,\ x \in \mathbb{R}^n \ ,$$

where $A \in \mathbb{R}^{m,n}$, $c \in \mathbb{R}^n$, and $b \in \mathbb{R}^m$. Such systems of linear inequalities were first considered by Fourier in 1827, who gave name to the Fourier-Motzkin elimination method to solve systems of linear inequalities. In parallel to many political, social, and economical developments within the 20th century that are too diverse to appropriately shed light upon here, the interest in optimization problems increased a lot starting from the 1940s. More data became available, systems started to interact, inherently increasing their complexity. Thus, simple heuristics weren't sufficient anymore and the need to find (provably) good solutions became significant. For a more extensive description of the history of linear programming, see [94].

As part of Smale's problems for the 21st century [95], the arguably biggest question concerning linear programs is whether they admit a strongly polynomial-time solution algorithm. Strongly polynomial-time here means that the runtime can only depend on the size of the constraint matrix A, i.e., the number of variables and constraints, but not on the encoding length of the entries. Within the last 80 years, many algorithmic frameworks to solve linear programs have been developed. We restrict our attention to a few ones that are the most important within the context of this thesis. The first and still one of the most popular algorithms is the simplex method, developed by Dantzig [35]. Despite its good performance in practice, it is open whether there exists a polynomial-time variant of the algorithm. The ellipsoid method due to Khachiyan [66] was the first algorithm for which a polynomial runtime could be shown. In practice, other methods are more efficient. Notably, we mention interior point methods, which were pioneered by Karmarkar [62]. Further, Tardos [96] published a proximity based algorithmic framework, whose running time only depends on the entries of A, and therefore in particular not on the entries of the vectors b and c. As such, it gives a strongly polynomial-time algorithm for relaxations of totally unimodular and totally Δ-modular integer programs. More recently, the guarantees on the framework have been improved and generalized to real-valued matrices by Dadush, Natura, and Végh [33].

While linear programs possess a considerable amount of modelling power and have been helpful tools to solve various mathematical problems, they are naturally limited by the linearity of objective functions, constraints, and in particular, variable values. This stands in contrast to the fact that many decisions in relevant problems are of a discrete nature. Variables often denote the assignment of inseparable goods, an integer count, or simply a Yes-No-decision. Each of these properties is inherently discrete, and can therefore not necessarily be trivially modelled within a linear programming framework. In order to deal with such instances, we consider integer programs, which can be described by a system of the form

$$\max c^\intercal x \text{ s.t. } Ax \leq b,\ x \in \mathbb{Z}^n \ ,$$

where $A \in \mathbb{R}^{m,n}$, $c \in \mathbb{R}^n$, and $b \in \mathbb{R}^m$. General exact algorithms employed by state of the art solvers often rely on cutting plane algorithms and branching rules, both of which heavily rely on linear programming and the respective solution algorithms mentioned above.

Integer programming has an abundance of applications. In particular, this includes many combinatorial optimization problems that can be naturally modeled within this framework. A range of these applications will appear later within the thesis. To give a very limited overview, we mention classical problems, such as graph coloring problems, the travelling salesperson problem, the facility location problem, scheduling problems, or matching problems. Many of these can also appear in combined or adapted versions that permit a more direct modelling of real-life scenarios and questions. Through modern data, technologies, and networks, many of these problems become of new practical relevance or it simply is necessary to solve them for much larger instance sizes. We present one such example in Chapter 2. In part due to their large modelling power, it can be shown that general integer programs are hard to solve. In fact, they represent one of Karp's original 21 NP-hard problems [63]. Therefore, the search for efficient algorithms both in a practical and theoretical sense has received a lot of attention.

We mention several polynomial-time algorithms for interesting special cases of integer programming. In particular, Papadimitriou [84] gave a pseudo-polynomial algorithm for integer programming with a fixed number of constraints. This result was improved by Eisenbrand and Weismantel [41] to a running time of $(m \cdot \|A\|_\infty)^{O(m)}$. Another stream of literature has been started by a breakthrough result of Lenstra [75], giving an algorithm to solve integer programs in time $2^{O(n^2)}$. This result has been subsequently improved by Kannan [61] to a running time of $n^{O(n)}$, and Dadush [34] to $2^{O(n)}n^n$. Very recently, Reis and Rothvoss [86] gave a significant improvement by reducing the running time to $(\log(n))^{O(n)}$. Further, integer programming with bounded subdeterminants has recently gained an increasing amount of attention. We describe this concept in detail in Chapter 4. Let us mention an efficient algorithm if the matrix has integer entries and all subdeterminants are bounded by 2 in absolute value by Artmann, Weismantel, and Zenklusen [8], as well as an efficient algorithm if the matrix has integer entries, all subdeterminants are bounded by a constant in absolute value, and there are at most two non-zero entries per row or column, by Fiorini, Joret, Weltge, and Yuditsky [44].

In this thesis, we look at the framework of integer programming from a few different perspectives. In Chapter 2, we consider the problem of placing charging stations for electric vehicles in cities. In its core, this problem can be seen as a facility location problem together with some problem-specific constraints. The first obstacle in this practical application of integer programming is the problem formulation. How can real-world data, conditions, and questions be cleanly translated into a mathematical optimization problem? How can this be done in an efficient way, i.e., by not just enumerating possibilities, that allows to solve large real-world instances? Formulating the problem as a combinatorial optimization problem is therefore our first contribution in this chapter. When modelling such a combinatorial optimization problem as an integer program, there is some further freedom. We can model certain relationships in different ways, for instance by adding auxiliary variables. In addition, many state of the art solvers use cutting plane subroutines in order to solve integer programs. In practice, there is a trade-off between generating a clean and small description of the problem, and trying to aid the solver by generating problem-specific cuts on the fly, or even add them to the model from the start. We investigate several different formulations of our given problem and analyze the advantage of certain reformulations when implemented, using a state of the art integer programming solver.

We present a more theoretical application of integer programming in Chapter 3. The main question here is, how to use optimization frameworks and in particular integer programs in order to prove mathematical theorems. In 2015, Kurz and Napel [74] proved that the *dimension* of the council of the European Union, when interpreted as a *simple game* is at most 7, and posed the exact determination of that value as a challenge to the community. Roughly, the dimension of a simple game can be understood as the number of linear inequalities that are needed in order to describe the set of winning coalitions within a voting system. The question can for instance be understood as a coloring problem on a certain hypergraph. Such problems can be modelled easily within the frameworks of integer programming, or boolean satisfiability with the help of a SAT solver. Within this work, we design specific optimization problems in order to understand combinatorial and geometric structures. In addition, we transform the solution of an integer program that has been calculated by a computer into a clean mathematical proof that can be checked by humans, using linear programming duality.

Finally, in Chapter 4 we consider a specific subclass of totally Δ-modular integer programs, for fixed $\Delta \in \mathbb{N}$. This chapter focusses on theoretical aspects of integer programming. We try to extend the class of integer programs that can be solved efficiently by investigating a natural, well-structured candidate. We use results from linear algebra, in particular geometry, matroid, and graph minor theory in order to gain structural insights and design algorithms for an interesting subclass.

In the following, we proceed to give more details on the separate perspectives and questions, and present our main results from each chapter.

1.1 A practical real-world problem formulated as an integer program

As mentioned above, in the first part of the thesis, we consider a placement problem for charging infrastructure for electric vehicles in cities. This planning problem is addressed from a planers' perspective that cares about the wellbeing of the drivers within a system.

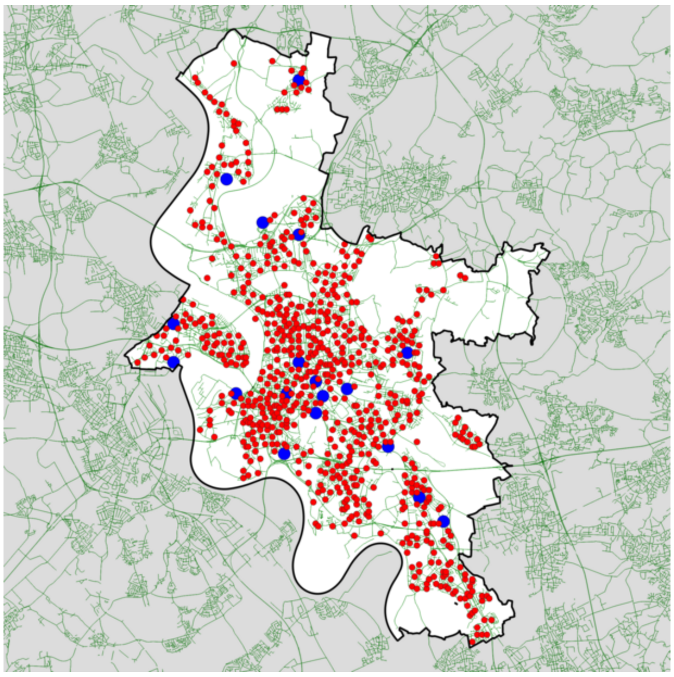


Figure 1.1: Optimized charging station positions for the city of Düsseldorf, Germany. Red locations correspond to slow charging stations, while blue locations denote fast charging stations.

By optimizing the number of satisfied drivers, also the energy throughput of the system is maximized.

Result. We develop an integer programming based approach to efficiently solve the charging station placement problem for cities with 600 000 inhabitants and an electrification rate of up to 15%. Our framework supports individual driver patterns, and temporal interactions, as well as individual charging modes and curves, a flexible choice of objectives, and exact positioning of the charging stations into account.

Our contribution within this project is three-fold. First, we give a combinatorial optimization problem, capturing realistic details of the problem, that make it more challenging than the usual facility location problem. This includes specific driver and charging characteristic on the one hand, like individual demand, different charging speeds, and realistic charging curves, as well as a capacity bound on the charging stations, which leads to interactions between the drivers when blocking charging ports on the other hand.

We proceed to give an integer programming model for the described optimization problem. Further, we give two different kinds of cuts that strengthen the relaxation of the model. A third enhancement is given by relaxing the integrality condition on a type of variable, whose effect on the solution appears to be restricted. We investigate and report on the effect of these three modifications, both on the runtime as well as on the gap between an optimal solution of the model compared to an optimal solution of the linear relaxation.

Finally, we are able to solve realistically sized real world instances, see Figure 1.1. We share the implementation of our optimization method, as well as a simulation framework for the evaluation of positions, in order to make it possible to replicate our results and apply them to further environments and settings. The code is available in our Github repository[1].

1.2 An integer programming powered proof

Article 16 of the treaty of Lisbon on the European Union states that in the voting system of the EU council, a coalition is winning if

1. it contains at least 55% of all members states *and*
2. it unites at least 65% of the total EU population,

or

3. it consists of at least 25 of the 28 member states.

In 2014, the EU consisted of 28 members (which is our reference for the composition of the EU, as well as the population data, in order to ensure comparability with the original publication by Kurz and Napel [74]). Therefore, the voting system can completely be determined based on the population data of these 28 member states, see Table 1.1.

The question for the dimension of the simple game defined by this voting system can be understood in the following way. Each country's vote corresponds to one variable of a 28-dimensional space. We associate 1 with a country, if they decide to vote in favor of a certain bill and 0 otherwise. Thus, the set of winning coalitions can be described as a (monotonic) subset of the 28-dimensional $\{0, 1\}$-cube. The dimension of a simple game is the minimum number of inequalities needed in order to separate winning from losing coalitions in this space. We were able to improve on the lower bound of 7 on the dimension, given by Kurz and Napel [74].

Result. The dimension of the simple game corresponding to the EU council under the treaty of Lisbon is at least 8.

It turns out that while the problem can be concisely represented using Table 1.1, the calculation of this number comes with computational challenges. This may in part be due to the fact, that the standard formulation of such a problem asks whether there *exists* a certain size set of linear inequalities, separating *all* winning points from the losing ones. This type of formulation with two quantors is described by Woeginger [104] in more detail and links the problem to Σ_2^p-hardness. To our knowledge, no corresponding hardness reduction for the determination of the dimension of a simple game is known,

[1] `https://github.com/tumBAIS/driverAwareChargingInfrastructureDesign`

#	Member state	Population	Percentage	#	Member state	Population	Percentage
1	Germany	80 780 000	15.9%	15	Austria	8 507 786	1.7%
2	France	65 856 609	13.0%	16	Bulgaria	7 245 677	1.4%
3	United Kingdom	64 308 261	12.7%	17	Denmark	5 627 235	1.1%
4	Italy	60 782 668	12.0%	18	Finland	5 451 270	1.1%
5	Spain	46 507 760	9.2%	19	Slovakia	5 415 949	1.1%
6	Poland	38 495 659	7.6%	20	Ireland	4 604 029	0.9%
7	Romania	19 942 642	3.9%	21	Croatia	4 246 700	0.8%
8	Netherlands	16 829 289	3.3%	22	Lithuania	2 943 472	0.6%
9	Belgium	11 203 992	2.2%	23	Slovenia	2 061 085	0.4%
10	Greece	10 992 589	2.2%	24	Latvia	2 001 468	0.4%
11	Czech Republic	10 512 419	2.1%	25	Estonia	1 315 819	0.3%
12	Portugal	10 427 301	2.1%	26	Cyprus	858 000	0.2%
13	Hungary	9 879 000	1.9%	27	Luxembourg	549 680	0.1%
14	Sweden	9 644 864	1.9%	28	Malta	425 384	0.1%

Table 1.1: Population data of the European Union on 01.01.2014, see also [74, Table 1].

and this relation is based purely on the formulation of the problem, but it may be an indication as to why the calculation of the exact number seems to be challenging. For more details on general complexity and hardness reductions, we refer to the book by Papadimitriou [83].

We remark that the question for the dimension of a simple game can also be understood in terms of the *relaxation complexity* of the related polytope. The relaxation complexity (rc) of a set of (convex) integer points $X \subseteq \mathbb{Z}^d$ with respect to a set of integer points $Y \subseteq \mathbb{Z}^d$ is defined as the minimum number of facets of a poyhedron P, such that $X \subseteq P$ and $(Y \setminus X) \cap P = \emptyset$, see Averkov, Hojny, and Schymura [11], originally introduced by Kaibel and Weltge [60]. Asking for the dimension of the simple game described above, with $\mathcal{W}$ denoting the points associated with winning coalitions amounts to asking for $\mathrm{rc}(\mathcal{W}, \{0,1\}^{28})$. Questions around relaxation complexity have gained recent attention, see Weltge [103], Averkov, Hojny, and Schymura [10, 11], as well as concerning the role of rationality, Aprile, Averkov, Di Summa, and Hojny [5].

1.3 Algorithms for a well-structured subclass of integer programs

In the final chapter, we consider integer programs with a certain condition on the subdeterminants of the constraint matrix, i.e., we assume that the constraint matrix has integer entries, all subdeterminants are bounded in absolute value by a constant Δ, and there exists one specific row that is present in all submatrices with absolute value of their determinant greater than 1. To be more precise, this means that the constraint matrix is totally Δ-modular, and even totally unimodular after the removal of one row. This additional row can be interpreted as a weight vector on the columns of the totally unimodular matrix.

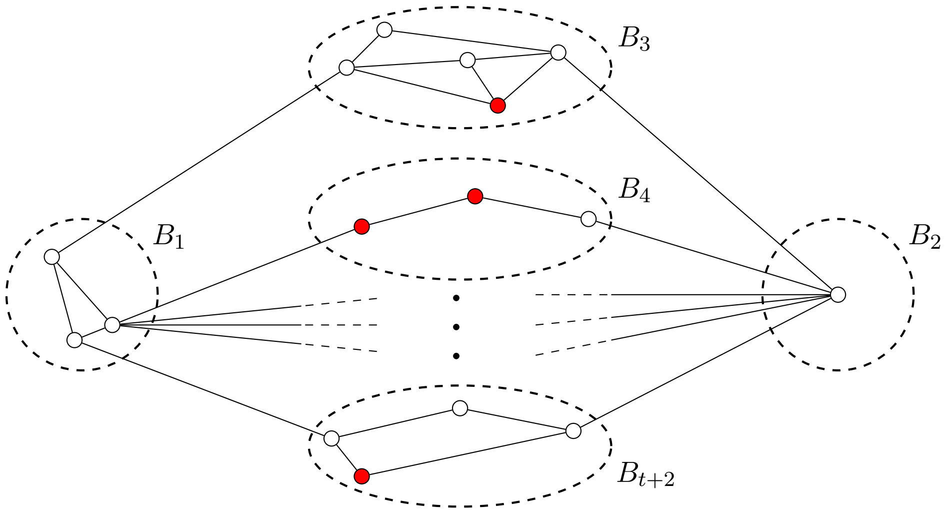


Figure 1.2: Sketch of a t-pumpkin in a graph G. The dashed circles correspond to connected subgraphs that can be contracted in order to obtain a $K_{2,t}$-minor. The red vertices denote labeled vertices.

We use a combination of proximity, based on the results of Cook, Gerards, Schrijver and Tardos [31], and structural graph theory, in particular a result by Böhme and Mohar [19] in order to give efficient algorithms for some subclasses of the problem described above.

Result. For every $\Delta > 0$ there exists a strongly polynomial-time algorithm for solving integer programs of the form

$$\begin{array}{rl} \max & c^{\intercal}x \\ \text{s.t.} & Mx \leq b \\ & d^{\intercal}x \leq W \\ & x \in \mathbb{Z}^n \end{array},$$

where the constraint matrix is totally Δ-modular, and M is a transposed incidence matrix, or M is a transposed network matrix, and induces a planar, 3-connected instance.

Seymour's decomposition of totally unimodular matrices [92] motivates the study of the particular problem for M (or $M^{\intercal}$) being a network matrix. Network matrices let us associate graphs with our problem. A weight on the graph is given by the additional row. In the case of M being a transposed network matrix, total Δ-modularity limits the weight of a connected subgraph, such that its complement is connected too. This characterization leads to a forbidden minor result in the associated graph. We can show that the size of a *pumpkin* is bounded in terms of Δ. A pumpkin is a $K_{2,t}$-minor with a non-zero weight (labeled) vertex in each of the t connected sets (the central sets in Figure 1.2). Pumpkins have found previous attention in research, and a forbidden minor result has further implications on the structure of graphs, in particular in the planar and bounded genus case, see Böhme and Mohar [19] as well as Böhme, Kawarabayashi, Maharry, and Mohar [20].

1.4 Preliminaries

We assume the reader to be familiar with the standard notations and basic results of combinatorial optimization, in particular linear and integer programming, as well as matroid and graph theory. For additional background, we reference the book by Korte and Vygen on combinatorial optimization [71], the book by Schrijver on linear and integer programming [90], the books by Truemper [100] and Oxley [81] on matroids, as well as the book by Diestel [38] on graph theory.

In addition, we use the notation $[k] := \{1, \dots, k\}$ for counting from 1 to an integer number k, and similarly for integer numbers $i < j$, we write $[i, j] := \{i, i+1, \dots, j\}$. For graphs, we denote the complete graphs on t vertices by K_t, and the complete bipartite graph on $i + j$ vertices, with a bipartition into i and j vertices by $K_{i,j}$.

2 Driver-aware charging infrastructure design

The content of this chapter is based on a joint publication with Maximilian Schiffer, Stephan Sorgatz, and Stefan Weltge [69].

2.1 Background

The promotion and deployment of electric vehicles is a key part of the transformation of the mobility sector towards more emission-free and sustainable transport. One of the main challenges is the coordination of charging processes and mobility behavior in general. For commercial fleets a multitude of studies and projects that support the recharging process exist. Contrarily, in the private sector, early adopters mainly rely on private charging solutions, e.g., wallboxes on their own property. Especially in urban environments, potential electric vehicle (EV) users do not necessarily have a private parking space, such that the supply of public charging infrastructure is crucial to increase the acceptance of EVs. Accordingly, identifying the right amount of and locations for public charging stations in cities is of particular interest in order to achieve the goals for the market diffusion of EVs. In fact, governments have already set aggregated targets for the development of charging infrastructure. However, municipalities are struggling with the implementation due to the high costs of installing charging stations and the inherent complexity of the problem, which stems from the fact that the infrastructure is addressed to the general public, i.e., obtaining good solutions requires to solve a large scale facility location problem.

Accordingly, designing charging station networks has gained significant interest in transport optimization and various approaches have been proposed to accomplish this planning task. The most common approaches are node-based and aggregate charging demand without a temporal dimension, or are path-based and account for charging demands in aggregated flows. However, these existing approaches suffer from two major drawbacks: first, both approaches do not consider driver behavior via individual patterns. Second, as a consequence of aggregation, even path-based models insufficiently consider temporal interactions between drivers that occupy the same charging stations. Mitigating these fundamental drawbacks by incorporating individual and realistic demands and temporal interactions into a mathematical planning approach remains a fundamental challenge to provide profound decision support that enables the market diffusion of EVs. First approaches that tackle these issues can only solve small-scale instances or are simulation-based approaches that cannot be handled within rigorous optimization methods.

Against this background, we develop a new approach to compute optimal placements of public charging stations, incorporating the aforementioned driver-based features. Moreover, our framework supports individual charging modes and curves, a flexible choice of objectives, and exact positioning. Despite incorporating this level of complexity, we present an integer programming based approach that remains at the one hand computationally tractable and at the other hand easy to implement. We demonstrate the effectiveness of our approach on instances based on traffic data of German major cities. We show that our method scales reasonably well for future electrification rates of up to 15% for cities of up to 600 000 inhabitants.

2.1.1 Literature Review

In the following, we give a short overview of the related literature for the placement of public charging stations, which comprises three main streams, considering flow-based demand for recharging facilities [56, 72], arc-based demand [25], or node-based demand. The flow-based and arc-based models find a cover of origin-destination-pairs, or arcs in a given network respectively. In particular the refueling station location problem with routing (RSLP-R) is an active area of research for which recently specialized branch-and-cut [6, 53] as well as branch-and-price [105] approaches were developed. In these studies, refueling stations are placed along long trips, which underlines their applicability for inter-regional travel. While it is possible to adapt these approaches to an urban context by mapping refueling possibilities to pre-existing breaks, several problem characteristics that are native to the traffic in cities are not considered in this stream of research. This includes for instance different charging speeds, the capacity of charging stations or the effect of break durations, i.e., that short stops may not be enough to fully recharge an EV.

In contrast, node-based approaches aim to cover discrete demand (clusters) at given points. Most node-based approaches work in a two-fold manner: in a first step, a (tempo)-spatial demand is estimated. This is usually done by aggregating the demand of several vehicles and forming demand clusters. These clusters are covered in a second step under consideration of problem-specific constraints, e.g., vehicle-to-grid technologies [36], the exact consumption [2, 55], or the electric grid [13]. In general, this approach is more appropriate for the urban context, but the static demand aggregation inherently neglects important features of the problem. Especially dynamic features like individual user behavior or interaction of users with respect to time are hard to portray in such models [1]. There exist some attempts to include these dynamic properties in the literature, for instance Cavadas et al. [26] introduce transferable demand and multiple periods, Adenaw and Lienkamp [1] combine a simulation approach with an evolutionary process to find the best charging stations, Shahraki et al. [93] accurately model individual demand for plug-in hybrid electric vehicles, and Andrews et al. [3] introduce a MIP that accounts for user interaction. As we will detail in Section 2.2, these approaches leave room for a more detailed analysis of dynamic properties.

2.1.2 Contribution

We aim to derive a mathematical programming based approach that allows to determine positions of charging stations in an urban environment such that EV drivers experience a convenient charging experience, i.e., they can maintain their individual daily schedule without significant deviations for charging. To this end, we formulate the problem of finding such positions in a purely combinatorial manner, accounting for individual driver patterns, and temporal interactions, as well as individual charging modes and curves, a flexible choice of objectives, and exact positioning. This formulation leads to mixed integer linear problem (MILP) models, for which we present strengthened formulations that allow to handle large instances and various objective functions. As a byproduct, this approach also yields dual bounds on the quality of our placements.

We show how to utilize this approach in practice by making use of traffic data that contains information of individual drivers on a typical day within a given city, which can easily be gathered from various traffic simulation frameworks, e.g., MATSim [57]. We present results of a case study for the city of Düsseldorf (Germany, $\approx$ 620k citizens), which show that our approach allows to solve instances with the current electrification rate ($\approx 1\%$), within few minutes and instances with electrification rates up to 15% in few hours. This equals placing up to 1500 charging ports optimally to cover more than 3800 driver profiles with recharging demand, which improves significantly upon the current state of the art approach that handles temporal interactions [26], which has been limited to placing 9 stations for 300 drivers.

Moreover, we evaluate the quality of the obtained solutions within a simulation environment to show the efficacy of our approach from a practitioners' perspective.

To foster future research and the use of our approach in practice, we open the implementation of our optimization method, our simulation environment, as well as the case study data on Github[1].

2.1.3 Organization

In Section 2.2, we describe our general approach, introduce the most relevant objects that we deal with and explain how they are exploited to incorporate the aforementioned driver-based features. We formulate the task of placing public charging stations as a combinatorial optimization problem in Section 2.3, which gives rise to a natural integer program. Moreover, we discuss reformulation techniques that are crucial in order to solve real-world instances efficiently. The second part of this chapter is concerned with computational experiments. In Section 2.4, we show how to derive a case study from existing traffic data. We propose a simulation framework to evaluate different positionings of charging stations in Section 2.5. Section 2.6 demonstrates the effectiveness of the proposed reformulations, and shows that the resulting model is capable to solve real-world instances. We close this chapter with remarks on possible applications and extensions of our work in Section 2.7.

[1] `https://github.com/tumBAIS/driverAwareChargingInfrastructureDesign`

2.2 Problem setting

In this section, we describe our general approach for placing public charging infrastructure in an urban environment. We introduce the most relevant objects that we deal with and explain how they are exploited to incorporate the aforementioned driver-based features.

We assume that we have access to a set of *drivers* D (of electric vehicles) and their schedules on a representative day (or set of days). Here, we identify each driver with her vehicle, typically referred to as driver vehicle unit (DVU). For each driver $d \in D$, we are given a list of *trips* $T(d)$, where each trip is defined by its start location, end location, start time, end time, and the amount by which the driver's state of charge (SOC) decreases during the trip. Implicitly, this yields information about stops in between trips, which, for example, correspond to shorter stays while shopping or longer stays while working, and hence to potential charging activities that do not affect a driver's actual schedule. Moreover, information about charging characteristics of d is available. We note that, while one may argue that this driver behavior is hardly deterministic over a longer time horizon, we will see in Section 2.6 that it is still possible to derive representative scenarios for strategic planning.

Individual charging modes and curves. For every *charging mode* $m \in M$, we are given a *charging curve* $f_{m,d}$. The set M refers to different charging modes and typically consists of two modes: AC charging (slow) and DC charging (fast). The function $f_{m,d}$ determines, for given x and t, the SOC of d after charging in mode m for t units of time, starting with an SOC of x.

Individual demands. We are further given each driver's SOC at the beginning of the day as well as bounds within which the SOC must stay over the course of the entire day. Every trip decreases the driver's SOC by a given certain amount, depending on the trip. We assume that the SOC can only be increased by charging at public charging stations, which results in a recharging demand for each driver. Note, that our planning problem only encompasses the placement of public charging infrastructure. Private charging possibilities can still be portrayed, for instance by adapting the SOC and the bounds for a driver that has access to home charging. Consequently, we also omit commercial fleets from our model, since they usually charge at specially dedicated charging stations and therefore do not need to rely on public charging infrastructure.

Accurate placements. In order to allow for charging operations, public charging stations must be made available at a subset of *potential locations* L. The latter is a finite set of locations that may arbitrarily arise from the operating area. Each location can be equipped with a certain type of charging station, which is defined by a number of charging ports of a specific mode. To obtain a high degree of accuracy in the placement, the set of locations has to be defined accordingly. This is for instance possible by using a grid approach, similar to the one employed by Cavadas et al. [26].

	charging station capacities & interactions	individual demand	charging modes & curves	different objectives	scalability
Hodgson [56], Bayram et al. [13]					✓
Kuby and Lim [72], Göpfert and Bock [53], Yıldız et al. [105]					✓
Capar et al. [25], Arslan et al. [6]				✓	✓
Hidalgo et al. [55]		✓	(✓)	✓	
Cavadas et al. [26]	(✓)				
Shahraki et al. [93]		✓	✓		(✓)
Andrews et al. [3]	✓		(✓)		
Our work	✓	✓	✓	✓	✓

Table 2.1: Characteristics of previous optimization-based approaches for the placement of public charging infrastructure.

Capacities and time interactions. Once charging stations have been placed, drivers may occupy single charging ports between their trips. Since time intervals between two consecutive trips typically refer to activities carried out by the drivers, we assume that drivers charge at (and hence occupy) a port for the whole time interval. At every time at most one driver may be connected to a port.

Different objectives. Our goal is to determine a subset of locations $L' \subseteq L$, and for each location $\ell \in L'$ the type of charging station installed at ℓ. A possible objective is to minimize the total cost associated to the selected charging stations under the constraint that every driver's demand can be satisfied. In this setting, a driver may only be allowed (or willing) to occupy charging ports at locations whose distance to the driver's current location is below a certain threshold. However, as we will see later, our approach can be easily adapted to various other objectives.

Table 2.1 shows the problem characteristics of the closest related works. As can be seen, existing approaches that are node or path-based are partially scalable, but neglect individual demand, realistic charging modes and curves, as well as temporal interactions. First approaches that aim to mitigate these shortcomings partially consider capacities but neglect individual demand and multiple objectives or consider the latter, but ignore capacities. All of these approaches are not scalable to large instances. Concluding, to the best of the authors' knowledge, none of the existing approaches accounts for individual driver patterns, temporal interactions, individual charging modes and curves, a flexible choice of objectives, as well as exact positioning, and is at the same time scalable to large instances.

2.3 Method

In this section, we derive a mathematical formulation for the task of positioning public charging stations that is based on the setting described in the previous section. We first formulate an optimization problem that includes all relevant objects and constraints in a combinatorial manner. This formulation gives rise to a natural integer program, which we discuss in Section 2.3.2. In Section 2.3.3, we describe modifications to the integer program that do not affect the set of feasible solutions but are crucial in order to solve real-world instances efficiently.

2.3.1 Combinatorial optimization problem

Recalling Section 2.2, we assume that we are given finite sets of drivers D, charging modes M, and locations for potential charging stations L. Moreover, we are given a set of trips $T(d)$ for every driver $d \in D$.

Breaks. Since we associate the time intervals between consecutive trips with a driver's charging opportunities, we extract a set of *breaks* $B(d)$ from the trip data for each driver. A break is characterized by its start time, end time, and location. We assume that breaks are individual objects for each driver, i.e., $B(d) \cap B(d') = \emptyset$ for $d \neq d' \in D$.

Nearby locations. For each break $b \in B(d)$, we compute a set of *nearby locations* $L(b) \subseteq L$ that driver d is willing to charge at during break b. For example, the set $L(b)$ may consist of all locations in L that are within a certain distance to the location of b. However, we do not make any specific assumptions on $L(b)$ in order to keep our formulation as general as possible.

Feasible charging plans. To identify at which breaks a driver $d \in D$ is charging (in a specific mode), we define a *charging plan* of d as a set of pairs $(b, m) \in B(d) \times M$. We say that a charging plan $P \subseteq B(d) \times M$ is *feasible* if its charging operations ensure that d can complete all trips while respecting the pre-defined bounds on the SOC at any time. Given the charging curves of d, we assume that we can efficiently determine whether a charging plan is feasible. We denote the set of all feasible plans of a driver d by $F(d) := \{P \subseteq B(d) \times M : P \text{ is feasible}\}$.

Charging stations. We consider a *charging station* as a tuple $(\ell, m, \Delta) \in L \times M \times \mathbb{Z}_{\geq 1}$ indicating its location ℓ, mode m, and number of ports Δ. We assume that we are given a finite set $\Omega \subseteq L \times M \times \mathbb{Z}_{\geq 1}$ denoting possible charging stations. Each charging station is associated with a (possibly individual) *cost* $c : \Omega \to \mathbb{R}_{\geq 0}$.

Time points. Finally, we define a set of *time points* T in order to monitor the capacity utilization of charging stations at these time points. We use the shorthand notation $t \in b$ to denote that a time point $t \in T$ is contained in the time interval of the break $b \in B(d)$.

Task. Our goal is to find a set of charging stations $S \subseteq \Omega$ that minimizes the total cost $\sum_{s \in S} c(s)$ under the following constraints.

1. For each driver $d \in D$ there must exist a feasible charging plan $P(d) \in F(d)$, where every charging process $(b, m) \in P(d)$ is assigned to a charging station $s(b, m) = (\ell, m, \Delta) \in S$ with $\ell \in L(b)$.

2. Moreover, these assignments have to respect the charging station capacities at any time, i.e.,
$$|\{(b, m) \in P(d) : d \in D,\ s(b, m) = s,\ t \in b\}| \leq \Delta$$
holds for every $t \in T$ and $s = (\ell, m, \Delta) \in S$.

We note that it is possible to adjust the constraints and objective, for instance by introducing a budget constraint and maximizing the amount of satisfied demand, or by working on several driver sets in parallel.

2.3.2 Integer program

In this section, we derive an integer programming formulation of the above combinatorial optimization problem. First, we introduce a binary variable for every charging station in Ω, i.e.,
$$x_{\ell,m,\Delta} \in \{0, 1\} \quad \forall\ (\ell, m, \Delta) \in \Omega,$$
where S corresponds to all (ℓ, m, Δ) with $x_{\ell,m,\Delta} = 1$. The objective function is easily expressed as
$$\text{minimize} \sum_{(\ell,m,\Delta) \in \Omega} c(\ell, m, \Delta) \cdot x_{\ell,m,\Delta}.$$
The constraints
$$\sum_{\substack{m,\Delta: \\ (\ell,m,\Delta) \in \Omega}} x_{\ell,m,\Delta} \leq 1 \quad \forall\ \ell \in L$$
ensure that we can create at most one charging station per location. In order to capture charging operations of drivers, we introduce binary variables
$$y_{d,b,\ell,m} \in \{0, 1\} \quad \forall\ d \in D,\ b \in B(d),\ \ell \in L(b),\ m \in M,$$
where $y_{d,b,\ell,m} = 1$ encodes that driver d charges at location ℓ with mode m during break b. Recall that a charging station (ℓ, m, Δ) can be occupied by at most Δ drivers simultaneously, which is expressed by the constraints
$$\sum_{d \in D} \sum_{\substack{b \in B(d): \\ \ell \in L(b),\, t \in b}} y_{d,b,\ell,m} \leq \sum_{\substack{\Delta: \\ (\ell,m,\Delta) \in \Omega}} \Delta \cdot x_{\ell,m,\Delta} \quad \forall\ \ell \in L,\ m \in M,\ t \in T. \tag{2.1}$$

Notice that for fixed $\ell^* \in L$ the sets $\{(d, b) : d \in D,\ b \in B(d),\ \ell^* \in L(b),\ t \in b\}$ can be identical for different time points $t \in T$, resulting in duplicate constraints. In fact, it suffices to impose the above constraints for the start times of all breaks b with $\ell^* \in L(b)$ only.

Finally, we need to make sure that every driver charges according to one of her feasible charging plans. A straightforward way to model this is by introducing binary variables

$$z_{d,P} \in \{0,1\} \quad \forall\, d \in D,\, P \in F(d),$$

where $z_{d,P} = 1$ encodes that driver d charges according to P. The requirement that every driver has to decide on exactly one feasible charging plan is expressed as

$$\sum_{P \in F(d)} z_{d,P} = 1 \quad \forall\, d \in D.$$

The constraints

$$\sum_{\ell \in L(b)} y_{d,b,\ell,m} = \sum_{\substack{P \in F(d):\\ (b,m) \in P}} z_{d,P} \quad \forall\, d \in D,\, b \in B(d),\, m \in M \tag{2.2}$$

ensure that driver d selects charging operations that match the chosen plan.

Note that the model variations mentioned in Section 2.3.1 can be captured in a similar way.

2.3.3 Strengthened formulations

It turns out that the above integer programming formulation can be significantly improved since its linear programming relaxation is rather weak in terms of its integrality gap. That is, it admits non-integer points satisfying all linear constraints but whose objective value is much smaller than the true optimum value. Fortunately, it is possible to derive additional linear inequalities that reduce the integrality gap significantly but do not affect the set of feasible integer solutions. We present a family of such cuts in Section 2.3.3.1.

Moreover, we discuss the constraints in our model that force drivers to follow feasible charging plans in Section 2.3.3.2. We observe that they often can be reformulated by eliminating several variables, which has an additional positive impact on the computation time needed to solve our model. The effectiveness of the strengthened formulations is demonstrated in Section 2.6.

2.3.3.1 Capacity cuts

To illustrate why our original formulation is weak, suppose that an (optimal) solution to the integer program decides to build a charging station $(\ell^*, m^*, \Delta) \in \Omega$ with $\Delta = k \geq 2$ ports. Suppose further that at every time at most one driver is charging at this station. This means that the left-hand sides of constraints (2.1) that correspond to ℓ^*, m^* are at most 1 for all $t \in T$. Since these constraints are the only ones that relate x and y directly, we can set $x_{\ell^*,m^*,\Delta} = \frac{1}{k}$ and still obtain a solution that is feasible to the linear programming relaxation. Clearly, setting a variable to $\frac{1}{k}$ instead of 1 may decrease the objective value significantly. A simple way to exclude such fractional points is by adding the inequalities

$$y_{d,b,\ell^*,m^*} \leq \sum_{\Delta:(\ell^*,m^*,\Delta)\in\Omega} x_{\ell^*,m^*,\Delta} \tag{2.3}$$

for all $d \in D$ and $b \in B(d)$ with $\ell^* \in L(b)$, which are certainly valid for all integer solutions. In our computational study, we show that adding these inequalities improves our model significantly.

We observe that, in the case where there is only one Δ with $(\ell^*, m^*, \Delta) \in \Omega$, inequalities (2.1) together with (2.3) cannot be further strengthened in the following sense:

Proposition 2.1. *Let $\mathcal{B} := \{b : d \in D,\, b \in B(d),\, \ell^* \in L(b)\}$ and suppose that $(\bar{y}, \bar{x}) \in \mathbb{R}^{\mathcal{B}}_{\geq 0} \times [0,1]$ satisfies $\sum_{b\in\mathcal{B}:t\in b} \bar{y}_b \leq \Delta\bar{x}$ for all $t \in T$ and $\bar{y}_b \leq \bar{x}$ for all $b \in \mathcal{B}$. Then $(\bar{y}, \bar{x})$ is a convex combination of binary vectors $(y', x') \in \{0,1\}^{\mathcal{B}} \times \{0,1\}$ that satisfy $\sum_{b\in\mathcal{B}:t\in b} y'_b \leq \Delta x'$ for all $t \in T$.*

Proof. If $\bar{x} = 0$, then this implies $\bar{y} = \mathbf{0}$ in which case we are done. Otherwise, let $y^* := \frac{1}{\bar{x}}\bar{y}$ and observe that $(\bar{y}, \bar{x}) = (1-\bar{x})(\mathbf{0}, 0) + \bar{x}(y^*, 1)$, that is, $(\bar{y}, \bar{x})$ is a convex combination of $(\mathbf{0}, 0)$ and $(y^*, 1)$. Thus, it remains to show that y^* is a convex combination of binary vectors $y' \in \{0,1\}^{\mathcal{B}}$ that satisfy $\sum_{b\in\mathcal{B}:t\in b} y'_b \leq \Delta$ for all $t \in T$.

To this end, notice that y^* satisfies $\sum_{b\in\mathcal{B}:t\in b} y^*_b = \frac{1}{\bar{x}} \sum_{b\in\mathcal{B}:t\in b} \bar{y}_b \leq \Delta$ for all $t \in T$ and $y^*_b = \frac{1}{\bar{x}}\bar{y}_b \leq 1$ for all $b \in \mathcal{B}$. In other words, y^* is contained in the polytope

$$Q := \left\{ \tilde{y} \in [0,1]^{\mathcal{B}} : \sum_{b\in\mathcal{B}:t\in b} \tilde{y}_b \leq \Delta \text{ for all } t \in T \right\}.$$

By ordering the constraints according to their time $t \in T$, we see that Q can be written as $\{y : Ay \leq h,\, \mathbf{0} \leq y \leq \mathbf{1}\}$, where h is an integer vector and A is a 0/1-matrix that has the *consecutive ones property.* Such matrices are known to be totally unimodular [47], which implies that all vertices of Q are integer, cf. [90, Theorem 19.3], and hence binary. Thus, y^* is a convex combination of binary vectors $y' \in \{0,1\}^{\mathcal{B}} \in Q$ as claimed. □

However, even with these constraints there are still several constellations in which the x-values can be decreased in the linear programming relaxation. For this reason, one may consider the following natural generalization of inequalities (2.3). For every $t^* \in T$ and every set $S \subseteq \{(d, b) : d \in D,\, b \in B(d),\, \ell^* \in L(b),\, t^* \in b\}$, consider the inequality

$$\sum_{(d,b)\in S} y_{d,b,\ell^*,m^*} \leq \sum_{\Delta:(\ell^*,m^*,\Delta)\in\Omega} \min(|S|, \Delta) \cdot x_{\ell^*,m^*,\Delta}. \tag{2.4}$$

Note that the inequalities (2.3) arise as a special case from (2.4) where $|S| = 1$. Again, it is easy to see that every integer solution satisfies (2.4). Still, the inequalities (2.4) do not suffice to generalize the observation of Proposition 2.1 to the case where there is more than one Δ with $(\ell^*, m^*, \Delta) \in \Omega$, and it is not clear how the inequalities can be further strengthened. Moreover, given the strength of the inequalities in (2.3) and the large number of the inequalities in (2.4), incorporating the latter inequalities in an efficient way remains challenging.

2.3.3.2 Charging plan cuts

Recall that we require every driver $d \in D$ to follow a feasible charging plan in $F(d)$. To this end, we introduced auxiliary variables $z_{d,P}$ for all $P \in F(d)$ and constraints (2.2) to link y and z. While this formulation is very explicit, we describe a reformulation that avoids the auxiliary variables z.

For a charging plan $P \in F(d)$ let $\chi(P) \in \{0,1\}^{B(d)\times M}$ denote its characteristic vector, i.e., $\chi(P)_{b,m} = 1$ if and only if $(b,m) \in P$. Consider the polytope $Q(d) := \operatorname{conv}\{\chi(P) : P \in F(d)\}$. Suppose we have computed an inequality description $Q(d) = \{q \in \mathbb{R}^{B(d)\times M} : Aq \geq h\}$ for $Q(d)$, where A is a matrix and h is a vector such that the rows of A and h correspond to some index set I and the columns of A correspond to $B(d) \times M$. The following statement shows that the auxiliary variables z and the constraints (2.2) can be avoided by imposing some specific linear inequalities on y only.

Proposition 2.2. *For a vector $y \in \mathbb{R}^\Gamma$ with $\Gamma := \{(b,\ell,m) : b \in B(d),\ \ell \in L(b),\ m \in M\}$, the following are equivalent.*

- *There exist $z \in [0,1]^{F(d)}$ with $\sum_{P\in F(d)} z_P = 1$ and*
$$\sum_{\ell\in L(b)} y_{b,\ell,m} = \sum_{\substack{P\in F(d):\\(b,m)\in P}} z_P$$
for all $b \in B(d)$ and $m \in M$.

- *The linear inequalities*
$$\sum_{(b,m)\in B(d)\times M} \left(\sum_{\ell\in L(b)} A_{i,(b,m)} y_{b,\ell,m} \right) \geq h_i \tag{2.5}$$
are satisfied for all $i \in I$.

Proof. Let $y \in \mathbb{R}^\Gamma$, and suppose there is some $z \in [0,1]^{F(d)}$ satisfying the first condition. For every $i \in I$, we have

$$\begin{aligned}
\sum_{(b,m)\in B(d)\times M} \left(\sum_{\ell\in L(b)} A_{i,(b,m)} y_{b,\ell,m} \right) &= \sum_{(b,m)\in B(d)\times M} A_{i,(b,m)} \sum_{\substack{P\in F(d):\\(b,m)\in P}} z_P \\
&= \sum_{(b,m)\in B(d)\times M} A_{i,(b,m)} \sum_{P\in F(d)} z_P \chi(P)_{b,m} \\
&= \sum_{P\in F(d)} z_P \sum_{(b,m)\in B(d)\times M} A_{i,(b,m)} \chi(P)_{b,m} \\
&\geq \sum_{P\in F(d)} z_P h_i = h_i,
\end{aligned}$$

where the inequality holds since $\chi(P)$ is contained in $Q(d)$ and hence satisfies $A\chi(P) \geq h$ for all $P \in F(d)$.

Suppose now that $y \in \mathbb{R}^{\Gamma}$ satisfies (2.5) for all $i \in I$. Define $q \in \mathbb{R}^{B(d)\times M}$ via $q_{b,m} = \sum_{\ell \in L(b)} y_{b,\ell,m}$ for all $b \in B(d)$ and $m \in M$. For every $i \in I$ we have

$$\sum_{(b,m)\in B(d)\times M} A_{i,(b,m)} q_{b,m} = \sum_{(b,m)\in B(d)\times M} \left(\sum_{\ell \in L(b)} A_{i,(b,m)} y_{b,\ell,m} \right) \geq h_i$$

and hence $Aq \geq h$. This means, that q is contained in $Q(d)$ and hence can be written as $q = \sum_{P\in F(d)} z_P \chi(P)$ for some $z \in [0,1]^{F(d)}$ with $\sum_{P \in F(d)} z_P = 1$. For every $b \in B(d)$ and $m \in M$, we obtain

$$\sum_{\ell \in L(b)} y_{b,\ell,m} = q_{b,m} = \sum_{P \in F(d)} z_P \chi(P) = \sum_{\substack{P \in F(d): \\ (b,m)\in P}} z_P.$$

□

In fact, we suggest to replace the auxiliary variables z and the constraints (2.2) by the inequality description of $Q(d)$, whenever the latter can be easily computed. For a general set of feasible charging plans $F(d)$ it is very difficult to derive an inequality description of $Q(d)$ in a closed form. However, one may use existing software tools to compute the convex hull of a given set of points. This is particularly feasible when the ambient dimension of $Q(d)$ is small. Since the ambient dimension of $Q(d)$ is equal to $|B(d)| \cdot |M|$, this approach is applicable for drivers $d \in D$ with a reasonably small number of breaks. Indeed, in our computational experiments, for most drivers the set $B(d)$ consists of at most five breaks, in which case the inequality description of $Q(d)$ can be quickly computed. Moreover, it turns out that this reformulation improves our model further.

2.4 Case study

In this section, we derive a case study from existing traffic data. We perform our evaluations on open traffic data[2] that has been generated for the city of Düsseldorf, Germany (≈ 619 000 inhabitants in 2019) provided by Rakow et al. [85]. Based on current traffic information, the authors used MATSim to simulate a collection of drivers together with their stops within a typical day in Düsseldorf. In what follows, we describe how to turn such information into driver data as needed for our method.

Our data contains 268 110 individual agents and their mobility plans on a typical work day. The set of agents encompasses all agents that spend a certain amount of time in Düsseldorf during the day. Out of these agents, we identify 113 852 as actual residents, i.e., agents that start their first trip within the city borders of Düsseldorf, which are

[2] `https://svn.vsp.tu-berlin.de/repos/public-svn/matsim/scenarios/countries/de/duesseldorf/`

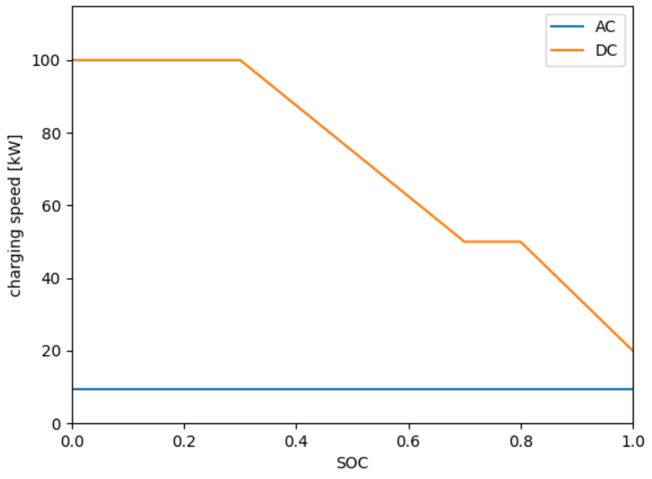


Figure 2.1: Effective charging speeds of AC and DC charging stations in our case study.

responsible for 477 277 trips, using different modes of transport. The true number of such trips was approximately 2.16 million in 2019 (calculations based on Gerike et al. [48]). Therefore, we estimate that our set of agents and consequently the portrayed traffic represents approximately one quarter of the true size.

Extracting only the trips that were done by car, we obtain 475 639 trips that are done by 128 059 different drivers. Recall, that this only represents approximately one quarter of the traffic, so we assume that our groundset of potential EVs consists of $N := 512\,236$ drivers. To identify a set of potential EV drivers, we restrict ourselves to drivers whose consecutive trips end and start at locations within a distance of 300m, which we also require for the last and first trip to hold.

An EV in our setting is characterized by its battery capacity, its range and its charging behavior. Since we are investigating an urban environment, we assume all cars to be rather compact with a battery capacity of 50kWh and an effective range of 260km, corresponding to a consumption of 19.23kWh per 100km. However, we remark that our approach would also allow to model arbitrary EVs and assign them to different drivers. We consider two modes of charging, AC and DC. For AC charging, the charging speed can be assumed to be almost independent of the SOC of the battery. Therefore, we assume a constant charging speed of 11kW and an efficiency of 0.85, which results in an effective charging speed of 9.35kW. The charging speed of DC charging typically follows a charging curve whose speed decreases with the SOCs. We use a piecewise linear function as a charging curve, as depicted in Figure 2.1.

We assume that during an average day, drivers avoid their state of charge to fall below 10% of its capacity. Based on this, we exclude all drivers whose SOC will fall below 10% even when starting with an SOC of 100% in the morning and DC charging at every break within the planning region. This results in a set of 100 854 remaining drivers, which we use as our final set of drivers $\mathcal{D}$ to sample from. Out of these drivers, 26 740 live within the city of Düsseldorf and we denote this set by $\mathcal{D}_c \subseteq \mathcal{D}$.

We assume that all drivers from $\mathcal{D} \setminus \mathcal{D}_c$ have a wallbox, i.e., they have the opportunity to recharge their vehicle at home. This is motivated by the facts that drivers in $\mathcal{D} \setminus \mathcal{D}_c$ live outside of our planning region and that houses in rural regions have a higher probability of being able to install a wallbox [12]. Moreover, we assume that 39% of the drivers in $\mathcal{D}_c$, selected uniformly at random, also have access to a wallbox. This percentage is based on the *dena* study by Bamberg et al. [12].

To generate an actual instance for our approach, we proceed as follows. Assuming a specific electrification rate $r \in [0, 1]$, we uniformly sample $rN \cdot {}^{|\mathcal{D}_c|}/_{|\mathcal{D}|}$ distinct drivers from $\mathcal{D}_c$ and $rN \cdot (1 - {}^{|\mathcal{D}_c|}/_{|\mathcal{D}|})$ drivers from $\mathcal{D} \setminus \mathcal{D}_c$.

We assume that all sampled drivers with a wallbox start with an SOC of 100%. For each driver d without a wallbox, let $\mu_d \geq 10\%$ be the minimum starting SOC needed to never reach an SOC below 10% when DC charging at every break within the planning region. The starting SOC of d is then chosen uniformly at random from the interval $[\max(\mu_d, 20\%), 100\%]$.

We define a charging plan for a driver d as feasible if it ensures that the SOC never falls below 10% and reaches at least $\max(\mu_d, 20\%)$ after the last break. To this end, we only compute minimally feasible plans, i.e., those plans that do not satisfy the previous criteria when switching from a charging break to a non-charging break, or from a DC charging break to an AC charging break. While calculating all minimal feasible charging plans is only tractable for a small number of breaks, we further declare charging plans to be feasible if they consist of at least four charging breaks.

To define the possible locations of charging stations, we use a grid approach (cf. [26, 32]) by overlaying the entire city region of Düsseldorf with a square grid with cells of side length 100m, resulting in a total of 21 745 possible locations. For each location, we allow $n_{\mathrm{AC}} \in \{2, 4, 6, 8\}$ AC charging ports, or $n_{\mathrm{DC}} \in \{4, 6, 8\}$ DC charging ports, and define its cost to be n_{AC} or $2n_{\mathrm{DC}}$, respectively.

Finally, for each break b of a driver, we define its set of nearby locations $L(b)$ as all locations within the grid whose distance is at most 200 meters. In order to reduce the size of the instance, we exclude those locations ℓ for which there exists another location ℓ' with $\{b \in B(d) : d \in D,\ \ell \in L(b)\} \subseteq \{b \in B(d) : d \in D,\ \ell' \in L(b)\}$.

2.5 Simulation

In this section, we describe the simple driver-based simulation that has been used to evaluate the positioning of charging stations obtained by our main method. The code for the simulation is available in our Github repository[3] alongside with our optimization method.

Recall that our main approach is based on the idea that a set of charging stations performs well if most drivers are able to charge during breaks without deviating from their original schedules, making a transition to EVs as easy as possible. Given a fixed

[3]`https://github.com/tumBAIS/driverAwareChargingInfrastructureDesign`

set of charging stations, we describe a simple simulation where each driver greedily aims at satisfying their own demand. The number of drivers that need to take 'large' detours in order to satisfy their demand is then regarded as a measure of performance.

Input. Apart from a fixed set of charging stations to be evaluated, our simulation is again based on the set of drivers $\mathcal{D}$, which is obtained as described in Section 2.4. Given an electrification rate, we sample a specific set of drivers D and their starting SOC as explained in Section 2.4. Moreover, we specify the maximum distance r_h a driver is willing to walk between their stop and a charging station without deviating from the schedule, and a maximum distance r_m that can be travelled in this manner at all. For our experiments, we use $r_h := 400$m and $r_m := 5000$m.

Main simulation loop. The main attention of our simulation is devoted to the decisions of individual drivers in D. We assume that each driver knows ahead of day about both the planned trips as well as the starting SOC of their EV. We say that a feasible charging plan is *compatible* (to the set of charging stations) if for every charging stop there is a charging station with the desired mode within distance r_m, and initially label it as *good* if the respective charging stations are within distance r_h. We assume that drivers prefer plans that both use few charging stops and have them as early as possible. For every driver, we determine an initial *preferred* charging plan accordingly from the set of good charging plans, or from the set of compatible charging plans if no good charging plan exists.

We sort the set of all breaks of drivers in D in increasing order with respect to the starting time. For each break of a driver d in this list, we proceed as follows:

1. If the current preferred plan of d is good and contains a charging operation at the current break, we search for a charging station within distance r_h that has a free port of the respective charging mode.
2. If such a charging station exists, d blocks a port for the whole duration of the break, and we proceed with the next break.
3. If such a station does not exist, d chooses a new good charging plan that still matches the drivers' charging operations so far and dismisses the current one, provided that such a new plan exists.
4. Otherwise, if no good charging plans remain, we remove the label 'good' from the current preferred charging plan and try to find a free charging station within a distance r_m.
5. If no such station exists, dismiss the current plan and choose another compatible plan that still matches the drivers' charging operations so far.
6. Only in the case, where no further compatible plan is available, and the current plan cannot be completed, we label a driver as "not compatible" and remove her breaks from the rest of the simulation.

Evaluation. Note that, at the end of the simulation, every driver that has not been labeled as "not compatible" was able to perform charging operations according to a feasible charging plan. Moreover, if the preferred plan of a driver d at the end of the simulation is still labeled as good, then d was able to avoid large detours at all. Thus, in order to evaluate how well the charging stations served the drivers' needs, we can compare the (i) number of drivers that were able to follow a feasible charging plan (i) without detours and (ii) with detours, and (iii) the number of drivers that were not able to follow a feasible charging plan at all (labeled as "not compatible").

Repeated simulation. We remark that the initial distribution of the SOCs described above may seem arbitrary. However, observe that at the end of each simulation loop, we can reconstruct the exact SOC of a driver at the end of the day by analyzing the performed charging processes throughout the day. These SOCs can be used as new initial SOCs for another simulation loop. Running the simulation loop several times yields SOCs that might serve as a more realistic input.

2.6 Results

In this section, we demonstrate the effectiveness of the strengthened formulations described in Section 2.3.3 and show that the resulting model is capable to solve real-world instances.

2.6.1 Effectiveness of strengthened formulations

To analyze the effectiveness of the strengthened formulations presented in Section 2.3.3, first recall that we proposed to add the *capacity cuts* (2.3). Second, we suggested to use *charging plan cuts*, i.e., inequality descriptions of $Q(d)$ for each driver d (see Section 2.3.3.2) instead of the auxiliary variables z and constraints (2.2). A third enhancement that turned out to be very efficient is to relax the binary variables y (which assign charging processes to locations) to *fractional* variables in $[0, 1]$. Clearly, this may result in selections of charging stations that do not permit a feasible assignment of charging processes to locations. However, in all our experiments, we found that this caused almost no change in the value of the optimal solution, but had a significant impact on the performance of our model.

To evaluate the impact of the proposed enhancements, for each electrification rate in $\{1\%, 2\%, \ldots, 6\%\}$, we independently sampled driver data for 10 days as described in the previous section. The size of the instances in our experiments spans a range between 200 and 1 200 drivers (that need to charge at a public charging station) who have between 1 000 and 6 000 stops in total. For each subset of the three enhancements, we ran the adapted model on all generated instances using Gurobi Optimization, LLC [52] on a standard laptop. First, we evaluated the computational times, averaged over the ten instances per electrification rate, until Gurobi determines a gap of at most 1%. Here, the gap is defined as $(P - D)/D$, where P denotes the objective value of the best known feasible solution and D denotes the dual bound. The results are depicted in Figure 2.2.

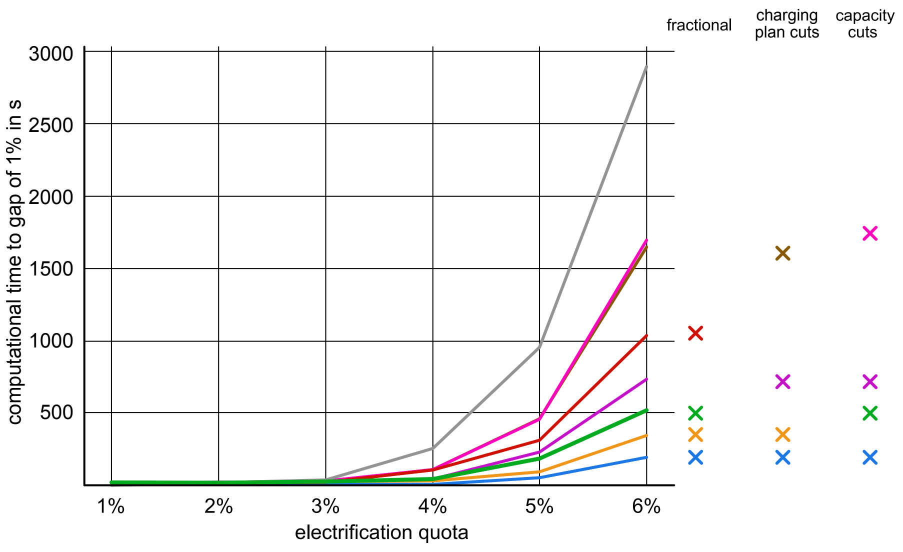


Figure 2.2: Comparison of average computational times for enhanced models. Each curve represents a variant of our original model, where the selected enhancements are depicted on the right.

We observe that in our calculations, adding additional enhancements always has a positive effect on the computational time of our model. The strength of the enhancements appears to increase from 'capacity cuts' to 'charging plan cuts' to the fractional relaxation of the y variables.

Secondly, we evaluated the quality of the linear programming relaxations of the respective models by evaluating the 'root LP gap' given by $(\mathrm{OPT}-D)/D$, where OPT denotes the optimum value and D denotes the dual bound directly after the linear programming relaxation has been solved. The results are depicted in Figure 2.3.

It should be noted that since Gurobi's preprocessing routines may further strengthen the linear programming relaxations, it is difficult to specify the actual linear program that determines the dual bound D. However, as above, we observe that adding enhancements is favorable in order to improve the root LP gap. In particular, adding the capacity cuts improves the gap most significantly. Note that without preprocessing, the other two enhancements would not have an effect on the dual bound.

In summary, we recommend to use all three presented enhancements in a practical application. For the 'fractional' enhancement, we observe the largest improvement in terms of computational time, which comes at the cost that the solution is not necessarily feasible. Since the exact future mobility demand is not known a priori and the new solution value is almost indistinguishable from the original, this cost seems negligible. The two different classes of cuts also improve the computational time by an extent that clearly outweighs their generation cost.

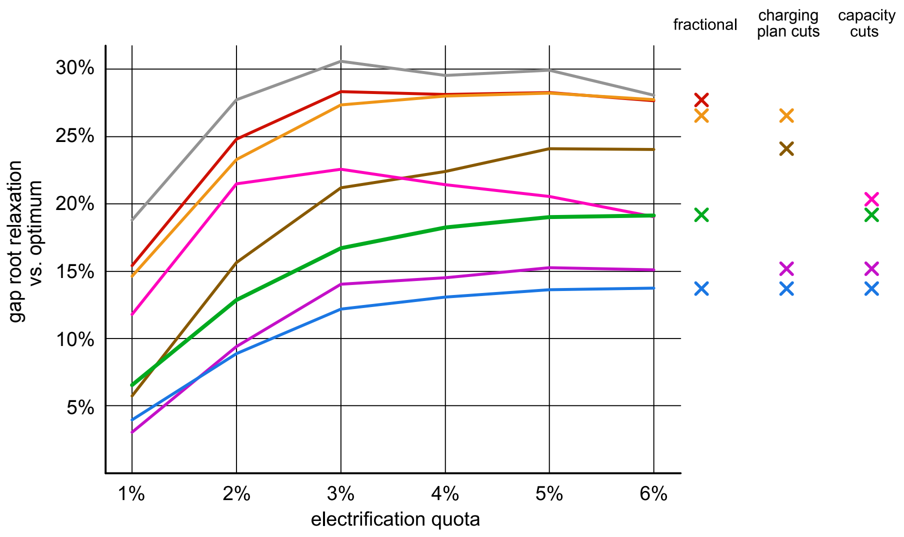


Figure 2.3: Comparison of average root LP gap for enhanced models. Again, each curve represents a variant of our original model, where the selected enhancements are depicted on the right.

2.6.2 Case study analysis

In this section, we demonstrate that our approach is able to determine positions for charging infrastructure in large urban environments that satisfy driver demands to a high degree.

To this end, we consider the two following slight modifications of the presented model. First, instead of minimizing the cost and satisfying all drivers, we maximize the number of satisfied drivers under the constraint that the cost is below a given budget. This allows us to compare our solutions with existing positions, and can be easily incorporated into our model by introducing binary variables for each driver indicating whether their demands could be satisfied. Second, instead of considering a single set of drivers, we incorporate multiple independent sets of drivers (which can be thought of as different days). This avoids overfitting to driver data of a single day.

Notice that, in our model, not only the positioning of charging stations but also the allocation of a fixed set of drivers to charging stations is optimized. Thus, in order to evaluate our method on realistic instances, it is necessary to understand how a solution performs on (i) an unknown set of drivers who (ii) occupy charging stations according to some natural behavior.

We propose to evaluate the positioning of charging stations by means of a driver-based simulation. A simple version, where each driver greedily aims at satisfying their own demand is described in Section 2.5. Recall that our goal is to determine charging stations that allow drivers to switch to EVs without having to significantly deviate from their original schedules. To this end, within the simulation, we count the number of drivers that need to take 'large' detours in order to satisfy their demand. Here, we say that a

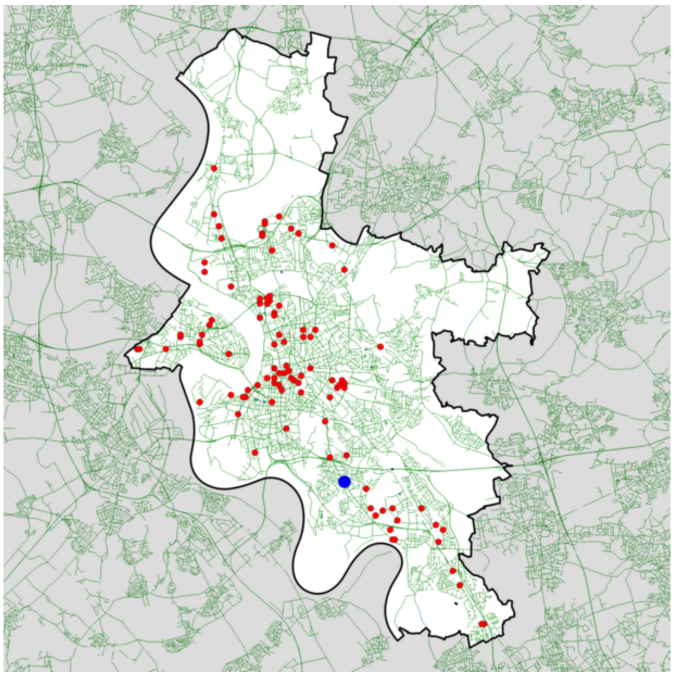


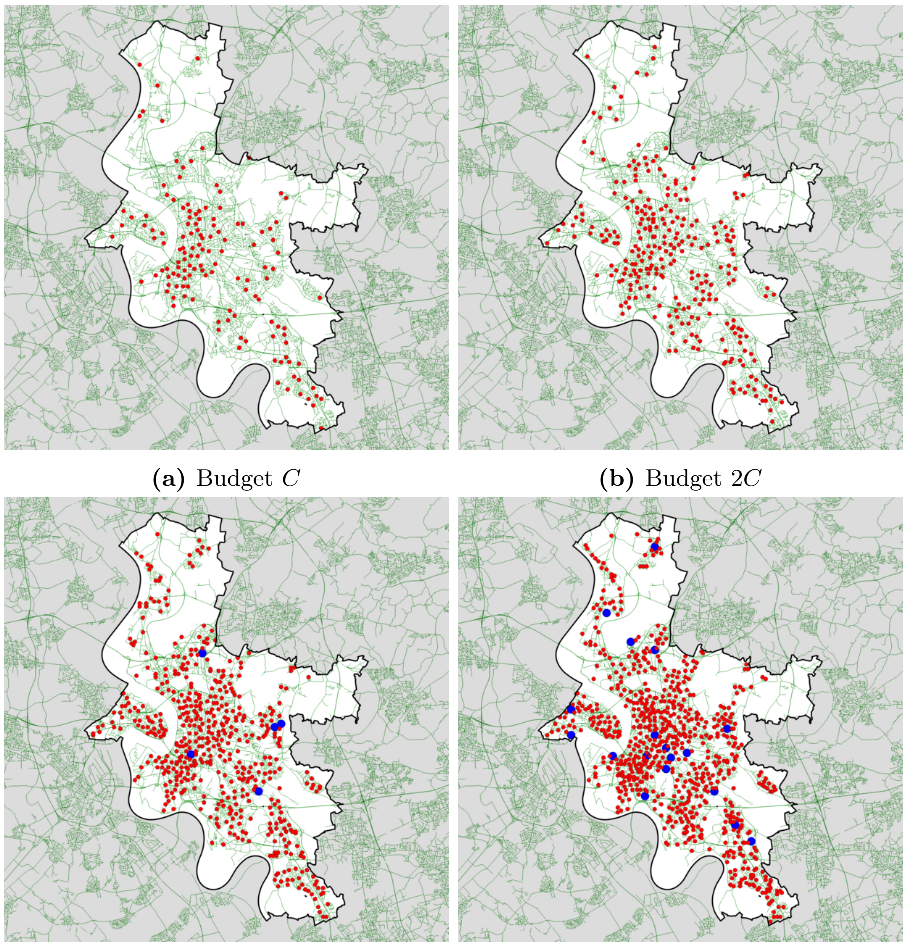
Figure 2.4: Currently existing charging infrastructure in the city of Düsseldorf (November 2021). Red points represent AC charging stations, and blue points represent DC charging stations.

driver took a large detour if she charges at a distance of more than 400 m to the position of her respective break. A walking distance in this range is a standard assumption in the related literature (cf. [89]). If the distance is more than 5 km, then we say that the driver's schedule is not compatible.

The baseline for our experiments are the positions of existing charging infrastructure in Düsseldorf[4] (cf. Figure 2.4), which has a total cost of $C = 272$ cost units as defined in Section 2.4. We compute four solutions with budgets worth C, $2C$, $4C$ and $6C$, each based on two random driver sets corresponding to electrification rates of 3%, 5%, 12%, and 18%, respectively. The specific models were chosen in a way that the respective budgets had to be fully used. While there is no clear indication what electrification rates and number of driver sets are optimal as an input, we found that solutions obtained from different variations of the input performed almost equally in our evaluations.

Each solution was found within at most 5 hrs using Gurobi version 9.5.1 on a standard laptop and is within 1% of the optimal solution. The solutions consist of 272, 544, 1068, and 1560 charging ports, respectively, and the arrangements of the respective charging stations are shown in Figure 2.5. We see that, in each solution, the charging infrastructure is much more evenly dispersed than in the reference arrangement (see Figure 2.4).

[4] `https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Energie/Unternehmen_Institutionen/E_Mobilitaet/Ladesaeulenregister.xlsx`


(a) Budget C **(b)** Budget $2C$

(c) Budget $4C$ **(d)** Budget $6C$

Figure 2.5: Charging station positions for selected solutions of the budget-constrained model. Red points represent AC charging stations, and blue points represent DC charging stations.

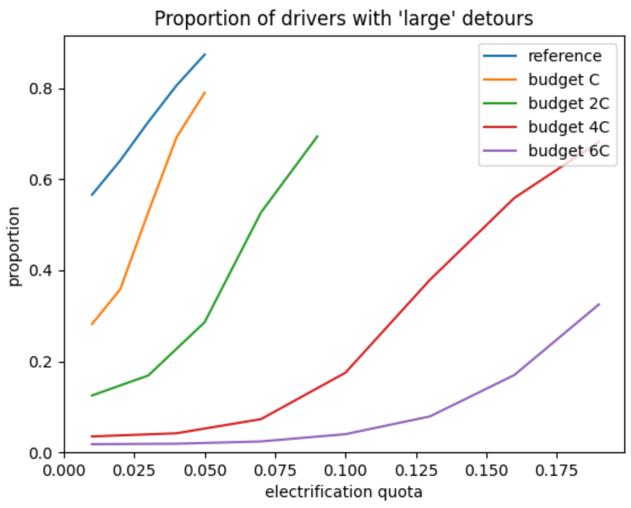


Figure 2.6: Proportion of drivers that need to take a detour of more than 400 m for at least one of their charging processes. Each datapoint is the average over 10 simulation runs.

This is in particular observable in Figure 2.5a, which operates on the same budget as the reference arrangement, but covers a larger area. Since in our model, DC charging stations are twice as expensive as AC stations, they are only placed within scenarios that permit a higher budget, i.e., for a budget of more than $4C$ (see Figure 2.5c and Figure 2.5d). The general structure of our solutions is an even distribution over the central part of the city (see in particular Figure 2.5a). Starting from a budget of $2C$, also outer areas are well covered (see Figure 2.5b). While a budget of $4C$ additionally allows for some DC charging stations (see Figure 2.5c), a budget of $6C$ suffices for an even distribution of both types of charging stations over the whole city area.

To evaluate the performance of our solutions, we depict the proportion of drivers that need to take at least one detour of at least 400 m in Figure 2.6. We clearly see that our solution with a budget of C outperforms the reference solution. Still, neither of the two positionings can satisfy the demand of an unknown set of drivers to a satisfying degree. In contrast, if we allow for a budget of $4C$, our solution provides sufficient charging infrastructure for populations that consist of up to 10% of EVs. The 6-fold budget suffices for an electrification rate of 15% consistently.

In Figure 2.7, we show the proportion of drivers whose schedules are not compatible, i.e., who need to take a detour of more than 5000 m for at least one of their charging processes. The graphic shows that the positionings of charging stations appear to have a ‘tipping point’: a specific electrification rate at which the number of incompatible drivers begins to drastically increase. At lower electrification rates, the proportion of incompatible drivers is almost constant, and is comprised of the drivers whose demands are not covered by the positioning. This graphic underlines the result of Figure 2.6: while a budget of $2C$ is not sufficient to cover the entire region of the city (even for small

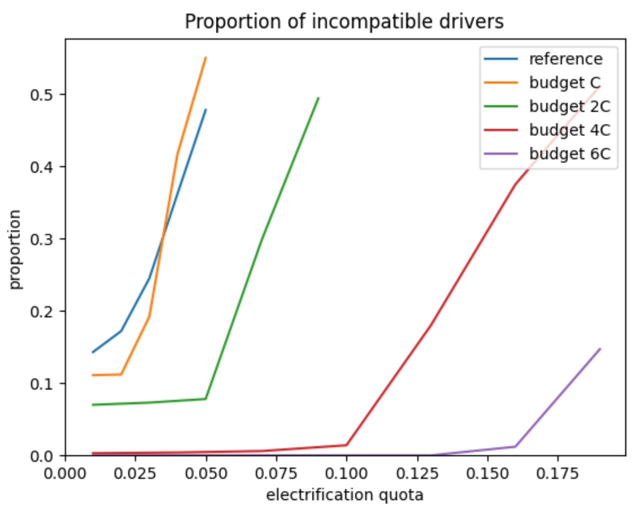


Figure 2.7: Proportion of drivers that are incompatible. Each datapoint is the average over 10 simulation runs.

electrification rates), our solution for the 4-fold budget performs well for electrification rates of up to 10%, and a 6-fold budget is sufficient for electrification rates of 15%.

2.7 Outlook

We introduced a mathematical programming based approach for determining positions of charging stations that allow EV drivers to maintain their individual daily schedule without significant deviations for charging. We formulated this problem as a combinatorial optimization problem and derived MILP models with strengthened formulations that allow to handle large instances and various objective functions. We demonstrated how this approach can be utilized in practice based on traffic data containing mobility information of individual drivers. We presented results of a case study for the city of Düsseldorf, which show that our approach allows to efficiently solve instances with electrification rates up to 15%.

Our work paves the way for future research from a methodological and from a practitioner's perspective. From a methodological perspective, it remains interesting to enhance our framework for solving even larger instances. This can be done by either further improving our exact approach or by developing competetive metaheuristics. In the latter case, our exact approach may serve for benchmarking purposes.

From a practitioner's perspective, several questions remain that require additional case studies. First, it remains interesting to analyze how existing charging infrastructure should be extended to meet increasing demand. The approach presented in this chapter is readily applicable to such a problem setting (by fixing decision variables for existing charging stations). Second, analyzing various case studies can help to detect structural properties that yield profound insights to inform city planners in practice. Third, extending our planning model to account for power network related constraints remains a challenging but also crucial avenue for future work.

3 Improved lower bound on the dimension of the EU council's voting rules

The content of this chapter is based on a joint publication with Stefan Weltge [68].

3.1 Background

Simple games are cooperative games that are commonly used to describe real-world voting systems. Considering a fixed, finite set M of voting members, a *simple game* is given by a collection $\mathcal{W}$ of subsets of M satisfying the monotonicity property: $C \in \mathcal{W}$ and $C \subseteq C' \subseteq M$ implies $C' \in \mathcal{W}$. The sets in $\mathcal{W}$ are called *winning coalitions*, and each subset of M that is not in $\mathcal{W}$ is called a *losing coalition*. It is convenient to require the empty coalition to be losing and the grand coalition of all voting members to be winning when dealing with real-world examples. A fundamental class of simple games are *weighted games* whose winning coalitions can be written as

$$\mathcal{W} = \left\{ C \subseteq M : \sum\nolimits_{m \in C} a_m \geq \beta \right\}$$

for some $a \in \mathbb{R}^M_{\geq 0}$ and $\beta \in \mathbb{R}$. Note that there exist winning and losing coalitions if and only if $0 < \beta \leq \sum_{m \in M} a_m$. It is a basic fact that every simple game is the intersection of finitely many weighted games, and hence we may define the *dimension* of a simple game $\mathcal{W}$ to be the smallest number of weighted games whose intersection is $\mathcal{W}$.

In a similar way, the *codimension* of a simple game was defined by replacing *intersection* with *union* in the above definition [46]. As a third measure for the complexity of the description of simple games, the *boolean dimension* was introduced, which allows arbitrary combinations of intersections and unions [43]. The three notions are similarly interesting from a mathematical point of view. Nevertheless, the notion of dimension stands out above the others for its analogy to the $\mathcal{H}$-representation of polyhedra.

Determining the dimension of (simple games associated to) real-world voting systems has been of particular interest in social choice theory, see, e.g., the books by Taylor and Zwicker [99] and Taylor and Pacelli [97]. Even though it is in general NP-hard to determine the exact dimension of a given simple game [37], the dimensions of many real voting rules are known. For instance, many real world examples actually have dimension one, which is easy to verify. Examples of dimension two are given by the US federal legislative system [98] and the amendment of the Canadian constitution [67]. A voting rule of dimension three has been adopted by the Council of the European Union under the treaty of Nice [45] and by the Legislative Council of Hong Kong [29].

A new record was set with the change of the EU (European Union) council's voting system by the Treaty of Lisbon in 2014. Based on the population data of 2014, Kurz and Napel [74] showed that its dimension is at least 7 and at most 13 368, and they posed the exact determination as a challenge to the community. In response, Chen, Cheung, and Ng [28] were able to reduce the upper bound to 24.

We provide the first improved lower bound and show that the dimension is at least 8. Although we will not rely on this interpretation in what follows, the idea behind our lower bound is based on the observation that the dimension of a simple game $\mathcal{W}$ can be seen as the chromatic number of a particular hypergraph H: the nodes of H are the losing coalitions, and a set of losing coalitions $\mathcal{N}$ forms a hyperedge iff $\mathcal{N} \cap \mathcal{W}' \neq \emptyset$ for every weighted game $\mathcal{W}' \supseteq \mathcal{W}$. The proof of Kurz and Napel [74] establishes that H contains a *clique* of cardinality 7, which directly implies that the chromatic number of H is at least 7. This idea has been used previously in the context of lower bounds on sizes of integer programming formulations [58, 60, 42]. While we have not found any *simple* subgraph of larger chromatic number, we will show that H contains a *hyper*graph on 15 nodes whose chromatic number is 8.

3.1.1 Organization

In Section 3.2 we introduce the concept of non-separable subsets of the losing coalitions of a simple game $\mathcal{W}$. A family $\mathcal{F}$ of such subsets can be thought of as a subgraph of the above hypergraph. Moreover, we consider the notion of a k-cover for such a set $\mathcal{F}$, which can be seen as a node-coloring of the respective subgraph with k colors. Accordingly, we will see that if the dimension of $\mathcal{W}$ is at most k, then there exists a k-cover for each $\mathcal{F}$. In Section 3.3 we consider the simple game associated to the EU council and give a construction of a set $\mathcal{F}$, for which no 7-cover exists. A proof of the latter fact will be given in Section 3.4. We give a short geometric intuition on the proof for the currently standing upper bound in Section 3.5. In the final Section 3.6, we comment on the structure of the subgraph, that is used to obtain the improved lower bound and on possible further improvements.

3.2 Strategy

In what follows, we consider simple games on a common fixed ground set M.

Definition 3.1. Let $\mathcal{W}$ be a simple game and $\mathcal{N}$ be any set of losing coalitions of $\mathcal{W}$. We say that $\mathcal{N}$ is *non-separable* with respect to $\mathcal{W}$ if every weighted game $\mathcal{W}' \supseteq \mathcal{W}$ satisfies $\mathcal{W}' \cap \mathcal{N} \neq \emptyset$.

From the definition it is immediate that a simple game is weighted if and only if no set of losing coalitions is non-separable. So, the existence of a single non-separable set yields that the dimension of a simple game is at least two. To obtain a larger lower bound, the following notion will be useful.

Definition 3.2. Let $\mathcal{W}$ be a simple game with losing coalitions $\mathcal{L}$, and let $\mathcal{N}_1, \ldots, \mathcal{N}_t \subseteq \mathcal{L}$ be non-separable with respect to $\mathcal{W}$. A *k-cover* of $(\mathcal{N}_1, \ldots, \mathcal{N}_t)$ is a collection of sets $\mathcal{L}_1, \ldots, \mathcal{L}_k \subseteq \mathcal{L}$ such that

1. $\mathcal{L}_1 \cup \cdots \cup \mathcal{L}_k = \mathcal{N}_1 \cup \cdots \cup \mathcal{N}_t$ and
2. $\mathcal{N}_i \nsubseteq \mathcal{L}_j$ for all $i \in \{1, \ldots, t\}$, $j \in \{1, \ldots, k\}$.

In order to obtain a lower bound on the dimension, we will exploit the following observation.

Lemma 3.3. *Let $\mathcal{W}$ be a simple game with non-separable sets $\mathcal{N}_1, \ldots, \mathcal{N}_t$. If $\mathcal{W}$ has dimension at most k, then there exists a k-cover for $(\mathcal{N}_1, \ldots, \mathcal{N}_t)$.*

Proof. If $\mathcal{W}$ has dimension at most k, then there exist k weighted games $\mathcal{W}_1, \ldots, \mathcal{W}_k$ such that $\bigcap_{i=1}^k \mathcal{W}_i = \mathcal{W}$. For $i \in \{1, \ldots, k\}$ define $\mathcal{L}_i$ as the intersection of the losing coalitions in $\mathcal{W}_i$ and $\mathcal{L}^* := \mathcal{N}_1 \cup \cdots \cup \mathcal{N}_t$.

We claim that $(\mathcal{L}_1, \ldots, \mathcal{L}_k)$ is a k-cover of $(\mathcal{N}_1, \ldots, \mathcal{N}_t)$. In order to show Property 1, first observe that $\mathcal{L}_1 \cup \cdots \cup \mathcal{L}_k \subseteq \mathcal{L}^*$ holds. Now, for any $\ell \in \mathcal{L}^* \subseteq \mathcal{L}$ we have $\ell \notin \mathcal{W}$ and hence there is an $i \in \{1, \ldots, k\}$ with $\ell \notin \mathcal{W}_i$, which implies $\ell \in \mathcal{L}_i$.

For Property 2, assume that $\mathcal{N}_i \subseteq \mathcal{L}_j$ holds for some $i \in \{1, \ldots, t\}$ and $j \in \{1, \ldots, k\}$. This means that each coalition in $\mathcal{N}_i$ is losing for $\mathcal{W}_j$, meaning that $\mathcal{N}_i$ and $\mathcal{W}_j$ are disjoint. This contradicts the fact that $\mathcal{N}_i$ is non-separable with respect to $\mathcal{W}$ since $\mathcal{W}_j \supseteq \mathcal{W}$ is weighted. □

In what follows, we will consider the simple game associated with the EU council and construct a collection of non-separable losing coalitions that does not permit a 7-covering. By Lemma 3.3 this implies that the dimension must be at least 8.

3.3 Our Construction

Let us give a formal definition of the simple game associated to the EU council based on the population data of 2014, as considered by Kurz and Napel [74]. In 2014, the European Union consisted of 28 members and hence we may fix $M := \{1, \ldots, 28\}$. In the voting system of the EU council, a coalition is winning if

1. it contains at least 55% of all members states *and*
2. it unites at least 65% of the total EU population,

or

3. it consists of at least 25 of the 28 member states.

Denoting the weighted game associated with rule i by $\mathcal{W}_i$ and the simple game that represents the voting system of the EU council by $\mathcal{W}_{\mathrm{EU}}$, we thus have

$$\mathcal{W}_{\mathrm{EU}} = (\mathcal{W}_1 \cap \mathcal{W}_2) \cup \mathcal{W}_3.$$

#	Member state	Population	Percentage	#	Member state	Population	Percentage
1	Germany	80 780 000	15.9%	15	Austria	8 507 786	1.7%
2	France	65 856 609	13.0%	16	Bulgaria	7 245 677	1.4%
3	United Kingdom	64 308 261	12.7%	17	Denmark	5 627 235	1.1%
4	Italy	60 782 668	12.0%	18	Finland	5 451 270	1.1%
5	Spain	46 507 760	9.2%	19	Slovakia	5 415 949	1.1%
6	Poland	38 495 659	7.6%	20	Ireland	4 604 029	0.9%
7	Romania	19 942 642	3.9%	21	Croatia	4 246 700	0.8%
8	Netherlands	16 829 289	3.3%	22	Lithuania	2 943 472	0.6%
9	Belgium	11 203 992	2.2%	23	Slovenia	2 061 085	0.4%
10	Greece	10 992 589	2.2%	24	Latvia	2 001 468	0.4%
11	Czech Republic	10 512 419	2.1%	25	Estonia	1 315 819	0.3%
12	Portugal	10 427 301	2.1%	26	Cyprus	858 000	0.2%
13	Hungary	9 879 000	1.9%	27	Luxembourg	549 680	0.1%
14	Sweden	9 644 864	1.9%	28	Malta	425 384	0.1%

Table 3.1: Population data of the European Union on 01.01.2014, see also [74, Table 1].

Note that $\mathcal{W}_2$ depends on the population of each member state. As in [74], we will work with the data depicted in Table 3.1. Out of the 2^{28} possible coalitions, 30 340 718 are winning. It can be seen that the following coalitions are losing with respect to $\mathcal{W}_{\mathrm{EU}}$.

$$\begin{aligned}
L_1 &:= M \setminus \{1,4,7,13,14\} & L_2 &:= M \setminus \{2,3,6,13\} \\
L_3 &:= M \setminus \{1,4,5,28\} & L_4 &:= M \setminus \{1,2,10,11,14\} \\
L_5 &:= M \setminus \{1,3,8,11,15\} & L_6 &:= M \setminus \{1,4,8,9,12\} \\
L_7 &:= M \setminus \{1,2,7,17,18\} & L_8 &:= M \setminus \{1,5,6,16,18\} \\
L_9 &:= M \setminus \{2,4,6,15,21\} & L_{10} &:= M \setminus \{1,3,9,10,12\} \\
L_{11} &:= M \setminus \{3,4,5,20,22\} & L_{12} &:= M \setminus \{2,3,7,8,9\} \\
L_{13} &:= M \setminus \{1,3,6,26\} & L_{14} &:= M \setminus \{2,3,5,19\} \\
L_{15} &:= M \setminus \{16,17,\ldots,28\}
\end{aligned}$$

In fact, note that each coalition L_i contains less than 25 members. Now, $L_1, \ldots, L_{14}$ are losing since each of them unites less than 65% of the total EU population, and L_{15} is losing since it contains less than 55% of all members.

Next, we construct non-separable subsets with respect to $\mathcal{W}_{\mathrm{EU}}$ that consist of the above losing coalitions. In order to verify that these subsets are indeed non-separable, the following lemma is helpful.

Lemma 3.4. *Let $\mathcal{W}$ be a simple game and let $\mathcal{W}^*$ and $\mathcal{N}$ be sets of some winning and losing coalitions for $\mathcal{W}$, respectively, such that $|\mathcal{W}^*| \geq |\mathcal{N}|$. If*

$$|\{W \in \mathcal{W}^* : m \in W\}| \leq |\{L \in \mathcal{N} : m \in L\}|$$

holds for all $m \in M$, then $\mathcal{N}$ is non-separable with respect to $\mathcal{W}$.

Proof. Consider any weighted game $\mathcal{W}' = \{C \subseteq M : \sum_{m \in C} a_m \geq \beta\} \supseteq \mathcal{W}$ with $a \in \mathbb{R}^M_{\geq 0}$ and $\beta \in \mathbb{R}$. Then we have

$$\sum_{L \in \mathcal{N}} \sum_{m \in L} a_m \geq \sum_{W \in \mathcal{W}^*} \sum_{m \in W} a_m \geq \beta \, |\mathcal{W}^*| \, .$$

The last inequality holds because all elements of $\mathcal{W}^*$ are contained in $\mathcal{W}'$. Thus, there must exist some $L \in \mathcal{N}$, such that

$$\sum_{m \in L} a_m \geq \beta \cdot \frac{|\mathcal{W}^*|}{|\mathcal{N}|} \geq \beta.$$

Therefore, we have $L \in \mathcal{W}'$ and hence $\mathcal{W}' \cap \mathcal{N} \neq \emptyset$. Since this holds for any weighted game $\mathcal{W}' \supseteq \mathcal{W}$, $\mathcal{N}$ is non-separable with respect to $\mathcal{W}$. □

We claim that the following 2-element subsets of the above losing coalitions are non-separable.

$$\begin{aligned}
&\{L_1, L_5\}, \{L_1, L_8\}, \{L_1, L_9\}, \{L_1, L_{10}\}, \{L_1, L_{11}\}, \{L_1, L_{13}\}, \{L_1, L_{14}\}, \{L_1, L_{15}\},\\
&\{L_2, L_3\}, \{L_2, L_4\}, \{L_2, L_5\}, \{L_2, L_6\}, \{L_2, L_7\}, \{L_2, L_8\}, \{L_2, L_{10}\}, \{L_2, L_{11}\}, \{L_2, L_{15}\},\\
&\{L_3, L_4\}, \{L_3, L_5\}, \{L_3, L_7\}, \{L_3, L_9\}, \{L_3, L_{10}\}, \{L_3, L_{12}\}, \{L_3, L_{13}\}, \{L_3, L_{14}\}, \{L_3, L_{15}\},\\
&\{L_4, L_6\}, \{L_4, L_8\}, \{L_4, L_9\}, \{L_4, L_{11}\}, \{L_4, L_{12}\}, \{L_4, L_{13}\}, \{L_4, L_{14}\}, \{L_4, L_{15}\},\\
&\{L_5, L_7\}, \{L_5, L_8\}, \{L_5, L_{11}\}, \{L_5, L_{14}\}, \{L_5, L_{15}\},\\
&\{L_6, L_7\}, \{L_6, L_8\}, \{L_6, L_9\}, \{L_6, L_{11}\}, \{L_6, L_{13}\}, \{L_6, L_{14}\}, \{L_6, L_{15}\},\\
&\{L_7, L_9\}, \{L_7, L_{10}\}, \{L_7, L_{11}\}, \{L_7, L_{13}\}, \{L_7, L_{14}\}, \{L_7, L_{15}\},\\
&\{L_8, L_9\}, \{L_8, L_{10}\}, \{L_8, L_{11}\}, \{L_8, L_{12}\}, \{L_8, L_{14}\}, \{L_8, L_{15}\},\\
&\{L_9, L_{10}\}, \{L_9, L_{11}\}, \{L_9, L_{12}\}, \{L_9, L_{13}\}, \{L_9, L_{14}\}, \{L_9, L_{15}\},\\
&\{L_{10}, L_{11}\}, \{L_{10}, L_{14}\}, \{L_{10}, L_{15}\},\\
&\{L_{11}, L_{12}\}, \{L_{11}, L_{13}\}, \{L_{11}, L_{15}\},\\
&\{L_{12}, L_{13}\}, \{L_{12}, L_{15}\},\\
&\{L_{13}, L_{14}\}, \{L_{13}, L_{15}\},\\
&\{L_{14}, L_{15}\} \qquad (3.1)
\end{aligned}$$

To see that each above set $\mathcal{N} := \{L_i, L_j\}$ is non-separable, we make use of Lemma 3.4 as follows. If $L_i, L_j \neq L_{15}$, we have that L_i and L_j are contained in $\mathcal{W}_1 \setminus \mathcal{W}_2$. Pick a set of states $A \subseteq L_i \cup L_j \setminus (L_i \cap L_j)$ of minimum total population such that $W_1 := A \cup (L_i \cap L_j)$ is contained in $\mathcal{W}_3 \subseteq \mathcal{W}$. For all above pairs it can be checked that $W_2 := (L_i \cup L_j) \setminus A$ is contained in $\mathcal{W}_1 \cap \mathcal{W}_2 \subseteq \mathcal{W}$. By construction, $\mathcal{N}$ and $\mathcal{W}^* := \{W_1, W_2\}$ satisfy the assumptions of Lemma 3.4 and hence $\mathcal{N}$ is indeed non-separable.

Otherwise, we may assume that $L_j = L_{15}$. For all above pairs, exchanging the two members with the least population in $L_i \setminus L_{15}$ with the member of largest population in $L_{15} \setminus L_i$, results in two winning sets W_1, W_2. Again, $\mathcal{N}$ and $\mathcal{W}^* := \{W_1, W_2\}$ satisfy the assumptions of Lemma 3.4, implying that $\mathcal{N}$ is non-separable.

Moreover, the following 3-element subsets of losing coalitions are also non-separable.

$$\{L_1, L_2, L_{12}\}, \{L_1, L_4, L_7\}, \{L_1, L_6, L_{12}\}, \{L_4, L_5, L_{10}\}, \{L_5, L_{10}, L_{12}\} \tag{3.2}$$

To see that these sets are non-separable, consider the following sets of winning coalitions.

$$\begin{aligned}
W_1 &:= M \setminus \{3,7,8,9,10,11,12,15\} & W_2 &:= M \setminus \{1,2,3\} \\
W_3 &:= M \setminus \{1,2,7,14\} & W_4 &:= M \setminus \{3,4,7,8,9\} \\
W_5 &:= M \setminus \{1,8,9,10,11,12,14,15\} & W_6 &:= M \setminus \{1,3,8,9\} \\
W_7 &:= M \setminus \{2,6,7,8,13\} & W_8 &:= M \setminus \{1,7,8,9,12,13,14\} \\
W_9 &:= M \setminus \{1,3,10,11\} & W_{10} &:= M \setminus \{1,2,4\} \\
W_{11} &:= M \setminus \{3,4,7,9,13,14\} & W_{12} &:= M \setminus \{1,7,10,11,13,14,17,18\}
\end{aligned}$$

Observing that the pairs

$$\begin{aligned}
&(\{W_2, W_7, W_{11}\}, \{L_1, L_2, L_{12}\}), \\
&(\{W_3, W_{10}, W_{12}\}, \{L_1, L_4, L_7\}), \\
&(\{W_4, W_8, W_{10}\}, \{L_1, L_6, L_{12}\}), \\
&(\{W_2, W_5, W_9\}, \{L_4, L_5, L_{10}\}), \text{ and} \\
&(\{W_1, W_2, W_6\}, \{L_5, L_{10}, L_{12}\})
\end{aligned}$$

satisfy the assumptions of Lemma 3.4, we see that the sets in (3.2) are indeed non-separable.

In the next section, we show that the non-separable sets in (3.1) and (3.2) do not admit a 7-cover. Recall that this implies that the dimension must be at least 8 by Lemma 3.3.

3.4 Proof that no 7-cover can exist

For the sake of contradiction, let us assume that the non-separable sets in (3.1) and (3.2) admit a 7-cover. This implies that there exist sets $\mathcal{L}_1, \dots, \mathcal{L}_7 \subseteq \{L_1, \dots, L_{15}\}$ such that

(i) each $\mathcal{L}_j$ is an inclusion-wise maximal subset of $\{L_1, \dots, L_{15}\}$ that does not contain any of the sets in (3.1) and (3.2), and

(ii) $\mathcal{L}_1 \cup \dots \cup \mathcal{L}_7 = \{L_1, \dots, L_{15}\}$.

It can be easily verified that the only sets satisfying (i) are the following.

$$\begin{aligned}
&\{L_1, L_2\}, \{L_1, L_3, L_6\}, \{L_1, L_4\}, \{L_1, L_7, L_{12}\}, \{L_2, L_9\}, \{L_2, L_{12}, L_{14}\}, \{L_2, L_{13}\}, \\
&\{L_3, L_8\}, \{L_3, L_{11}\}, \{L_4, L_5\}, \{L_4, L_7\}, \{L_4, L_{10}\}, \{L_5, L_6, L_{10}\}, \{L_5, L_6, L_{12}\}, \\
&\{L_5, L_9\}, \{L_5, L_{10}, L_{13}\}, \{L_6, L_{10}, L_{12}\}, \{L_7, L_8\}, \{L_8, L_{13}\}, \{L_{11}, L_{14}\}, \{L_{15}\}
\end{aligned} \tag{3.3}$$

In what follows, for a weight-vector $w = (w_1, \dots, w_{15}) \in \mathbb{R}^{15}$, let us define the *weight* of a set $\mathcal{L}' \subseteq \{L_1, \dots, L_{15}\}$ as $w(\mathcal{L}') := \sum_{i: L_i \in \mathcal{L}'} w_i$.

Suppose first that none of the sets $\mathcal{L}_1, \dots, \mathcal{L}_7$ is equal to $\{L_1, L_3, L_6\}$. In this case, consider the weight-vector

$$w = (1/2, 0, 1, 1/2, 0, 1, 1/2, 0, 1, 0, 0, 0, 1, 1, 1)$$

and observe that the weight of each set in (3.3) that is distinct from $\{L_1, L_3, L_6\}$ is at most 1. Thus, the weight of each set $\mathcal{L}_1, \dots, \mathcal{L}_7$ is at most 1, and we obtain

$$7 < \tfrac{15}{2} = w(\{L_1, \dots, L_{15}\}) = w(\mathcal{L}_1 \cup \dots \cup \mathcal{L}_7) \leq w(\mathcal{L}_1) + \dots + w(\mathcal{L}_7) \leq 7,$$

a contradiction.

It remains to consider the case that one of the sets $\mathcal{L}_1, \dots, \mathcal{L}_7$ is equal to $\{L_1, L_3, L_6\}$, say $\mathcal{L}_1$. Consider the weight-vector

$$w = (0, 1/3, 0, 2/3, 1/3, 0, 1/3, 2/3, 2/3, 1/3, 1, 2/3, 1/3, 0, 1)$$

and observe that the weight of each set in (3.3) is at most 1, and that $w(\mathcal{L}_1) = 0$. Thus, we have

$$6 < \tfrac{19}{3} = w(\{L_1, \dots, L_{15}\}) = w(\mathcal{L}_1 \cup \dots \cup \mathcal{L}_7) \leq w(\mathcal{L}_2) + \dots + w(\mathcal{L}_7) \leq 6,$$

another contradiction. This completes our proof.

3.5 Strategy for the upper bound by Chen et al. [28]

We proceed to give the general idea for the proof by Chen, Cheung, and Ng [28], which reduced the upper bound on the dimension from 13 368 to 24. They give their Theorem 1, which is crucial for the upper bound in terms of weighted games. We add a short geometric intuition, how the theorem can structurally be applied to the given problem.

The integral observation for the upper bound result is that the set $\mathcal{W}_3 \setminus (\mathcal{W}_1 \cap \mathcal{W}_2)$ is both small and well-structured. Almost all coalitions that contain 25 members also unite at least 65% of the population. Exceptions to this are only

$$\begin{aligned} W_1 &:= M \setminus \{1,2,3\} & W_2 &:= M \setminus \{1,2,4\} \\ W_3 &:= M \setminus \{1,2,5\} & W_4 &:= M \setminus \{1,2,6\} \\ W_5 &:= M \setminus \{1,3,4\} & W_6 &:= M \setminus \{1,3,5\} \\ W_7 &:= M \setminus \{1,3,6\} & W_8 &:= M \setminus \{1,4,5\} \\ W_9 &:= M \setminus \{1,4,6\} & W_{10} &:= M \setminus \{2,3,4\}. \end{aligned}$$

Note that $\{7, \dots, 28\} \subseteq W_i$ for $i \in [10]$. When interpreting coalitions as the corners of the 28-dimensional $\{0,1\}$-cube, this implies that when projecting to the hyperplanes $x_j = 0$ for $j \in \{7, \dots, 28\}$, $\mathcal{W}_1 \cap \mathcal{W}_2$ already captures all of $\mathcal{W}_{\mathrm{EU}}$. Therefore, we can fix the intersection between the weighted game (which can be interpreted as a halfspace) given by rule 2 and any such hyperplane. We can find a modified hyperplane, pertaining this intersection but not separating any coalition in W_i for $i \in [10]$. This uses the fact that all winning coalitions are on the same side of the hyperplane. Therefore, 22 weighted games (one corresponding to each hyperplane), together with the weighted game given by rule 1 suffice to completely describe the case where at least one element of $\{7, \dots, 28\}$ is missing from the coalitions. This leaves a problem in 6 variables to be solved. It turns out, that there is a single additional weighted games that suffices to cover this remaining problem, resulting in a total number of 24 weighted games.

3.6 Outlook

As mentioned in Section 3.1, the dimension of $\mathcal{W}_{\mathrm{EU}}$ is equal to the chromatic number of a hypergraph that is formed by all losing coalitions. Actually, it is sufficient to consider the subgraph that is induced by maximally losing coalitions. Unfortunately, this subgraph still contains 270 179 nodes and *determining* its chromatic number seems computationally intractable. However, in order to obtain a *lower bound* on the chromatic number one may consider any smaller subgraph. Natural candidates for small subgraphs with a large chromatic number are subgraphs with many hyperedges of small cardinality.

Kurz and Napel [74] considered the *simple* subgraph induced by $\mathcal{L}^*$, which consists of all losing coalitions L such that $|L| \in \{23, 24\}$ or $L = L_{15}$. Note that our losing coalitions $L_1, \dots, L_{15}$ all belong to $\mathcal{L}^*$. In fact, this subgraph contains many edges: If a coalition L with $|L| \in \{23, 24\}$ is losing, then its population is below 65% of the total EU population. For two such losing coalitions it is quite likely that exchanging a member with a high population against some members with a small population results in two winning coalitions. In this case the losing coalitions share an edge (see Lemma 3.4). In a similar manner, it is easy to see that L_{15} is adjacent to every other coalition in $\mathcal{L}^*$.

As observed in [74], this subgraph contains many 7-cliques, showing that its chromatic number is at least 7. However, it is possible to find a 7-coloring for this graph. When also considering hyperedges of arbitrary size, we showed in Section 3.4 that one needs at least 8 colors. In order to provide a short combinatorial proof for the lower bound, it is not practical to work with the whole set $\mathcal{L}^*$, which contains 950 coalitions, even when only considering maximal losing coalitions. Therefore, we iteratively reduced the size of the set, while ensuring that the bound was still intact. For the remaining set of coalitions, we analyzed the corresponding dual of the fractional coloring linear program in order to obtain the weights used in Section 3.4.

We mention that it is possible to separate all coalitions in $\mathcal{L}^*$ from $\mathcal{W}_{\mathrm{EU}}$ with the help of 8 weighted games. Thus, in order to obtain an improved lower bound, it is necessary to include a broader set of losing coalitions and to find a more diverse set of blocking hyperedges. We expect some room for improvement, especially on the currently standing upper bound by Chen, Cheung, and Ng [28], but find it hard to predict the true value of the dimension of $\mathcal{W}_{\mathrm{EU}}$.

4 Totally Δ-modular IPs with TU constraint matrix and one additional row

The content of this chapter is based on discussions with Manuel Aprile, Samuel Fiorini, Stefan Weltge, and Yelena Yuditsky.

4.1 Background

Total unimodularity is one of the most important concepts in integer optimization. Roughly speaking, it gives a condition on the constraint matrix of integer programs such that all vertex solutions of the associated polyhedron are integral for any integer right-hand side of the problem. As such, total unimodularity allows to solve certain integer programs as linear programs, which are known to be polynomial-time solvable, for instance by the ellipsoid method [66], or even in strongly polynomial time due to the algorithm given by Tardos [96], whose running time depends only on the size of the entries in the constraint matrix. Well-known problems with such a formulation are for instance the bipartite matching problem or the maximum flow problem. To be more precise, an integer matrix A is said to be *totally unimodular*, if all its square submatrices A' fulfill $\det(A') \in \{-1, 0, 1\}$. This includes the 1×1 submatrices, i.e., single entries of A. In general, total unimodularity is very well researched and part of the main textbooks on integer programming and combinatorial optimization, see e.g. [90, 71]. Among the most important results on such matrices, we have an extensive characterization theorem [90, Thm. 19.3], their decomposition due to Seymour [92][90, Thm. 19.6], and a polynomial-time algorithm to recognize whether a given matrix is totally unimodular [90, Thm. 20.3].

A generalization of total unimodularity that has recently gained a lot of attention is given by the concept of total Δ-*modularity*. In this case, the condition on subdeterminants is relaxed such that they are bounded between $-\Delta$ and Δ. Note that since all entries of A are integer, the same holds for each subdeterminant. It is a natural question to ask, how many of the concepts above can be generalized from total unimodularity to total Δ-modularity. In particular the optimization question has been frequently asked and is a challenging open question.

Question 4.1. Does there exist a polynomial-time algorithm for solving integer programs with a totally Δ-modular constraint matrix for fixed $\Delta > 0$?

An analog to Question 4.1 is open for Δ-modular matrices, i.e., integer matrices where the same condition is relaxed to full-rank subdeterminants. Within this work, we focus on total Δ-modularity only. Question 4.1 has recently been resolved for a few interesting cases. Let us begin by remarking that the case $\Delta = 1$ is trivially true, since the definition coincides with total unimodularity, and we can apply the algorithm by Tardos [96]. The case for $\Delta = 2$ has been completely solved by Artmann, Weismantel, and Zenklusen [8], who build up on previous work by Veselov and Chirkov [101]. More recently, Nägele, Santiago, and Zenklusen [79], as well as Nägele, Nöbel, Santiago, and Zenklusen [80] made progress on testing feasibility of *strictly* Δ*-modular* integer programs for the cases $\Delta = 3$ and $\Delta = 4$ respectively by giving a randomized polynomial-time algorithm. Strictly Δ-modular matrices are a subclass of Δ-modular matrices, where full-rank subdeterminants are constrained to be in $\{-\Delta, 0, \Delta\}$, for which an analog of Question 4.1 is also open. Further, Fiorini, Joret, Weltge, and Yuditsky [44] were able to answer Question 4.1 in the affirmative for general fixed Δ if the constraint matrix fulfills the additional property that there are at most two non-zero entries in each row or column. The last work covers for instance the stable set problem with a bounded number of node-disjoint odd cycles (bounded odd cycle packing number).

In addition to the above optimization question, the concept of total Δ-modularity has proved helpful in different areas of geometry, combinatorics and optimization. In particular, we have a proximity bound on the distance of optimal solutions of the integer program to optimal solutions of its linear relaxation, see Cook, Gerards, Schrijver, and Tardos [31], which was recently improved by Celaya, Kuhlmann, Paat, and Weismantel [27]. Further, there is a polynomial upper bound on the diameter of polytopes of the form $P = \{x \mid Ax \leq b\}$ for a totally Δ-modular matrix A and an integer vector b. Such a bound was first shown by Dyer and Frieze [40] for $\Delta = 1$, and was later improved and extended to general fixed Δ by Bonfias, Di Summa, Eisenbrand, Hähnle, and Niemeier [21]. In addition, there is evidence of further results for certain subclasses of integer programs defined by totally Δ-modular matrices. Especially for the stable set problem, due to the close connection with the odd cycle packing number, there is a PTAS if $\Delta \in 2^{O(\sqrt{\log(n)/\log\log(n)})}$ (with n the number of vertices) due to Bock, Faenza, Moldenhauer, and Ruiz-Vargas [17], as well as an extended formulation of size $O(n^2)$ if $\Delta = 2$ due to Conforti, Fiorini, Huynh, and Weltge [30]. Finally, let us mention recent work on the maximal number of distinct columns in totally Δ-modular matrices, conducted by Glanzer, Weismantel, and Zenklusen [50], Averkov and Schymura [9], as well as Paat, Stallknecht, Walsh, and Xu [82].

The general Question 4.1 is out of scope for this thesis. Therefore, for the following work, we restrict ourselves to an interesting subcase.

Question 4.2. Does there exist a polynomial-time algorithm for solving integer programs with a totally Δ-modular constraint matrix for fixed $\Delta > 0$, if the constraint matrix becomes totally unimodular after the removal of one row?

It turns out that the natural transpose of the question, i.e., on totally Δ-modular matrices that become totally unimodular after the removal of one column is actually easy to solve. The Cook et al. proximity bound [31], see Theorem 4.5 gives an upper and lower bound on the additional variable (column). We can then solve the problem by using dynamic programming on the value of this variable and solving $O(n\Delta)$ totally unimodular integer programs.

Question 4.2 defines a second class of integer programs that we now take a closer look at. We investigate problems on integer constraint matrices that consist of a totally unimodular matrix with one additional row. While this class of integer programs may sound very restricted, we can model some interesting problems in this way. We mention the partially ordered knapsack problem, see Problem 4.23, which is strongly NP-complete and has been studied by Kolliopoulos and Steiner [70] with regards to approximations and an FPTAS for a special case, as well as Boyd [23] with regards to its polyhedral structure. In addition, the exact matching problem, see Problem 4.20 has gained a lot of recent attention, and its hardness is unresolved, even for the bipartite case which is relevant for us. Jia, Svensson, and Yuan [59] very recently showed that the corresponding polytope has exponential extension complexity. On the positive side, there is a randomized algorithm due to Camerini, Galbiati, and Maffioli [24], as well as exact polynomial time algorithms for the special cases of planar graphs (Yuster [106]) and dense graphs (Gurjar, Korwar, Messner, and Thierauf [51], as well as El Maalouly and Steiner [76]).

As mentioned above, there is a decomposition theorem of totally unimodular matrices due to Seymour [92], see Theorem 4.16, describing how they can be built from (transposed) network matrices. We partially answer Question 4.2 for some restrictions to the case of transposed network matrices, see Theorem 4.25 and Theorem 4.34. By considering these base blocks, we obtain a direct interpretation of the respective problems in terms of graphs. This allows us to make use of celebrated results from graph and minor theory to solve our problem. In the following, we give a short overview over the most relevant techniques and theory that relate to our problem.

We assume that the reader is familiar with the standard notions of graph theory and refer to Diestel [38] for an extensive reference. Let us assume for the context of the introduction that our graphs are simple and undirected unless explicitly denoted otherwise. One fundamental containment relation that we will be working with in Section 4.5 is the concept of minors. We say for two graphs G and H that H is a *minor* of G if it is isomorphic to some graph that can be obtained from G by deleting and contracting edges. We call a class of graphs $\mathcal{G}$ *minor-closed* if $\mathcal{G}$ is closed under taking minors, i.e., for H a minor of G and $G \in \mathcal{G}$ we have $H \in \mathcal{G}$.

One of the arguably most famous minor-closed graph classes are planar graphs, i.e., graphs that can be drawn on the surface of the plane without any pair of its edges intersecting. We have a nice characterization of planar graphs in terms of minors due to Kuratowski [73], and Wagner [102]. Essentially, a graph can be drawn in the plane if and only if it doesn't contain K_5 or $K_{3,3}$ as a minor. We mention that there is an extension to graphs that can be embedded on a surface of bounded genus that pertain many structural advantages. In particular, embedded graphs come with an orientation which can be exploited in order to gain additional structural knowledge, see for instance Section 4.5.3. In addition, the 4-color theorem (see [38, Theorem 5.1.1]) gives a tight upper bound on the chromatic number of planar graphs. Together with the forbidden minor characterization, this result implies one of the base cases of Hadwiger's conjecture [54], one of the most important open questions in graph theory, see Seymour [91] for a survey. The conjecture suggests that if G does not contain K_{t+1} as a minor, then G is t-colorable.

The concept of graph classes that are closed under minors is particularly interesting due to a theorem of Robertson and Seymour [88], resolving a conjecture attributed to Wagner. They prove that any class of graphs that is closed under taking minors can be characterized by a finite set of forbidden minors. The work of Robertson and Seymour also implies an algorithm with cubic running time for recognizing whether a graph contains a fixed graph H as a minor. The running time was later improved to quadratic by Kawarabayashi, Kobayashi, and Reed [64]. This opens the door for fixed parameter tractable algorithms for testing properties of graphs that persist under minor taking, i.e., that induce a minor-closed class. Note that while the mentioned work proves that such FPT algorithms exist for a large number of problems, finding the corresponding minors, and consequently the algorithms is a problem in itself, as the proof by Robertson and Seymour is not constructive.

One important part of the minor testing algorithm (in fact the bottleneck of the runtime) is an algorithm for the k disjoint rooted path problem (k-DRP), see Problem 4.3.

Problem 4.3. An instance of k-DRP $\mathcal{A}$ is described by a graph $G = (V, E)$ and a set of $k \in \mathbb{N}$ pairs of vertices $(s_1, t_1), \dots, (s_k, t_k)$. $\mathcal{A}$ is a *YES*-instance if there is node-disjoint $s_i - t_i$-paths in G for all $i \in [k]$.

The idea of the algorithm to solve k-DRP that was given by Robertson and Seymour [87] can be roughly sketched as follows. If the *treewidth* of G is small, the problem can be efficiently solved. Otherwise it can be shown that G contains a large clique or grid minor. Now, either the connectivity between the terminals and a large clique minor is sufficiently high. Then, the clique can be used to reroute the path. In all other cases, the authors efficiently find an *irrelevant* vertex, i.e., a vertex whose removal does not change the answer of the problem. This strategy can be recursed, until the question is answered or low tree-width is reached.

This brings us to tree decompositions and tree-width, which are the last concept that we want to discuss within the scope of this introduction. A *tree decomposition* of some graph G can be described by some tree T and a map that assigns vertices of G to subtrees of T. For any edge $\{v, w\} \in E(G)$, the intersection of the images of v and w cannot be empty. Roughly speaking, a tree decomposition relates an arbitrary graph to a tree, while pertaining its connectivity information. The *tree-width* of G is defined as the minimum over all such maps of the maximum size of the preimage of a vertex of T (minus 1, for normalization purposes). Tree decompositions are a helpful base for dynamic programming algorithms, see for instance Bodlaender [18]. This is due to the fact that a tree can be traversed without the risk of getting unwanted dependencies by circuits, while vertices that are connected through an edge in the original graph still appear in the same tree-vertex at least once.

4.1.1 Organization

In Section 4.2, we describe general results on integer programs that we ask about in Question 4.2. This includes a characterization of the constraint matrices we consider, and a strengthening of the proximity bound by Cook et al. [31] for our particular type of problem together with a decomposition of the solution vector. In the following Section 4.3 we present Seymour's decomposition of TU matrices [92], to motivate the study of network matrices and their transposes. Here, we translate the previous general results to these base blocks. We study the particular case of transposed incidence matrices in Section 4.4 and answer Question 4.2 in the affirmative for this subproblem. Section 4.5 deals with the more general case of transposed network matrices and uses graph and minor theory in order to solve a further subcase of Question 4.2. Finally we conclude this chapter in Section 4.6 with open questions and directions for further research.

4.2 Results for the general case

We start by formally defining total Δ-modularity, and then establish Lemma 4.8, which characterizes total Δ-modularity for matrices that become totally unimodular after removing one row.

Definition 4.4. An integer matrix $A \in \mathbb{Z}^{m,n}$ is said to be *totally* Δ*-modular* for some fixed integer $\Delta > 0$ if all its square submatrices A' satisfy $-\Delta \leq \det(A') \leq \Delta$. The *maximum subdeterminant* of A is the maximum of $|\det(A')|$ over all square submatrices A' of A. Consequently, A is totally Δ-modular if and only if its maximum subdeterminant is at most Δ.

Let us assume in the following that A is totally Δ-modular, and has a specific row whose removal makes it totally unimodular, i.e.,

$$A = \begin{pmatrix} M \\ d^\intercal \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} b_M \\ W \end{pmatrix},$$

for M totally unimodular, b_M, d appropriately sized integer vectors, and some integer W. We claim that we can assume without loss of generality that A contains a $n \times n$ identity matrix, denoted by I_n, as well as its opposite. We consider the IP $\max\{p^\intercal x \mid Ax \leq b, x \text{ integer}\}$, for an appropriately sized integer vector p, and $b^\intercal := (b_M^\intercal, W)$. We also define $P := \{x \in \mathbb{R}^n \mid Mx \leq b_M\}$. This is an integral polyhedron, since M is a TU matrix and b_M is integer.

We can make sure that the IP is feasible by minimizing $d^\intercal x$ on P and check that there is at least one point $x \in P$ with $d^\intercal x \leq W$, and otherwise report infeasibility. We then solve the LP relaxation $\max\{p^\intercal x \mid Ax \leq b\}$. If the LP relaxation turns out to be unbounded, then the IP itself is also unbounded and we report this. Else, we find an optimal solution x^* to the LP. By the proximity result of Cook, Gerards, Schrijver, and Tardos [31] (see Theorem 4.5 below), we know that there exists an optimal solution to the IP within ℓ_∞-distance at most Δn of x^*. Hence, we may add integral upper and lower bounds on each x-variable without changing the optimum value of the IP. Adding the corresponding rows to $Mx \leq b_M$, we keep the property that M is a TU matrix and that A is totally Δ-modular. In fact, this corresponds to adding to A the $n \times n$ identity matrix and its opposite.

Theorem 4.5. *[Cook et al. [31], Thm. 1.]*
Let $A \in \mathbb{Z}^{m,n}$ be a totally Δ-modular matrix and let b and p be vectors such that $Ax \leq b$ has an integral solution and $\max\{p^\intercal x \mid Ax \leq b\}$ exists. Then for each optimal solution x^* to $\max\{p^\intercal x \mid Ax \leq b\}$, there exists an optimal solution $\bar{x}$ to $\max\{p^\intercal x \mid Ax \leq b,\ x \in \mathbb{Z}^n\}$ with $||x^* - \bar{x}||_\infty \leq n\Delta$.

We fix the following notation for the remainder of the chapter: Let $A \in \mathbb{Z}^{m+1,n}$ be a totally Δ-modular matrix of the form

$$A = \begin{pmatrix} M' \\ d^\intercal \end{pmatrix}, \text{ where } M' = \begin{pmatrix} I_n \\ -I_n \\ M \end{pmatrix}$$

for some totally unimodular matrix M. Note that $M' \in \{-1,0,1\}^{m,n}$ is also totally unimodular. Ultimately, we want to solve the IP

$$\begin{array}{rl} \max & p^\intercal x \\ \text{s.t.} & Ax \leq b \\ & x \in \mathbb{Z}^n \end{array} . \tag{4.1}$$

4.2.1 Characterization

The theory of both total unimodularity and total Δ-modularity is closely coupled with matroid and graph minor theory, see e.g. [90]. For an extensive introduction into matroid theory, see [81]. In this work, we focus our attention on graphic matroids, which naturally occur from Seymour's decomposition of TU matrices [92], and matroids represented by a given matrix. We say that a matroid $(E, \mathcal{I})$ is represented by a matrix M, if the elements E correspond to the columns of M, and a set of columns is in $\mathcal{I}$ if and only if the corresponding vectors are linearly independent. Since all matroids that we consider arise from totally unimodular matrices, they are regular and can thus be represented over any field, see [81]. Recall that a circuit of a matroid is defined as a minimal dependent set, i.e., a subset of the elements $C \subseteq E$ is a circuit if and only if $C \notin \mathcal{I}$ and $C \setminus x \in \mathcal{I}$ for all $x \in C$.

We define a *circuit vector* of M' as a signed incidence vector of a circuit of the matroid represented by (M', I_m), restricted to the column space of M'. To be more precise, we say that $c \in \{-1,0,1\}^n$ is a circuit vector of M' if and only if there is a vector $c' \in \{-1,0,1\}^{n+m}$ with $c'_{[n]} = c$ such that $(M'\, I_m)c' = 0$ and the support of c' is a circuit C of the matroid represented by $(M'\, I_m)$. In this case, we say that c induces C.

Lemma 4.6. *Every circuit of the matroid represented by $(M'\, I_m)$ induces a unique (up to sign) circuit vector of M'.*

Proof. Consider some circuit $C \subseteq [n+m]$ of the matroid represented by $(M'\, I_m)$. Denote the corresponding columns of $(M'\, I_m)$ by c_i for $i \in C$. Since the columns are linearly dependent and have rational entries, we can find integer coefficients $\alpha_i \in \mathbb{Z}$ for $i \in C$, such that they sum up to the m-dimensional zero-vector, i.e., $\sum_{i \in C} \alpha_i c_i = 0_m$. Assume without loss of generality that $\gcd\{\alpha_i : i \in C\} = 1$. Clearly, then also $\sum_{i\in C}(\alpha_i \mod 2)c_i \equiv 0_m \mod 2$. Thus, since all entries in $(M'\, I_m)$ are in $\{-1,0,1\}$, we obtain a set of columns, such that each row-index has an even number of non-zero entries.

The characterization of TU-matrices given by Ghouila-Houri [49], see [90, Thm. 19.3 (iv)] gives a signing of the corresponding columns such that they sum up to 0. Since C is inclusion-minimally dependent, this signing is unique. □

The following lemma shows that we can alternatively define circuit vectors as the kernel of row-submatrices of M' of rank $n-1$.

Lemma 4.7. *Let $c \in \{-1,0,1\}^n$ be a non-zero vector. Then, c is a circuit vector of M' if and only if $r_i^\intercal c = 0$ for $n-1$ linearly independent rows $r_1, \dots, r_{n-1}$ of M'.*

Proof. Assume first that $r_1^\intercal c = \cdots = r_{n-1}^\intercal c = 0$, where $r_1, \dots, r_{n-1}$ are linearly independent rows of M'. Due to a similar argument as in the proof of Lemma 4.6, this implies that in $\operatorname{supp}(c)$, each of the $n-1$ rows has an even number of non-zero entries. Since M' is totally unimodular, we can apply the characterization of TU-matrices given by Ghouila-Houri [49], see [90, Thm. 19.3 (iv)]. Applied to the support of c, this gives us some vector $v \in \{-1,0,1\}^n$ with $\operatorname{supp}(c) = \operatorname{supp}(v)$ and $M'v \in \{-1,0,1\}^m$. In particular, since the number of non-zero entries of the rows in r_i with respect to the columns of $\operatorname{supp}(v)$ is even, it holds that $r_i^\intercal v = 0$ for $i \in [n-1]$. In addition, since the r_i are linearly independent for $i \in [n-1]$, the submatrix of M' obtained by choosing the corresponding rows has rank $n-1$, and therefore its kernel is 1-dimensional. Therefore, either $v = c$, or $v = -c$.

It remains to show that the support of c' is a circuit, i.e., that it is minimally dependent. Therefore, assume that our choice of support on I_m is already minimal. In order to find a smaller dependent set, the support on M' needs to differ, yielding a new solution $\tilde{c}'$ to $(M'\, I_m)x = 0$. When restricted to the columns of M', we obtain a new solution $\tilde{c}$. This solution also satisfies $r_i^\intercal \tilde{c} = 0$ for $i \in [n-1]$, contradicting the fact that the kernel of the associated matrix is 1-dimensional.

Now, take any circuit vector $c \in \{-1,0,1\}^n$ and its extension $c' \in \{-1,0,1\}^{n+m}$. Since M' has full column rank, the support of $c'_{[n+1,n+m]}$ is non-empty. For each $j \in [m]$ with $c'_{n+j} \neq 0$, remove the corresponding row from M'. Denote the resulting submatrix of M' by M''. Due to Lemma 4.6, we know that $M''c = 0$. It remains to show that $\operatorname{rank}(M'') = n-1$. Any minimally dependent set of columns of M'' cannot contain an index outside of $\operatorname{supp}(c)$, because of the identity matrices on top of M'. If there is a dependent set of columns of M'' that is a proper subset of $\operatorname{supp}(c)$, this induces a smaller dependent set in M', contradicting the fact that c is a circuit vector. Therefore, $\operatorname{supp}(c)$ is the unique circuit in the matroid represented by M'', showing that $\operatorname{rank}(M'') = n-1$. □

We are ready to give a full characterization of the subdeterminants of A. Note that we are interested in subdeterminants of maximum absolute value only. Therefore, we can assume that the submatrices contain the last row, since the rest of the matrix is totally unimodular and its subdeterminants are bounded by 1 in absolute value.

Lemma 4.8. *The maximum absolute value of a subdeterminant of A (containing the row $d^\intercal$) is equal to the maximum absolute value of $d^\intercal c$, where c is a circuit vector of M'.*

Proof. Consider a $(k \times k)$ submatrix $\tilde{A}$ of A containing the last row. The $(k-1 \times k)$ submatrix of $\tilde{A}$ obtained by removing the last row is totally unimodular and has rank at most $k-1$; if the rank is less than $k-1$, then $\det(\tilde{A}) = 0$, hence we can assume that the rank is exactly $k-1$. Hence the $(k-1 \times k)$ submatrix contains a unique minimal set of linearly dependent columns. This can be extended to a circuit C of $(M'\, I_m)$, by adding columns of I_m. Note that if there is a dependent subset of C, this would induce a second solution in the set of linearly dependent columns in $\tilde{A}$. Now, consider vector c given by Lemma 4.6, and in $\tilde{A}$ replace one of the columns in C with $\tilde{A}c_{\tilde{A}}$, i.e., with the signed sum of columns in C, where the sign is given by the circuit vector c. Note that this does not change the absolute value of the determinant of $\tilde{A}$.

We claim that the column we obtained is 0 everywhere but in the last component, where it is equal to $d^\intercal c$. For the second part of the claim, observe that the extension to C only uses elements (columns) of I_m. For the first part of the claim, observe that the set of columns in C is minimally dependent in $\tilde{A}$. Since the extension to a circuit of the matroid only happens on the columns of I_m, the sum for each row of $\tilde{A}$ must be in $\{-1, 0, 1\}$. Following the arguments of Lemma 4.6, we can see that each row has an even number of non-zero entries with respect to the circuit columns of $\tilde{A}$, implying that they are already 0. Laplace expansion on this new column shows that $|\det(\tilde{A})| = |d^\intercal c|$.

On the other hand, consider a circuit vector c of M' and let $k = |\mathrm{supp}(c)|$. Thanks to the minimality of the circuit, the rank of M'_C is $k-1$. Let $\tilde{A}$ be the $(k \times k)$ submatrix of A containing $k-1$ linearly independent rows of M'_C and the last row $d^\intercal$. With the same column operation as before, we see that the determinant of $\tilde{A}$ is equal to $d^\intercal c$ in absolute value. $\square$

Generally speaking, such characterization results of total Δ-modularity are interesting for a larger class of problems than we discuss here. By giving additional properties on the possible problem structure, they are the first step to a potential polynomial algorithm solving the corresponding optimization problems. Such results are for instance known for the stable set problem, where a connection to the odd-cycle packing number can be shown and has been exploited [17, 44].

We remark that our results for matrices that become totally unimodular after removing one row can be extended to $k \in \mathbb{N}$ rows in a meaningful way. Similar to the concept of circuit vectors, maximum subdeterminants arise from submatroids of the matroid represented by $(M'\, I_m)$ that can be obtained by deleting a number of elements (columns) in order to obtain a certain relationship between the number of elements in the submatroid, and its rank. In this submatroid, the object of interest is the circuit space, i.e., the linear subspace that can be generated by the incidence vectors of all circuits (over $\mathbb{Z}_2$). If we choose $k' \leq k$ of the additional rows in a subdeterminant, we can show that the dimension of a relevant corresponding circuit space is k'. We create an auxiliary

$k' \times k'$ matrix M'', whose rows correspond to the k' additional rows and whose columns correspond to a basis of the circuit space. Now, the subdeterminant is equal to the determinant of M'', where each entry corresponds to the dot product of the respective additional row and the element of the circuit base.

4.2.2 Proximity and solution decomposition

In this section, we prove a strong decomposition and proximity result for totally Δ-modular IPs on TU matrices with one additional row. We strengthen the totally Δ-modular proximity result by Cook et al. [31] for our particular case of a TU matrix with one additional row to a distance (in ∞-norm) of at most $O(\Delta)$. Such a proximity result alone reduces the search space for an optimal solution to be in $\Delta^{O(n)}$, which is still too large to efficiently search. The additional decomposition into circuit vectors provides a potential path towards an efficient algorithm by understanding their structure. Note that while we follow along the notation of the previous section, we do not explicitly use that A contains the n-dimensional identity matrix as well as its negative. The following results hold for any totally Δ-modular rank-n matrix which is totally unimodular after removing one row. Note that as before, we define $P := \{x \in \mathbb{R}^n \mid M'x \leq b_{M'}\}$ the (integral) polytope of the totally unimodular part of our problem.

Lemma 4.9. *Each one-dimensional face of P is generated by a circuit-vector $r \in \{-1, 0, 1\}^n$. In particular, if a one-dimensional face is generated by $r \in \{-1, 0, 1\}^n$, then $|d^\intercal r| \leq \Delta$.*

Proof. Any one-dimensional face of P is determined by $n-1$ linearly independent inequalities of $M'x \leq b_{M'}$ which are tight, i.e., it is generated by a circuit vector, see Lemma 4.7. Lemma 4.8 shows that the scalar product of a circuit vector with d is bounded by Δ in absolute value. □

Lemma 4.10. *[Nägele, Santiago, and Zenklusen [79], Lemma 29]*
Let $T \in \{-1, 0, 1\}^{m,n}$ be a totally unimodular matrix such that the cone $C := \{x \in \mathbb{R}^n : Tx \leq 0\}$ is pointed, and let $y \in C \cap \mathbb{Z}^n$. Then one can determine in strongly polynomial time integer extremal rays $y_1, \ldots, y_k \in \mathbb{Z}^n$ of C and coefficients $\lambda_1, \ldots, \lambda_k \in \mathbb{Z}_{\geq 0}$ such that $y = \sum_{i=1}^k \lambda_i y_i$.

As before, we assume that P is an integral polytope containing at least one integer point $x \in \mathbb{Z}^n$ with $d^\intercal x \leq W$. Let x^* denote an extreme point of $\{x \in P : d^\intercal x \leq W\}$ maximizing $p^\intercal x$, i.e., an optimal solution to the linear relaxation of the integer program (4.1). If x^* is integral, we are done. Else, x^* is the intersection of a one-dimensional face F of P and the hyperplane $H := \{x \in \mathbb{R}^n : d^\intercal x = W\}$.

Furthermore let $x^0 \in F$ with $d^\intercal x^0 < W$ be the closest feasible integer point on F to x^*. Lemma 4.9 shows that $W - d^\intercal x^0 < \Delta$. Now consider some optimal solution x^{opt} to the integer program (4.1). We can of course write $x^{\text{opt}} = x^0 + x^{\text{diff}}$ for some integer difference vector. Let $K \subseteq \mathbb{R}^n$ be the cone defined by

$$K := \left\{ x \in \mathbb{R}^n : \begin{array}{ll} M'_i x \leq 0 & \text{if } M'_i x^{\text{diff}} < 0 \\ -M'_i x \leq 0 & \text{if } M'_i x^{\text{diff}} \geq 0 \end{array} \right\},$$

where M'_i denotes the ith row of M'. By definition, it is clear that $x^{\text{diff}} \in K$. Since M' has full rank, K is pointed. As the constraint matrix of K can be obtained from M' by multiplying some rows with -1, it is totally unimodular. Therefore, Lemma 4.10 gives a decomposition of x^{diff} into integer extremal rays of K with integer coefficients. By Lemma 4.9, the vectors in the decomposition have bounded weight and correspond to circuit vectors. Create a multiset of vectors $\mathcal{S} := \{\lambda_i \times y_i : i \in [k]\}$, where λ_i denotes the multiplicity of the vector y_i in $\mathcal{S}$. Any submultiset $\mathcal{S}' \subseteq \mathcal{S}$ induces a point in $x' \in P$ by $x' = x^0 + \sum_{S \in \mathcal{S}'} S$, since all linear constraints of P are maintained.

Lemma 4.11. *Let $\mathcal{W} = \{w_1, w_2, \ldots, w_k\}$ denote a multiset of integers, all between $-\Delta$ and Δ such that $|\sum_{i=1}^k w_i| \le \Delta$, and $k \ge 2\Delta + 2$. Then there exists a nonempty $I \subseteq [k]$ with $|I| \le 2\Delta + 1$ such that $\sum_{i \in I} w_i = 0$.*

Proof. Clearly we can assume that $0 \notin \mathcal{W}$, otherwise we are done. Order the set $\mathcal{W}$ by starting from an element of largest value and applying iteratively the following rule: if the sum of the elements listed so far is positive (resp. non-positive), look for a negative (resp. positive) element in $\mathcal{W}$ and set it to be the next one in the list. If at some point we cannot find an element of the desired sign, say if the current sum is positive and there is no negative element left in $\mathcal{W}$, since the sum of all elements of $\mathcal{W}$ is at most Δ in absolute value we must have only a bounded number of elements left, which we add in arbitrary order to our list. Notice that proceeding this way all partial sums lie between $-\Delta$ and Δ, i.e., there exists a permutation $\pi : [k] \to [k]$ such that $|\sum_{i=1}^{\ell} w_{\pi(i)}| \le \Delta$ for all $\ell \in [k]$.

Now, by the pigeon hole principle since $|\mathcal{W}| > 2\Delta + 1$ there must be two values $\ell_1 < \ell_2 \in [k]$ with $\ell_2 \le \ell_1 + 2\Delta + 1$ such that $\sum_{i=1}^{\ell_1} w_{\pi(i)} = \sum_{i=1}^{\ell_2} w_{\pi(i)}$, hence the multiset $\{w_{\pi(\ell_1+1)}, \ldots, w_{\pi(\ell_2)}\}$ is the desired submultiset. □

Lemma 4.12. *There is an optimal solution x^{opt} of IP* (4.1) *such that $x^{\text{diff}} = \sum_{i=1}^k \lambda_i y_i$ as in Lemma 4.10, and also $\sum_{i=1}^k \lambda_i \in O(\Delta)$.*

Proof. Let x^0 be the rounded solution to the linear relaxation of (4.1) as defined before, and x^{opt} be an optimal solution of the integer program (4.1) such that $\sum_{i=1}^k \lambda_i$ given by Lemma 4.10 is as small as possible. Again, $x^{\text{diff}} := x^{\text{opt}} - x^0$. As before, consider $\mathcal{S} := \{\lambda_i \times y_i : i \in [k]\}$. Consider the weight $d^\intercal S$ of every $S \in \mathcal{S}$: each is an integer between $-\Delta$ and Δ. We claim that $|\sum_{S \in \mathcal{S}} d^\intercal S| \le \Delta$.

We have $\sum_{S \in \mathcal{S}} d^\intercal S \le \Delta$, since otherwise $W \ge d^\intercal x^{\text{opt}} = d^\intercal x^0 + d^\intercal x^{\text{diff}} = d^\intercal x^0 + \sum_{S \in \mathcal{S}} d^\intercal S > d^\intercal x^0 + \Delta$ and this contradicts the fact that $d^\intercal x^0 \ge W - \Delta$.

We now show that $\sum_{S \in \mathcal{S}} d^\intercal S \ge -\Delta$. Toward a contradiction, assume that $\sum_{S \in \mathcal{S}} d^\intercal S < -\Delta$. Hence, $d^\intercal x^{\text{opt}} < W - \Delta$. We inspect the vectors S one by one. For every $S \in \mathcal{S}$, $x^0 + \sum_{S' \in \mathcal{S} \setminus \{S\}} S' = x^{\text{opt}} - S$ is a feasible IP solution by Lemma 4.10 and $d^\intercal (x^{\text{opt}} - S) = d^\intercal x^{\text{opt}} - d^\intercal S < W - \Delta + \Delta = W$. This in turn implies that $p^\intercal S > 0$ for all $S \in \mathcal{S}$, since otherwise the optimality of x^{opt} would be contradicted. If there exists some $S \in \mathcal{S}$ such that $d^\intercal S \le 0$ then $x^* + \varepsilon S$ is feasible for the LP relaxation for some $\varepsilon > 0$ small enough (due to convexity of P), which contradicts the optimality of x^*. We conclude that $d^\intercal S > 0$ for all j, which contradicts our assumption $\sum_{S\in\mathcal{S}} d^\intercal S < -\Delta$. This concludes the proof of the claim.

Now we can apply Lemma 4.11 to the multiset of integers $\mathcal{W} := \{d^\intercal S : S \in \mathcal{S}\}$. If $|\mathcal{W}| \geq 2\Delta + 2$, then we can find a proper nonempty submultiset $\mathcal{S}'$ whose weight is 0. By Lemma 4.10, we know that $x^0 + \sum_{S\in\mathcal{S}\setminus\mathcal{S}'} S$ is a feasible solution to the IP, and in addition on the same translate of H as x^{opt}. By minimality of x^{opt} and $\mathcal{S}$, this implies $\sum_{S\in\mathcal{S}'} p^\intercal S > 0$. This again contradicts the optimality of x^*, since $x^* + \varepsilon \sum_{S\in\mathcal{S}\setminus\mathcal{S}'} S$ is feasible for the LP relaxation for $\varepsilon > 0$ small enough, but has profit more than that of x^*. □

4.3 Seymour's decomposition

Seymour's decomposition of totally unimodular matrices [92] gives a way to decompose such matrices into smaller blocks that can be understood better in terms of graph theory. It is a technique often employed for questions revolving around totally unimodular matrices, e.g., by Artmann, Weismantel, and Zenklusen [8], or Aprile and Fiorini [4]. Instead of dealing with an arbitrary totally unimodular matrix, this approach allows us to work with network matrices, and their transposes, which have a graph representation. These matrices are combined by certain block-like operations of low rank. We give a short introduction into the relevant objects that will be used in this chapter. For a more detailed reference, see Artmann [7, Section 1.5] or Schrijver [90, Section 19.4], on which the following part is based. Note that we do not solve the whole totally unimodular case in this thesis, but use Seymour's decomposition theorem as a motivation to work on the base blocks and block-like combination operations.

Any *network matrix* can be represented by a directed tree $T = (V, A_T)$ and a directed graph $D = (V, A_D)$ on the same set of vertices. Note that T is defined to be a spanning tree (in the undirected sense) on the set of vertices V. The network matrix represented by T and D is defined by the fundamental cycles that the arcs of D close in T.

Definition 4.13. Let T and D be given as before, and consider $M \in \{-1,0,1\}^{A_T,A_D}$. We define the corresponding network matrix as

$$M_{i,j} := \begin{cases} 1 & \text{if the fundamental cycle defined by } j \text{ uses } i \text{ in backward direction} \\ -1 & \text{if the fundamental cycle defined by } j \text{ uses } i \text{ in forward direction} \\ 0 & \text{otherwise} \end{cases},$$

for $i \in A_T$ and $j \in A_D$.

Note that while T and D uniquely determine a network matrix (up to exchanging the order of rows or columns), a given network matrix can have different graph representations. We define a simple undirected graph G from a network matrix M as $G = (V, E)$, where E is the set of edges obtained by undirecting each arc in A_T and A_D.

Remark 4.14. Network matrices generalize both arc-node incidence matrices of directed graphs and edge-node incidence matrices of bipartite undirected graphs. This can be seen by adding an auxiliary vertex v_0 to a given graph and defining the tree T as a star with center v_0. As such, network matrices play a role in many important combinatorial optimization problems, see for instance Problem 4.20 and Problem 4.23.

Together with $M_1, M_2 \in \{-1,0,1\}^{5,5}$ in (4.2), network matrices form the building blocks of totally unimodular matrices, see Theorem 4.16 below. This motivates the study of network matrices and their transposes, and what total Δ-modularity means for them in our setting. First we need to define, with which operations it is possible to combine said blocks.

$$M_1 = \begin{pmatrix} 1 & -1 & 0 & 0 & -1 \\ -1 & 1 & -1 & 0 & 0 \\ 0 & -1 & 1 & -1 & 0 \\ 0 & 0 & -1 & 1 & -1 \\ -1 & 0 & 0 & -1 & 1 \end{pmatrix} \quad M_2 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{4.2}$$

Definition 4.15. Consider the following TU-preserving combination operations on matrices and vectors:

$$\text{replace} \begin{pmatrix} \varepsilon & c^\intercal \\ b & D \end{pmatrix} \text{ by } \begin{pmatrix} -\varepsilon & \varepsilon c^\intercal \\ \varepsilon b & D - \varepsilon b c^\intercal \end{pmatrix} \tag{pivot}$$

$$A \oplus_1 B := \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix} \tag{1-sum}$$

$$\begin{pmatrix} A & a \end{pmatrix} \oplus_2 \begin{pmatrix} b^\intercal \\ B \end{pmatrix} := \begin{pmatrix} A & ab^\intercal \\ 0 & B \end{pmatrix} \tag{2-sum}$$

$$\begin{pmatrix} A & a & a \\ c^\intercal & 0 & 1 \end{pmatrix} \oplus_3 \begin{pmatrix} 1 & 0 & b^\intercal \\ d & d & B \end{pmatrix} := \begin{pmatrix} A & ab^\intercal \\ dc^\intercal & B \end{pmatrix}, \tag{3-sum}$$

for $\varepsilon \in \{-1,1\}$, a, b, c, d appropriately sized column vectors and A, B appropriately sized matrices, such that each matrix on the left side is totally unimodular.

Theorem 4.16. *[Seymour [92], (14.3), see also Schrijver [90], Thm. 19.6]*
Let A be totally unimodular. Then, one of the following applies:

- *A is a network matrix*
- *A is one of M_1 and M_2*
- *there is a combination of* (1-sum), (2-sum) *and* (3-sum) *that decomposes A into smaller blocks*
- *we can apply a series of* simple *operations until one of the above applies:*
  - *use a* (pivot) *operation*
  - *delete a duplicate row or column*
  - *permute rows or columns*

- *delete a row or column with at most one non-zero entry*
- *change the sign of a row or column*
- *replace A with $A^\intercal$*

4.3.1 k-sums

Seymour's decomposition uses k-sums of matrices (or matroids, originally) as means to compose arbitrary totally unimodular matrices from simpler base blocks. This motivates the study of these operations: How can this additional structure be benificial to us, and how can we deal with these k-sums within an optimization framework? Lemma 4.17 answers the first question in terms of (transposed) network matrices, by relating k-sums of the matrix to connectivity properties of the corresponding graphic matroid and graph respectively. Note that in the context of this work, we use the term of connectivity to denote vertex-connectivity only. Lemma 4.18 builds up on this by showing that we can assume some basic connectivity properties on M', i.e., we can deal with 1-sums and reduce to the case where the corresponding matroid $(M' \, I_m)$ is 2-connected. Note that this proof relies on the fact that the blocks given by a 1-sum are independent except for the additional constraint. Therefore, it cannot be immediately generalized to 2- and 3-sums.

Lemma 4.17. *Let M be a network matrix and consider the simple undirected graph G obtained from M in the same way as above.*

If M cannot be decomposed by means of a k-sum for k in $\{1, 2, 3\}$ after a series of the operations in Theorem 4.16, then G is $k+1$-vertex-connected.

Note that the simple operations involve transposing M, so the same statement holds for $M^\intercal$.

Proof. Consider the matroid represented by $(M \, I_m)$. Adding the identity matrix as columns to M corresponds to adding an element to the matroid for each arc in A_T. Columns of this matrix are linearly dependent if and only if the corresponding edges contain a cycle in G. Therefore, the matroid represented by $(M \, I_m)$ is the graphic matroid of G.

For these matroids, the claimed connectivity properties are well-known, see e.g. [100, Theorem. 3.2.25, Corollary 3.2.29, Lemma 8.2.6, Lemma 8.3.12]. □

Lemma 4.18. *Let $\mathcal{A}$ be an algorithm that solves* (4.1) *if the totally unimodular part cannot be decomposed by means of a* (1-sum)*.*

Then there is a dynamic programming algorithm invoking $\mathcal{A}$ at most $O(\Delta^4 n^3)$ times and with polynomial overhead that solves (4.1)*.*

Proof. Assume M' decomposes via 1-sums into k blocks $M'_1, \dots, M'_k$, such that the blocks M'_i are associated with disjoint sets of rows $m_i \subseteq [m]$ and columns $n_i \subseteq [n]$. We solve the LP relaxation of (4.1) and denote the optimal solution by x^*. As in Section 4.2.2, we round x^* to a feasible integer solution x^0. We associate a weight with each block by defining $W_i = d_{n_i}^\intercal x_{n_i}^0$ for all $i \in [k]$. By Lemma 4.12, we know that there exists some optimal solution to (4.1), denoted by x^{opt}, such that $\|x^0 - x^{\text{opt}}\|_\infty \leq 2\Delta$. Since $\|d\|_\infty \leq \Delta$, this implies that $|d_{n_i}^\intercal x_{n_i}^{\text{opt}} - W_i| \leq 2\Delta^2 |n_i|$ for each $i \in [k]$ and corresponding results for partial sums as well as the whole vector.

Define a modified problem via

$$\rho(\tilde{W}, i) := \max p_{n_i}^\intercal x_{n_i} \text{ s.t. } M_i' x_{n_i} \leq b_{m_i},\ d_{n_i}^\intercal x_{n_i} \leq \tilde{W},\ x_{n_i} \in \mathbb{Z}^{n_i}. \tag{4.3}$$

Note that $\mathcal{A}$ solves instances of the form (4.3). Define a dynamic programming table, whose columns represent the blocks for $i \in [k]$ and whose rows represent the accumulated weight difference to x^0 with respect to the additional constraint $d^\intercal x \leq W$. To be more precise, we define nodes $N_{\tilde{W},i}$ for $i \in [k]$ and $\tilde{W} \in \left[-2\Delta^2 \sum_{j=1}^{i} |n_j|, 2\Delta^2 \sum_{j=1}^{i} |n_j|\right]$, as well as an auxiliary node $N_{0,0}$. We connect two nodes $N_{W',i}, N_{W'',j}$ by an arc $(N_{W',i}, N_{W'',j})$ if $i + 1 = j$ and $|W' - W''| \leq 2\Delta^2 |n_j|$. Each arc is assigned a profit denoted by p, which is defined as $p(N_{W',i}, N_{W'',j}) := \rho(W'' - W', j)$.

Since our dynamic programming table corresponds to a directed acyclic graph with at most $2\Delta^2 n^2$ nodes and $4\Delta^4 n^3$ arcs, we can run a longest path algorithm from $N_{0,0}$ to $N_{W-\sum_{i=1}^k W_i, k}$. Observe that each $N_{0,0}$-$N_{W-\sum_{i=1}^k W_i, k}$-path in the graph corresponds to a feasible solution of (4.1). In addition, since we optimize over the blocks independently, given the right combination of weights, this algorithm performs at least as good as x^{opt} on each block certifying optimality of our solution.

The longest path problem in a directed acyclic graph can for instance be solved with the Bellman-Ford algorithm [14]. Its runtime is given by the number of nodes times the number of arcs, which is upper bounded by $O(\Delta^6 n^5)$. Together with solving the LP relaxation once, this gives a polynomial overhead. In addition, $\mathcal{A}$ is used to calculate the weight on the $O(\Delta^4 n^3)$ arcs. □

4.3.2 Network matrices

The main results of Section 4.2.1 and Section 4.2.2 motivate the further study of circuit vectors. In particular, it is interesting to further investigate the combinatorial structure given by them. In Lemma 4.6, we characterize circuit vectors as the circuits of some related (regular) matroid. Matroids can be seen as a generalization of graphs that preserve a lot of their combinatorial structure. However, as we will see in Section 4.4 and Section 4.5, we can make use of structural graph theory results that help solve the integer program (4.1) in some special cases. These graph theoretical results do not trivially translate to matroids. Therefore, we repeat our general characterization result from Section 4.2.1 for the special cases of network matrices and their transposes respectively, which gives us clean results on the directed graph obtained by taking the union of T and D.

Lemma 4.19. *Let M be a network matrix inducing T and D, and consider $D' = (V, A_T \cup A_D)$.*

Then the set of supports of circuit vectors of M correspond exactly to the cycles in D' (in the undirected sense). Furthermore, the signs of a circuit vector give a direction on the arcs of the corresponding cycle, such that it is a directed cycle.

Proof. Recall the definition of a circuit vector as the circuits of the matroid represented by $(M\, I_m)$. Adding the identity matrix as columns to M corresponds to adding a parallel arc to A_D for each arc in A_T. This implies that the elements of the matroid represented by $(M\, I_m)$ correspond exactly to the arcs of D'. Elements of the matroid are dependent, if and only if the corresponding arcs contain a cycle. In addition, an orientation of a cycle gives a signing to the corresponding elements such that the columns sum up to 0, see [90, Section 19.3, Example 4, (32)]. Thus, the set of supports of circuit vectors of M corresponds exactly to the set of cycles in D', and the signing on the elements of a circuit vector gives an orientation on the cycles. □

In particular, the preceding lemma implies that if A can be described by a network matrix with one additional row $d^\intercal$, then we can view $d^\intercal$ as a weighting function on the arcs of D', with all weights of arcs in A_T being 0. By Lemma 4.8, A is totally Δ-modular, this restricts the weight of a cycle in the graph D' that represents the network matrix to at most Δ in absolute value.

An interesting problem that can be described as an integer program that is a network matrix with one additional row is the bipartite exact matching problem, see Problem 4.20 below. Note that the classical formulation of the perfect matching polytope is obtained from a constraint matrix that is the node-edge incidence matrix of the corresponding graph.

Problem 4.20. An instance $\mathcal{A}$ of the bipartite exact matching problem is described by a bipartite undirected graph $G = (V, E)$, a coloring function on its edges $c : E \to \{0, 1\}$ and an integer $k \in \mathbb{N}$. $\mathcal{A}$ is a *YES*-instance if there is a perfect matching $\mathcal{M} \subseteq E$ in G, such that $\sum_{e \in \mathcal{M}} c(e) = k$.

The exact matching problem asks for a matching fulfilling an exact weight constraint. While this would take 2 additional inequalities, which is not allowed in our setting, we can make use of the fact that the decision version of this problem is already interesting and unsolved (as opposed to an optimization version). Therefore, it suffices to model the upper bound of the weight as a constraint, and maximize the weight simultaneously. If we find a solution that reaches the upper bound, the instance is a *YES*-instance.

Corollary 4.21. *The constraint matrix for the bipartite exact matching problem formulated as an integer program as described above is totally Δ-modular for some fixed $\Delta \in \mathbb{N}$ if and only if every path $P = (e_1, \dots, e_k)$ in G has imbalance $I(P)$ at most Δ. The* imbalance *of a path is defined as*

$$I(P) := \left| \sum_{i \in [\lceil k/2 \rceil]} c(e_{2i-1}) - \sum_{i \in [\lfloor k/2 \rfloor]} c(e_{2i}) \right| .$$

Note that a node-edge incidence matrix of a bipartite graph can be interpreted as a network matrix by adding an apex v_a to the set of nodes, choosing the spanning tree as (v, v_a) for $v \in V$, and assigning appropriate directions to the edges. In particular, any path in the graph can be completed to a cycle through v_a.

4.3.3 Transposed network matrices

As network matrices, tranposed network matrices are important building blocks of totally unimodular matrices. We proceed to analyze these matrices in the same way as TU matrices and network matrices before.

Let $G = (V, E)$ be a graph (directed or undirected). For $S \subseteq V$, we define $\bar{S} := V \setminus S$.

Lemma 4.22. *Let M be a network matrix inducing T and D, and consider $D' = (V, A_T \cup A_D)$.*

Then the set of supports of circuit vectors of $M^\intercal$ induces minimal cuts. That is, removing the edges corresponding to the support of the circuit vector, we obtain two subgraphs that are weakly connected. Denote the vertices of the subgraphs by S and $\bar{S}$. Then, the signs of a circuit vector give a direction on the arcs of the corresponding minimal cut, such that all arcs either go from or to S.

Proof. The proof follows along the lines of the proof of Lemma 4.19. Adding the identity matrix as columns to M corresponds to subdividing each arc in A_D into an arc of A_T and the original arc by adding an auxiliary vertex. This implies that the elements of the matroid represented by $(M\, I_m)$ correspond exactly to the arcs of D' (the arcs in A_T obtained by subdivision correspond to the respective arc in A_D). Elements of the matroid are dependent if and only if the corresponding arcs contain a cut of the graph. In addition, orienting the arcs on a cut in the same direction gives a signing to the corresponding elements such that the columns sum up to 0. This can for instance be seen as a consequence of the proof of Lemma 4.6, where we show that circuits correspond to inclusion-minimal subsets of columns, such that each row-index has an even numbers of entries in these columns.

Thus, the set of supports of circuit vectors of M corresponds exactly to the set of minimal cuts, and the signing on the elements of a circuit vector gives a direction of the cut. $\square$

Similar as before, Lemma 4.22 implies that if A can be described by a transposed network matrix with one additional row $d^\intercal$, then we can view $d^\intercal$ as a weighting function on the arcs of D', with all weights of arcs in A_D being 0. By Lemma 4.8, A is totally Δ-modular, this restricts the weight of a minimum cut in the graph D' that represents the transposed network matrix to at most Δ in absolute value.

Here again, we describe a well-known combinatorial problem that falls into the described category of integer programs arising from a transposed network matrix with one additional row.

Problem 4.23. An instance $\mathcal{A}$ of the partially ordered knapsack problem is defined by a directed acyclic graph $G = (V, A)$, giving a precedence order on the elements V, a profit function $p : V \to \mathbb{N}$, a weight function $w : V \to \mathbb{N}$, and a maximum capacity $W \in \mathbb{N}$. We describe the optimization version of the problem. Therefore, we search for the subset $\mathcal{K} \subseteq V$ maximizing $\sum_{v \in \mathcal{K}} p(v)$ such that the weight constraint $\sum_{v \in \mathcal{K}} w(V) \leq W$ is fulfilled, and the precedence order is accounted for, i.e., for each arc $(v, w) \in A$, $w \in \mathcal{K}$ implies $v \in \mathcal{K}$.

The precedence constraints can be formulated by a transposed incidence matrix with the 0-vector as a right-hand side. The knapsack capacity constraint is modeled by the additional row. The following corollary characterizes total Δ-modularity for transposed incidence matrices, and therefore also for the partially ordered knapsack problem.

Corollary 4.24. *Consider a matrix M arising from a transposed incidence matrix of the graph $G = (V, A)$ with one additional row w, assigning weights to each vertex $v \in V$. Then, M is totally Δ-modular if every weakly connected subgraph H of G fulfills*

$$\left| \sum_{v \in V(H)} w(v) \right| \leq \Delta.$$

Note that a transposed incidence matrix of a directed graph can be interpreted as a network matrix by adding an apex v_a to the set of nodes, choosing the spanning tree as (v, v_a) for $v \in V$, and assigning appropriate directions to the edges. In particular, any edge-cut in the original graph extends to a minimal cut in the extended graph.

4.4 Transposed incidence matrices

Instead of the full transposed network matrix case, we start to analyze an important subcase, which are transposed incidence matrices with one additional row. Due to the direct interpretation of a transposed incidence matrix as one graph, and the clean characterization (see Corollary 4.24), this is a very instructive case. Note that as remarked above, this already includes the totally Δ-modular partially ordered knapsack problem. To be precise, we answer two questions in this section: Given an integer program of the form (4.1), where M is a transposed incidence matrix (possibly together with bounds on the variables), how can we check efficiently if it is indeed totally Δ-modular? And how can we solve it in strongly polynomial time? We consider the questions in Section 4.4.3 and Section 4.4.2 respectively. In particular, we show the following optimization results.

Theorem 4.25. *There exists a fixed parameter tractable algorithm in terms of Δ for solving integer programs of the form* (4.1) *where A is totally Δ-modular, and M is a transposed incidence matrix.*

Corollary 4.26. *There exists a fixed parameter tractable algorithm in terms of Δ for solving the partially ordered knapsack problem defined by $G = (V, A)$, with integer weights d and profits p if every weakly connected subgraph of G has weight at most Δ in absolute value.*

4.4.1 Structural results

We start by analyzing the meaning of total Δ-modularity for this subcase further. Therefore, let $G = (V, E)$ be a graph together with integer vertex weights $d \in \mathbb{Z}^V$. We denote the absolute value of the *maximum weight of a connected subgraph* of G by $\alpha(G, d)$.

Let $D = (V, A)$ be the (directed) graph representing M. Lemma 4.18 implies for transposed incidence matrices that we can assume D to be (weakly) connected. This can for instance be seen by interpreting the transposed incidence matrix as a transposed network matrix, where the tree is a star formed by an auxiliary apex. This apex is not part of D, which reduces 2-connectedness to connectedness for this setting. In addition, within the scope of this section we denote the graph that we obtain by undirecting the arcs of D by G. Note that for G, obtained by a transposed incidence matrix, together with a weight vector which is given by the additional row, we know that the maximal absolute value of a subdeterminant is given by $\alpha(G, d)$. Thus, we proceed to investigate the weighted graph given by G and d.

We first describe a subgraph contraction operation that essentially goes one step further than the classical graph contraction operation. Usually, contracting all edges within a connected subgraph corresponds to replacing said subgraph with a star, such that the center vertex corresponds to the contracted subgraph which has an edge to each neighbor of the subgraph. We extend this operation by also removing the center vertex and replacing it with a clique on its neighbors. More formally, this operation can be described in the following way.

Definition 4.27. Let $G = (V, E)$ be a graph and consider some vertex subset $U \subsetneq V$ inducing a connected subgraph of G. We define $G/^0U := (V \setminus U, E \cup E'' \setminus E')$, where $E' := \{\{v, w\} \in E \mid v \in U \text{ or } w \in U\}$ the edges incident to U and $E'' := \{\{v, w\} \mid v \in N(U), w \in N(U), v \neq w\}$ a clique on the neighborhood of U. We denote the corresponding operation as 0-*contraction*.

The 0-contractions behave well with respect to α when applied to connected subgraphs where each vertex has weight 0.

Lemma 4.28. *Let $U \subseteq V$ with $d(u) = 0$ for all $u \in U$ such that U induces a connected subgraph of G. Then, $\alpha(G, d) = \alpha(G/^0U, d)$.*

Proof. We prove the claim by showing that some induced subgraph H of $G/^0U$ is connected if and only if there is some subset U' of U, such that $V(H) \cup U'$ induces a connected subgraph of G.

First assume that H is connected and take any two vertices $v, w \in V(H)$. Consider a v, w-path in H. We replace any edge that was obtained by 0-contracting U by a corresponding path in U. This is possible since U induces a connected subgraph. Therefore, we obtain a v, w-walk in G using only vertices of $V(H)$ and U. We can use the same argument in the other direction to show that a path of vertices in $V(H) \cup U$ can be shortcutted using edges obtained from 0-contracting U to only use vertices of H. □

Denote the set of vertices of weight 0 with respect to d by $C := \{v \in V \mid d(v) = 0\}$. Further, we denote the connected components of G restricted to the vertices in C by

$C_1, \dots, C_k$. We define the 0-*contraction* of G by $G^0 := G/^0C_1/^0 \dots /^0C_k$. Note that $d(v) \neq 0$ for all $v \in V(G^0)$.

Corollary 4.29.

$$\alpha(G,d) = \alpha(G^0,d).$$

Proof. This follows directly from Lemma 4.28 □

Corollary 4.29 implies that the largest subdeterminants of A are in fact still represented in G^0. Since all weights in G^0 are non-zero, its structure is easier to analyze, as the following two lemmas show. When we talk about the degree of some vertex in $V(G^0)$, we always mean its degree in G^0 unless explicitly denoted otherwise.

Lemma 4.30.

$$\max_{v \in V(G^0)} \deg(v) \leq 2\alpha(G,d).$$

Proof. Assume there is a vertex $v^* \in V(G^0)$ with $\deg(v^*) > 2\alpha(G,d)$. Denote its neighbors by $v_1, \dots, v_k$. We partition the neighbors in two classes: $V^+ := \{v_i \mid d(v_i) > 0 \text{ for } i \in [k]\}$ and $V^- := \{v_i \mid d(v_i) < 0 \text{ for } i \in [k]\}$. Clearly, both $v \cup V^+$ and $v \cup V^-$ induce a connected subgraph of G^0, and $d(v^* \cup V^+) - d(v^* \cup V^-) \geq \deg(v^*) > 2\alpha(G,d)$, implying that one of the values must be greater than $\alpha(G,d)$ in absolute value, a contradiction. □

Lemma 4.31.

$$|\{v \in V(G^0) \mid \deg(v) > 2\}| \in O(\alpha(G,d)^3).$$

Proof. Throughout the proof, we denote the distance of two distinct vertices v, w in G^0 as the number of edges on a shortest v,w-path in G^0. Following the arguments from Lemma 4.30, any spanning tree of G^0 can have at most $2\alpha(G,d)$ leaves. This holds, since we can partition the set of leaves in the same way into positive and negative weights and argue that the tree minus the leaves can be combined to a connected subgraph with any subset of leaves.

Now, consider any inclusion-maximal packing (distance ≥ 3) of vertices of degree > 2. Construct a spanning tree by including all edges incident to vertices of the packing, and completing to an aribtrary spanning tree. Observe that the edges incident to the packing don't form a cycle, because of the distance requirement. In any tree, the number of leaves is greater than the number of vertices of degree > 2. Therefore, the size of the packing is at most $2\alpha(G,d)$.

Because of maximality of the packing, any vertex of degree > 2 must be of distance at most 2 from a vertex of the packing. Combining the degree bound of $2\alpha(G,d)$ and the size of the packing of at most $2\alpha(G,d)$, we see that G can have at most $2\alpha(G,d) + 4\alpha(G,d)^2 + 8\alpha(G,d)^3$ vertices of degree > 2. □

Thus, we can describe the structure of G^0 in the following way. There is a collection of $O(\alpha(G,d)^3)$ vertices of degree more than 2, denoted by V^*. The rest of the vertices are on paths that start from some vertex in V^* and either end in a leaf or in some other vertex in V^*. In order to avoid unnecessary case distinction, if $\max_{v \in V(G^0)} \deg(v) \leq 2$, i.e., G^0 is a path or a cycle, we set $V^* := \{v\}$ for an arbitrary $v \in V(G^0)$. Thus, we can assume that V^* is non-empty.

4.4.2 Optimization

We showed in the previous section that structurally, G^0 has nice properties. Lemma 4.31 for instance immediately shows bounded treewidth of G^0, which may motivate the development of a dynamic programming algorithm for our problem. Note that the same is not true for G, since the original graph could contain a large clique or grid minor that contains only few non-zero vertices and is not well-connected to the rest of the graph. Since the 0-vertices are part of our problem, it is not sufficient to restrict the optimization algorithm to the structure of G^0. Therefore, we proceed to characterize, how components of 0-weight vertices can be integrated in the above structure when reversing the 0-contraction operation. We distinguish three cases, depending on the size of the intersection between the neighborhood of a component and $V(G^0) \setminus V^*$, see Figure 4.1. Therefore, consider a component of 0-weight vertices denoted by C_i. The following cases are relevant.

(a) $|N(C_i) \cap (V(G^0) \setminus V^*)| = 0$

(b) $|N(C_i) \cap (V(G^0) \setminus V^*)| = 1$

(c) $|N(C_i) \cap (V(G^0) \setminus V^*)| = 2$

Note that if $|N(C_i) \cap (V(G^0) \setminus V^*)| \geq 3$, this induces a K_3 on the vertices of $V(G^0) \setminus V^*$. Since G^0 is connected and V^* is non-empty, these vertices must have at least one further edge, contradicting the fact that $\deg(v) \leq 2$ for $v \in V(G^0) \setminus V^*$.

Algorithm. We are ready to describe a dynamic programming algorithm in order to prove Theorem 4.25.

We start by computing an optimal solution to the linear relaxation of the integer program of the form (4.1), with M a transposed incidence matrix (possibly together with bounds on the variables). We round the solution to a nearby feasible integer solution x^0 as in Section 4.2.2. Recalling Lemma 4.12, this implies that there is an optimal solution x^{opt} to the problem such that $\|x^0 - x^{\text{opt}}\|_\infty \in O(\Delta)$.

This holds in particular for the variables corresponding to vertices in V^*. Since their number is in $O(\Delta^3)$ and the corresponding guessing range is in $O(\Delta)$ for each of them, there is a function f, such that $f(\Delta)$ guesses suffice for these variables.

We proceed to describe what happens when this set of guesses is fixed, i.e., we have fixed values $x_v = \tilde{x}_v$ for $v \in V^*$. Note that type-(a) components of 0-weight vertices only depend on variables corresponding to vertices in V^* and have no influence on the additional constraint. Thus, checking feasibility and calculating an optimal solution for these components amounts to solving a totally unimodular integer program on the corresponding variables. To be more precise, consider some type-(a) component of 0-weight vertices and denote it by C_i. Denote the submatrix containing the rows interacting with

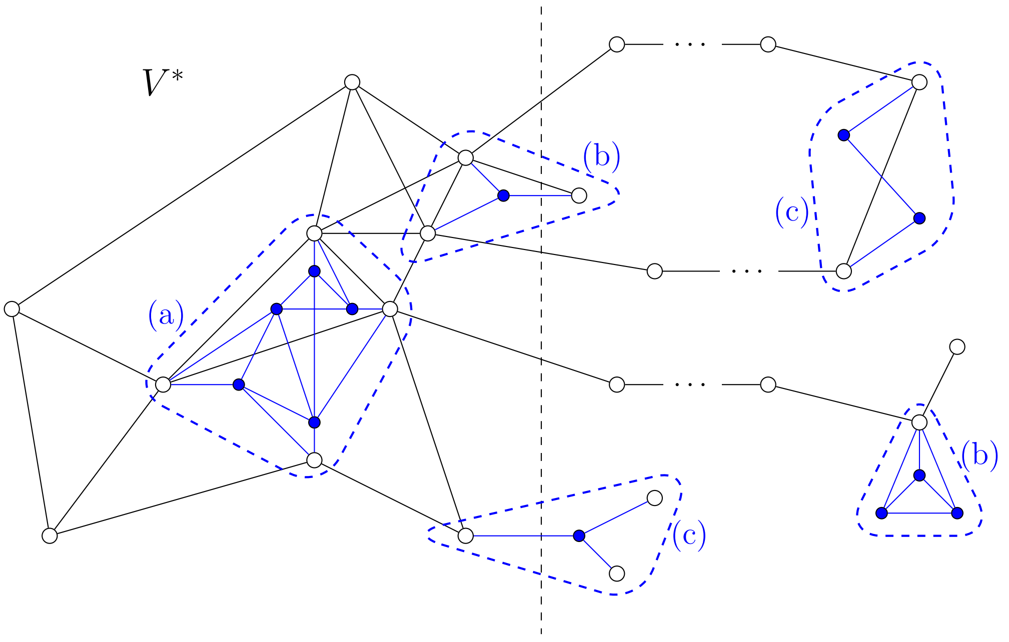


Figure 4.1: Sketch of G illustrating the possible configurations of components of 0-weight vertices in G. Vertices of weight 0 are depicted in blue. We obtain G^0 by deleting the blue vertices and adding cliques on the vertices inside the dashed blue lines.

the variables corresponding to C_i, and all columns corresponding to C_i or $N(C_i)$ by M_{C_i}. These subproblems can then be written as

$$\max\ p^\intercal x \text{ s.t. } M_{C_i}x \le b,\ x_v = \tilde{x}_v \text{ for } v \in V^*,\ x \in \mathbb{Z}^{C_i \cup N(C_i)}\ ,$$

where p and b are naturally restricted to the set of variables and constraints respectively. Since the additional row is not contained in M_{C_i} by definition, this constraint matrix is totally unimodular.

Once we have fixed all guesses on the variables corresponding to vertices in V^* and type-(a) components, we run the following dynamic programming algorithm in order to complete the guesses to a complete solution. We do this for all the $f(\Delta)$ possible guesses and simply return the best solution we obtain in this way. This adds a factor of $f(\Delta)$ to the total runtime of the algorithm. Due to Lemma 4.12, we are guaranteed to find an optimal solution within the search space.

Dynamic programming network. Observe that the connected components of $V(G^0) \setminus V^*$ are paths in G^0. Denote $n' := |V(G^0) \setminus V^*|$. We reindex the vertices to obtain a sequence $v_1, \dots, v_{n'}$ such that $|i - j| = 1$ if $\{v_i, v_j\} \in E(G^0)$, i.e., vertices that are connected by an edge in G^0 appear in succesive fashion.

We create a dynamic programming network $\mathcal{N}$. The network has layers corresponding to the vertices v_i for $i \in [n']$. Each node within a layer carries two additional pieces of information:

- the value of x_{v_i} (compared to the base solution x^0), and
- the accumulated weight margin with respect to the upper bound W.

Note that due to Lemma 4.12, we know that the guessing range for each variable x_{v_i} is at most $4\Delta + 3$. In addition, since the entries of d are bounded by Δ in absolute value, we see that the total weight difference between x^0 and x^{opt} can be bounded by $(4\Delta^2+3\Delta)n$ for any subset of entries. Thus, we see that $\mathcal{N}$ contains $O(\Delta^3 n^2)$ nodes. We add arcs between vertices of layers i and $i+1$ that correspond to guesses on the variable $x_{v_{i+1}}$, for which we have at most $4\Delta + 3$ choices. Each arc will obtain a value based on the profit that can be generated by this choice. Note that therefore the outdegree of every node is in $O(\Delta)$, and $\mathcal{N}$ contains $O(\Delta^4 n^2)$ arcs.

We proceed to formally define the above network. Define $W_0 := d^\intercal x^0 + \sum_{v\in V^*}(\tilde{x}_v - x^0_v)d_v - W$, the weight margin of our initial solution after guessing on the variables corresponding to V^*. As long as this margin is negative, our intermediate solutions are feasible with respect to the last constraint. We define the nodes $N_{i,\tilde{W},\tilde{x}}$ for $i \in [n']$, $\tilde{W} \in [-(2\Delta^2+\Delta)n+W_0, (2\Delta^2+\Delta)n+W_0]$, $\tilde{x} \in [-(2\Delta+1), 2\Delta+1]$, as well as auxiliary nodes $N_{0,W_0,0}$ and $N_{n'+1,0,0}$. Let $i \in [n']$. We add arcs with profit 0 from $N_{n',W',x'}$ to $N_{n'+1,0,0}$ for all choices of x' if $W' \le 0$. In addition, we add an arc from $N_{i-1,W',x'}$ to $N_{i,W'',x''}$ if

- the original integer program has a solution with $x_v = \tilde{x}_v$ for all $v \in V^*$, $x_{v_{i-1}} = x^0_{v_{i-1}} + x'$, and $x_{v_i} = x^0_{v_i} + x''$ (this can be checked by optimizing a totally unimodular integer program as outlined in the beginning of Section 4.2), and
- $W'' = W' + x'' d_{v_i}$.

The profit on these arcs can be calculated by the following subroutine. Denote the set of type-(b) components of 0-weight vertices by $\mathcal{C}^b$ and similarly, type-(c) components by $\mathcal{C}^c$. Let $i \in [n']$. We define $\mathcal{C}^b(i) := \{C \in \mathcal{C}^b \mid v_i \in N(C)\}$ and $\mathcal{C}^c(i) := \{C \in \mathcal{C}^c \mid v_{i-1} \in N(C) \text{ and } v_i \in N(C)\}$, where $\mathcal{C}^c(1) := \emptyset$. Recall that components of 0-weight vertices are replaced by a clique on their neighbors in G^0. Therefore, the neighbors are connected by an edge and appear succesively within our reindexed order. Thus, these sets give a partition of $\mathcal{C}^b$ and $\mathcal{C}^c$ respectively. The relevant variables for each step correspond to the vertices in $\mathcal{C}(i) := \mathcal{C}^b(i) \cup \mathcal{C}^c(i)$. Further, the variables that correspond to v_i or vertices in the neighborhood of v_i in G^0, excluding v_{i+1} will already be fixed. To be more formal, we denote these vertices by $V(i) := \{v_i\} \cup N(v_i) \setminus v_{i+1}$. Note that due to the degree constraints, $|V(i)| \le 3$.

Let us create totally unimodular subproblems for each $i \in [n']$. Denote the submatrix containing the rows interacting with the variables corresponding to $\mathcal{C}(i)$, and all columns corresponding to vertices in $\mathcal{C}(i)$ or $V(i)$ by M_i.

$$\rho(i, (\tilde{x}_v)_{v\in V(i)}) = \max\ p^\intercal x \text{ s.t. } M_i x \le b,\ x_v = \tilde{x}_v \text{ for } v \in V(i),\ x \in \mathbb{Z}^{V(i)\cup\{v\in C \mid C\in\mathcal{C}(i)\}}\ ,$$

where p and b are naturally restricted to the set of variables and constraints respectively. The profit on the arc $(N_{i-1,W',x'}, N_{i,W'',x''})$ is determined by $\rho(i, x_v = \tilde{x}_v \text{ for } v \in V^*, x_{v_{i-1}} = x^0_{v_{i-1}} + x', x_{v_i} = x^0_{v_i} + x'')$.

It is easy to check that each $N_{0,W_0,0} - N_{n'+1,0,0}$-path corresponds to a feasible solution of our problem of the form (4.1), with M a transposed incidence matrix. In addition, the profit along such a path corresponds to the profit of a corresponding solution. Due to the proximity result in Lemma 4.12, we are guaranteed to find an optimal solution in the search space. The runtime of the dynamic programming algorithm is in $g(\Delta)n^{O(1)}$ by construction of the dynamic programming network.

Remark 4.32. Note that our dynamic programming does not make use of the full power of Lemma 4.12. In fact, using Lemma 4.31 we can show that for fixed Δ, G^0 only induces a polynomial number of circuit vectors. Due to the solution decomposition part of Lemma 4.12, this immediately implies a polynomial-time brute force algorithm, testing all possible augmentations and solving totally unimodular integer programs in order to solve the components of 0-weight vertices. This algorithm is not fixed parameter tractable in terms of Δ though, since it does not take advantage of the low dependency of vertices in $V(G^0) \setminus V^*$.

4.4.3 Recognition

We give a short sketch on how to check whether a transposed incidence matrix with one additional row is totally Δ-modular for fixed Δ. Note that due to Corollary 4.29, it suffices to check whether G^0 contains a connected subgraph of weight more than Δ in absolute value. If G^0 is not connected itself, we deal with every connected component separately. Due to Lemma 4.30 and Lemma 4.31, we can reject instances where G^0 contains a vertex of degree more than 2Δ, or more than $8\Delta^3 + 4\Delta^2 + 2\Delta$ vertices of degree more than 2. Thus, we can assume G^0 to be of the same structure as before, with V^* denoting the set of vertices of degree at least 3. Further, by combining Lemma 4.30 and Lemma 4.31, we can see that the sum of the degrees in V^* is bounded. Thus, also the number of paths starting in V^* is bounded by a function of Δ.

We claim that these restrictions make it computationally tractable to find the largest absolute value of a connected subgraph. Observe that instead of searching for the largest absolute value, it suffices to maximize the weight of a connected subgraph for the original weights and their negative. Thus, it suffices to describe an algorithm for finding the largest weight of a connected subgraph.

We denote the paths in $V(G^0) \setminus V^*$ by $P_1, \dots, P_k$. Note that by definition, each path induces a connected component on the vertices of $V(G^0) \setminus V^*$. Since both the number of vertices in V^*, and the number of paths in $V(G^0) \setminus V^*$ are bounded by a function of Δ, the same holds for the set of all possible subsets. Thus, we can iterate over all these subsets and check whether it is possible to extend them by partial paths to a connected subgraph of large weight. Choose any such subset, defined by a subset of high-degree vertices $U^* \subseteq V^*$ and a subset of $k' \leq k$ paths, say without loss of generality $P_1, \dots, P_{k'}$.

Observe that each connected set in G^0 can be partitioned into some vertices in V^*, some complete paths, as well as some partial paths. Since each path induces a connected component on the vertices of $V(G^0) \setminus V^*$, partial paths cannot connect two components that would be disconnected otherwise. Thus, we begin by checking if the induced subgraph defined by U^* and $P_1, \dots, P_{k'}$ is connected. Otherwise, we proceed with the next subset.

For each path in $P_{k'+1}, \dots, P_k$, if it is connected to U^*, we optimize the maximum weight interval that can be connected to the rest of the graph. Since there is only a quadratic number of intervals on each path, this can be done efficiently. Also, the decisions on each path are independent, since connectivity on the partial paths is only relevant locally.

Observe that the runtime of the algorithm we presented is in $O(f(\Delta)n^2)$ for some function f, i.e., it is fixed parameter tractable in terms of Δ.

4.5 Transposed network matrices

We saw in Section 4.4 that we can solve IPs of the form (4.1) if M is a transposed incidence matrix. Since transposed network matrices are a natural generalization of transposed incidence matrices, see Remark 4.14, it is an interesting question of how many of the concepts carry over and can be generalized to transposed network matrices. Recall that with a (transposed) network matrix, we associate a directed graph $D' = (V, A_T \cup A_D)$. In the transposed case, adding the two incidence matrices to the top of M', see Section 4.2.1, corresponds to adding the arcs of A_T (in both directions) to the rows, i.e., the arcs of A_D. Thus, interestingly we can find a linear transformation that transforms M' to the transposed incidence matrix of D'. Note that such a change of variables need not be subdeterminant-preserving, which is why we cannot reduce the case of transposed network matrices to transposed incidence matrices. In the following, we start to describe this linear change of variables and its implications on the graphic representation of our transposed network matrix.

4.5.1 Change of variables

We describe a change of variables that transforms the IP $\max\{p^\intercal x | M'x \leq b_{M'},\ d^\intercal x \leq W,\ x \text{ integer}\}$ where M' arises from a transposed network matrix as described in Section 4.2.1 to the same type of problem, where M' is replaced with the transposed incidence matrix of a related graph. Recall that M' itself is still a transposed network matrix, whose rows correspond to the arcs in A_D and the arcs in A_T in both directions. The corresponding variables correspond to the arcs of the tree, i.e., $x \in \mathbb{Z}^{A_T}$. Thus, A_D already includes this complete collection of arcs. Both T and D are defined on the same set of vertices V. Fix a *root* vertex $r \in V$. After replacing some variables $x(a)$ by their opposite $-x(a)$ (this corresponds to multiplying the corresponding columns by -1), we may assume that T is an r-arborescence, i.e., every arc of T points toward root r. For each vertex $v \in V$, let $P(v) = P(v, r) \subseteq A_T$ denote the unique v–r path in T (in particular, $P(r)$ is empty) and let $y(v) := \sum_{a \in P(v)} x(a)$ (in particular, $y(r) = 0$).

When transforming from arcs to vertices, it is helpful to also translate the property of a minimal cut in the following way. Consider some (potentially directed) graph $D = (V, A)$. We call a vertex subset $S \subseteq V$ *doubly connected* if both induced subgraphs $D[S]$ and

$D[\bar{S}]$ are weakly connected. We use the shorthand *docset* to refer to a doubly connected set.

The following lemma helps to relate the objective vector $p^\intercal$ and the weight vector $d^\intercal$ of the original problem to corresponding vectors of the new problem. These new vectors preserve the values of objective and weights under the linear transformation.

Lemma 4.33. *Let $x \in \mathbb{Z}^{A_T}$ and $y \in \mathbb{Z}^V$ be related as above. Let $d \in \mathbb{Z}^{A_T}$ be an arbitrary vector. If $w(v) = \sum_{a \in \delta_T^+(v)} d(a) - \sum_{a \in \delta_T^-(v)} d(a)$ holds for each $v \in V$, then*

$$\sum_{v \in V} w(v)y(v) = \sum_{a \in A_T} d(a)x(a)\,.$$

Proof. Notice that $\sum_{v : a \in P(v)} w(v) = d(a)$ holds for each arc $a \in A_T$ since the left-hand side contains two opposite terms $d(f)$ and $-d(f)$ for each arc f of the subtree rooted at the tail of a, as well as the extra term $d(a)$. Hence,

$$\begin{aligned}
\sum_{v \in V} w(v)y(v) &= \sum_{v \in V} \left(w(v) \sum_{a \in P(v)} x(a) \right) \\
&= \sum_{a \in A_T} \left(\sum_{v : a \in P(v)} w(v) \right) x(a) \\
&= \sum_{a \in A_T} d(a)x(a)\,.
\end{aligned}$$

□

Note that Lemma 4.33 gives a nice characterization for the transformation of weight and profit vectors respectively: The weight (or profit) of a variable that corresponds to a vertex in the new space is equal to the weight of the corresponding (directed) cut, separating said vertex from the rest of the graph in $(V, A_T \cup A_D)$. Recall that a circuit vector c in the transposed network matrix case corresponds to a (directed) minimal cut.

Such a minimal cut induces a docset. Due to the transformation of the weight vector as defined above, we can see that the weight of such a cut $d^\intercal c$ behaves nicely under the transformation and corresponds to the sum of the weights of the vertices in the induced docsets.

Further, we see that all constraints of $M'x \le b_{M'}$ can be written as $\sum_{a \in P(u)} x(a) - \sum_{a \in P(v)} x(a) \le b(u,v)$ for $(u,v) \in A_D$, where $b(u,v)$ equals b_i for the corresponding row index i. Switching to y-variables, these constraint become $y(u) - y(v) \le b(u,v)$. For the last constraint of $d^\intercal x \le W$, we simply use Lemma 4.33 and write it as $\sum_{v \in V} w(v)y(v) \le W$. Therefore, in y-variables, the IP $\max\{p^\intercal x | M'x \le b_{M'},\ d^\intercal x \le W,\ x \text{ integer}\}$ becomes:

$$\begin{array}{rll}
\max & \sum_{v \in V} q(v)y(v) & \\
\text{s.t.} & y(u) - y(v) \le b(u,v) \quad \forall (u,v) \in A_D & \\
& \sum_{v \in V} w(v)y(v) \le W & \qquad (4.4)\\
& y(r) = 0 & \\
& y \in \mathbb{Z}^V\,. &
\end{array}$$

Above, $q(v) := p(\delta_T^+(v)) - p(\delta_T^-(v))$ for $v \in V$. Notice that $\sum_{v \in V} w(v) = \sum_{v \in V} p(v) = 0$.

We can now formulate the central theorem of this section and prove a small reduction result on the recognition of such matrices.

Theorem 4.34. *For every $\Delta > 0$ there exists a strongly polynomial-time algorithm for solving integer programs of the form* (4.1) *where A is totally Δ-modular, M is a transposed network matrix, and the simple undirected graph G obtained by undirecting the arcs in A_T and A_D is planar and* 3*-connected.*

Recall that we defined k-DRP in Problem 4.3.

Lemma 4.35. *Checking whether a transposed network matrix with one additional row is still totally unimodular is at least as hard as* 2*-DRP.*

In a similar fashion, we can show that checking whether a totally unimodular matrix with k additional rows is 2^{k-1}-modular is at least as hard as k-DRP.

Proof. Note that due to the weight propagation inside the tree, any definition of integer node-weights that sum up to 0 leads to a valid definition of weights for the arcs in A_T. Let an instance of 2-DRP be given by a connected graph $G = (V, E)$, and two sets of terminals $(s_1, t_1), (s_2, t_2)$. We define node-weights $w : V \to \mathbb{Z}$ as follows: $w(s_1) = w(t_1) = 1$, $w(s_2) = w(t_2) = -1$, and $w(v) = 0$ otherwise. Now, the maximum weight docset contains both s_1 and t_1, but none of s_2 and t_2. Such a docset exists if and only if the 2-DRP instance is a *YES*-instance. □

Note that checking for total unimodularity can be done efficiently in general, see e.g.[90, Section 20]. Also, 2-DRP can be solved itself in linear time [65], and even k-DRP admits an efficient algorithm for fixed k [87, 64]. Still, the reduction gives insight on the techniques and theory related to the recognition question and hints that this may be a non-trivial question. It seems plausible to also relate our recognition question to the labeled minor containment problem, see e.g. [64].

4.5.2 Structural results

The following results concentrate on the simple undirected graph G that is obtained by undirecting D (as before, including the arcs of T). Using the change of variables, we obtain a profit vector $q \in \mathbb{Z}^V$ and a weight vector $w \in \mathbb{Z}^V$. We call the vertices $v \in V(G)$ such that $w(v) \neq 0$ *terminals*, and the other vertices *non-terminals*. We let $\beta(G, w)$ denote the maximum weight $w(S) := \sum_{v \in S} w(v)$ of a docset. Therefore, the original matrix A is totally Δ-modular if and only if $\beta(G, w) \leq \Delta$.

The aim of this section is to analyze the possible structure of G with respect to the terminals. In the related literature, graphs that contain a special class of vertices are also called *labeled*, see e.g. [19, 20]. We investigate such labeled graphs with the hope to find substructures that guarantee large $\beta(G, w)$, independent of the specific weights in w. Therefore, we start by defining the notion of a *pumpkin*. These objects have received prior attention in graph-theory, both in labeled and unlabeled form, see e.g. [19, 20, 39, 22]. Roughly speaking, a pumpkin is a model of $K_{2,t}$ in G such that each of the t last branch sets contains a terminal.

Definition 4.36. A *t-pumpkin* is a collection $\{B_1, B_2, B_3, \dots, B_{t+2}\}$ of disjoint connected subsets of G such that B_i has an edge to B_j for each $i \leq 2$ and $j \geq 3$, and moreover each of $B_3, \dots, B_{t+2}$ contains a terminal.

Lemma 4.37. *If G contains a pumpkin of size $t \in \mathbb{N}$, then $4 \cdot \beta(G, w) \geq t$.*

In order to prove Lemma 4.37, we need an additional technical lemma. This lemma allows us to construct certain kind of docsets in an iterative way.

Lemma 4.38. *Let G be a 2-connected graph, with a docset $H \subseteq V(G)$ and a docset $I \subsetneq H$. Then, there is some vertex v in $H \setminus I$, such that $I \cup v$ is a docset.*

Proof. Consider the (non-empty) set of neighbors of I in H and denote it by $N_H(I)$. Choose any vertex v_G in $\overline{H}$, and calculate shortest paths from each vertex of $N_H(I)$ to v_G that do not touch I (this is possible, since $\overline{I}$ is connected). We create an auxiliary directed containment graph J on the vertices of $N_H(I)$, i.e., $(i, j) \in (N_H(I) \times N_H(I))$ is an arc of J if the shortest path from i to v_G contains j. Observe that J is acyclic, because we chose shortest paths, i.e., J has a source. Given some source $v^* \in J$, it remains to show that $I \cup v^*$ is a docset, making v^* a valid choice for v. Clearly, $I \cup v^*$ is connected, so for a contradiction, assume $\overline{I \cup v^*}$ decomposes into more than one connected component $B_1, \dots, B_k$. Since $\overline{H}$ is connected, it is completely contained in a connected component, say wlog B_1. Because of 2-connectedness, $N_H(I) \cap B_i \neq \emptyset$ (otherwise, v^* would be a 1-cut) for each $i \in [2, k]$, i.e., each B_i contains a neighbor of I, denoted by v_i. For each v_i, we calculated a shortest path to v_G in $\overline{I}$ before, not containing v^* by construction, a contradiction. □

Proof of Lemma 4.37. Assume that G contains a t-pumpkin for some $t \in \mathbb{N}$. Denote some t-pumpkin maximizing the number of non-zero bags by $B = \{B_1, \dots, B_{t+2}\}$, i.e., it maximizes $|\{i : i \in [3, t+2], w(B_i) \neq 0\}|$ among all t-pumpkins of G. We will show that at most $\frac{t}{2}$ of the bags have zero-weight.

For a contradiction, consider some bag B_{i^*} with $w(B_{i^*}) = 0$ for $i^* \in [3, t+2]$. For $t \geq 2$, B_{i^*} is a docset. Choose $H = B_{i^*}$ and $I = \{v\}$ for some $v \in B_{i^*} \cap N(B_1)$. While I contains no terminal, apply Lemma 4.38 in order to grow I by one vertex. Observe that when I contains exactly one terminal, then $B_{i^*} \setminus I$ is non-empty, since $w(B_{i^*}) = 0$. In addition, we have that $w(I) \neq 0$ and $w(B_{i^*} \setminus I) \neq 0$.

Upon removal of I, B_{i^*} may decompose into connected components. Clearly, at least one of them has non-zero weight, denote it by J. Since $\overline{I}$ is connected, J connects to some B_i for $i^* \neq i$. Define $J' := \overline{J} \cap B_{i^*}$, which is connected and has non-zero weight. In addition, J' connects to B_1 by construction of I.

If J connects to B_2, define $B'_{i^*} := J'$ and $B'_2 := B_2 \cup J$. Otherwise, if J connects to B_1, then J' must connect to B_2. Then, define $B'_{i^*} := J'$ and $B'_1 := B_1 \cup J$. In both cases, we obtain a same-size pumpkin with $w(B'_{i^*}) \neq 0$ and no other weight of B_i changing for $i^* \neq i \in [3, t+2]$, a contradiction.

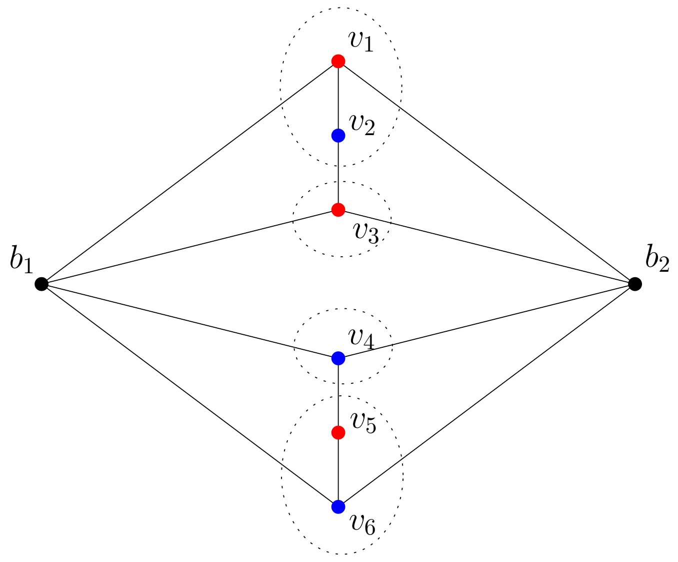


Figure 4.2: Graph G containing a pumpkin of size 4. Red vertices are assigned a weight of 1, blue vertices a weight of -1 and black vertices have weight 0.

Otherwise, we can see that J must connect to some B_i for $i^* \neq i \in [3, t+2]$. Notice that this implies that J' is connected to both B_1 and B_2. If $w(B_i) = 0$, define $B'_{i^*} := J'$ and $B'_i := B_i \cup J$. We again obtain a same-size pumpkin with $w(B_{i^*}) \neq 0$ and $w(B_i) \neq 0$, a contradiction.

Assume that none of the above options exist for all B_i with $w(B_i) = 0$ for $i \in [3, t+2]$. This implies in particular that each zero-weight branchset has a partition into connected subsets (J, J'), such that J connects to some non-zero weight B_i for $i \in [3, t+2]$ and J' connects to both B_1 and B_2. Create a bipartite auxiliary graph H on the branchsets of G in the following way: the vertices of H correspond to branchsets B_i for $i \in [3, t+2]$. For each zero-weight component B_{i^*}, determine (J, J'). If J is a neighbor of B_i for some $i \in [3, t+2]$, then add the edge $\{i, i^*\}$ to H.

Now, consider some non-zero branchset $B_{\tilde{i}}$ and its neighborhood in H. If $B_{\tilde{i}}$ has $k > 1$ neighbors $B_{i_1}, \ldots, B_{i_k}$ in H, denote the corresponding partitions by $(J(i_j), J'(i_j))$. Define $B_{\tilde{i}'} := B_{\tilde{i}} \bigcup_{j \in [k]} J(i_j)$ and $B_{i_j} := J'(i_j)$.

Observe that each of the above redefinitions keeps the size of the pumpkin intact, while increasing the number of non-zero-weight components by at least one, a contradiction. In particular this means that in H, each non-zero component has at most one zero-component neighbor. Since H is bipartite, and each zero-component has a non-zero neighbor, this implies that at most half of the components have zero-weight.

Denote $B^+ := \{B'_i : i \in [3, t+2], w(B_i) \geq 0\}$ and $B^- = \{B'_i : i \in [3, t+2], w(B_i) < 0\}$. Then, both $B'_1 \cup B^+$ and $B'_1 \cup B^-$ induce docsets with a weight difference of at least $t/2$. $\square$

Remark 4.39. It turns out that the factor of 4 in Lemma 4.37 on the relationship between the maximum size of a pumpkin in a graph G, and $\beta(G, w)$ is tight. For this, consider the graph G in Figure 4.2 with $V(G) = \{b_1, b_2, v_1, \ldots, v_6\}$, such that b_1 and b_2 represent the *ends* of the pumpkin. The sets of vertices v_1, v_2, v_3, and v_4, v_5, v_6 each induce a path of length three. In addition, the ends of the path are connected to the ends of the pumpkin, i.e., $\{b_i, v_1\}, \{b_i, v_3\}, \{b_i, v_4\}, \{b_i, v_6\} \in E(G)$ for $i = 1, 2$. Further, $w(b_1) = w(b_2) = 0$, $w(v_1) = w(v_3) = w(v_5) = 1$, and $w(v_2) = w(v_4) = w(v_6) = -1$.

As can be seen from the dashed lines, G contains a pumpkin of size 4. Still $\beta(G, w)$ is only 1. To prove this, it suffices to show that the maximum weight of a docset is bounded by 1, since the total weights sum up to 0.

We separately consider the possible numbers of positive vertices in a docset S. Assume S contains all 3 positive vertices v_1, v_3, v_5. Then, in order to induce a connected subgraph, and having the complement connected, S must contain at least 2 vertices of v_2, v_4, v_6. Otherwise, if S contains v_1 and v_3 but not v_5, then for $\bar{S}$ to be connected, we need $v_2 \in S$. If S contains v_5 and one of v_1 and v_3, then for S to be connected, we need either $v_4 \in S$ or $v_6 \in S$. Finally, if none of this is fulfilled, S contains at most one vertex of positive weight. Thus, $w(S) \leq 1$ for all docsets $S \subseteq V(G)$.

By adding copies of the paths v_1, v_2, v_3 and v_4, v_5, v_6 and connecting them to b_1 and b_2 in the same way, this construction can be extended to show that there exist graphs G containing a $4k$-pumpkin with $\beta(G, w) = k$.

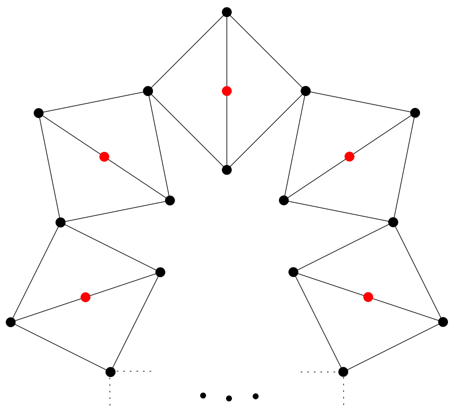
4.5.3 Optimization of planar 3-connected instances

From now on, assume that G is planar and 3-connected, and $\beta(G, w) \leq \Delta$, aligning with Theorem 4.34. By Lemma 4.37, G has no $(4\Delta + 1)$-pumpkin. In the following, we use structural graph theory results by Böhme and Mohar [19] as well as Bienstock and Dean [15] in order to show that the bound on the size of a pumpkin implies that all terminals can be covered by a bounded number of faces, see Theorem 4.40.

Bienstock and Dean [15] relate the maximum size of a *packing* of a set of labeled vertices with the minimum size of a *face cover* of that set by showing that their values differ at most by a multiplicative constant. A packing of vertices denotes a subset of the labeled vertices, such that no two of them are incident to the same face. A face cover of the set of labeled vertices denotes a set of faces, such that each of the vertices is incident to at least one of the selected faces.

Böhme and Mohar [19] studied pumpkins in the context of work by Mohar [78] on the genus problem for apex graphs. Interestingly, their notion of a labeled $K_{2,t}$ minor completely coincides with our notion of pumpkins. They show that if the minimum size face cover is large, say of order k, then we can either find a pumpkin of order $\sqrt{k}$, or a cycle containing at least $\sqrt{k}$ terminals. If there is a cycle of this form, they continue to show that this also implies a large pumpkin (of much smaller order, but still unbounded for increasing k). Their proof uses the result of Bienstock and Dean [15] in order to obtain a large packing from the given large face cover.

Together, this implies the following:


(a) A 2-connected graph G with a large packing of terminals that does not contain a large pumpkin. The terminal vertices are depicted in red.

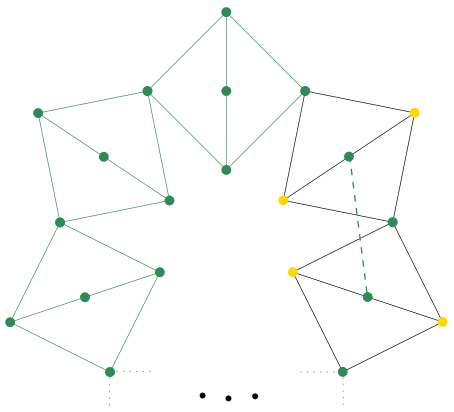

(b) A $K_{4,3}$-minor within G, if two terminals share a face. Green edges are contracted, leaving 3 green vertices, each of them neighboring all 4 yellow vertices.

Figure 4.3: An obstruction to extending Theorem 4.40 to 2-connected graphs.

Theorem 4.40. *[Böhme and Mohar [19], Thm. 1.2]*
Let $G = (V, E)$ be a 3-connected graph with a set of terminals $Z \subseteq V$. If G contains no k-pumpkin, then we can cover all elements of Z with a set of at most $f(k)$ faces, where $f : \mathbb{N} \to \mathbb{N}$ and each terminal is incident to at least one of the faces in the set.

Remark 4.41. The 3-connectedness of G is crucial for the argument of Böhme and Mohar [19] to work. They give such an example within their work, which can be reduced in order to make the structure more clear.

Consider the graph given in Figure 4.3a, consisting of k identical copies of the same graph on 5 vertices, which is connected by identifying vertices of succesive graphs. It can be easily checked that the largest pumpkin in G has size at most 4 (using Lemma 4.37 and an alternating signing of the terminals, giving $\beta(G, w) = 1$), while for this drawing, we need k faces in order to cover all terminals, since no two terminals share a face. Figure 4.3b shows that the second argument holds for any planar drawing of G. Indeed, assume that there is a planar drawing such that two terminals share a face. Then, the drawing remains valid, even if we add an edge between these terminals, implying that the graph is still planar. In Figure 4.3b we show a $K_{4,3}$-minor within this modified graph. Thus, no planar drawing of G can have two terminals within the same face.

The following two lemmas make use of these results in order to restrict the interaction between terminals and docsets.

Lemma 4.42. *For every face f and docset S of G, the vertices of S incident to f are consecutive on the boundary of f.*

We remark that a slightly weaker statement extends to graphs embeddable on a surface of genus Γ. Due to a result of Mohar [77] that excludes $K_{3,k}$ minors on surfaces of genus Γ, with $k \in \Theta(\Gamma)$, we can bound the number of intervals in which a docset intersects a region of the graph by k.

Proof. Assume for a contradiction that the vertices of S incident to f decompose into at least 2 sets $f_1, \dots, f_k$ that are each consecutive on the boundary of f. Observe that also $f \setminus (f_1 \cup f_2)$ decomposes into two parts that are consecutive on the boundary of f, denoted by g_1, g_2. Additionally, we define an auxiliary vertex h within f and connect it to each vertex of f.

Contract f_1, g_1, f_2, g_2 to single vertices. These vertices are necessarily connected in a circular manner by their definition as a partition of a common face. Furthermore, since S is a docset, both f_1 and f_2 as well as g_1 and g_2 are connected. Finally, h is connected to each f_1, g_1, f_2, g_2, giving a K_5 minor. Observe that this auxiliary graph is obtained by contraction and with the addition of h. As such, it is planar if the original graph is planar, a contradiction. □

Lemma 4.43. *Let Z denote the set of all terminals of G. If G has a set of k faces such that every terminal is incident to at least one of these faces, then the number of possible intersections $Z \cap S$ of the terminal set with a docset S is at most n^{2k}, where n denotes the number of vertices of G.*

Proof. This follows straight from Lemma 4.42. □

We now have everything in place to give a strongly-polynomial time algorithm for our described problem as announced in Theorem 4.34. Strictly speaking, the change of variables is not necessary since it is possible to analyze the structures and obtain the subsequent results in the original space. It makes the results and their interpretation easier to follow, so we included it in the final version of the algorithm.

We first follow along the lines of Section 4.2.2 in order to obtain a fractional solution x^*, as well as a rounded feasible integer solution x^0 to our problem. By employing our change of variables, see Section 4.5.1, we obtain a new problem on a transposed incidence matrix, as well as a feasible integer solution y^0. Note that due to the linear transformation, and the equivalence of minimal cuts and docsets, Lemma 4.12 also holds for y^0. This implies that there is an optimal solution y^{opt} to our version of (4.1) that can be obtained by augmenting y^0 in the direction of $O(\Delta)$ docsets. Ultimately, we guess $y(v)$ for all $v \in Z$. Lemma 4.43 shows that for Δ constant, there is a polynomial number of guesses to consider. It turns out that we can find the corresponding faces and therefore all guesses efficiently, since we assume our graph to be 3-connected, which implies a fixed embedding. In addition, Bienstock and Monma [16] give an algorithm to find a corresponding embedding if it exists, even if G is not 3-connected.

For each of these guesses, we check whether the constraint $w^\intercal y \leq W$ is satisfied. We can do this, since the components of y that are not fixed at this stage have a coefficient of zero in the additional constraint. If the constraint is not satisfied, we reject the guess. Otherwise, we solve IP (4.4), fixing all variables $y(v)$ for $v \in Z$ to their guessed value, and removing the additional constraint $w^\intercal y \leq W$ which we know is satisfied. We get an IP over a TU constraint matrix, with integer right-hand sides. Once we have processed all guesses, we simply take the best solution that was found. The complexity of this algorithm is $n^{O(f(\Delta))}$, where n denotes the number of vertices of G, which is strongly polynomial in n for fixed Δ.

Remark 4.44. We remark that in addition to planar 3-connected instances, we can also solve the following types of instances in strongly polynomial time:

- the number of terminals is in $O(\log(n))$. Then, the number of intersections between terminals and docsets is bounded by a polynomial in n, allowing for a similar approach as before.
- the treewidth of G is bounded in terms of Δ. This allows for a dynamic programming algorithm comparable to the one presented in Section 4.4.2.
- the 0-weight vertices form a stable set. This implies that the treewidth is bounded, since the grid minor would contain a large pumpkin otherwise.

4.5.4 Recognition of planar 3-connected instances

Recognizing whether an instance is of the aforementioned type (planar, 3-connected, and totally Δ-modular) is an easy consequence of the results from the previous section. We can check for network matrices that induce a planar 3-connected graph due to [90, Theorem 20.1, Theorem 20.2]. Further, we can use the unique embedding or the result by Bienstock and Monma [16] to check whether we need more than $f(\Delta)$ faces in order to cover all terminals. If we need more faces, we can immediately reject the instance. Otherwise, Lemma 4.43 gives us a polynomial number of docsets to check, whether each of their weights is bounded by Δ.

4.6 Outlook

We close this chapter by giving an overview over possible further research directions relating to Question 4.2. From our presentation of results and the motivation of using Seymour's decomposition, there is some obvious questions. In particular it would be interesting to find strongly polynomial time optimization algorithms for the rest of the base blocks, i.e., general transposed network matrices as well as network matrices. Observe that in the related setting, studied by Nägele, Santiago, and Zenklusen [79], and Nägele, Nöbel, Santiago, and Zenklusen [80], the non-transposed case also proved to be the more challenging one, since they could only give a randomized algorithm for it. Furthermore, it seems relevant to try and generalize Lemma 4.18 to 2-sums and 3-sums.

In addition to these important open questions, our work permits some possible generalizations. In particular it would be intriguing to see if our optimization algorithm for planar and 3-connected instances can be generalized to instances that can be described by a graph of fixed, bounded genus. In our analysis, the result by Böhme and Mohar [19] plays a key role. This argument seems to generalize to higher genus, if in addition large *facewidth* is assumed, see Böhme, Kawarabayashi, Maharry, and Mohar [20, Section 3] where a corresponding statement is given without proof. The facewidth (also known as representability) of a graph embedded on a surface of bounded genus is defined as the minimum number of intersections of a closed, non-contractible curve with the vertices of the graph, where the curve and the graph can only intersect in its vertices. Given this result, it is not necessarily clear though, how to find a respective embedding that allows for a small face cover of the terminals, even if existence is guaranteed.

A second important ingredient of our algorithm presented in Section 4.5.3 is the concept of *pumpkins*. They are a substructure that permit to find large subdeterminants, only based on the knowledge which vertices are terminals. Thus, they represent structures that cannot appear in our problem. It would be interesting to see if we can find a corresponding structure for network matrices or the general totally unimodular case.

In a similar spirit, it is not clear whether pumpkins are the only, or best structure that permits a forbidden minor result. The results by Böhme and Mohar [19] imply something like this for the planar, 3-connected case: if we have a configuration of terminals in G, such that no matter their weight (apart from being non-zero), $\beta(G, w)$ is large, then there also is a large pumpkin in G. It is not immediately clear, whether this can be generalized to arbitrary graphs.

Finally, we remark on the recognition question of total Δ-modularity. This question was left open by Artmann, Weismantel, and Zenklusen [8] in their paper on 2-modularity. It would be intriguing to see, whether it is possible to recognize 2-modularity at least in our strengthened setting. In addition, the recognition question for network matrices with one additional row seems interesting, i.e., is a matrix of the previously described form totally Δ-modular for some fixed Δ? Basically, we ask whether there is a cycle exceeding a certain length in a weighted and directed graph, with some minor additional information. This question is close to provably hard questions, but includes some additional information that may allow for a polynomial-time algorithm.

Bibliography

[1] L. Adenaw and M. Lienkamp. A model for the data-based analysis and design of urban public charging infrastructure. In *2020 Fifteenth International Conference on Ecological Vehicles and Renewable Energies (EVER)*, pages 1–14, 2020.

[2] N. Andrenacci, R. Ragona, and G. Valenti. A demand-side approach to the optimal deployment of electric vehicle charging stations in metropolitan areas. *Applied Energy*, 182:39–46, 2016.

[3] Matthew Andrews, Mustafa K Dogru, John D Hobby, Yue Jin, and Gabriel H Tucci. Modeling and optimization for electric vehicle charging infrastructure. In *IEEE innovative smart grid technologies conference*. Citeseer, 2013.

[4] Manuel Aprile and Samuel Fiorini. Regular matroids have polynomial extension complexity. *Mathematics of Operations Research*, 47(1):540–559, 2022.

[5] Manuel Aprile, Gennadiy Averkov, Marco Di Summa, and Christopher Hojny. The role of rationality in integer-programming relaxations. *arXiv:2206.12253*, 2022.

[6] Okan Arslan, Oya Ekin Karaşan, A Ridha Mahjoub, and Hande Yaman. A branch-and-cut algorithm for the alternative fuel refueling station location problem with routing. *Transportation Science*, 53(4):1107–1125, 2019.

[7] Stephan Artmann. *Optimization of bimodular integer programs and feasibility for three-modular base block IPs.* PhD thesis, ETH Zurich, 2020.

[8] Stephan Artmann, Robert Weismantel, and Rico Zenklusen. A Strongly Polynomial Algorithm for Bimodular Integer Linear Programming. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 1206–1219. Association for Computing Machinery, 2017.

[9] Gennadiy Averkov and Matthias Schymura. On the Maximal Number of Columns of a Δ-modular Matrix. In *Integer Programming and Combinatorial Optimization*, pages 29–42, 2022.

[10] Gennadiy Averkov, Christopher Hojny, and Matthias Schymura. Computational aspects of relaxation complexity: possibilities and limitations. *Mathematical Programming*, pages 1–28, 2021.

[11] Gennadiy Averkov, Christopher Hojny, and Matthias Schymura. Efficient MIP techniques for computing the relaxation complexity. *Mathematical Programming Computation*, pages 1–32, 2023.

[12] Carsten Bamberg, Jascha Lackner, Stefan Siegemund, and Alex Auf der Maur. dena-STUDIE: Privates Ladeinfrastrukturpotenzial in Deutschland, 2020. URL `https://www.dena.de/fileadmin/dena/Publikationen/PDFs/2020/dena-STUDIE_Privates_Ladeinfrastrukturpotenzial_in_Deutschland.pdf`.

[13] I. Safak Bayram, George Michailidis, Michael Devetsikiotis, and Fabrizio Granelli. Electric power allocation in a network of fast charging stations. *IEEE Journal on Selected Areas in Communications*, 31(7):1235–1246, 2013.

[14] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1): 87–90, 1958.

[15] Daniel Bienstock and Nathaniel Dean. On obstructions to small face covers in planar graphs. *Journal of Combinatorial Theory, Series B*, 55(2):163–189, 1992.

[16] Daniel Bienstock and Clyde L Monma. On the complexity of covering vertices by faces in a planar graph. *SIAM Journal on Computing*, 17(1):53–76, 1988.

[17] Adrian Bock, Yuri Faenza, Carsten Moldenhauer, and Andres Jacinto Ruiz-Vargas. Solving the stable set problem in terms of the odd cycle packing number. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[18] Hans L Bodlaender. Dynamic programming on graphs with bounded treewidth. In *Automata, Languages and Programming: 15th International Colloquium Tampere, Finland, July 11–15, 1988 Proceedings 15*, pages 105–118. Springer, 1988.

[19] Thomas Böhme and Bojan Mohar. Labeled $K_{2,t}$ minors in plane graphs. *Journal of Combinatorial Theory, Series B*, 84(2):291–300, 2002.

[20] Thomas Böhme, Ken-ichi Kawarabayashi, John Maharry, and Bojan Mohar. $K_{3,k}$-minors in large 7-connected graphs. *preprint*, 2008.

[21] Nicolas Bonifas, Marco Di Summa, Friedrich Eisenbrand, Nicolai Hähnle, and Martin Niemeier. On sub-determinants and the diameter of polyhedra. In *Proceedings of the twenty-eighth annual symposium on Computational geometry*, pages 357–362, 2012.

[22] Nicolas Bousquet, Théo Pierron, and Alexandra Wesolek. A note on highly connected $K_{2,\ell}$-minor free graphs. *arXiv:2301.02133*, 2023.

[23] E Andrew Boyd. Polyhedral results for the precedence-constrained knapsack problem. *Discrete Applied Mathematics*, 41(3):185–201, 1993.

[24] Paolo M. Camerini, Giulia Galbiati, and Francesco Maffioli. Random pseudo-polynomial algorithms for exact matroid problems. *Journal of Algorithms*, 13(2): 258–273, 1992.

[25] Ismail Capar, Michael Kuby, V. Jorge Leon, and Yu-Jiun Tsai. An arc cover–path-cover formulation and strategic analysis of alternative-fuel station locations. *European Journal of Operational Research*, 227(1):142–151, 2013.

[26] Joana Cavadas, Gonçalo Homem de Almeida Correia, and João Gouveia. A MIP model for locating slow-charging stations for electric vehicles in urban areas accounting for driver tours. *Transportation Research Part E: Logistics and Transportation Review*, 75:188–201, 2015.

[27] Marcel Celaya, Stefan Kuhlmann, Joseph Paat, and Robert Weismantel. Improving the Cook et al. Proximity Bound Given Integral Valued Constraints. In *Integer Programming and Combinatorial Optimization*, pages 84–97. Springer International Publishing, 2022.

[28] Yuming Chen, Wai Shun Cheung, and Tuen Wai Ng. An upper bound on the dimension of the voting system of the European Union Council under the Lisbon rules. *arXiv:1907.09711*, 2019.

[29] Wai-Shun Cheung and Tuen-Wai Ng. A three-dimensional voting system in Hong Kong. *European Journal of Operational Research*, 236(1):292–297, 2014.

[30] Michele Conforti, Samuel Fiorini, Tony Huynh, and Stefan Weltge. Extended formulations for stable set polytopes of graphs without two disjoint odd cycles. *Mathematical Programming*, 192(1):547–566, 2022.

[31] William Cook, Albertus MH Gerards, Alexander Schrijver, and Éva Tardos. Sensitivity theorems in integer linear programming. *Mathematical Programming*, 34 (3):251–264, 1986.

[32] Bálint Csonka and Csaba Csiszár. Determination of charging infrastructure location for electric vehicles. *Transportation Research Procedia*, 27:768–775, 2017. ISSN 2352-1465. 20th EURO Working Group on Transportation Meeting, EWGT 2017, 4-6 September 2017, Budapest, Hungary.

[33] Daniel Dadush, Bento Natura, and László A Végh. Revisiting Tardos's Framework for Linear Programming: Faster Exact Solutions using Approximate Solvers. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 931–942. IEEE, 2020.

[34] Daniel Nicolas Dadush. *Integer programming, lattice algorithms, and deterministic volume estimation.* PhD thesis, Georgia Institute of Technology, 2012.

[35] George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1951.

[36] Michele De Gennaro, Elena Paffumi, and Giorgio Martini. Customer-driven design of the recharge infrastructure and vehicle-to-grid in urban areas: A large-scale application for electric vehicles deployment. *Energy*, 82:294–311, 2015.

[37] Vladimir G. Deineko and Gerhard J. Woeginger. On the dimension of simple monotonic games. *European Journal of Operational Research*, 170(1):315–318, 2006.

[38] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg; New York, fourth edition, 2010.

[39] Guoli Ding. Graphs without large $K_{2,n}$-minors. *arXiv:1702.01355*, 2017.

[40] Martin Dyer and Alan Frieze. Random walks, totally unimodular matrices, and a randomised dual simplex algorithm. *Mathematical Programming*, 64(1-3):1–16, 1994.

[41] Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the steinitz lemma. *ACM Transactions on Algorithms (TALG)*, 16(1):1–14, 2019.

[42] Yuri Faenza and Laura Sanità. On the existence of compact ε-approximated formulations for knapsack in the original space. *Operations Research Letters*, 43(3): 339–342, 2015.

[43] Piotr Faliszewski, Edith Elkind, and Michael Wooldridge. Boolean combinations of weighted voting games. *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 1:185–192, 2009.

[44] Samuel Fiorini, Gwenaël Joret, Stefan Weltge, and Yelena Yuditsky. Integer programs with bounded subdeterminants and two nonzeros per row. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 13–24, 2022.

[45] Josep Freixas. The dimension for the European Union Council under the Nice rules. *European Journal of Operational Research*, 156(2):415–419, 2004.

[46] Josep Freixas and Dorota Marciniak. A minimum dimensional class of simple games. *Top*, 17(2):407–414, 2009.

[47] Delbert Fulkerson and Oliver Gross. Incidence matrices and interval graphs. *Pacific journal of mathematics*, 15(3):835–855, 1965.

[48] Regine Gerike, Stefan Hubrich, Frank Hubrich, Frank Ließke, Sebastian Wittig, and Rico Wittwer. Sonderauswertung zum Forschungsprojekt "Mobilität in Städten–SrV 2018", 2020. URL `https://tu-dresden.de/bu/verkehr/ivs/srv/ressourcen/dateien/SrV2018_Staedtevergleich.pdf`.

[49] Alain Ghouila-Houri. Caractérisation des matrices totalement unimodulaires. In *Comptes Rendus Hebdomadaires des Séances de l'Académie des Science (Paris)*, pages 1192–1194, 1962.

[50] Christoph Glanzer, Robert Weismantel, and Rico Zenklusen. On the number of distinct rows of a matrix with bounded subdeterminants. *SIAM Journal on Discrete Mathematics*, 32(3):1706–1720, 2018. doi: 10.1137/17M1125728.

[51] Rohit Gurjar, Arpita Korwar, Jochen Messner, and Thomas Thierauf. Exact perfect matching in complete graphs. *ACM Transactions on Computation Theory (TOCT)*, 9(2):1–20, 2017.

[52] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. URL `https://www.gurobi.com`.

[53] Paul Göpfert and Stefan Bock. A branch&cut approach to recharging and refueling infrastructure planning. *European Journal of Operational Research*, 279(3):808–823, 2019.

[54] Hugo Hadwiger. Über eine Klassifikation der Streckenkomplexe. *Vierteljschr. Naturforsch. Ges. Zürich*, 88(2):133–142, 1943.

[55] Pablo A. López Hidalgo, Max Ostendorp, and Markus Lienkamp. Optimizing the charging station placement by considering the user's charging behavior. In *2016 IEEE International Energy Conference (ENERGYCON)*, pages 1–7, 2016.

[56] M. John Hodgson. A flow-capturing location-allocation model. *Geographical Analysis*, 22(3):270–279, 1990.

[57] A. Horni, K. Nagel, and K. Axhausen, editors. *The Multi-Agent Transport Simulation MATSim*. Ubiquity Press, London, 2016. ISBN 978-1-909188-77-8.

[58] Robert G Jeroslow. On defining sets of vertices of the hypercube by linear inequalities. *Discrete Mathematics*, 11(2):119–124, 1975.

[59] Xinrui Jia, Ola Svensson, and Weiqiang Yuan. The exact bipartite matching polytope has exponential extension complexity. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1635–1654. SIAM, 2023.

[60] Volker Kaibel and Stefan Weltge. Lower bounds on the sizes of integer programs without additional variables. *Mathematical Programming*, 154(1-2):407–425, 2015.

[61] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of operations research*, 12(3):415–440, 1987.

[62] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[63] Richard Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[64] Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2): 424–435, 2012.

[65] Ken-ichi Kawarabayashi, Zhentao Li, and Bruce Reed. Connectivity preserving iterative compaction and finding 2 disjoint rooted paths in linear time. *arXiv:1509.07680*, 2015.

[66] Leonid Genrikhovich Khachiyan. A polynomial algorithm in linear programming. In *Doklady Akademii Nauk*, volume 244, pages 1093–1096. Russian Academy of Sciences, 1979.

[67] D Marc Kilgour. A formal analysis of the amending formula of Canada's Constitution Act, 1982. *Canadian Journal of Political Science/Revue canadienne de science politique*, 16(4):771–777, 1983.

[68] Stefan Kober and Stefan Weltge. Improved lower bound on the dimension of the EU council's voting rules. *Optimization Letters*, 15:1293–1302, 2021.

[69] Stefan Kober, Maximilian Schiffer, Stephan Sorgatz, and Stefan Weltge. Driver-aware charging infrastructure design. *arXiv:2212.05084*, 2022.

[70] Stavros G Kolliopoulos and George Steiner. Partially ordered knapsack and applications to scheduling. *Discrete Applied Mathematics*, 155(8):889–897, 2007.

[71] Bernhard H Korte and Jens Vygen. *Combinatorial optimization*, volume 1. Springer, 2011.

[72] Michael Kuby and Seow Lim. The flow-refueling location problem for alternative-fuel vehicles. *Socio-Economic Planning Sciences*, 39(2):125–145, 2005.

[73] Casimir Kuratowski. Sur le probleme des courbes gauches en topologie. *Fundamenta mathematicae*, 15(1):271–283, 1930.

[74] Sascha Kurz and Stefan Napel. Dimension of the Lisbon voting rules in the EU Council: a challenge and new world record. *Optimization Letters*, 10(6):1245–1256, 2015. doi: 10.1007/s11590-015-0917-0.

[75] Hendrik W Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of operations research*, 8(4):538–548, 1983.

[76] Nicolas El Maalouly and Raphael Steiner. Exact matching in graphs of bounded independence number. *arXiv:2202.11988*, 2022.

[77] Bojan Mohar. An obstruction to embedding graphs in surfaces. *Discrete mathematics*, 78(1-2):135–142, 1989.

[78] Bojan Mohar. Face covers and the genus problem for apex graphs. *Journal of Combinatorial Theory, Series B*, 82(1):102–117, 2001.

[79] Martin Nägele, Richard Santiago, and Rico Zenklusen. Congruency-constrained TU problems beyond the bimodular case. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2743–2790. SIAM, 2022.

[80] Martin Nägele, Christian Nöbel, Richard Santiago, and Rico Zenklusen. Advances on Strictly Δ-Modular IPs. *arXiv:2302.07029*, 2023.

[81] James G Oxley. *Matroid theory*, volume 3. Oxford University Press, USA, 2006.

[82] Joseph Paat, Ingo Stallknecht, Zach Walsh, and Luze Xu. On the column number and forbidden submatrices for Δ-modular matrices. *arXiv:2212.03819*, 2022.

[83] C.H. Papadimitriou. *Computational Complexity*. Theoretical computer science. Addison-Wesley, 1994.

[84] Christos H Papadimitriou. On the complexity of integer programming. *Journal of the ACM (JACM)*, 28(4):765–768, 1981.

[85] Christian Rakow, Ihab Kaddoura, Ronald Nippold, and Peter Wagner. Investigation of the system-wide effects of intelligent infrastructure concepts with microscopic and mesoscopic traffic simulation. In *27th ITS World Congress, Hamburg, Germany*, 2021.

[86] Victor Reis and Thomas Rothvoss. The subspace flatness conjecture and faster integer programming. *arXiv:2303.14605*, 2023.

[87] Neil Robertson and Paul D Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995.

[88] Neil Robertson and Paul D Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004.

[89] Maximilian Schiffer, Gerhard Hiermann, Fabian Rüdel, and Grit Walther. A polynomial-time algorithm for user-based relocation in free-floating car sharing systems. *Transportation Research Part B: Methodological*, 143:65–85, 2021.

[90] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[91] Paul Seymour. Hadwiger's conjecture. *Open problems in mathematics*, pages 417–437, 2016.

[92] Paul D Seymour. Decomposition of regular matroids. *Journal of combinatorial theory, Series B*, 28(3):305–359, 1980.

[93] Narges Shahraki, Hua Cai, Metin Turkay, and Ming Xu. Optimal locations of electric public charging stations using real world vehicle travel patterns. *Transportation Research Part D: Transport and Environment*, 41:165–176, 2015.

[94] Gerard Sierksma and Yori Zwols. *Linear and integer optimization: theory and practice.* CRC Press, 2015.

[95] Steve Smale. Mathematical problems for the next century. *Mathematics: frontiers and perspectives*, pages 271–294, 2000.

[96] Éva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986.

[97] Alan D Taylor and Allison M Pacelli. *Mathematics and politics: strategy, voting, power, and proof.* Springer Science & Business Media, 2008.

[98] Alan D Taylor and William Zwicker. Weighted voting, multicameral representation, and power. *Games and Economic Behavior*, 5(1):170–181, 1993.

[99] Alan D Taylor and William Zwicker. *Simple Games: Desirability Relations, Trading, Pseudoweightings.* Princeton University Press, 1999.

[100] Klaus Truemper. *Matroid decomposition*, volume 6. Citeseer, 1992.

[101] Sergey I Veselov and Aleksandr J Chirkov. Integer program with bimodular matrix. *Discrete Optimization*, 6(2):220–222, 2009.

[102] Klaus Wagner. Über eine Eigenschaft der ebenen Komplexe. *Mathematische Annalen*, 114(1):570–590, 1937.

[103] Stefan Weltge. *Sizes of linear descriptions in combinatorial optimization.* PhD thesis, Otto von Guericke Universität, Magdeburg, 2015.

[104] Gerhard J Woeginger. The trouble with the second quantifier. *4OR*, 19(2):157–181, 2021.

[105] Barış Yıldız, Okan Arslan, and Oya Ekin Karaşan. A branch and price approach for routing and refueling station location model. *European Journal of Operational Research*, 248(3):815–826, 2016.

[106] Raphael Yuster. Almost exact matchings. *Algorithmica*, 63:39–50, 2012.