DECOMPOSITION OF INTEGER PROGRAMS WITH MATCHABILITY STRUCTURE

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Mathematiker
Florian Herbert Werner Dahms
aus Berlin

Berichter: Universitätsprofessor Dr. ir. Arie M.C.A. Koster
Universitätsprofessor Dr. rer. nat. Sven O. Krumke
Universitätsprofessor Dr. rer. nat. Marco Lübbecke

Tag der mündlichen Prüfung: 25.11.2016

PREFACE

This thesis deals with combinatorial optimization problems, which exhibit some kind of (possibly hidden) matching structure. Examples for problems where such structures appear are educational timetabling, multiple knapsack, list coloring, machine scheduling, and a particular railway shunting problem. Each of these will be examined in more detail. The algorithms developed for these applications all require a considerable amount of mathematical theory to work and are therefore exemplary cases where theoretical research translates directly into applicable results.

The first part of the thesis will introduce the necessary theoretical groundwork, ranging from theory of graphs and polyhedra to the decomposition of (integer) linear programming problems. Many well known graph theoretical results are combined in novel ways to yield the concepts needed for developing algorithms that can generically exploit matching structures. The established theory gives a sound foundation for algorithms that can be used to solve a wide range of practical problems. Some of these are covered later but the results are not confined to these. In fact it is expected that many problems exhibit the required structures and the reader may find in this thesis the necessary groundwork to base her own methods upon. To be useful in this regard, much effort was expended in order to make the theorems as widely applicable as possible.

Overall there are two major areas where the thesis provides novel research. The first of these areas is related to the theory of partial transversals and their polyhedral application in bipartite matchings and subgraphs. It is shown how a separation algorithm for the partial transversal polytope is related to a Benders' decomposition algorithm for the same problem. From this relation some algorithmic improvements are derived. Furthermore these insights then lead to a combinatorial Benders' algorithms that is capable of dealing with hypergraphs instead of regular graphs while being a generalization of the algorithm developed for regular graphs beforehand.

The second larger research topic deals with column generation algorithms where the problem structure exhibits certain symmetries. In very symmetric problems, e. g., in the binpacking problem, one can vastly reduce the number of generated variables by aggregating the variable space. If the problems symmetry is slightly distorted, a novel approach is presented how such an aggregation can still succeed. This approach builds upon the already covered theory around the partial transversal polytope and uses it to ensure that disaggregation will remain possible. Some very beautiful theory leads to the insight that such a procedure can be implemented without a theoretical drawback in the problem complexity.

Alongside these two larger research topics, the thesis also contains many small insights, which are presented at the appropriate places. For example it is shown that for popular matchings it is not possible to separate an equivalent to the partial transversal polytope in a simple way (as simple as it was for the partial transversal polytope). It is also shown how the Benders' optimality cuts for a weighted matching can be efficiently approximated with a polynomial number of cuts in

certain special cases. Furthermore several NP-completeness proofs are presented for various problems.

The second part of this thesis will contain the already mentioned example problems, introduce them in detail, and give computational studies for some of the algorithms developed in the first part. For some of the problems, modifications of the algorithms will be introduced to make them more suited for the specific application. Hopefully the reader will find here inspiration on how the results of this thesis can be applied to her problems.

The thesis presents a large amount of required theory, much of it backed up by original mathematical proofs, which shall grant the reader a deeper understanding of the underlying concepts. Throughout the thesis many theorems are proven. If not stated otherwise the proof will be the original work of the thesis' author (in total there are such original proofs for 15 theorems, 4 lemmas and 5 corollaries). If the theorem was not proven by the author and the original publication of the theorem was known to the author of this thesis, the corresponding publication is cited instead of the proof (potentially alongside other references to proofs which are better understandable according to the opinion of this thesis' author). If the original publication was not known (e. g., because the theorem has diffused into being common mathematical knowledge without a definite source) a good textbook reference was cited.

PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

- "Optimal Freight Train Classification using Column Generation" by Markus Bohlin, Florian Dahms, Holger Flier, and Sara Gestrelius[34]
- "Optimized shunting with mixed-usage tracks" by Markus Bohlin, Sara Gestrelius, Florian Dahms, Matúš Mihalák, and Holger Flier[35]
- "A Two-Stage Decomposition of high School Timetabling applied to cases in Denmark" by Matias Sørensen and Florian Dahms[149]
- "Optimization Methods for Multistage Freight Train Formation" by Markus Bohlin, Sara Gestrelius, Florian Dahms, Matúš Mihalák, and Holger Flier[36]

It turns out that an eerie type of chaos can lurk just behind a facade of order – and yet, deep inside the chaos lurks an even eerier type of order.

— Douglas R. Hofstadter

ACKNOWLEDGMENTS

During the work on this thesis I was supported by many friends and colleagues to whom I would like to express my gratitude.

In particular I would like to express thanks to my PhD advisor Marco Lübbecke for giving direction in times where my research lost its track and teaching me many of the intricacies of implementing proper combinatorial algorithms.

Thank goes to Martin Bergner and his impressive in depth knowledge of GCG and SCIP which made it possible that we were able put the theory of the heterogeneous aggregation algorithm (chapter 5) into a working prototype.

I would like to thank Marco Lübbecke as well as Arie Koster and Sven Krumke for taking the time of thoroughly reviewing this thesis.

I am very thankful to Michael Bastubbe, Christina Büsing, Sarah Kirchner, Christian Puchert, Annika Thome, Felix Willamowski and Jonas Witt for carefully reading the preliminary text of this thesis and providing many helpful corrections and suggestions.

Thanks are also due to Judith Behrooz who helped me in the grueling fight against the wheels of bureaucracy.

Furthermore I would like to thank my parents, who made sure that I never lost my fascination with the sciences and always stayed curious.

Finally I would like to thank my fiancée Anne Junk, whom I gravely neglected way to much while working on the research for this thesis, for her ongoing love and support.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Part I

THEORY

1 INTRODUCTION

In order to settle notation, this chapter will briefly cover some basic mathematical concepts necessary for later parts. Nevertheless the reader will be required to already posses a sound mathematical education. This chapter is not supposed to be a thorough introduction into the respective fields, but references to literature with more exhaustive explanations are provided.

BASIC NOTATION

Logic

Common logical notation – and ($\wedge$), or ($\vee$), implication ($\Rightarrow$), equivalence ($\Leftrightarrow$), and negation ($\neg$) – are used in their regular sense. In sentences the equivalence is (compliant with common practice) often abbreviated as iff (for "if and only if").

In many definitions as well as mathematical models, the standard quantifiers "for all" ($\forall$) and "exists" ($\exists$) are used in a canonical way, i. e., $\forall x \in X, \Phi(x)$ yields true iff the logical function Φ returns true for each $x \in X$ and $\exists x \in X, \Phi(x)$ yields true iff Φ yields true for at least one element $x \in X$. When defining mathematical programming models, large parts of the literature set the quantifiers after the statement they apply to, e. g.,

$$x_i \geq 0 \qquad \forall i \in I$$

This convention will also be followed in this thesis when defining mathematical programming models over multiple lines.

Sets

The set of real numbers is given by $\mathbb{R}$, the set of integers by $\mathbb{Z}$ and the set of rational numbers by $\mathbb{Q}$. The sets $\mathbb{R}_+$ and $\mathbb{Z}_+$ denote the non negative real and integer numbers respectively. Equivalently the non positives are denoted by $\mathbb{R}_-$ and $\mathbb{Z}_-$.

More complex sets will be defined in the common set-builder notation where $\{x : \Phi(x)\}$ is the set of all elements for which the logical function Φ evaluates as true. A set S is called convex, if for $x, y \in S$ and for $0 \leq \lambda \leq 1$ the statement $\lambda x + (1 - \lambda)y \in S$ is true.

When dealing with subsets of sets the general convention will be to use the same letter for both sets but to denote the subset with a bar, i. e., for S a subset might be called $\bar{S} \subseteq S$. Note that in cases where multiple subsets are needed different letters or other distinguishing features will be used.

Functions

In optimization it is often the goal to find a maximal (or minimal element of a given set). For some set $S \subset \mathbb{R}$ let $\max S$ denote an element in S of maximal value (and $\min S$ of minimal value). Note that for arbitrary sets such an element does not need to exist. Within this thesis the corresponding sets will be closed and either have a maximum (or minimum), or they will be empty or unbounded (in the corresponding direction). If the set S is defined over an arbitrary ground set X with a function $f : X \to \mathbb{R}$ such that $S = \{f(x) : x \in X\}$, then an element x leading to a maximum value in S can be accessed using the operator $\arg\max_{x \in X} f(x)$. Similarly a minimizing argument is given by $\arg\min_{x \in X} f(x)$.

The notation $(x)^+$ will be used to refer to the positive part of x, i. e., $(x)^+ := \max\{0, x\}$. Respectively $(x)^- := \min\{0, x\}$ refers to the negative part of x.

Vectors and Matrices

For a vector $a \in X^n$, with X being one of the vector spaces $\mathbb{Z}$, $\mathbb{R}$ or $\mathbb{Q}$, let a_i with $1 \leq i \leq n$ denote the i-th entry of a. Let $dim(X^n) = n$ be the dimension of the underlying vector space. For two vectors $a, b \in X^n$ the inequality $a \leq b$ is fulfilled iff $\forall i \in [n], a_i \leq b_i$. If not stated otherwise, vectors are assumed to be column vectors. Also sometimes vector dimensions will not be explicitly stated if they are clear from the context. When constructing a new vector c from several smaller vectors a, b the notation $c = (a^\intercal, b^\intercal)^\intercal$ becomes cluttered with transposition signs, which will therefore be left out to improve readability: $c = (a, b)$.

There are several commonly used vectors and matrices: the vector with all zero entries is denoted by 0, the vector of all ones as 1. The identity matrix (i. e., the matrix consisting of ones on the diagonal and zeros otherwise) is denoted as I. For any vector $a \in X^n$ let $diag(a) \in X^{n \times n}$ be defined as the matrix with a on the diagonal and zero entries everywhere else.

Often it is useful to index a vector by the elements of a set. Given a set S and a vector $x \in X^{|S|}$ then it is assumed that each dimension of x is associated with one of the elements in S and the corresponding entry can be accessed as x_s for some $s \in S$. Given some subset $\bar{S} \subseteq S$ it is another common and useful notation to define $x(\bar{S}) = \sum_{s \in \bar{S}} x_s$.

For some collection of sets $\mathcal{S} \subseteq 2^S$, it is often interesting to look at its incidence vectors. For some element $\bar{S} \in \mathcal{S}$ the incidence vector $x^{\bar{S}}$ is defined as the vector in $\mathbb{R}^{|S|}$, such that its entries are 1 if they correspond to an element in $\bar{S}$ and 0 otherwise, i. e.,

$$x_s^{\bar{S}} = \begin{cases} 1 & \text{if } s \in \bar{S} \\ 0 & \text{otherwise.} \end{cases}$$

Now the set of all incidence vectors of $\mathcal{S}$ is defined as

$$\chi_{\mathcal{S}} = \{x : (x \in \mathbb{R}^{|S|}) \wedge (\exists \bar{S} \in \mathcal{S}, x = x^{\bar{S}})\}.$$

For two sets of vectors $A, B \in X^n$, their Minkowski sum is defined as the set

$$A + B = \{a + b : (a \in A) \wedge (b \in B)\}.$$

COMPUTATIONAL COMPLEXITY

The task of computational complexity theory is to determine and measure how difficult a certain problem class is and to specify the efficiency of a specific algorithm for it. This thesis will be mainly concerned with the concept of worst case complexity (which will be meant when talking about complexity if not stated otherwise). For a more thorough introduction into this field, the reader is referred to Garey and Johnson (1979)[77].

Assume we are given a computational problem class where I is the set of all its problem instances. Each instance $i \in I$ is encoded using some kind of alphabet. Now for each instance we can assign it a number $n(i)$ indicating the size of the instance in this encoding. The exact extent of the alphabet does not matter as scalar factors will not be of interest later on. In general, a computational problem consists of classifying instances as "yes" or "no" (e. g., determining whether a matrix is invertible or not). Therefore an optimization problem needs to be rephrased in the form "Is this instance solvable with an objective value of at most/at least x?". Having an algorithm to answer this question could then be used to quickly find the optimal value x via bisection, under the assumption that the possible values for x are integral or rational.

If we now look at an algorithm to solve instances from I, we want to know how its run time relates to the size of the input instance. We assume the algorithm runs on some form of computer that is equipped with infinite memory (to ensure that instance size won't be an issue). The definition of a computer is based upon the concept of a Turing machine, which won't be explained here in detail (see, e. g., Garey and Johnson (1979)[77] for further information on this topic). In worst case analysis, we want to find an upper bound (ideally the smallest) for this run time. For example, say that our algorithm will not use more than $f(n)$ units of time for an input of size n. As already mentioned we do not care about scalar factors, as this would mean to get into too many details of the algorithm's implementation and the underlying machine model. Instead we want to get an idea in which "efficiency ballpark" the algorithm is playing. Therefore the common big-O notation will be used. A function f is said to be of asymptotic growth lesser or equal to another function g ($f(n) \in O(g(n))$ or $g(n) \in O(f(n))$) if for some sufficiently large c we have $|f(n)| \leq c \cdot |g(n)|$ for every value n of sufficient size (we usually do not care about very small instance sizes, like $n = 0$). The runtime complexity of an algorithm is described by a function f such that the algorithm runtime is in $O(f(n))$ where n is the instance size.

For a certain problem class one usually wants to find an algorithm having the lowest possible runtime complexity for this problem class. While finding the runtime complexity for some algorithm is usually quite simple, it is much more involved to prove that no algorithm of lower complexity than a certain bound exists. Generally a problem for which there is an algorithm solving it in polynomial time (i. e., in $O(n^k)$ for some constant k) is considered tractable. These polynomially solvable problems are denoted by the problem class P.

Another relevant class of problems is NP. A problem is in NP, if for all its "yes"-instances one could generate a certificate that can be used to prove that it is in fact a "yes"-instance. Furthermore it has to be possible to perform this proof in polynomial time (i. e., $O(n^k)$). Problems in P are obviously also in NP, as here the problem instance already is such a certificate (given that it is a "yes"-instance). On

the other hand it is currently unknown whether $\mathsf{P} \subset \mathsf{NP}$ or $\mathsf{P} = \mathsf{NP}$. It is often assumed that $\mathsf{P} \subset \mathsf{NP}$, a belief also held by the author of this thesis. In fact the reader might not find much value in large parts of this thesis should it turn out that $\mathsf{P} = \mathsf{NP}$.

In the case that $\mathsf{P} \subset \mathsf{NP}$, there exist problems where a solution can easily be verified but which cannot be solved in polynomial time. Candidates for these problems are grouped in the class of NP-complete problems. This complexity class is defined using the concept of reducibility. One problem can be reduced to another one if there is a polynomial algorithm to transform an input of the first to an input of the second one such that a "yes"-instance for the first problem will be translated to a "yes"-instance for the second problem and a "no"-instance will be translated equivalently to another "no"-instance. A problem class is NP-complete if any other problem in NP can be reduced to it. For the Satisfiability problem, Cook's theorem shows that it is NP-complete[50,77]. A problem in NP can be shown to be NP-complete if it is within NP and some problem that is already established to be NP-complete can be reduced to it. For a list of NP-complete problems see Garey and Johnson (1979)[77].

Related to the class of NP-complete problems are the NP-hard problems. These include problems to which all problems in NP can be reduced, but which are not necessarily in NP themselves.

A lot of interesting and relevant planning and optimization problems turn out to be NP-complete or NP-hard. As there is no known polynomial time algorithm to solve them, they are often labeled as intractable. This notion can be quite misleading as it turns out that for many of these problems one can in fact find algorithms to solve instances of practical relevance within reasonable time. Here the key usually is to exploit the structure that practical instances have in some way, to avoid the worst case runtime. This thesis will focus on such structures that are related to matchings in regular and hypergraphs, and which appear in several relevant applications.

GRAPHS

Unless stated otherwise a graph $G = (V, E)$ will be given by a set of vertices V and edges E.

An edge is a pair of vertices. If the pairs are ordered, the graph is called directed. If the pairs are unordered the graph is called undirected. By default, graphs will be undirected and directed graphs will be explicitly denoted as such. Edges will often be treated as if they were sets (i. e., unordered pairs) in order to use common set operations (like intersection, checking for inclusion of an element, etc.). This will help simplify notation in certain cases and let us treat regular graphs and hypergraphs (which will be introduced later) more similarly.

The edges of a graph can be weighted by introducing a weight $w_e \in \mathbb{R}$ for each edge $e \in E$.

A graph is called bipartite if its vertices can be partitioned into two disjoint sets such that every edge has one vertex in each of the two sets. To highlight a bipartite graph, the two disjoint sets of vertices can be provided in the definition, e. g., $G = (U \cup V, E)$.

For a given graph $G = (V, E)$ the function

$$N_G(v) = \{v' : (v' \in V \setminus \{v\}) \wedge (\exists e \in E, (v \in e) \wedge (v' \in e))\}$$

defines the neighborhood of a given vertex v in graph G. Note that v is not part of its own neighborhood. This definition is extended for nonempty sets of vertices $\bar{V} \subseteq V$ as $N_G(\bar{V}) = (\bigcup_{v \in \bar{V}} N_G(v)) \setminus \bar{V}$. The inverse of N_G is defined for some nonempty subset $\bar{V} \subseteq V$ as

$$N_G^{-1}(\bar{V}) = \{v : (v \in V) \wedge (N_G(v) \subseteq \bar{V})\}.$$

The idea behind N_G^{-1} is to give some kind of "pseudo inverse", as N_G may be neither injective nor surjective and therefore does not need to have a proper inverse function. The following theorem states some properties that N_G^{-1} fulfills and which justify the definition of it:

Theorem 1.1. *For a given bipartite graph $G = (V \cup U, E)$ and some subset $\bar{V} \subseteq V$ the following properties hold:*

- $N_G^{-1}(N_G(\bar{V})) \supseteq \bar{V}$
- $N_G(N_G^{-1}(\bar{V})) \subseteq \bar{V}$
- *If there is no $\bar{V}' \subseteq V, \bar{V}' \neq \bar{V}$ with $N_G(\bar{V}') = N_G(\bar{V})$ then*

  $$N_G^{-1}(N_G(\bar{V})) = \bar{V}$$

- *If there exists $\bar{U} \subseteq U$ such that $\bar{V} = N_G(\bar{U})$ then*

  $$N_G(N_G^{-1}(\bar{V})) = \bar{V}$$

Proof. The properties are proven in the order they appear in the theorem.

Let $v \in \bar{V}$. As $\bar{V} \cap N_G(v) = \emptyset$ due to the bipartiteness of G and by definition of N_G and we get $N_G(v) \subseteq N_G(\bar{V})$ which implies $v \in N_G^{-1}(N_G(\bar{V}))$ and therefore $N_G^{-1}(N_G(\bar{V})) \supseteq \bar{V}$.

Let $v \in N_G(N_G^{-1}(\bar{V}))$. Then there exists $v' \in N_G^{-1}(\bar{V})$ such that $v \in N_G(v')$. As $v' \in N_G^{-1}(\bar{V})$ implies $N_G(v') \subseteq \bar{V}$, it follows that $v \in \bar{V}$.

Next let $v \in N_G^{-1}(N_G(\bar{V}))$. This implies $N_G(v) \subseteq N_G(\bar{V})$, which in turn means that $N_G(\bar{V}) = N_G(\bar{V} \cup \{v\})$. If there is no $\bar{V}' \subseteq V, \bar{V}' \neq \bar{V}$ with $N_G(\bar{V}') = N_G(\bar{V})$, the only possibility is that $v \in \bar{V}$.

Finally let $\bar{U} \subseteq U$ such that $N_G(\bar{U}) = \bar{V}$. From this theorems first property we get that $\bar{U} \subseteq N_G^{-1}(\bar{V})$. This implies $\bar{V} = N_G(\bar{U}) \subseteq N_G(N_G^{-1}(\bar{V}))$. □

From Theorem 1.1 we can conclude that N_G being injective implies that N_G^{-1} is a proper left inverse and if N_G is surjective then N_G^{-1} is a proper right inverse. If N_G is only injective or surjective on some subsets, then N_G^{-1} acts as the respective proper inverse restricted to those subsets. Both N_G and N_G^{-1} are illustrated for a bipartite graph in Figure 1.

For directed graphs one can distinguish between the incoming and the outgoing neighbors, where

$$N_G^{\mathrm{i}}(v) = \{v' : (v' \in V \setminus \{v\}) \wedge ((v', v) \in E)\}$$

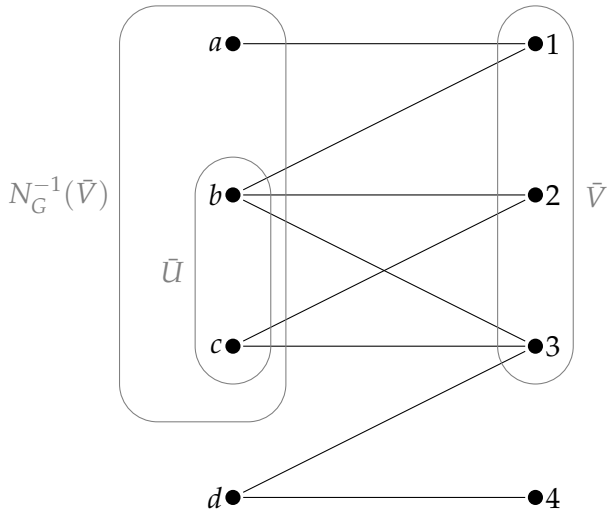


Figure 1: N_G and N_G^{-1}: $\bar{U} = \{b, c\}$, $N_G(\bar{U}) = \{1, 2, 3\} = \bar{V}$, $N_G^{-1}(\bar{V}) = \{a, b, c\}$

are the incoming and

$$N_G^{\circ}(v) = \{v' : (v' \in V \setminus \{v\}) \wedge ((v, v') \in E)\}$$

are the outgoing neighbors.

The set of edges that are adjacent to a certain vertex v is denoted by $E(v) = \{e : (e \in E) \wedge (v \in e)\}$. For a set of vertices $\bar{V} \subseteq V$ the adjacent edges are those containing one of the vertices in $\bar{V}$ but not both:

$$E(\bar{V}) = \bigcup_{v \in \bar{V}} E(v) \setminus \{e : \exists v_1 \in \bar{V}, \exists v_2 \in \bar{V}, (e \in E) \wedge (v_1 \in e) \wedge (v_2 \in e)\}.$$

A subgraph of a graph consists of a subset of the original graph's vertices and edges. For a graph $G = (V, E)$ the subgraph induced by a subset of the vertices $\bar{V} \subseteq V$ is the graph $\bar{G} = (\bar{V}, \bar{E})$ with all edges of G for which all endpoints lie in $\bar{V}$, i. e., $\bar{E} = \{e : \exists v, w \in \bar{V}, (e = \{v, w\} \in E)\}$.

A path in a graph $G = (V, E)$ is a sequence of alternating vertices and edges $p = (v_1, e_1, v_2, \ldots, e_{k-1}, v_k)$ (with $v_i \in V$ and $e_i \in E$) such that no vertex is visited more than once ($v_i \neq v_j$ for each $i \neq j$) and such that the edges connect the neighboring vertices ($\forall i \in [k-1], e_i = (v_i, v_{i+1})$). Note that the direction of the edge does make a difference for a path in a directed graph. Related to paths are cycles, which are defined in the same way as paths, except for having the same start and end vertex $v_1 = v_k$ (note that a single vertex is not considered a cycle). A graph is said to contain an odd cycle if there exists a cycle of odd length in it.

A vertex $v_2 \in V$ is said to be reachable from $v_1 \in V$ if there exists a path from v_1 to v_2. The set of all vertices which are reachable from a given vertex $v \in V$ is denoted as $\mathcal{R}(v)$. Note that $v \in \mathcal{R}(v)$ as it is reachable by a path of length 0.

A subset of vertices $\mathcal{C} \subseteq V$ is called a connected component if for $v_1, v_2 \in \mathcal{C}$ it holds that v_2 is reachable from v_1 and $\mathcal{C}$ is of maximal cardinality with this property, i. e., there is no vertex that can be added to $\mathcal{C}$ without destroying the connectedness. For a graph G, let $\mathcal{C}(G)$ denote the set of all connected components in G. For a vertex $v \in V$ let $\mathcal{C}(v) \subseteq V$ be the connected component containing v. Note that in an undirected graph $\mathcal{R}(v) = \mathcal{C}(v)$, which does not always hold in directed graphs.

A subset of vertices is called independet if there exists no edge in the graph covering two vertices from the set.

Hypergraphs

The notation of hypergraphs resembles its counterpart for graphs. Again a hypergraph $G = (V, E)$ is given by a set of vertices and a set of edges. The difference is, that the edges $E \subseteq 2^V$ can contain an arbitrary amount of vertices instead of exactly two. The notion of a neighborhood N_G can be defined exactly as for regular graphs.

A bipartite hypergraph $G = (U \cup V, E)$ has its vertices partitioned into two disjoint sets U and V such that every edge has exactly one vertex in U. For clarity the subset of vertices, for which this property holds will always be stated first when defining a bipartite hypergraph. It is furthermore assumed that every edge contains at least 2 vertices. Note, that in the literature there exist different definitions of bipartiteness for hypergraphs. This thesis will stick with the aforementioned definition.

Graph problems

Many relevant algorithmic problems can be modeled using graphs. A dominant theme of this thesis are matchings, which are selections of edges such that each vertex in the graph is covered by at most one of the chosen edges. As matchings are fundamental for this thesis they have a dedicated chapter (see Chapter 2).

Two very simple problems are finding the maximal connected component $\mathcal{C}(v)$ and the set of all reachable vertices $\mathcal{R}(v)$ for a given vertex in a graph. These sets can be computed by performing a depth first search as shown by Tarjan (1972)[152]. Note that to find a connected component in a directed graph one needs to maintain some additional data structures to avoid assigning vertices to a component from which there is no path back to the other vertices. Tarjan (1972)[152] shows that this can be done without increasing the runtime complexity. The runtime complexity of these depth first search algorithms is $O(|V| + |E|)$. By repeating this procedure for remaining vertices in the graph one can calculate the set of all maximal connected components $\mathcal{C}^{\max}(G)$ with the same runtime complexity.

Two graph problems related to matchings are maximum s-t-flows and minimum s-t-cuts. These are defined on weighted directed graphs $G = (V, E)$ where V contains two dedicated vertices s (called source) and t (called sink) and where all edge weights are non negative.

An s-t-flow is a function $f : E \to \mathbb{R}_+$, assigning a value to each edge (which denotes the amount of flow flowing over the corresponding edge). An s-t-flow is feasible if each vertex has the same amount of flow coming out of the vertex as going into it (except for the source and the sink) and for which the flow on any edge does not exceed the edge's capacity:

$$\forall v \in V \setminus \{s, t\}, \quad \sum_{v' \in N^{\mathrm{i}}(v)} f((v', v)) = \sum_{v' \in N^{\mathrm{o}}(v)} f((v, v'))$$
$$\forall e \in E, \quad f(e) \leq w_e.$$

The maximum s-t-flow problem now searches for an s-t-flow with maximum total value, where the value of a flow is defined as the total amount of flow leaving its source (or, equivalently, entering its sink): $\sum_{v \in N_G^o(s)} f((s,v))$. The first published algorithm for the maximum flow problem is the Ford-Fulkerson algorithm with a runtime complexity of $O(|V| \cdot |E|^2)$ [71]. The best known maximum s-t-flow algorithm today achieves a runtime complexity of $O(|V| \cdot |E|)$ [127].

A byproduct from the Ford-Fulkerson algorithm is the integral flow theorem:

Theorem 1.2 (Integral flow theorem)**.** *For a directed weighted graph $G = (V, E)$ where $\forall e \in E, w_e \in \mathbb{Z}_+$ there exists a maximum s-t-flow f such that $\forall e \in E, f(e) \in \mathbb{Z}_+$.*

Proof. The theorem follows directly from the way the Ford-Fulkerson algorithm works, as it will produce an integral flow under the preconditions of the theorem (see Ford and Fulkerson (1956) [71] or Ahuja et al. (1993) [5]). □

Related to s-t-flows is the concept of s-t-cuts. An s-t-cut is a set $C \subseteq V$ such that $s \in C$ and $t \notin C$. The value of an s-t-cut is defined as the sum of the edge weights on the border between C and $V \setminus C$:

$$\sum_{\substack{(v_1,v_2) \in E \\ v_1 \in C \\ v_2 \notin C}} w_{(v_1,v_2)}.$$

A relevant problem is now to find an s-t-cut of minimum value. An interesting result shown by Ford and Fulkerson (1956) [71] is that in a given graph the maximum value of an s-t-flow is equal to the minimum value of an s-t-cut:

Theorem 1.3 (Max-flow min-cut)**.** *Given a weighted, directed graph $G = (V, E)$. Then the value of a maximum s-t-flow is equal to the value of a minimum s-t-cut in G.*

Proof. See Ford and Fulkerson (1956) [71] or Ahuja et al. (1993) [5]. □

One can use an algorithm for determining a maximum s-t-flow in order to also find a corresponding minimum s-t-cut. Given a maxium s-t-flow f this can be done by using the auxiliary graph $G' = (V, E')$ where $E' = \{e : (e \in E) \wedge (w_e - f(e) > 0)\}$ are the edges for which the flow does not use the entire edge capacity. Now a minimum s-t-cut is a set $\mathcal{R}(s)$ of all vertices reachable from s. To see that this procedure works as promised, see Ahuja et al. (1993) [5]. Using this algorithm the minimum s-t-cut problem can be solved in the same runtime complexity as the best current maximum s-t-flow algorithm.

POLYHEDRAL THEORY

The definitions in this section are based upon those given by Schrijver (1998) [146], a book which is also recommended for further information about the topic.

The theory of polyhedra is fundamental for several concepts in linear programming, which will be introduced later in this thesis. A polyhedron H is the intersection of finitely many closed half-spaces. A closed half-space can be seen as the set of vectors on one side of or on a hyperplane, or more formally $\{x : (a^\intercal x \leq b) \wedge (x \in \mathbb{R}^n)\}, a, b \in \mathbb{Q}^n, a \neq 0$. Therefore we can define a polyhedron as

$$H = \{x : (Ax \leq b) \wedge (x \in R^n)\}$$

where $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^n$. Every such matrix-vector pair defines a polyhedron. Note that the restriction to $\mathbb{Q}$ implies that these polyhedra are rational polyhedra, a necessary prerequisite for some theorems which this thesis relies on.

Two related concepts are polytopes and cones. Polytopes are commonly defined using the concept of the convex hull: the convex hull $\text{conv}(Y)$ of a set of vectors Y is the smallest convex set containing all those vectors. This is equivalent to

$$\text{conv}(Y) = \{x : \exists k \geq 1, \exists x_1, \ldots, x_k \in Y, \exists \lambda_1, \ldots, \lambda_k \geq 0, \\ (x = \sum_{i=1}^{k} \lambda_i x_i) \wedge (\sum_{i=1}^{k} \lambda_i = 1)\}.$$

A set $T \subset \mathbb{R}^n$ is a polytope, if there exists a finite set $P \subset \mathbb{R}^n$ of vectors such that $T = \text{conv}(P)$.

A polyhedral cone is defined by a matrix $A \in \mathbb{Q}^{m \times n}$ as

$$C = \{x : (Ax \leq 0) \wedge (x \in \mathbb{R}^n)\}.$$

The polyhedral cone is therefore the intersection of a finite number of linear half-spaces $\{x : a^\intercal x \leq 0\}, 0 \neq a \in \mathbb{Q}^n$. It is possible to show that a polyhedral cone can be represented as the conic (i. e., non negative) combination of a finite set of vectors, similar to the finite representation of a polytope:

Theorem 1.4. *Given a polyhedral cone $C = \{x : Ax \leq 0\}$. Then there exists a finite set R of vectors such that*

$$C = \{x : (x = \sum_{r \in R} \lambda_r r) \wedge (\lambda \in \mathbb{R}_+^{|R|})\}.$$

Proof. See Farkas (1902)[69], Minkowski (1910)[121] or Weyl (1934)[163]. □

There exists a strong relationship between polyhedra, polytopes and polyhedral cones:

Theorem 1.5 (Decomposition theorem for polyhedra)**.** *A set H is a polyhedron iff $H = T + C$ for some polytope T and some polyhedral cone C.*

Proof. See Motzkin (1936)[122]. □

Putting all these pieces together we now know that every polyhedron can be decomposed into a polytope and a polyhedral cone which in turn can each be represented by a finite set of generating vectors. This leads to the following, very important theorem:

Theorem 1.6 (Minkowski)**.** *For every polyhedron $H = \{x : Ax \leq b\}$ there exist finite sets P and R (usually denoted as extreme points and extreme rays) such that*

$$H = \{x : (x = \sum_{p \in P} \lambda_p p + \sum_{r \in R} \mu_r r) \wedge (\sum_{p \in P} \lambda_p = 1) \wedge ((\lambda, \mu) \in \mathbb{R}_+^{|P|+|R|})\}.$$

Proof. See, e. g., Nemhauser and Wolsey (1988)[125]. □

Note that the sets P and R contain vectors of the underlying vector space and not just representatives for them (which is a common alternative notation found in the relevant literature).

One conclusion from these findings is the finite basis theorem for polytopes which states that a polyhedron is bounded if and only if it is a polytope[121,151,163].

In later parts of this thesis there will be a particular interest in the polytope defined by the convex hull of a set of incidence vectors $\text{conv}(\chi_{\mathcal{S}})$ over some family of sets $\mathcal{S}$. Having a perfect description of this polytope would allow us to easily optimize over the set $\mathcal{S}$. Note that this convex hull only includes incidence vectors of $\mathcal{S}$, as shown by the following lemma:

Lemma 1.7. *For any family of sets $\mathcal{S}$ and some set $\bar{S}' \notin \mathcal{S}$ the incidence vector $x^{\bar{S}'}$ is not contained in the convex hull of $\chi_{\mathcal{S}}$.*

Proof. Assume for contradiction that there is $\bar{S}' \notin \mathcal{S}$ such that $x^{\bar{S}'} \in \text{conv}(\chi_{\mathcal{S}})$. Then $x^{\bar{S}'}$ must be a convex combination of the vectors in $\chi_{\mathcal{S}}$. Therefore there exists a vector $\lambda \geq 0$ such that $\sum_{\bar{S}\in\mathcal{S}} \lambda_{\bar{S}} = 1$ and $\sum_{\bar{S}\in\mathcal{S}} \lambda_{\bar{S}} x^{\bar{S}} = x^{\bar{S}'}$. As all entries of $x^{\bar{S}}$ are either zero or one, it follows that $(\lambda_{\bar{S}} > 0) \wedge (x_i^{\bar{S}} = 1)$ implies $x_i^{\bar{S}'} = 1$, as $x_i^{\bar{S}'}$ may not be fractional. Equivalently follows that $(\lambda_{\bar{S}} > 0) \wedge (x_i^{\bar{S}} = 0)$ implies $x_i^{\bar{S}'} = 0$ and therefore $\lambda_{\bar{S}} > 0$ implies $x^{\bar{S}} = x^{\bar{S}'}$, meaning that $\bar{S}' \in \mathcal{S}$. □

MATROIDS

A matroid, as defined by Whitney (1935)[164], is a pair $(S, \mathcal{I})$ of a finite set S and $\mathcal{I} \subseteq 2^S$, $\mathcal{I} \neq \emptyset$, for which holds

$$(\bar{S} \in \mathcal{I} \wedge \bar{S}' \subseteq \bar{S}) \Rightarrow (\bar{S}' \in \mathcal{I}) \tag{1.1}$$

$$(\bar{S}_1, \bar{S}_2 \in \mathcal{S} \wedge |\bar{S}_1| < |\bar{S}_2|) \Rightarrow (\exists u \in \bar{S}_2 \setminus \bar{S}_1, \bar{S}_1 \cup \{u\} \in \mathcal{I}). \tag{1.2}$$

The elements of $\mathcal{I}$ are called independent. Condition (1.1) states, that subsets of independent sets must again be independent and condition (1.2) states, that for any independent set, which is not of maximal size within $\mathcal{I}$, it is possible to add an element from a larger independent set such that independence is preserved. These concepts are closely related to linear independence in vector spaces. Similarly one defines a base B of a subset $\bar{S} \subseteq S$ as an inclusion-wise maximal independent subset of $\bar{S}$. By condition (1.2) it is easy to see that any two bases must be of equal size[128]. The rank $r(\bar{S})$ of subset $\bar{S}$ is defined as the size of a basis of $\bar{S}$.

Polymatroids

For the definition of a polymatroid we first need the notion of a submodular function. Given some ground set S then a function $f : 2^S \to \mathbb{R}$ is called submodular, if

$$f(\bar{S}_1) + f(\bar{S}_2) \geq f(\bar{S}_1 \cap \bar{S}_2) + f(\bar{S}_1 \cup \bar{S}_2) \qquad \forall \bar{S}_1, \bar{S}_2 \subseteq S.$$

A polymatroid can now be defined by some submodular function $f : 2^S \to \mathbb{R}$ as the following polytope[65,147]:

$$T_f = \{x : (x \in \mathbb{R}_+^{|S|}) \wedge (\forall \bar{S} \subseteq S, x(\bar{S}) \leq f(\bar{S})\}.$$

Polymatroids exhibit some very useful properties. For example, it is possible to optimize some linear function over a polymatroid using a simple greedy algorithm. Also one can easily describe all facets of the polytope.

For some matroid $M = (S, \mathcal{I})$ one can show that the incidence vectors of $\mathcal{I}$ form a polymatroid. More formally:

Theorem 1.8. *For a matroid $M = (S, \mathcal{I})$ the convex hull of its incidence vectors is given by the polymatroid*

$$\text{conv}(\chi_{\mathcal{I}}) = \{x : (x \in \mathbb{R}_+^{|S|}) \wedge (\forall \bar{S} \subseteq S, x(\bar{S}) \leq r(\bar{S}))\}.$$

where r is the rank function of M.

Proof. See Edmonds (1970)[65].

Note that the specified polytope is a polymatroid, because the rank function of M is submodular. □

The polymatroid yielded from Theorem 1.8 is also called the matroid polytope.

A theorem by Edmonds (1970)[65] shows that the facets of a polymatroid can be characterized using the following two properties:

- A subset $\bar{S} \subseteq S$ is f-flat if $\forall s \in S \setminus \bar{S}, f(\bar{S} \cup \{s\}) > f(\bar{S})$
- A subset $\bar{S} \subseteq S$ is f-inseparable if $\nexists(\bar{S}_1, \bar{S}_2), (\bar{S}_1 \cup \bar{S}_2 = \bar{S}) \wedge (\bar{S}_1 \cap \bar{S}_2 = \emptyset) \wedge (f(\bar{S}) = f(\bar{S}_1) + f(\bar{S}_2))$

Using these properties, the facets of a polymatroid can be determined by the following Theorem 1.9.

Theorem 1.9. *For some submodular function $f : 2^S \to \mathbb{R}$ with $f(\emptyset) = 0$ and $\forall s \in S, f(\{s\}) > 0$, the facets of P_f are given by*

$$\forall s \in S, x_s \geq 0$$
$$\forall \bar{S} \subseteq S, \bar{S} \text{ is } f\text{-flat and } f\text{-inseparable}, x(\bar{S}) \leq f(\bar{S}).$$

Proof. See Edmonds (1970)[65]. Another good proof can be found in Schrijver (2003)[147] on p. 777, Theorem 44.4. □

LINEAR PROGRAMMING

The goal of linear programming is to find a vector in a polyhedron that maximizes (or minimizes) a given linear objective, (given that such a vector exists). Instances of linear programming problems are denoted as linear programs (LPs) and are fully characterized by the objective sense (i. e., max or min), the underlying vector space (which in this thesis will be $\mathbb{R}^n$ for some n), its objective function (c), and its constraints (i. e., the set of affine half-spaces that define the polyhedron). In the remainder of this thesis LPs will usually have their variables constrained to be non negative. An LP can be stated for example as $\max\{c^\intercal x : (Ax \leq b) \wedge (x \in \mathbb{R}_+^n)\}$.

Any x in the given polyhedron is denoted as a feasible solution and a feasible solution x^*, for which $c^\intercal x^* = \max\{c^\intercal x : Ax \leq b\}$, as an optimal solution. The numeric value $c^\intercal x$ for a feasible x is called the objective value of x, the value $c^\intercal x^*$ of an optimal solution x^* is called the optimum. An LP may be unbounded or

infeasible, i. e., there does not need to exist an optimal solution, e. g., in the cases $\max\{1^\intercal x : x \in \mathbb{R}_+^n\}$ or $\max\{x : (x \leq -1) \wedge (x \in \mathbb{R}_+)\}$.

For more complex LPs the condensed notation from above may become difficult to read. In such cases the definition of an LP will be spread over multiple lines having the objective in the first line and the polyhedron defining constraints in the following lines, as is common practice when defining linear programs.

Linear programs can be expressed in multiple, equivalent ways. An inequality constraint $a_i^\intercal x \leq b_i$ can easily be transformed into an equality constraint $a_i^\intercal x + s = b_i$ by introducing an additional variable $s \in R_+$ (often denoted as slack variable). Conversely an equality constraint $a_i^\intercal x = b_i$ can be transformed into the two inequality constraints $a_i^\intercal x \leq b_i$ and $-a_i^\intercal x \leq -b_i$. LPs with a maximization objective, less than or equal constraints, and non negative variables, i. e., of the form

$$\begin{aligned} \max\ & c^\intercal x \\ \text{s.t}\ & Ax \leq b \\ & x \in R_+^n \end{aligned}$$

will be said to be in standard form. If the constraints are equality constraints and therefore the LP has the form

$$\begin{aligned} \max\ & c^\intercal x \\ \text{s.t}\ & Ax = b \\ & x \in R_+^n \end{aligned}$$

it will be said to be in normal form. As mentioned before, it is easy to transform an LP between standard, normal, and other formulations. They can therefore be used interchangeably, depending which is most convenient in the situation at hand.

An important concept from the theory of linear programming is duality. For any so called primal LP $\max\{c^\intercal x : (Ax \leq b) \wedge (x \in \mathbb{R}_+^n)\}$ one can define a corresponding dual LP $\min\{b^\intercal y : (A^\intercal y \geq c) \wedge (y \in \mathbb{R}_+^m)\}$ (whose dual would in turn be the original primal LP). For such a pair of LPs there are two fundamental results, denoted as the weak and strong duality for linear programming. The weak duality states, that the objective value of any feasible y of the dual LP is always an upper bound for the objective value of a feasible x in the primal LP. The strong duality states, that the respective optimal values are actually equal. Of course this requires the existence of an optimum in both LPs. In fact, if one of the LPs is unbounded, the other must therefore have the empty set as the defining polyhedron. Proofs for the duality theorems can be found in Gale et al. (1951)[74].

Theorem 1.10 (Weak Duality Theorem of Linear Programming)**.** *Given a matrix $A \in \mathbb{Q}^{m \times n}$ and two vectors $c \in \mathbb{Q}^n, b \in \mathbb{Q}^m$ and two feasible solutions $x \in \{x : (Ax \leq b) \wedge (x \in \mathbb{R}_+^n)\}$ and $y \in \{y : (A^\intercal y \geq c) \wedge (y \in \mathbb{R}_+^m)\}$. Then $c^\intercal x \leq b^\intercal y$.*

Theorem 1.11 (Strong Duality Theorem of Linear Programming)**.** *Given a matrix $A \in \mathbb{Q}^{m \times n}$ and two vectors $c \in \mathbb{Q}^n, b \in \mathbb{Q}^m$. Iff there exists*

$$x^* \in \{x : (Ax \leq b) \wedge (x \in \mathbb{R}_+^n)\}$$

such that

$$c^\intercal x^* = \max\{c^\intercal x : (Ax \leq b) \wedge (x \in \mathbb{R}_+^n)\}$$

then there exists

$$y^* \in \{y : (A^\intercal y \geq c) \wedge (y \in \mathbb{R}^m_+)\}$$

such that

$$c^\intercal x^* = b^\intercal y^* = \min\{b^\intercal y : (A^\intercal y \geq c) \wedge (y \in \mathbb{R}^m_+)\}$$

An important building block for the duality theory is Farkas' Lemma, which states that a point is either contained in a given (polyhedral) cone or can be separated from this cone by some hyperplane:

Lemma 1.12 (Farkas' Lemma). *For $A \in \mathbb{Q}^{m\times n}$ and $b \in \mathbb{Q}^m$ exactly one of the two following statements holds true:*

$$\exists x \in R^n, (Ax = b) \wedge (x \geq 0)$$
$$\exists y \in R^m, (A^\intercal y \geq 0) \wedge (b^\intercal y < 0).$$

Proof. See Farkas (1902)[69] for the original paper. A more recent proof can be found in Schrijver (1986)[146]. □

As there are different ways to represent a polyhedron, the Farkas' Lemma can also be formulated in many different variants. For example if we start with $(Ax \leq b) \wedge (x \geq 0)$ this is equal to $(Ax + Is = b) \wedge ((x, s) \geq 0)$ and therefore a solution for this system exists if there is no y such that $(A^\intercal y \geq 0) \wedge (y \geq 0) \wedge (b^\intercal y < 0)$ which again is equal to saying that $(A^\intercal y \geq 0) \wedge (y \geq 0)$ implies $(b^\intercal y \geq 0)$.

SOLVING LINEAR PROGRAMS

To find an optimal solution for a given linear program, the prevalent algorithm is the simplex method which dates back to the year 1947 when it was invented by George Dantzig[55]. The simplex method is also fundamental for solving the linear programing relaxation when handling integer programs (see Section 1.8). For a thorough description of the simplex method (and other linear programming algorithms) the reader may consult any text book about linear programming, e. g., Schrijver (1986)[146], Nemhauser and Wolsey (1988)[125], or Bertsimas and Tsitsiklis (1997)[26].

In brief the simplex algorithm works by traversing the extreme points of the feasible polyhedron until it cannot find a direction where the objective can be improved, which means that the current extreme point is optimal. A core component here is that each extreme point of a polyhedron $\{x : (Ax \leq b) \wedge (x \in \mathbb{R}^n_+)\}$ can be described by a subset $B \subseteq [n]$ of the variable indices. When A_B is the submatrix of A restricted to the columns indicated by B, then one can find for each extreme point (at least) one such subset (which will be called a basis) such that $A_B x_B = b$ (the corresponding extreme point for an appropriate basis can be found by solving the system of linear equations $A_B^{-1} b$). The simplex now traverses the polyhedron's extreme points by exchanging indices in the basis appropriately.

In a proper implementation the simplex method is guaranteed to reach an optimal solution in finite time. While it is not of polynomial time complexity[103] (at least up till now there exists no modification for which such a worst case bound could be proven), it turns out that in the average case it performs very well[146].

There exist algorithms to solve linear programs with polynomial complexity. They are usually denoted as interior point methods as they follow a path through the feasible polytope towards an optimal point instead of walking over the polytopes edges, as in the simplex method. The first such algorithm is the ellipsoid method by Khachiyan[102] followed by the projective algorithm by Karmarkar[99]. Nowadays interior point methods can compete with the simplex method performance wise. Note that the interior point methods do not need to find an optimal extreme point, but can also terminate in the middle of an optimal face of the polyhedron. For several applications later it will be important to find optimal extreme points and not just optimal feasible solutions. If an optimal extreme point is needed, additional effort is required. This can also be achieved in polynomial time, as shown by Megiddo (1991)[117] – therefore finding an optimal extreme point of a polyhedron can be achieved in polynomial time.

(MIXED) INTEGER PROGRAMMING

While linear programming is already a powerful tool in itself, a lot of applications do not allow solutions with fractional variable values. Therefore we will now look at optimization problems which (partially) restrict the feasible region from polyhedrons to the integer points within a given polyhedron. Such a problem can be stated in the generic form

$$\max\{c^\intercal x : (Ax \leq b) \wedge (x \in \mathbb{R}_+^{n_1} \times \mathbb{Z}_+^{n_2})\}$$

which will be called a mixed integer linear program (MILP). Often (e. g., in all applications of this thesis) no fractional variables are needed. In this case we speak of an integer linear program (ILP): $\max\{c^\intercal x : (Ax \leq b) \wedge (x \in \mathbb{Z}_+^n)\}$.

Solving integer programming problems

The most efficient algorithms nowadays for solving ILPs and MILPs are based on the branch & bound framework. The basic idea is to relax the integrality constraints of the problem, that were making the problem hard in the first place, thereby obtaining a linear programming relaxation. This LP relaxation can in practice be efficiently solved to optimality, e. g., using the previously mentioned simplex algorithm. If the solution of the LP relaxation turns out to be feasible for the original MILP, one can immediately stop and accept the solution as optimal, because the MILP optimum will never be able to surpass its LP relaxation in terms of the objective. Solutions feasible for the MILP will also be called "integral" (even though in the mixed integer case some of the variables do not necessarily need to be in $\mathbb{Z}$).

In many cases the LP relaxation will not have an integral solution, as for most interesting problems the model constraints will not describe the convex hull of its integer points (see Figure 2). If they would do so (at least around the optimal solution/solutions), one can expect the LP solution to be integral. Section 1.9 gives a class of problems where this is the case. But for many problems such a perfect formulation is expected not to be easily obtainable (having a compact description of the convex hull for some NP-hard problem would imply P=NP).

If the LP relaxation is not integral, one can perform one of the following three major strategies:

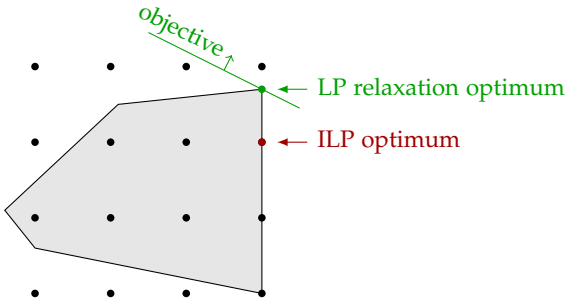


Figure 2: The optimal solution of an ILPs LP relaxation, compared with the ILP optimum

HEURSITICS: Using a heuristic one can try to find some feasible solution with a good objective value. The heuristic might use the pool of already found solutions or the solution of the LP relaxation as a guidance. The objective value of the best known feasible solution is called the primal bound. A multitude of integer programming heuristics exist, for example

- try to round LP relaxation (rounding heuristics)
- iteratively round and fix one variable and resolve the LP relaxation (diving heuristics)
- change the objective function to punish fractional variables and resolve the LP relaxation (objective diving heuristics)
- look in some vicinity of already found solutions for better ones, often solving a smaller integer program in the process (large neighborhood search heuristics)

A survey of the heuristics implemented in the "Solving Constraint Integer Programs" (SCIP) solver was written by Berthold (2008)[25].

CUTTING PLANES: One can try to find some inequality which cuts off the current LP optimum but does not remove any integral points from the feasible region. Adding this inequality to the model might then lead to a new solution of the LP relaxation which might now be integral or for which another separating cut can be found. Some cutting plane methods need certain structures of the model to work (e. g., knapsack covers, which use knowledge about the facets of the knapsack polytope), others can be used for any integer program. Some of these methods can be shown to converge to an optimal solution within a finite number of steps but in practice often do perform poorly if used only on their own. A survey of some modern day cutting plane methods can be found in Marchand et al. (2002)[114].

BRANCHING: Another way to cut off the current LP optimum is to perform a branching decision. In this way the feasible region is divided into two (or more) distinct parts (called branches) where the current LP solution is not included in any of those. A common branching decision is to choose some variable, which is fractional in the current LP solution (let it's index be i and its value in the LP optimum be x_i^*). Now one can divide the problem into two branches, one with the constraint $x_i \leq \lfloor x_i^* \rfloor$, and the other with the constraint $x_i \geq \lceil x_i^* \rceil$ added. Each branch is again an MILP and can now be solved on its own. Note that the number of branches grows quickly with

each additional branching decision and it might in the worst case require a number of such decisions that is exponential in the number of variables. Therefore it is important to discard of branches, if it can be shown that they do not contain an optimal solution. A subproblem can be discarded, if its LP relaxation has an objective worse than the current primal bound or if its LP relaxation is infeasible.

The best LP relaxation objective over all current branches is called the dual bound. The dual bound guarantees that no solution with a better objective exists. Finding a feasible solution achieving the dual bound means that the problem was solved to optimality. The relative difference between the primal and the dual bound is called optimality gap and gives a measure of how much room for improvement in the solution quality is potentially left (note that one might already have the optimal solution at hand, but does not know so as the dual bound is not sufficiently good yet).

The performing of branching is an important part in solving MILP models. Algorithms building upon this idea are denoted as "branch & bound algorithms".

To arrive at an MILP solving procedure which performs sufficient in practice, many techniques need to be combined in order to offset the limitations of each other, leading to very complex implementations. Some state of the art MILP solvers are Gurobi (developed by Gurobi Optimization, Inc.), CPLEX (developed by IBM) and SCIP (developed by the Zuse Institute Berlin), of which SCIP's source code is openly available.

When comparing different solvers a commonly used measure for the progress of the algorithm is the optimality gap. This quantity measures the relative distance between the primal and the dual bound. There are slight differences in the definition of the gap when it comes to different solvers. In the experiments of this thesis, Gurobi and SCIP are used for different algorithms. Their definitions for the optimality gap are (with db being the dual bound and pb being the primal bound):

GUROBI:

$$\frac{|db - pb|}{|pb|}$$

SCIP:

$$\begin{cases} \dfrac{|pb - db|}{\min\{|pb|, |db|\}} & \text{if } \dfrac{pb}{|pb|} = \dfrac{db}{|db|} \\ \infty & \text{otherwise} \end{cases}$$

Special constraint types

Certain kinds of linear constraints appear in many linear programs, are well studied and most integer programming solvers have methods implemented to exploit their structure.

Constraints of the form $a^\intercal x \leq b$ with $a \geq 0$ and $b \geq 0$ are called knapsack constraints, as an integer programming formulation for the knapsack problem (packing items of different weight into a knapsack of bounded capacity) usually has exactly one such constraint, while requiring $x \in \{0, 1\}$.

A special class of knapsack constraints are set packing constraints where it is furthermore required that $a \in \{0,1\}^n$ and $b = 1$. Again related to these are set partitioning constraints $a^\intercal x = 1$ and set covering constraints $a^\intercal x \geq 1$, each with $a \in \{0,1\}^n$. These constraints are used to model the set [packing/partitioning/-covering] problem where one is given a family of sets, out of which one shall choose a subset such that every element from the union of all sets appear in [at most/exactly/at least] one of the chosen sets.

It is not uncommon for integer programming models to have many (or even all) constraints exhibit one of the structures just described. For certain algorithms introduced later it will be important that at least a certain part of the problem does consist only of certain constraint types.

TOTAL UNIMODULARITY AND TOTAL DUAL INTEGRALITY

For any given combinatorial optimization problem, it is desirable to find a "good" formulation for solving it. A common conception of a "good" formulation for an ILP or MILP is a formulation where the LP relaxation is close to (or ideally exactly) the convex hull of the feasible solutions. This section will look at a class of problems where the LP relaxation will always be ideal, and which can therefore be solved very efficiently. These problems will have integral polyhedra for which the extreme points consist of only integral vectors and therefore the Simplex method will result in an integral solution, no matter what the objective function is.

Given some polyhedron $H = \{x : (Ax = b) \wedge (x \geq 0)\}$ where A has full row rank. Now a solution found by the Simplex method (see Section 1.7) will have a corresponding basis B such that $x_B = A_B^{-1}b$ and $x_N = 0$. Whether this solution is integral or not hinges on whether $A_B^{-1}b$ is integral or not. Properties leading to an integral solution can be found using Cramer's rule:

Lemma 1.13 (Cramer's Rule). *For $A \in \mathbb{R}^{n\times n}$ nonsingular and $b \in \mathbb{R}^n$ with $x = A^{-1}b$ we have that*

$$\forall i \in [n] \qquad x_i = \frac{\det(A^i)}{\det(A)}$$

where A^i is the matrix with column i being equal to b and all other columns are the original columns from A.

Proof. See Cramer (1750)[52]. □

Assuming that $A \in \mathbb{Z}^{m\times n}$ and $b \in \mathbb{Z}^m$, Cramer's rule ensures that our solution x is integral if $\det(A_B) \in \{-1,1\}$[27]. If this property holds for each basis B any solution must be integral. Such matrices are called unimodular. When dealing with a problem in standard form, this property must be modified in order to ensure integrality even if not all constraints are active in the respective solution. This leads to the definition

Definition 1.14. A matrix $A \in \mathbb{Z}^{m\times n}$ is totally unimodular if for each square submatrix $\bar{A}$ we have $\det(\bar{A}) \in \{-1,0,1\}$.

Note that each entry in A is for itself a 1×1 submatrix and therefore $A \in \{-1,0,1\}^{m\times n}$. From Cramer's rule it quickly follows that unimodular and totally unimodular matrices lead to polyhedra where each extreme point is an integral

point. On the other hand one can show that total unimodularity is also in some sense sufficient for integrality as stated by the following theorem:

Theorem 1.15. *The matrix $A \in \mathbb{Z}^{m \times n}$ is totally unimodular if and only if $H = \{x : (Ax \leq b) \wedge (x \geq 0)\}$ has only integer extreme points for each $b \in \mathbb{Z}^m$ such that $H \neq \emptyset$.*

Proof. See, e. g., Bertsimas and Weismantel (2005)[27]. □

Due to these properties, it is very desirable for some integer programming problem to have a totally unimodular constraint matrix. The following theorem helps us identify totally unimodular matrices:

Theorem 1.16. *A matrix $A \in Z^{m \times n}$ is totally unimodular if and only if it is possible to partition any collection J of rows (columns) of A into disjoint sets J_1 and J_2 such that for the rows (columns) a_j of A holds*

$$(\sum_{j \in J_1} a_j) - (\sum_{j \in J_2} a_j) \in \{-1, 0, 1\}^k$$

with $k \in \{m, n\}$ depending whether we are working with the rows or columns.

Proof. See Ghouila-Houri (1962)[79], or, for a more recent (and English) proof, see Bertsimas and Weismantel (2005)[27]. □

From Theorem 1.16 it follows that, e. g., the node-edge incidence matrix of a bipartite graph or of a directed graph is totally unimodular[27]. This largely contributes to many optimization problems being very efficiently solvable if restricted to such graph structures. An example is the bipartite matching (which will be covered in Chapter 2).

Another concept defining integral polyhedra, is that of total dual integrality:

Definition 1.17. A polyhedron $\{x : Ax \leq b\}$ with rational $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$ is totally dual integral (TDI), iff for each integer vector $c \in \mathbb{Z}^n$, the corresponding LP dual

$$\min\{b^\intercal y : (A^\intercal y = c) \wedge (y \in \mathbb{R}^m_+)\}$$

has an integral optimum if the minimum is finite.

If A is a totally unimodular matrix, each rational vector b will yield a TDI system[146]. It can be shown that if $\{x : Ax \leq b\}$ is TDI, the LP $\max\{c^\intercal x : (Ax \leq b) \wedge (x \in R^n_+)\}$ will always have an optimal integral solution (see, e. g., Schrijver (1986)[146]).

2

MATCHING THEORY

Matchings are among the most well studied concepts in graph theory. Example applications include the search for possible couples among the applicants of a dating website, matching patients to appropriate doctors, distributing students among study groups and many more.

This thesis deals with optimization problems that are not pure matchings but contain a matching in some part of the overall structure (e. g., the assignment of lectures to rooms that is part of a timetabling problem). The methods developed here for solving such problems rely upon the theory that will be given in this chapter. One of the most fundamental building blocks of this theory is the König-Hall Theorem (which can be found later as Theorem 2.6). In order to be useful in the context of more complex integer programming formulations, the properties of the matching polytope, the partial transversal polytope, and the perfectly matchable subgraph polytope will be studied. Among these the partial transversal polytope is of particular interest for the upcoming applications in later chapters. Finally, some results will be shown how the polyhedral theory for regular matchings can be carried over to hypergraphs and how it fails to do so for popular matchings.

BASIC DEFINITIONS

A matching will be defined as follows: given a graph $G = (V, E)$ a matching $M \subseteq E$ is a set of edges such that no two edges share a vertex, i. e., $(e, e' \in M) \wedge (e \neq e') \Rightarrow (e \cap e' = \emptyset)$. The set of all matchings in a given graph will be called $\mathcal{M}(G)$.

Often one is interested in matchings that are not only valid but also have certain additional properties. Let $\nu(G) = \max\{|M| : M \in \mathcal{M}(G)\}$ be the maximum size a matching in G can have. Then a maximum matching M for G is a matching that is of such maximum cardinality, i. e., $|M| = \nu(G)$. Also of relevance will be matchings on subgraphs of the original graph. For example, given a subset $\bar{V} \subseteq V$ of a graph's vertices, we want to know the cardinality of a maximum matching, covering all the vertices in $\bar{V}$. More precisely we look for a maximum matching in the subgraph $G' = (V, E')$ with $E' = \{e : (e \in E) \wedge (\exists v \in \bar{V}, v \in e)\}$ (i. e., edges covering at least one vertex in $\bar{V}$). The size of such a maximum matching will be denoted as $\nu(\bar{V})$.

If the graph G is weighted with weights w_e for each edge e, each matching can also be associated with a weight and one can define $w(M) = \sum_{e \in M} w_e$ as the weight of matching M. A relevant use case later will be to find matchings that are on the one hand of maximal cardinality but on the other hand minimize their weight, i. e., among all maximum matchings we look for those with minimum weight. This problem will be denoted as minimum weight maximum matching problem.

A perfect matching M is such that it covers every vertex of G. A perfect matching is always a maximum matching. Note that all these definitions for matchings can be applied just as well for hypergraphs.

ALGORITHMS FOR MATCHINGS

The first algorithm for solving the maximum matching problem on any regular graph was published by Jack Edmonds[66]. The algorithm iteratively improves an existing matching (which is empty at the start) along augmenting paths (which will be defined later), while contracting cycles of odd length into single vertices. It has a runtime complexity of $O(|V|^4)$. An algorithm with a better runtime was published by Micali and Vazirani (1980)[120] and has a runtime complexity of $O(\sqrt{|V|}|E|)$. For dense graphs this complexity can – in theory – be improved using the algorithm by Mucha and Sankowski (2004)[123], which is bounded by the complexity of matrix multiplication, currently yielding a runtime complexity of approximately $O(n^{2.3729})$ (see Le Gall (2014)[109]), but which is not useful in practice due to the huge (but constant) overhead involved.

If the underlying graph is bipartite, some specialized algorithms for the maximum matching problem exist which do not need some more complex ideas required for general graphs. The theoretical runtime complexities nevertheless are the same here, i. e., we can expect to efficiently solve a bipartite maximum matching in $O(\sqrt{|V|}|E|)$ time.

Apart from these combinatorial algorithms, one can apply (integer) linear programming to find matchings with certain properties. This is only of minor interest if all we want to do is find a maximum matching in a given graph. The aforementioned special purpose algorithms generally do this job much more efficient. But if the matching problems appear as a part of some larger problem it can be beneficial to exploit this matching structure as we will see later. Starting with Section 2.3, the polyhedral properties of problems related to matchings will be studied more closely.

As some of the underlying concepts are useful in some of the later proofs, a brief introduction into Edmonds' augmenting path algorithm for maximum matchings will be given now. A special focus is given to bipartite graphs, as these are more relevant for the remaining parts of this thesis. When restricted to bipartite graphs, the algorithm becomes simpler, because it does not need to contract odd cycles, as these do not occur in bipartite graphs.

Edmonds' augmenting path algorithm

The augmenting path algorithm is based upon iteratively improving a matching along so called augmenting paths, which are defined as follows:

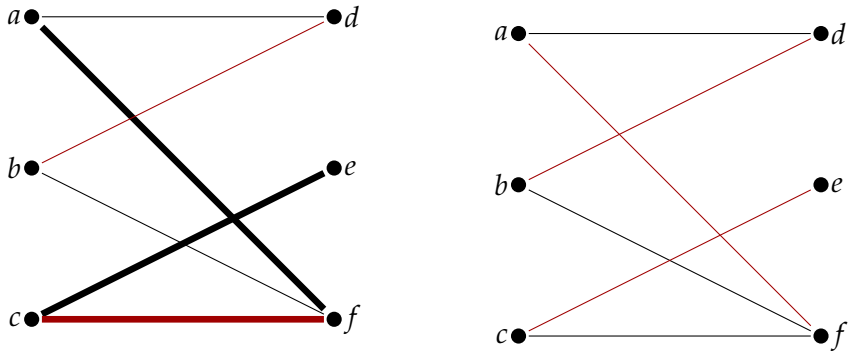
Definition 2.1. Given a graph $G = (V, E)$ and a matching $M \subseteq E$. A path p in G is called *M-alternating* if every second edge of it is contained in M, i. e., the edges are alternating between those in M and those in $E \setminus M$.

A path p is called *M-augmenting*, if it is alternating and its first and last vertex are not covered by M.

Given a matching M_1 in graph G and an M_1-augmenting path p. Then one can create a new matching M_2 by inverting M_1 along the augmenting path p:

$$M_2 = \{e : (e \in p) \wedge (e \notin M_1)\} \cup \{e : (e \in M_1) \wedge (e \notin p)\}.$$

As p is M_1-alternating, M_2 is indeed a matching, because only every second edge in p is covered by M_2. As p is M_1-augmenting we get $|M_2| = |M_1| + 1$. Figure 3a



(a) An initial matching (red) and an augmenting path (bold lines)

(b) The augmented matching

Figure 3: An example for an augmenting path iteration

shows an example for an augmenting path with the resulting augmented matching depicted in Figure 3b.

An augmenting path is not only a sufficient way to improve a matching but it is also necessary that such a path exists if the matching can be improved:

Theorem 2.2. *Given some graph $G = (V, E)$ and a matching $M \subseteq E$. Then M is a maximum matching iff there is no corresponding M-augmenting path in G.*

Proof. See, e. g., Lovász and Plummer (2009)[113]. □

The algorithm will therefore work correctly if it can reliably find existing augmenting paths. The construction of this will now be described very briefly. For more details, please refer to Lovász and Plummer (2009)[113]. Assume we have a graph $G = (V, E)$ and some current matching $M \subseteq E$. The algorithm for finding augmenting paths mainly requires a forest representation of a portion of the graph. Each tree of this forest will have one of the unmatched nodes as its root. The forest is iteratively build up using the vertices covered by the current matching until either an augmenting path is found or all vertices have been included in the forest and the algorithm can report that no augmenting path exists.

At each step of the algorithm one of the unprocessed leaves of one of the current trees is considered. For this vertex $v \in V$ its (yet unprocessed) edges $\{v, v'\}$ are considered. If v' is not yet part of the forest, it is appended to v in the forest. Note that v' must be part of the matching M and therefore there is some vertex v'' such that $\{v', v''\} \in M$. Vertex v'' will not be included in the forest yet, and is appended to v'.

If the vertex v' is already part of the forest, then there are three cases:

1. The distance between v' and the root of its tree is odd. In this case there is nothing to do.

2. The distance between v' and the root of its tree is even and the root of v' is different than the root of v. In this case we found an augmenting path between the two root nodes.

3. The distance between v' and the root of its tree is even and the root of v' is the same as the root of v. In this case we found a cycle of odd length going over the vertices v and v'. We can contract the vertices of this cycle into a

single vertex (having all outgoing edges of the former vertices). It can be shown[66] that searching an augmenting path in the resulting reduced graph is equivalent to searching the augmenting path in G.

Note that the third case can only happen in non bipartite graphs as it means that the algorithm has found a cycle of odd length, which does not exist in bipartite graphs.

THE MATCHING POLYTOPE

A maximum matching of a bipartite graph $G = (V, E)$ can easily be found if one maximizes the quantity $1^\intercal x$ of some vextor x residing inside a polytope describing all feasible matchings via their incidence vectors. This polytope can be described as follows[147]:

$$P_M = \{x : (x \in \mathbb{R}_+^{|E|}) \wedge (\forall v \in V, \sum_{e \in E(v)} x_e \leq 1)\}. \tag{2.1}$$

It can be shown that polytope P_M gives exactly the polytope of all incidence vectors for a feasible matching:

Theorem 2.3. *The convex hull of the incidence vectors describing a feasible matching in G is given by polytope P_M in (2.1).*

Proof. From G being bipartite it can be inferred that the Matrix describing P_M is totally unimodular, which yields the result. For a proper proof see Thm. 18.2 by Schrijver (2003)[147]. □

Using this knowledge, it is now easy to find a maximum matching via linear programming. The corresponding maximum matching LP is

$$\max\{1^\intercal x : (\forall v \in V, \sum_{e \in E(v)} x_e \leq 1) \wedge (x \in \mathbb{R}_+^{|E|})\}. \tag{2.2}$$

An interesting result is that the dual of this LP

$$\min\{1^\intercal y : (\forall \{v_1, v_2\} \in E, y_{v_1} + y_{v_2} \geq 1) \wedge (y \in \mathbb{R}_+^{|V|})\}$$

is the LP relaxation for the minimum vertex cover problem, i. e., the problem of finding a smallest set of vertices such that every edge is covered by at least one of these vertices. For a bipartite graph it can be shown that this formulation is again exactly the convex hull of all incidence vectors encoding a valid vertex cover[147].

From these insights, alongside the strong LP duality (see Theorem 1.11), it is now easy to conclude

Theorem 2.4. *For a bipartite graph G, the size of a maximum matching is equal to the size of a minimum vertex cover.*

Proof. This theorem was already proven by Kőnig (1936)[104]. □

Note that if the underlying graph is not bipartite this result will not hold. In fact the minimum vertex cover problem on general graphs is NP-hard[77]. Moreover polytope P_M (2.1) is totally unimodular if and only if the underlying graph is bipartite and only in this case does it describe the matching polytope (see, e. g., Cor. 18.1b and Thm. 18.2 by Schrijver (2003)[147]).

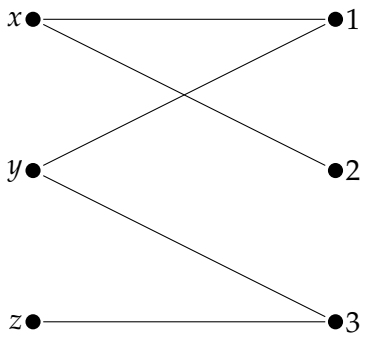


Figure 4: An example of a bipartite graph corresponding to the family of sets $\mathcal{A} = \{A_1 = \{x, y\}, A_2 = \{x\}, A_3 = \{y, z\}\}$

THE PARTIAL TRANSVERSAL POLYTOPE

One concept closely related to the matching polytope is the partial transversal (PT) polytope. A (partial) transversal is defined as follows[147]:

Definition 2.5. Given a set of sets $\mathcal{A} = \{A_1, \ldots, A_n\}$. A set of n distinct elements $T = \{a_1, \ldots, a_n\}$ is called a *transversal*, if it contains exactly one element from each set in $\mathcal{A}$, i. e., $\forall i \in [n], a_i \in A_i$.

A set of $k \leq n$ distinct elements T' is a partial transversal if it is a transversal of a subset $\bar{\mathcal{A}} \subseteq \mathcal{A}$.

Another name for a transversal is "system of distinct representatives". There is a simple bijective relation between a set $\mathcal{A}$ as in Definition 2.5 and a bipartite graph $G = (U \cup V, E)$. Given $\mathcal{A} = \{A_1, \ldots, A_n\}$ one can define G by

$$U = \bigcup_{i \in [n]} A_i, V = [n], E = \{\{u, v\} : (v \in [n]) \wedge (u \in A_v)\}.$$

Now it is easy to see that a transversal in $\mathcal{A}$ corresponds to a matching in this graph that covers all vertices from U. A partial transversal therefore corresponds to a matching which covers a certain subset of U. There may be multiple matchings for a transversal, as a transversal is not ordered and therefore it might be possible to match the elements from a transversal with different elements from $[n]$. On the other hand a matching in a bipartite graph $G = (U \cup V, E)$ can easily be associated with exactly one (partial) transversal in $\{N_G(v) : v \in V\}$. The set for which the transversal is searched will always be the first one mentioned when defining a bipartite graph G (usually U).

Figure 4 illustrates this relationship on a small example. On the set

$$\mathcal{A} = \{A_1 = \{x, y\}, A_2 = \{x\}, A_3 = \{y, z\}\}$$

there exists the transversal $T = \{a_1 = y, a_2 = x, a_3 = z\}$. The set $T' = \{a'_1 = x, a'_2 = y\}$ is a partial transversal, as it is a transversal on $\bar{\mathcal{A}} = \{A_1 = \{x, y\}, A_3 = \{y, z\}\} \subseteq \mathcal{A}$. Figure 4 is the bipartite graph corresponding to $\mathcal{A}$. In this graph, T corresponds to the matching $M = \{\{x, 2\}, \{y, 1\}, \{z, 3\}\}$ and T' corresponds to $M' = \{\{x, 1\}, \{y, 3\}\}$.

Our interest will now be to characterize whether a matching exists in a given bipartite graph covering one side of the vertices or a specified subset of it, i. e.,

wheter a transversal or a certain partial transversal exists. Probably the most well known characterization of the existence of such a matching is the König-Hall Theorem[87,104] also known as Hall's Marriage Theorem, which will be stated first in a version for transversals and then for bipartite graphs. This theorem is actually an equivalent reformulation of the already stated Theorem 2.4. While the theorem can be generalized to transversals of infinite size the following version will suffice for our purposes:

Theorem 2.6. *There exists a transversal for* $\mathcal{A} = \{A_1, \ldots, A_n\}$ *iff*

$$\forall I \subseteq [n], |I| \leq |\bigcup_{i \in I} A_i|.$$

Proof. See Hall (1935)[87] or (for a more modern presentation) Schrijver (2003)[147]. □

As the existence of a transversal implies the existence of a matching in the corresponding bipartite graph, one can easily derive the following graph theoretical equivalent:

Corollary 2.7. *Given a bipartite graph* $G = (U \cup V, E)$. *There exists a matching in* G *covering every node from* U *iff*

$$|\bar{U}| \leq |N_G(\bar{U})| \qquad \forall \bar{U} \subseteq U. \tag{2.3}$$

There exists a perfect matching in G *iff additionally*

$$|U| = |V|. \tag{2.4}$$

Proof. The first part for (not necessarily perfect) matchings follows directly from the relationship between partial transversals and matchings. The additional condition for perfect matchings can easily be concluded as a matching covering all U must also cover all V if and only if the equality holds. □

The characterizations from Theorem 2.6 and Corollary 2.7 basically consist of linear inequalities. We can exploit this to derive the polytope describing the set of all partial transversals of a graph. First we define a set to hold all partial transversals that exist on a given graph.

Definition 2.8. The set of all partial transversals of a graph G is given by $\mathcal{T}(G)$.

Similar to the matching polytope we describe a partial transversal T by an incidence vector $x \in \mathbb{R}_+^{|U|}$ where $x_u = 1$ if $u \in T$. Let $\chi_{\mathcal{T}(G)}$ be the set of all incidence vectors corresponding to a partial transversal in G. One can now find the convex hull $conv(\chi_{\mathcal{T}}(G))$ in the following way:

Theorem 2.9. *The partial transversal polytope for a bipartite graph* $G = (U \cup V, E)$ *is given by*

$$\text{conv}(\chi_{\mathcal{T}(G)}) = \{x : (x \in \mathbb{R}_+^{|U|}) \wedge (x \leq 1) \wedge (2.5)\}$$

with

$$\forall \bar{V} \subseteq V, \qquad x(U \setminus N_G(\bar{V})) \leq |V| - |\bar{V}|. \tag{2.5}$$

Proof. See Schrijver (2003)[147], corollary 22.9a. The proof also shows that the given system is TDI (see also Definition 1.17). □

The constraints (2.5) in Theorem 2.9 can also be formulated in several equivalent variants, some of which are useful in later theorems.

Theorem 2.10. *The set of constraints (2.5) is feasible for the same* $x \in \mathbb{R}_+^{|U|}$ *as*

$$\forall \bar{V} \subseteq V, \qquad x(N_G^{-1}(\bar{V})) \leq |\bar{V}| \tag{2.6}$$

which in turn is equivalent to

$$\forall \bar{U} \subseteq U, \qquad x(\bar{U}) \leq |N_G(\bar{U})|. \tag{2.7}$$

Proof. This can be shown in three steps where in each step it is shown that one set of constraints contains the next one.

(2.5) $\Rightarrow$ (2.6): Let $x \in \mathbb{R}_+^{|U|}$ be feasible for (2.5). Also let $\bar{V} \subseteq V$ and $\bar{V}^C = V \setminus \bar{V}$. First observe that

$$\begin{aligned}(u \in N_G^{-1}(\bar{V})) &\Rightarrow (N_G(u) \subseteq \bar{V}) \\ &\Rightarrow (N_G(u) \cap \bar{V}^C = \emptyset) \\ &\Rightarrow (u \notin N_G(\bar{V}^C)) \\ &\Rightarrow (u \in U \setminus N_G(\bar{V}^C))\end{aligned}$$

From here one can now easily conclude

$$x(N_G^{-1}(\bar{V})) \leq x(U \setminus N_G(\bar{V}^C)) \leq |V| - |\bar{V}^C| = |\bar{V}|$$

(2.6) $\Rightarrow$ (2.7): Let $x \in \mathbb{R}_+^{|U|}$ be feasible for (2.6). Now let $\bar{U} \subseteq U$ and $\bar{V} = N_G(\bar{U})$. From $\bar{U} \subseteq N_G^{-1}(\bar{V})$ it follows that

$$x(\bar{U}) \leq x(N_G^{-1}(\bar{V})) \leq |\bar{V}| = |N(\bar{U})|$$

(2.7) $\Rightarrow$ (2.5): Let $x \in \mathbb{R}_+^{|U|}$ be feasible for (2.7). Let again $\bar{V} \subseteq V$ and $\bar{V}^C = V \setminus \bar{V}$. The first step is to show that

$$N_G(U \setminus N_G(\bar{V})) \subseteq \bar{V}^C$$

To do so note that

$$\begin{aligned}(v \in N_G(U \setminus N_G(\bar{V}))) \Rightarrow \\ (\exists u \in U, (u \notin N_G(\bar{V})) \wedge (v \in N_G(u)))\end{aligned}$$

Assume that $v \in \bar{V}$. Then $v \in N_G(u)$ contradicts $u \notin N_G(\bar{V})$. Therefore we must have $v \in \bar{V}^C$. This leads to

$$x(U \setminus N_G(\bar{V})) \leq |N_G(U \setminus N_G(\bar{V})) \leq |\bar{V}^C| = |V| - |\bar{V}|$$

which concludes the theorem.

□

Theorem 2.10 allows us to use different equivalent formulations for the partial transversal (PT) polytope which will be useful in some applications later.

Yet another way of representing the PT polytope is in the form of a polymatroid (see Section 1.5). The first step is to see that the set of partial transversals forms a matroid:

Theorem 2.11. *Given a bipartite graph $G = (U \cup V, E)$. Then $M_T = (U, \mathcal{T}(G))$ is a matroid.*

Proof. See Edmonds and Fulkerson (1965)[67] for a complete proof. The first matroid condition (1.1) is clear, as the subset of a partial transversal must again be a partial transversal by definition. The second condition (1.2) follows from Theorem 2.2, which shows that each partial transversal can be extended using augmenting paths until it reaches the size of a maximum matching, while keeping the original vertices in the partial transversal. □

Corollary 2.12. *The partial transversal polytope of a bipartite graph $G = (U \cup V, E)$ is given by*

$$\operatorname{conv}(\chi_{\mathcal{T}(G)}) = \{x : (x \in \mathbb{R}_+^{|U|}) \wedge (\forall \bar{U} \subseteq U, x(\bar{U}) \leq \nu(\bar{U}))\}$$

and is a polymatroid.

Proof. First take a look at the rank function of our matroid M_T. For some subset $\bar{U} \subseteq U$, $r(\bar{U})$ is the size of a largest subset from $\bar{U}$ which is in $\mathcal{T}$, i. e., the largest partial transversal contained in $\bar{U}$. Therefore $r(\bar{U})$ gives exactly the size of a maximum matching in the subgraph induced by the vertices $\bar{U}$. So $r(\bar{U}) = \nu(\bar{U})$.

Now the corollary follows directly from Theorem 1.8. □

Which of these equivalent formulations of the partial transversal polytope is most useful depends on the situation. For example the formulation obtained from Corollary 2.12 has the disadvantage that it depends on the number $\nu(\bar{U})$ for all subsets $\bar{U} \subseteq U$. Calculating this is equivalent to calculating a maximum matching for the subsets $\bar{U}$ which is more difficult than, e. g., just counting the neighbors of $\bar{U}$. But if we get the $\nu(\bar{U})$ anyhow, that formulation is quite useful. This can be the case when we try to generate the partial transversal polytope with a separation algorithm.

Separating the partial transversal polytope

When generating the constraints of the partial transversal polytope, it makes sense to differentiate whether we are considering a fractional or an integer vector that we want to separate. For fractional vectors, violated constraints of the kind given by Theorem 2.10 can be found via a min-s-t-cut algorithm. If the vector is already integral it is easier to calculate a maximum matching and generate the polymatroid constraints from Corollary 2.12. Both variants are described below in greater detail.

Note that one should not separate constraints of the type $x \leq 1$. Such variable bounds are better included in the formulation directly from the start for efficiency reasons. Many MILP solvers can handle binary variables more efficiently than general integer variables.

FRACTIONAL VECTORS The separation routine for matchings has long been known and is described by Qi (1987)[134]. This paper also gives additional details about the min-s-t-cut based separation routine described now.

Assume we are given a fractional vector $x \in \mathbb{R}_+^{|U|}$. One can now find a violated constraint of the form (2.6) using a min-s-t-cut algorithm as follows. First construct a directed, weighted helper graph $G_h = (\{s,t\} \cup U \cup V, E_h)$ having edges and edge weights:

$$\begin{aligned} \forall u \in U, ((s,u) \in E_h) \wedge (w_{(s,u)} = x_u) \\ \forall v \in V, ((v,t) \in E_h) \wedge (w_{(v,t)} = 1) \\ \forall (u,v) \in E, ((u,v) \in E_h) \wedge (w_{(u,v)} = \infty). \end{aligned}$$

Let C be the vertex set defining a minimum weight s-t-cut in G_h such that $s \in C$. Assume $w(C) < x(U)$. Then

$$x(N_G^{-1}(V \cap C)) = x(U \cap C) > w(C) - x(U \setminus C) = |V \cap C|.$$

Therefore x can be separated using the constraint $x(N_G^{-1}(V \cap C)) \leq |V \cap C|$. On the other hand if there is no s-t-cut with weight lesser than $x(U)$ then there is no violated constraint and x is therefore feasible.

INTEGRAL VECTORS Let $x \in \{0,1\}^{|U|}$ and let $U' = \{u : x_u = 1\}$. Our goal is to find a subset $\bar{U} \subseteq U$ such that $x(\bar{U}) > \nu(\bar{U})$ or show that no such $\bar{U}$ exists. This can be done by calculating a matching in the subgraph of G, restricted to the vertices $U' \cup V$ which will be denoted as $G' = (U' \cup V, E')$. Let now M be a maximum matching in G'. First assume that $|M| = |U'|$. In this case M is proof that x represents a partial transversal and therefore no constraint can be violated.

Next assume that $|M| < |U'|$. As M being a maximal matching implies $|\nu(U')| = |M|$, we immediately get $x(U') > \nu(U')$. Note that the corresponding constraint $(x(U') \leq \nu(U'))$ is not necessarily a facet of the partial transversal polytope. In Chapter 4 the algorithm for separating the partial transversal polytope will be explored further, after we have equipped ourselves with some additional theory. A method will be shown how these cuts can be improved to yield facets of the partial transversal polytope.

Using a matching algorithm instead of calculating a minimum weight s-t-cut has the advantage that we just need a computational runtime effort of $O(\sqrt{|V|}|E|)$ instead of $O(|V||E|)$ required for the s-t-cut[127]. But note that in many applications neither the time required for solving the matching nor for the minimum weight s-t-cut will have a large overall impact on the solver runtime, as usually other parts of the problem will require the lion's share of the computational effort.

Facets of the partial transversal polytope

Using the formulation from Corollary 2.12 together with Theorem 1.9 provides a way to check whether a subset $\bar{U} \subseteq U$ defines a facet via the inequality $x(\bar{U}) \leq \nu(\bar{U})$ or not. Also they can be helpful in order to list the facets for certain, specially structured, partial transversal polytopes or to show that a polynomial number of constraints suffices for its description.

The following results link the definitions of inseparability and flatness to more graph theoretical concepts.

Lemma 2.13. *Given a bipartite graph $G = (U \cup V, E)$. If $\bar{U} \subseteq U$ with $|\bar{U}| > 1$ is ν-inseparable, then $\nu(\bar{U}) = |N(\bar{U})|$.*

Proof. Obviously the inequality $\nu(\bar{U}) \leq |N(\bar{U})|$ has to hold. To show that $|N(\bar{U})| \leq \nu(\bar{U})$, assume that there exists a maximal matching $M \subseteq E$ over $\bar{U}$ and a vertex $\hat{v} \in N(\bar{U})$ such that there is no edge $(u, \hat{v}) \in M$ for some $u \in U$. Now choose some $\hat{u} \in N(\hat{v}) \cap \bar{U}$ and define $\bar{U}_1 = \{\hat{u}\}$. Next let $\bar{U}_2 = \bar{U} \setminus \bar{U}_1$. It follows that $\bar{U}_2 \neq \emptyset$ as $|\bar{U}| > 1$. Of course $\nu(\bar{U}_1) = 1$. Also it has to hold that $\nu(\bar{U}_2) = \nu(\bar{U}) - 1$, which can be seen by examining a maximal matching M' over $\bar{U}_2$. If $|M'| = \nu(\bar{U})$, then we can add the edge $(\hat{u}, \hat{v})$ to M' and would get a matching over $\bar{U}$ with size $\nu(\bar{U}) + 1$.

But $\nu(\bar{U}) = \nu(\bar{U}_1) + \nu(\bar{U}_2)$ is a contradiction to $\bar{U}$ being ν-inseparable, which concludes the proof. □

Lemma 2.14. *Given a bipartite graph $G = (U \cup V, E)$. If $\bar{U} \subseteq U$ with $|\bar{U}| > 1$ is ν-inseparable and ν-flat, then $\bar{U} = N^{-1}(N(\bar{U}))$.*

Proof. As $|\bar{U}| > 1$ and $\bar{U}$ is ν-inseparable, Lemma 2.13 grants that $\nu(\bar{U}) = |N(\bar{U})|$. Assume that $\bar{U} \subsetneq N^{-1}(N(\bar{U}))$ and let $\hat{u} \in N^{-1}(N(\bar{U})) \setminus \bar{U}$. As $N(\bar{U} \cup \{\hat{u}\}) = N(\bar{U})$ and $\nu(\bar{U}) = |N(\bar{U})|$, it follows that $\nu(\bar{U} \cup \{\hat{u}\}) = \nu(\bar{U})$ which is a contradiction to the ν-flatness of $\bar{U}$. □

One conclusion from Lemma 2.13 and Lemma 2.14 is, e. g., that formulation (2.6) from Theorem 2.10 includes the facets of the partial transversal polytope and is therefore a valid description. This can be seen as an alternate route to that conclusion.

Corollary 2.15. *Given a bipartite graph $G = (U \cup V, E)$. Then the following formulation includes all facets of the partial transversal polytope*

$$\{x : (x \in \mathbb{R}_+^{|U|}) \wedge (x \leq 1) \wedge (\forall \bar{V} \subseteq V, x(N^{-1}(\bar{V})) \leq |\bar{V}|)\}$$

Proof. First note that for $\bar{U} \subseteq U$ with $|\bar{U}| = 1$ with $\bar{U}$ being ν-flat and ν-inseparable, the facets are included in $x \leq 1$. If $|\bar{U}| > 1$, then Lemma 2.13 and Lemma 2.14 yield a set $\bar{V} = N(\bar{U}) \subseteq V$ such that $x(N^{-1}(\bar{V})) \leq |\bar{V}|$ is the facet equivalent to $x(\bar{U}) \leq \nu(\bar{U})$. Therefore all of these facets are included in the given formulation. □

In a similar way, one can show for bipartite graphs, exposing a certain structure, that the number of facets of the partial transversal polytope is polynomially bounded and that one can produce a compact formulation for it. An example for such a structure is given in the following definition.

Definition 2.16. A bipartite graph $G = (U \cup V, E)$ is *V-ordered*, iff there exists a total ordering $\leq$ on the vertices V, such that for $v_1 \in V$ and $u \in N(v_1)$ holds

$$\forall v_2 \in V_{\geq}(v_1), \quad u \in N(v_2)$$

where $V_{\geq}(v_1) := \{v : (v \in V) \wedge (v_1 \leq v)\}$. Note that the set $V_{\geq}(v_1)$ hold the vertices appearing after v_1 in the ordering.

To illustrate Definition 2.16, imagine a bipartite graph representing the matching of lectures to lecture rooms. A lecture can be matched into a room if the room is large enough to host all students in the class. In this case the ordering would be equivalent to ordering the rooms by their sizes.

For V-ordered graphs it can be shown that the number of facets of the partial transversal polytope is linearly bounded by the size of V and a simple formulation, also linearly bounded in size, containing all those facets can be given.

Theorem 2.17. *Given a bipartite graph $G = (U \cup V, E)$ that is V-ordered. Then*

$$\{x : (x \in \mathbb{R}_+^{|U|}) \wedge (x \leq 1) \wedge (\forall v \in V, x(N^{-1}(V_{\geq}(v))) \leq |V_{\geq}(v)|)\}$$

contains all facets of the partial transversal polytope.

Proof. With Lemma 2.13 and Lemma 2.14 as well as Corollary 2.15 already established, all that remains to be shown is that for $\bar{U} \subseteq U$ with $\bar{U} = N^{-1}(N(\bar{U}))$ exists $v \in V$ such that $N(\bar{U}) = V_{\geq}(v)$.

Choose $v \in N(\bar{U})$ such that $\forall v' \in N(\bar{U}), v \leq v'$, i. e., v is a minimal neighbor of $\bar{U}$ according to the ordering of V. By G being V-ordered, each $v' \in V_{\geq}(v)$ must also be a neighbor of $\bar{U}$. On the other hand $v' \in V \setminus V_{\geq}(v)$ cannot be in $N(\bar{U})$ as v was minimal with this property. Therefore $\bar{U} = V_{\geq}(v)$. □

THE PERFECTLY MATCHABLE SUBGRAPH POLYTOPE

A concept very related to partial transversals are perfectly matchable subgraphs. When working with partial transversals, we considered bipartite graphs and the subgraphs stemming from removing vertices on one (fixed) side of the graph, the focus will now be on determining which subgraphs in general will still permit a perfect matching (as opposed to partial transversals where we did not need matchings to be perfect). Again the goal will be to find a polyhedral description of the corresponding incidence vectors. The work presented here is based upon the work by Balas and Pulleyblank[17,18].

The focus will first be upon bipartite graphs, but the concepts will then be generalized to arbitrary graphs. Given some bipartite graph $G = (U \cup V, E)$ let $x \in \{0,1\}^{|U|+|V|}$ be some incidence vector of the graph's vertices. For this vector the corresponding subgraph $G(x)$ is the subgraph induced by the vertices encoded by x, i. e., $G(x) = (\bar{U} \cup \bar{V}, \bar{E})$ with

$$\begin{aligned}
\bar{U} &= \{u : (u \in U) \wedge (x_u = 1)\} \\
\bar{V} &= \{v : (v \in U) \wedge (x_v = 1)\} \\
\bar{E} &= \{\{u, v\} : (\{u, v\} \in E) \wedge (u \in \bar{U}) \wedge (v \in \bar{V})\}.
\end{aligned}$$

Now our interest will be in determining whether such an incidence vector induces a subgraph for which a perfect matching exists or not. Let $\mathcal{P}(G)$ be the set containing all incidence vectors which do induce such a perfectly matchable subgraph. As with the partial transversal polytope one can find a full description of this set's convex hull (which will be denoted as the perfectly matchable subgraph polytope):

Theorem 2.18. *Given a bipartite graph $G = (U \cup V, E)$. Then the perfectly matchable subgraph polytope is fully described by*

$$\text{conv}(\chi_{\mathcal{P}(G)}) = \{x : (2.8) \wedge (x(U) = x(V)) \wedge (x \leq 1) \wedge (x \in \mathbb{R}_+^{|U|+|V|})\}$$

with (2.8) being

$$\forall \bar{U} \subseteq U, \quad x(\bar{U}) \leq x(N(\bar{U})). \tag{2.8}$$

Proof. See Balas and Pulleyblank (1983)[17], who give three different proofs for this theorem. □

Note how the polytope given in Theorem 2.18 is very similar to the partial transversal polytope from Theorem 2.9, especially when using the constraints (2.7) for its description. This similarity is not arbitrary as both are based upon the König-Hall Theorem (Theorem 2.6). In the perfectly matchable subgraph case one additionally needs a constraint ensuring that on both sides of the graph the same number of vertices is present (a trivial requirement for a perfect matching to exist) and the constraints (2.8) need to count the vertices on the right hand side, which are fixed when dealing with the partial transversal polytope.

When the underlying graph is not bipartite ($G = (V, E)$) additional effort is required, as the graph may exhibit complicating structures (which in this case are odd cycles). It was shown by Balas and Pulleyblank (1989)[18] that Theorem 2.18 can be generalized to arbitrary graphs. To do so we require the following subsets of vertices:

$$\begin{aligned}\mathcal{D}(G) = &\{\bar{V} : (\bar{V} \subseteq V) \wedge (\forall e \in E, |e \cap \bar{V}| \leq 1)\} \\ &\cup \{\bar{V} : (\bar{V} \subseteq V) \wedge (|\bar{V}| \text{ is odd}) \wedge (\bar{V} \text{ contains odd cycle})\}.\end{aligned}$$

The set $\mathcal{D}(G)$ contains all independent vertex sets as well as the vertex subsets of odd size which contain an odd cycle, i. e., which do not correspond to a bipartite subgraph. For some vertex subset $\bar{V} \subseteq V$ let $\mathcal{C}^{\max}(\bar{V})$ denote the set of maximal connected components in the subgraph induced by $\bar{V}$.

Using this it is possible to define the perfectly matchable subgraph polytope for an arbitrary graph:

Theorem 2.19. *Given a graph $G = (V, E)$. Then the perfectly matchable subgraph polytope is fully described by*

$$\text{conv}(\chi_{\mathcal{P}(G)}) = \{x : ((2.9)) \wedge (x \leq 1) \wedge (x \in \mathbb{R}_+^{|V|})\}$$

with (2.9) being

$$\forall \bar{V} \in \mathcal{D}(G), \quad x(\bar{V}) - x(N_G(\bar{V})) \leq |\bar{V}| - |\mathcal{C}^{\max}(\bar{V})|. \tag{2.9}$$

Proof. See Balas and Pulleyblank (1989)[18]. □

Theorem 2.19 generalizes the preceding Theorem 2.18 and the later one can be easily deduced from the more general setting.

In the applications considered in this thesis, only the partial transversal polytope will be required to model the corresponding problems. But in similar applications it might easily happen that perfect matchability is required, or that the underlying graph will not be bipartite. In such cases it may be possible to extend the concepts presented here appropriately to suit the respective setting.

BIPARTITE HYPERGRAPH MATCHINGS

The definition of a matching can be directly applied to hypergraphs, i. e., taking a subset of edges such that each vertex is covered by at most one edge. But while calculating most matching problems to optimality can be easily done on non hypergraphs, it turns out that matchings on hypergraphs are generally NP-complete – e. g., by reduction from 3 dimensional matching (see Karp (1972)[100]). A subclass of hypergraphs are bipartite hypergraphs which were defined in Section 1.3. Matchings in bipartite hypergraphs find application, for example, in timetabling problems (see Chapter 6). As this thesis aims at solving exactly such problems, the focus of this section will be upon bipartite hypergraphs. Efficient algorithms for exploiting bipartite hypergraph structures will be described in Section 4.2.

As defined in the introduction (see Section 1.1) a bipartite hypergraph $G = (U \cup V, E)$ is a hypergraph such that every edge contains exactly one vertex from U, i. e., for $e \in E$ we have $|e \cap U| = 1$. A matching in a (bipartite) hypergraph is defined exactly as for a regular graph, i. e., a subset $M \subseteq E$ where distinct edges $e_1, e_2 \in M, e_1 \neq e_2$ do not share a vertex, i. e., $e_1 \cap e_2 = \emptyset$. Matchings in bipartite hypergraphs (and therefore also for hypergraphs in general) are NP-complete, which can be easily shown by reduction from 3-dimensional matching, a problem that is among Karp's original 21 NP-complete problems[100]. For 3-dimensional matching the underlying hypergraph $G = (U \cup V \cup W, E)$ consists only of edges in $U \times V \times W$, implying that G is also a bipartite hypergraph. Checking whether a subset of a given size is a matching in a hypergraph can easily be done in polynomial time by testing the elements for disjointness, implying that bipartite hypergraph matching is indeed NP-complete.

The concept of a partial transversal can be extended to bipartite hypergraphs as follows: a subset $\bar{U} \subseteq U$ is a hypergraph partial transversal iff there exists a matching M such that every element in $\bar{U}$ is covered by an edge in M. Again one might wonder if the polytope of characteristic vectors in $\{0,1\}^{|U|}$, that are defining hypergraph partial transversals, can be described by inequalities similar to the partial transversal polytope. While this question is still open, so far it can be easily shown that a description similar to the polymatroid formulation from Corollary 2.12 yields a polytope where the integer points are exactly the incidence vector we were looking for:

Theorem 2.20. *Given a bipartite hypergraph $G = (U \cup V, E)$, define the polytope*

$$Q_{\mathcal{T}(G)} = \{x : (x \in \mathbb{R}_+^{|U|}) \wedge (\forall \bar{U} \subseteq U, x(\bar{U}) \leq \nu(\bar{U}))\}.$$

Then $x \in Q_{\mathcal{T}(G)}$ and $x \in \mathbb{Z}^{|U|}$ is equivalent to $x \in \chi_{\mathcal{T}(G)}$.

Proof. First note that for $\bar{U}_1 \subseteq \bar{U}_2 \subseteq U$ it holds that $\nu(\bar{U}_1) \leq \nu(\bar{U}_2)$ as the size of a maximum matching can only increase when adding vertices to the corresponding subgraph. Furthermore, the subset of a partial hypergraph transversal is again a partial hypergraph transversal (as the corresponding matching can be derived by removing edges from the previous one).

Let $x \in \chi_{\mathcal{T}(G)}$ be an incidence vector of a partial hypergraph transversal and $\bar{U}_x = \{u : x_u = 1\}$ be the set of vertices encoded by x. Assume there exists a vertex subset $\bar{U} \subseteq U$ such that $x(\bar{U}) > \nu(\bar{U})$. Now it holds that

$$x(\bar{U} \cap \bar{U}_x) = x(\bar{U}) > \nu(\bar{U}) \geq \nu(\bar{U} \cap \bar{U}_x)$$

which implies that $\bar{U} \cap \bar{U}_x$ is not a partial hypergraph transversal. This in turn implies that $\bar{U}_x$ is not a partial hypergraph transversal, which is a contradiction and therefore $x \in Q_{\mathcal{T}(G)}$

Next let $x \in Q_{\mathcal{T}(G)}$ and $x \in \mathbb{Z}^{|U|}$. As for $u \in U$ we have the constraint $x_u \leq \nu(\{u\}) \leq 1$, it follows that $x \in \{0,1\}^{|U|}$ and therefore x is in fact an incidence vector. Let again $\bar{U}_x = \{u : x_u = 1\}$ be the vertex set encoded by x. Let M be a matching of size $\nu(\bar{U}_x)$ over U_x. As $|\bar{U}_x| = x(\bar{U}_x) \leq \nu(\bar{U}_x)$ the matching must contain at least $|\bar{U}_x|$ vertices and therefore cover $\bar{U}_x$ completely, implying that U_x is a partial hypergraph transversal and therefore $x \in \chi_{\mathcal{T}(G)}$.

□

While bipartite hypergraph matching is NP-complete in general, the question remains whether some relevant special cases might be solvable in polynomial time. It will turn out that even very restricted cases are already NP-complete. On the other hand hypergraphs with applications in timetabling (see Chapter 6) tend to be very close to the edge dividing hypergraph matchings in P and those that are NP-complete. In order to explore the border between the complexities, the following classes of restrictions will be considered in various combinations:

- edge cardinality
- node degree
- k-partite – the nodes of V can be partitioned into sets $V_1, \ldots, V_{k-1}$, such that every edge contains at most one vertex from each set (but still exactly one from U). Note that this k-partiteness is in addition to the already bipartite hypergraph where U is always its own partition. Bipartiteness in the sense of this thesis shall not be confused with 2-partite (which would be a regular bipartite graph)
- consecutiveness – the nodes of V can be ordered linearly such that for all $e \in E$ the nodes $e \cap V$ are consecutive in that order, without interruption
- layered – we can partition the nodes of V into sets $V_1, \ldots, V_k$ such that every edge only covers one of the partitions: $e \in E \Rightarrow \exists i, e \subseteq U \cup V_i$

The three last classes of restrictions will make especially sense in the context of timetabling problems (see Chapter 6) where events need to be matched to timeslots and rooms, and we have on the one side vertices for each event and on the other side vertices for each combination of a timeslot and a room. In these settings we will have a layer per room. When k timeslots are available, the problem will be k-partite. And when events span multiple timeslots these will usually be consecutive (ordered by the starting time of the corresponding timeslot). Also many instances will feature edge cardinalities that are mainly between 2 and 3. In Chapter 6 this application will be further explored, but the reader might find it helpful to keep these concepts in mind as a motivation for the preceding definitions.

EDGES OF CARDINALITY 2 OR 2-PARTITE This is the ordinary bipartite matching and polynomially solvable, e. g., by one of the algorithms mentioned in Section 2.2.

EDGES OF CARDINALITY 3, 3-PARTITE This variant is NP-complete, as it is exactly the aforementioned 3-dimensional matching.

NODES OF DEGREE AT MOST 2, 3-PARTITE This case is polynomially solvable (see Garey and Johnson (1979)[77], page 221, problem SP1.

NODES OF DEGREE AT MOST 3, 3-PARTITE This case is again NP-complete (see Garey and Johnson (1979)[77], page 221, problem SP1).

NODE DEGREE IN U IS 1 AND 2 IN V This case is NP-complete by reduction from independent set. An instance of independent set is given by a regular graph $G = (U, E)$. The decision problem is, whether for some positive integer k there exists an independent set, i. e., a subset $\bar{U} \subseteq U$ such that no two vertices share an edge ($u, v \in \bar{U} \Rightarrow \{u, v\} \notin E$), of cardinality $|\bar{U}| = k$. This problem is NP-complete[77].

For the reduction we construct a bipartite hypergraph $G_H = \{U \cup E, F\}$, where $F = \{\{u\} \cup (\bigcup_{u \in e \in E} \{e\}) : u \in U\}$. So for every vertex there is an edge containing that vertex alongside all its connected edges. Now every independent set in G directly translates into a matching in G_H and vice versa. The hypergraph G_H fulfills the required properties, as every vertex in U is covered by exactly one edge and every vertex in E (note that E are vertices in G_H while being edges in G) is covered by exactly two edges.

NODE DEGREE IN V IS 1 This case is polynomially solvable. For each node in U we chose an arbitrary edge. This is possible as no other edge can share the nodes from V.

NODE DEGREE IN U IS 1, CONSECUTIVE This case can be solved similar to interval coloring. If we ignore the nodes from U (each of them has only one edge anyhow so it does only matter if that edge is chosen, not which one) the edges can be represented as intervals due to the consecutive ordering of the V nodes. Now we can find a matching if we can color those intervals with 1 color which can be done in time $\mathcal{O}(|V|)$ by inspecting every vertex for multiple edges.

CONSECUTIVE, LAYERED This case is NP-complete. The reduction is based upon the list coloring problem on interval graphs, for which an NP-completeness proof was done by Biró et al. (1992)[30]. A list coloring on interval graphs instance is given by a set I of intervals $(s, e) \in I$ where $s, e \in T$ are start and end timeslots drawn from a finite set $T \subseteq \mathbb{N}$ of timeslots such that $s \leq e$. For intervals (s_1, e_1) and (s_2, e_2) let $(s_1, e_1) \cap (s_2, e_2)$ be the intersection of the two intervals (i. e., if $s_1 \leq s_2$ this is $\{t : s_2 \leq t \leq e_1\}$ and correspondingly in the case $s_2 \leq s_1$). Furthermore we are given a set C of colors and each interval $i \in I$ has a list of colors $C_i \subseteq C$ to which it may be assigned. The goal is to find an assignment $m : I \to C$ such that each interval is assigned a color from its list ($m(i) \in C_i$) and such that for two intervals with the same color the intervals do not intersect ($(i_1 \neq i_2 \wedge m(i_1) = m(i_2)) \Rightarrow (i_1 \cap i_2 = \emptyset)$).

Now construct the following bipartite hypergraph matching instance. Let $G = (I \cup T \times C, E)$ with edges defined as

$$E = \{\{i\} \cup \{(c,t) : (t \in T) \wedge (s_i \leq t \leq e_i)\} : ((s_i, e_i) = i \in I) \wedge (c \in C_i)\}.$$

Note that we get one hyperedge per interval and feasible color, therefore the number of edges is bounded by $|I| \cdot |C|$. Each matching in G assigns to the matched intervals a color and the properties of the matching ensure that no overlapping intervals receive the same color. Equivalently an assignment of colors corresponds to a feasible matching in G. Therefore the list coloring problem can be solved iff G has a maximum matching of size $|I|$.

POPULAR MATCHINGS

This section will take a closer a look at a problem variant of the classical matching that uses a quite different objective than the simple maximization of the number of matched vertices or the weight of the chosen edges. The work presented in this section was conducted together with Oliver Scheel, a bachelor student of this thesis' author (also see Oliver Scheels bachelor's thesis[144]). The concept of popular matchings was introduced by Gärdenfors (1975)[76] in the context of the stable marriage problem. The first polynomial time algorithm, as well as much of theoretical results needed for our work, were introduced by Abraham et al. (2007)[2].

The concern of popular matchings is to to find a matching such that for no other matching more vertices would be better off than in the present one. The satisfaction of a vertex will be measured relative to the objective the vertex could achieve in other matchings. Given a weighted bipartite graph $G = (U \cup V, E)$ where, for example, U is a set of workers who shall each be matched to a job from V. An edge $\{u, v\}$ indicates that $u \in U$ can perform job $v \in V$ and the weight $w_{\{u,v\}}$ tells us the priority with which u would like to be matched to v, lower priorities indicating higher satisfaction with the job. Now a worker from U would prefer a matching M_1 over a matching M_2 if she is assigned a lower weight edge in M_1 than in M_2 or if she was not matched in M_2 at all but in M_1. Note that only one side of the bipartite graph will be considered when comparing the matchings quality – in the example the jobs from V do not utter any preferences. In the bipartite graphs the set of vertices, which is giving the preferences, shall be denoted as U.

In order to define popular matchings more formally, let us be given a bipartite graph $G = (U \cup V, E)$. There are two variants of the problem:

WITHOUT TIES There may not be two edges with the same weight adjacent to one $u \in U$: $\forall u \in U, v_1, v_2 \in V, (\{u, v_1\}, \{u, v_2\} \in E) \wedge (v_1 \neq v_2) \Rightarrow w_{\{u,v_1\}} \neq w_{\{u,v_2\}}$

WITH TIES There may be ties and therefore there are no restrictions on the edge weights.

One possible (different than using popular matchings) way to model such a problem would be by calculating a minimum weight maximum matching, which would minimize the total amount of priorities. A drawback of this model is, that for the benefit of a single vertex, many others might be worse off than they could be, if the other vertex were excluded.

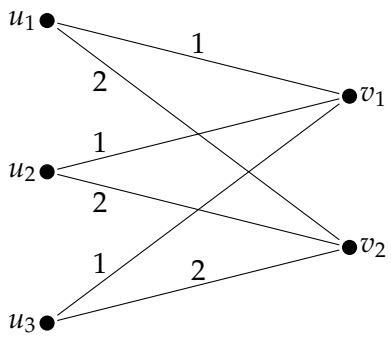


Figure 5: An example of a graph without popular matching

Furthermore in many situations there is no absolute scale for measuring the vertices individual priorities. Minimizing the total sum of the matchings weight implicitly assumes that all weights use the same absolute scale, which often is not the case when the vertices are allowed to specify their priorities themselves.

To circumvent these shortcomings, one can measure the popularity of one matching over another one as the relative number of vertices from U preferring the matching over the other one. A vertex $u \in U$ will prefer a matching M_1 over a matching M_2 if either it is not matched in M_2, i. e., $\forall v \in V, \{u, v\} \notin M_2$ or if it is matched with lower weight in M_1, i. e., $(\{u, v_1\} \in M_1) \wedge (\{u, v_2\} \in M_2) \wedge (w_{\{u,v_1\}} < w_{\{u,v_2\}})$. Now a matching M_1 is more popular than a matching M_2 if more members of U prefer M_1 over M_2 than the other way round.

A matching M is called popular if there is no matching M' that is more popular than M. Note that a popular matching does not need to exist in a given graph. Such a case is illustrated in Figure 5 where 3 members of U have the exact same preferences. Now we can see that matching $M_2 = \{\{u_2, v_1\}, \{u_3, v_2\}\}$ is more popular than $M_1 = \{\{u_1, v_1\}, \{u_2, v_2\}\}$ as M_2 is preferred by u_2 and u_3. But $M_3 = \{\{u_3, v_1\}, \{u_1, v_2\}\}$ is more popular than M_2 as M_3 which is preferred by u_1 and u_3. In turn M_1 is preferred by u_1 and u_2 over M_3. Due to the graph's symmetry we get the same results for other permutations.

The work by Abraham et al. (2007)[2] gives some detailed properties that a popular matching will exhibit (if it exists). These characterizations slightly differ for matchings with and without ties. For the main result of this section only matchings without ties are relevant (as matchings with ties are more general and the result can be easily carried over). For a popular matching in a bipartite graph $G = (U \cup V, E)$ without ties, for each vertex in U only two edges are actually relevant. The first one will be the most preferred one for the vertex. The corresponding vertex from V will be denoted as the f-vertex:

$$f(u) = \arg\min_{v \in N(u)} w_{\{e,v\}}.$$

The second relevant edge will be the most preferred one which is not connected to some f-vertex (of any vertex in U). The corresponding vertex from V will be denoted as the s-vertex. Note that there might not exist any such vertex in which case a virtual s-vertex will be introduced. Matchings to such vertices will later

represent popular matchings where the corresponding vertices from U are left unmatched.

$$s(u) = \begin{cases} \arg\min_{v \in N(u) \setminus \{v : \exists u, f(u) = v\}} w_{\{v,u\}} & \text{if } N(u) \setminus \{v : \exists u, f(u) = v\} \neq \emptyset \\ v_u^{\text{virt}} & \text{otherwise.} \end{cases}$$

The following theorem was shown by Abraham et al. (2007)[2], yielding that one can restrict the search for a popular matching to graphs consisting only of U and their f- and s-vertices:

Theorem 2.21. *Given a bipartite graph $G = (U \cup V, E)$ without ties. Let a corresponding helper graph $G_h = (U \cup V_h, E_h)$ be defined with*

$$V_h = \bigcup_{u \in U} \{f(u), s(u)\}$$

and

$$E_h = \bigcup_{u \in U} \{\{u, f(u)\}, \{u, s(u)\}\}.$$

A matching $M \subseteq E$ is a popular matching in G if and only if $M \cup \{\{u, v_u^{virt}\} : (u \in U) \wedge (\nexists e \in M, u \in e)\}$ is a matching in G_h covering all u and all f-vertices.

Proof. See Abraham et al. (2007)[2], theorem 2.5. □

By Theorem 2.21 one can search for a popular matching by finding a regular matching (fulfilling the requirements from the theorem) in the helper graph G_h. This makes it much easier to detect graphs where no popular matching is possible. As a side effect the theorem can also be used to efficiently compute popular matchings (or prove the non existence of such). This can be done by calculating a maximum matching in G_h. Note that in case not all f-vertices are covered, one can always exchange the edges in the matching such that a vertex, having the non covered f-vertex as its f-vertex is matched to this vertex instead of the s-vertex it was previously matched to. In the example shown in Figure 5 one can now easily see that no popular matching exists as the helper graph will look exactly like the original graph but has three vertices in U but only two in V_h.

The larger the graph is, the more likely it becomes that it exhibits a substructure preventing a popular matching (see Scheel (2014)[144] for an experimental study on this effect). Removing nodes from U might fix such issues – in Figure 5 removing any vertex from U would again allow a popular matching. Let therefore a subset $\bar{U} \subseteq U$ be called popular matchable iff there exists a popular matching in the subgraph $G(\bar{U} \cup V)$.

As the concept of a popular matchable subset looks a lot like a modified version of the partial transversal, one might wonder whether the polytope that contains exactly the incidence vectors of the popular matchable subsets can be described in a similar way as the partial transversal polytope. Let $\chi_{\text{pop}(G)} \subseteq \{0,1\}^{|U|}$ be the set of incidence vectors corresponding to subsets of U, for which the corresponding subgraph of G has a popular matching. Now our goal is to find a simple description of $\text{conv}(\chi_{\text{pop}(G)})$ (which will be called the popular partial transversal polytope) similar to that one of the partial transversal polytope.

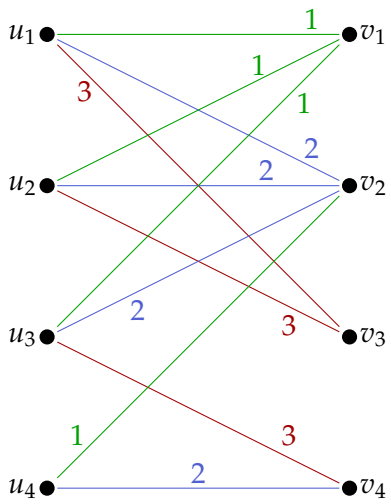


Figure 6: An example graph where knapsack constraints are insufficient to describe the popular partial transversal polytope

It turns out that such a description must be more complex than it was for partial transversals. The following counter example shows that constraints of the knapsack type are not sufficient. It was developed together with my bachelor student Oliver Scheel and can also be found in his bachelor's thesis[144].

Theorem 2.22. *There exists a bipartite graph $G = (U \cup V, E)$ without ties such that $\operatorname{conv}(\chi_{pop(G)})$ cannot be described exclusively with constraints of the knapsack type (i. e., constraints of the form $a^\intercal x \leq b$ for $a \geq 0$ and $b \geq 0$).*

Proof. Consider the graph $G = (U \cup V, E)$ shown in Figure 6 which is described by the vertices and edges

$$U = \{u_1, u_2, u_3, u_4\}, \quad V = \{v_1, v_2, v_3, v_4\}$$
$$E = \{\{u_1, v_1\}, \{u_1, v_2\}, \{u_1, v_3\}, \{u_2, v_1\}, \{u_2, v_2\}, \{u_2, v_3\}, \{u_3, v_1\}, \{u_3, v_2\}, \{u_3, v_4\}, \{u_4, v_2\}, \{u_4, v_4\}\}$$

with edge weights

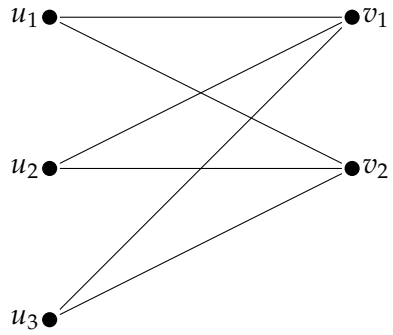
$$w_{\{u_1,v_1\}} = w_{\{u_2,v_1\}} = w_{\{u_3,v_1\}} = w_{\{u_4,v_2\}} = 1$$
$$w_{\{u_1,v_2\}} = w_{\{u_2,v_2\}} = w_{\{u_3,v_2\}} = w_{\{u_4,v_4\}} = 2$$
$$w_{\{u_1,v_3\}} = w_{\{u_2,v_3\}} = w_{\{u_3,v_4\}} = 3.$$

Now consider the subgraph resulting when only choosing the vertices $\{u_1, u_2, u_3\}$ from U. For this subgraph the corresponding helper graph is shown in Figure 7a. By Theorem 2.21 no popular matching can exist as there are more vertices from U present than from V.

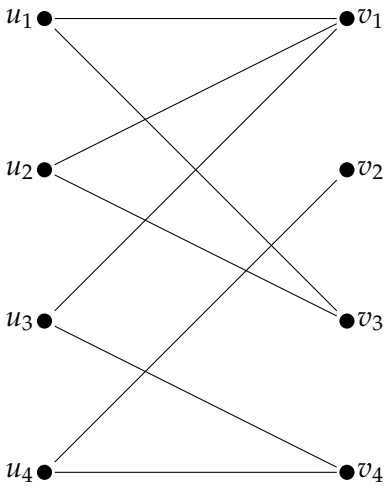
On the other hand for the entire graph with all U vertices, the helper graph is shown in Figure 7b. Here a popular matching does exist, e. g.,

$$M = \{\{u_1, v_1\}, \{u_2, v_3\}, \{u_3, v_4\}, \{u_4, v_2\}\}$$

which means that the addition of vertex u_4 actually makes a popular matching possible that would not have existed without it.



(a) The helper graph for the subgraph with vertices u_1, u_2, and u_3



(b) The helper graph for the entire original graph

Figure 7: Helper graphs for two subsets of the U vertices

The subset $\{u_1, u_2, u_3\}$ is described by the incidence vector $(1, 1, 1, 0)^\intercal$ while that of U is described by $(1, 1, 1, 1)^\intercal$. Now for $a \in \mathbb{R}_+^{|U|}$ it holds that

$$a^\intercal(1, 1, 1, 0)^\intercal \leq a^\intercal(1, 1, 1, 1)^\intercal$$

which implies that no knapsack constraint of the form $a^\intercal x \leq b$ can at the same time be valid for $(1, 1, 1, 1)^\intercal$ and violated by $(1, 1, 1, 0)^\intercal$. □

The counter example from Theorem 2.22 also serves as a case showing that the popular partial transversal polytope cannot be a polymatroid as the constraints describing a polymatroid are a subset of the knapsack type constraints. While so far the description of the popular partial transversal polytope is not known, it is now clear that finding it will require a different approach than finding the partial transversal polytope.

3 DECOMPOSITION METHODS

In practice optimization problems can easily become very large, often to a point where simply putting an MILP model of it into a general purpose solver will not yield good results in a reasonable amount of time. But such problems often have a lot of inherent structure which can be exploited to solve them more efficiently. For example if one ignores certain constraints or fixes a set of variables, the resulting formulation might become much easier to solve (e. g., by using a special purpose combinatorial algorithm). This smaller problem will be denoted as the subproblem. In order to find an optimal solution to the original model one needs another stage where either the ignored constraints are taken into account or the variable fixings are determined. This stage will be denoted as the master problem. Generally the master problem will have to make several calls to the subproblem in order to achieve optimality.

Usually it is not trivial to determine a good decomposition, i. e., a selection of variables or constraints which shall be deferred to the subproblem, for a given formulation. Obviously relaxing all or no constraints, or fixing all or no variables does not yield any benefit. But between these two extrema may exist several meaningful decompositions which all pose a trade off between the time needed for solving the subproblems versus the effort put in the master problem.

This chapter will give a short introduction into the two decomposition schemes known as the Dantzig-Wolfe decomposition[56] and its dual, the Benders' decomposition[24]. Dantzig-Wolfe deals with complicating constraints by ignoring them in the subproblem which then communicates its solutions back to the master problem by a technique called column generation. In the Benders' setting the subproblem is derived by fixing complicating variables and the information of the subproblem is incorporated in the master problem by adding additional constraints. In both cases one first derives a reformulation of the original problem that is based upon the extreme points and rays describing the subproblem (see Theorem 1.6). This reformulation is then solved via implicitly enumerating those extreme points and rays.

Even though these decomposition schemes originate from pure LPs, they often really start to shine when applied to MILPs. Each of the two techniques will first be introduced for the LP case and then adapted to the MILP setting.

FURTHER READING

This chapter only covers the concepts necessary for a proper understanding of the following topics of this thesis. In order to further delve into the details of decompositions and related concepts, the reader shall be referred to the following literature:

- The book "Column Generation"[63] explains the technique of column generation from the very basics to some recent developments.

- Chapter 13 of the book "50 Years of Integer Programming"[97] gives a good overview about Dantzig-Wolfe and Benders' decomposition, column generation, and the Lagrangean relaxation.

DANTZIG-WOLFE DECOMPOSITION

Our aim is to solve an MILP of the form

$$\max\{c^\intercal x : (Ax \leq b) \wedge (Dx \leq d) \wedge (x \in X)\} \tag{3.1}$$

where X can be $\mathbb{R}^n_+$, $\mathbb{Z}^n_+$ or $\mathbb{R}^{n_1}_+ \times \mathbb{Z}^{n_2}_+$. Matrix D is assumed to be of dimension $m_1 \times n$ and A of dimension $m_2 \times n$.

In many applications $\{x : (Dx \leq d) \wedge (x \in X)\}$ is chosen such that solving an optimization problem on this domain is easy, while A and b encode complicating constraints. As stated before, it is not immediately clear what an easy problem is. A family of subproblems that is noteworthy are those that have block diagonal structure. The clear advantage this provides is that blocks can be treated separately from each other. Later in this section this case will be covered in more detail.

For the start we will look at the case where $X = \mathbb{R}^n_+$, i. e., we deal with a regular linear program. In Section 3.3 this will then be extended to the MILP setting.

Solving the LP relaxation of (3.1) means optimizing a linear function over the intersection of two polyhedra. The constraints given by D and d define the following polyhedron:

$$H_{\text{sub}} = \{x : (Dx \leq d) \wedge (x \in \mathbb{R}^n_+)\}.$$

The basic building block for Dantzig-Wolfe as well as Benders' decomposition is the Minkowski-Weyl Theorem (Theorem 1.6) which gives us a representation of a polyhedron by a finite set of extreme points (P) and rays (R):

$$H_{\text{sub}} = \{\sum_{p\in P} \lambda_p p + \sum_{r\in R} \mu_r r : (\sum_{p\in P} \lambda_p = 1) \wedge ((\lambda, \mu) \in \mathbb{R}^{|P|+|R|}_+)\}.$$

Under the assumption that we have all of these extreme points P and rays R at hand one can now write our original problem as

$$\begin{aligned} \max \ & \sum_{p\in P} (c^\intercal p)\lambda_p + \sum_{r\in R} (c^\intercal r)\mu_r & \\ \text{s.t} \ & \sum_{p\in P} (Ap)\lambda_p + \sum_{r\in R} (Ar)\mu_r & \leq b \\ & \sum_{p\in P} \lambda_p & = 1 \\ & (\lambda, \mu) & \in \mathbb{R}^{|P|+|R|}_+. \end{aligned} \tag{3.2}$$

This new formulation is called the master problem. Of course stated this way the new formulation seems to actually make things worse as we now have to enumerate all extreme points and rays of H_{sub} in order to write down the master problem. As these may be many, one needs a way to shortcut this. The method of choice for achieving this is delayed column generation (or just column generation) which will be explained in the next paragraph.

Column generation

For an algorithm that is able to solve the master problem (3.2) to be of practical usefulness, it is usually crucial to avoid enumerating all the extreme points and rays. The trick to achieve this is to only work with a subset of P and R, and adding more points and rays only when they are needed. So assume we have some subsets $\bar{P} \subseteq P$ and $\bar{R} \subseteq R$ which may also be empty. When only working with these we get the so called restricted master problem:

$$\begin{aligned} \max \ & \sum_{p\in\bar{P}} (c^\intercal p)\lambda_p + \sum_{r\in\bar{R}} (c^\intercal r)\mu_r & & \\ \text{s.t} \ & \sum_{p\in\bar{P}} (Ap)\lambda_p + \sum_{r\in\bar{R}} (Ar)\mu_r & \leq b & \\ & \sum_{p\in\bar{P}} \lambda_p & = 1 & \\ & (\lambda, \mu) & \in \mathbb{R}_+^{|\bar{P}|+|\bar{R}|}. & \end{aligned} \tag{3.3}$$

The restricted master problem can be solved as a regular linear program. Of course an optimal solution (λ^*, μ^*) of (3.3) is not necessarily also an optimal solution to the master problem (3.2). Furthermore the restricted master problem might even be infeasible while the master problem is not. In order to find out whether we need to add some extreme point or ray to $\bar{P}$ or $\bar{R}$ we take a look at the dual of (3.2):

$$\begin{aligned} \min \ & b^\intercal \pi + \rho & & & \\ \text{s.t} \ & (Ap)^\intercal \pi + \rho & \geq c^\intercal p & \quad \forall p \in P & (3.4) \\ & (Ar)^\intercal \pi & \geq c^\intercal r & \quad \forall r \in R & (3.5) \\ & (\pi, \rho) \in \mathbb{R}_+^{m_2} \times \mathbb{R}. & & & \end{aligned}$$

Of course when removing all constraints but those corresponding to the points and rays in $\bar{P}$ and $\bar{R}$ from this formulation, we get the dual of the restricted master problem.

First assume that we have an optimal solution (λ^*, μ^*) of (3.3). Its corresponding dual solution be (π^*, ρ^*). If this dual solution satisfies all constraints (3.4) and (3.5) then (λ^*, μ^*) must also be optimal for the master problem. Otherwise the violated constraints correspond to extreme points and rays from which we need to add at least one to $\bar{P}$ or $\bar{R}$. That means we want to either find a $p' \in P$ or an $r' \in R$ for which the corresponding inequality holds:

$$\begin{aligned} (c^\intercal - \pi^\intercal A)p' &> \rho \\ (c^\intercal - \pi^\intercal A)r' &> 0. \end{aligned}$$

This can be achieved by solving the linear program

$$\max \{(c^\intercal - \pi^\intercal A)x : x \in H_{\text{sub}}\}. \tag{3.6}$$

The linear program (3.6) is often also referred to as the pricing problem but in order to keep the terminology consistent together with the Benders' decomposition, it will be called subproblem in this thesis.

Now let us look at the possible results we can get from solving the subproblem. If the subproblem is infeasible, then H_{sub} must be empty. Therefore there also exists no solution to our original problem. Next if (3.6) is unbounded we can get an extreme ray r' of H_{sub} such that $(c^\intercal - \pi^\intercal A)r' > 0$ which we can then add to $\bar{R}$. The last possibility is that the subproblem yields an optimal solution p'. If $(c^\intercal - \pi^\intercal A)p' > \rho$, then we add p' to $\bar{P}$. Otherwise we can conclude that there are no violated constraints of the dual master problem. In that case we have found an optimal solution to the original problem. Note that optimizing (3.6) ensures that we do not miss any extreme point or ray that might be valuable.

Lastly let us look at the case where (3.3) turns out to be infeasible. This might mean that the entire problem is infeasible but it also could (and often does) mean that we need to add variables in order to reach a first feasible solution. Such extreme rays or points can be identified by solving a problem similar to (3.6). Instead of the dual solution we use Farkas' Lemma (Lemma 1.12) to derive a vector (π, ρ) satisfying

$$\begin{aligned} &\forall p \in \bar{P}, (Ap)^\intercal \pi + \rho \geq 0 \\ &\forall r \in \bar{R}, (Ar)^\intercal \pi \geq 0 \\ &\qquad \pi \geq 0 \\ &\qquad d^\intercal \pi + \rho < 0. \end{aligned} \tag{3.7}$$

To see that such a (π, ρ) has to exist if (3.3) is infeasible while (3.2) has a feasible solution, we look at the standard form of the constraints

$$\begin{aligned} \sum_{p \in \bar{P}} (Ap)\lambda_p + \sum_{r \in \bar{R}} (Ar)\mu_r + s &= b \\ \sum_{p \in \bar{P}} \lambda_p &= 1 \\ (\lambda, \mu, s) &\in \mathbb{R}_+^{|\bar{P}|+|\bar{R}|+m_2} \end{aligned}$$

which fulfills the criteria of Lemma 1.12 and therefore a (π, ρ) satisfying (3.7) has to exist.

Now we need to solve a slightly modified subproblem (also known as Farkas pricing):

$$\min \{(\pi^\intercal A)x : x \in H_{\text{sub}}\}. \tag{3.8}$$

This is effectively the old sub problem where the c coefficients are set to 0. Again if (3.8) is unbounded we add a corresponding extreme ray to $\bar{R}$ and if there is an optimal solution x^* with $(\pi^\intercal A)x^* < -\rho$, it is added to $\bar{P}$. Otherwise the master problem turns out to be infeasible, as there exists no extreme point or ray that is able to cut off the Farkas vector (π, ρ) from our dual problem.

When implementing a column generation algorithm, one can usually rely on the dual values as well as an appropriate Farkas' vector being provided by the linear programming solver.

Blockdiagonal subproblem

It is very common to look for subproblems which have block diagonal structure. This provides two main advantages. First the blocks can be solved completely independent which means that we can solve many small problems instead of one large

one. Especially when the subproblems are NP-hard (see Section 3.3) a reduction in problem size is expected to provide a huge advantage in terms of algorithm runtime. The second advantage is the fact that the extreme points and rays of the subproblem can be split into smaller components. This way the master problem "can choose on its own" how to combine the partial extreme points and rays which may lead to fewer calls of the subproblem.

If D has a block diagonal structure and is split into k blocks the original problem will be of the form:

$$\max \left\{ \sum_{i\in[k]} c^{i\intercal} x^i : (\sum_{i\in[k]} A_i x^i \leq b) \wedge (\forall i \in [k], D_i x^i \leq d^i) \wedge (x^i \in X_i) \right\}.$$

Every x^i now has its own subproblem $H^i_{\text{sub}} = \{x : (D_i x \leq d^i) \wedge (x \in X_i)\}$ which defines the feasibility for the subproblem constraints. Therefore in the decomposition we do not need to consider the extreme points from the overall subproblem $H_{\text{sub}} = H^1_{\text{sub}} \times H^2_{\text{sub}} \times \ldots \times H^k_{\text{sub}}$ but we can look at the smaller polytopes separately. Let P^i and R^i be the respective sets of extreme points and rays for H^i_{sub}. Then our master problem becomes

$$\begin{aligned}
\max \ & \sum_{i\in[k]} (\sum_{p\in P^i} (c^{i\intercal} p)\lambda^i_p + \sum_{r\in R^i} (c^{i\intercal} r)\mu^i_r) & & & \\
\text{s.t} \ & \sum_{i\in[k]} (\sum_{p\in P^i} (A_i p)\lambda^i_p + \sum_{r\in R^i} (A_i r)\mu^i_r) & \leq b & & \\
& \sum_{p\in P^i} \lambda^i_p & = 1 & \forall i \in [k] & \\
& (\lambda, \mu) & \in \mathbb{R}_+^{\sum_{i\in[k]}(|P^i|+|R^i|)}. & &
\end{aligned} \tag{3.9}$$

Again, due to the potentially large number of variables this model is often solved using column generation. As before we start with a restricted master problem which is again equivalent to (3.9) but with subsets $\bar{P}^i \subseteq P^i$ and $\bar{R}^i \subseteq R^i$ of the extreme points and rays. The subproblem can again be determined by looking at the dual of the master problem

$$\begin{aligned}
\min \ & b^\intercal \pi + \sum_{i\in[k]} \rho_i & & & \\
\text{s.t} \ & (A_i p)^\intercal \pi + \rho_i & \geq c_i^\intercal p & \forall i \in [k], \forall p \in P^i & (3.10) \\
& (A_i r)^\intercal \pi & \geq c_i^\intercal r & \forall i \in [k], \forall r \in R^i & (3.11) \\
& (\pi, \rho) \in \mathbb{R}^{m_2}_+ \times \mathbb{R}^k. & & &
\end{aligned}$$

This means for each $i \in [k]$ we need to solve the subproblem

$$\max \{(c_i^\intercal - \pi^\intercal A_i)x : x \in H^i_{\text{sub}}\}. \tag{3.12}$$

And again infeasibility of any one of (3.12) leads to infeasiblity of the master problem. For an unbounded subproblem we can add an extreme ray and otherwise we add an extreme point if the subproblems objective value exceeds ρ_i. Also the modifications for Farkas pricing apply as before. Note that we only know that the

master problem has found an optimal solution if we cannot find any extreme ray or point for any of the subproblems.

Also note that, if we did not divide the subproblem, any combination of the extreme points and rays from H^i_{sub} leads to a new extreme point or ray from H_{sub}. Therefore exploiting the block diagonal structure of the subproblem is not only beneficial to the execution time of the subproblem but also reduces the number of variables in the master problem. Also the master problem can now choose the combination of extreme points and rays on its own and might therefore need less calls to the subproblem.

Another interesting case arises when the blocks are identical or almost identical ($c_i = c_j, A_i = A_j$, and possibly $D_i = D_j, d_i = d_j$ for $i \neq j$). Then a technique called aggregation and its extension, the heterogeneous aggregation, can be applied. These topics will be covered in detail in Chapter 5.

DANTZIG-WOLFE DECOMPOSITION FOR MILP

This section considers the case where some of the variables need to be integral, i. e., $X = \mathbb{R}^{n_1}_+ \times \mathbb{Z}^{n_2}_+$. Incorporating the integrality in the subproblem is – from a theoretical perspective – relatively simple, as we need to restrict the subproblems feasible region to only the allowed points:

$$H_{\text{sub}} = \{x : (Dx \leq d) \wedge (x \in X)\}.$$

Over this region we now need to solve the subproblem $\max\{(c^\intercal - \pi^\intercal A)x : x \in H_{\text{sub}}\}$ to get the extreme points and rays to add to the master problem. As long as we are able to solve the subproblem optimally, it does not matter that we do not have a polyhedral description of H_{sub}, as we can reach any desired extreme point or ray. One can solve this subproblem to optimality using some MILP solver, but doing so might be less performant than using a combinatorial algorithm that is specialized for the structure exhibited by H_{sub}. The existence of a good special purpose solver for some part of the original problem might actually be one of the main reasons to employ Dantzig-Wolfe decomposition.

While it is comparatively simple to get the integral extreme points and rays from the subproblem, it is much more complicated to ensure that the master problem only combines them in a way such that the overall solution is again integral. A solution of the master problem is denoted as integral if the corresponding original solution is integral. Note that this does not necessarily require the master problem variables to be in $\mathbb{Z}$. Consequently, when stating master problem formulations in the remaining parts of this thesis, the integrality constraints will not be explicitly stated. Instead the required integrality constraints are given for the original formulation.

When the solution of the master problem is not integral, one way to proceed is using a branch & bound procedure to exclude some fractional part of the solution. In the context of column generation, such algorithms are often denoted as branch & price. What makes the implementation of such a method challenging is the proper choice of the branching decisions to be made in each node of the search tree. Directly applying the methodology from classical branch & bound would mean to choose some fractional variable λ^*_p or μ^*_r and then branch based upon the variable bounds $\lambda_p \leq \lfloor \lambda^*_p \rfloor$ and $\lambda_p \geq \lceil \lambda^*_p \rceil$ (or equivalently for μ^*_r). So far such a

branching rule might not be feasible, as it could cut off an integral solution that is only represented by fractional master variables. To circumvent this issue one can employ discretization.

Discretization

So far it holds that for each integral original solution there exists a corresponding master solution and each master solution with integer variables will correspond to an integral original solution. What is missing is an assertion that an optimal integral original solution will correspond to a master solution with only integer variables, as an integral solution may lie inside of the subproblem polyhedron's convex hull and not be one of its extreme points.

A technique to circumvent this is to also add master variables for the subproblems interior integral solutions. Therefore, instead of the set P of all subproblem extreme points, one needs the set F, which shall contain all integral solutions of the subproblem. Note that, in the case of an unbounded or a mixed integer problem, F may not be finite. In order to avoid the complexities associated herewith (even though it is possible to use discretization in a generic MILP setting[161]), it is required for the remainder of this thesis that whenever the set F is required, only subproblems are used which ensure that F is finite.

Now in the master problem the new set F can be used instead of P, potentially increasing the number of variables but not changing the master problem's structure. Using F will be called discretization, while using only the extreme points P is called the convexification approach. When the problem variables have a binary domain the two approaches become equivalent, as no feasible integer points can be between two extreme points. Luckily many practical problems have this structure. This thesis will generally stick with the extreme point formulation whenever possible and only use discretization explicitly when it is necessary for the technique that is being explained. Note also that the set F does not only contain extreme points. As the majority of this thesis is based upon convexification, the term "extreme point" will also be used in cases where discretization is applicable.

With discretization one can be assured that any optimal solution of the original problem will correspond to a master problem solution where all variables take integer values. In theory this enables us to use a classical branching rule upon the master problem variables. It also helps with other, more complicated branching rules, as explained in Chapter 5.

Note that it may be more difficult to find interior integral points of the subproblem, as this requires to add additional constraints to the problem, e. g., using the branching decisions as hard constraints in the subproblem. This can destroy the structure of the subproblem, rendering a previously used (fast) combinatorial algorithm unsuited for the changed structure.

Branching

Branching directly on the master problem variables (when using discretization) generally leads to a poorly balanced search tree. Assume that in the current solution $\lambda_p^* = 0.5$ but we require it to be binary. Then the $\lambda_p \geq \lceil \lambda_p^* \rceil$ branch will choose exactly this solution which is a very strong constraint. On the other

hand the $\lambda_p \leq \lfloor \lambda_p^* \rfloor$ branch needs to exclude this and only this solution. This is usually a much weaker constraint. Furthermore it means that the subproblem in this branch needs to also exclude this particular solution from being added again (which would be a likely result when leaving the subproblem unchanged). This can easily destroy the combinatorial structure the subproblem might have had beforehand, and thereby render the employed special purpose algorithm unusable.

One commonly used method to handle branching translates the solution of the column generation method back into the space of the original variables: $x^* = \sum_{p \in P} p\lambda_p^* + \sum_{r \in R} r\mu_r^*$. Now one can add branching constraints on these variables as one would have done without the Dantzig-Wolfe decomposition. I. e., we choose one fractional variable x_j^* where we want to enforce integrality and solve two new problems for which we add the constraints

$$\sum_{p \in P} p_j \lambda_p^* + \sum_{r \in R} r_j \mu_r^* \leq \lfloor x_j^* \rfloor$$

or

$$\sum_{p \in P} p_j \lambda_p^* + \sum_{r \in R} r_j \mu_r^* \geq \lceil x_j^* \rceil$$

respectively. When solving the branches with these new constraints one now has two options. One option is that the new constraints are placed in the master problem. Note that this will result in a new dual value. This dual value does not affect the combinatorial structure of the subproblem as it will only be a constant factor that needs to be added to the subproblems objective. The other possibility is to add the new constraints to the subproblem. In this case one gets a potentially stronger bound but as mentioned before such changes in the subproblem can destroy its combinatorial structure. Therefore the strategy of choice highly depends on the problem at hand.

BENDERS' DECOMPOSITION

While the Dantzig-Wolfe decomposition was dealing with a set of complicating constraints, the goal of the Benders' decomposition is to help with a set of complicating variables. Assume that we are given the following generic problem:

$$\max\{c^\intercal x + d^\intercal y : (Ax + Dy \leq b) \wedge ((x, y) \in X \times Y)\}. \tag{3.13}$$

Here the x variables are in some way complicating. This means that if we assume to be given some fixed values x, the problem

$$\alpha(x) = \max\{d^\intercal y : (Dy \leq b - Ax) \wedge (y \in Y)\} \tag{3.14}$$

can be solved easily. Problem (3.14) is the subproblem of the Benders' formulation which we will need to solve repeatedly in order to find an optimum for the overall problem (3.13). The overall procedure will be to first choose x, then solve (3.14) and see if we need to adjust our choice of x.

As before we first look at the case where $X = \mathbb{R}_+^{n_1}$ and $Y = \mathbb{R}_+^{n_2}$, and extend this to the MILP setting in Section 3.5.

The choice of x has two aspects. We need to choose it such that it is still possible to find y such that $Ax + Dy \leq b$, i. e., x has to be feasible. Furthermore x shall be

chosen among all feasible possibilities such that $c^\intercal x + \alpha(x)$ is maximized, i. e., x shall be optimal. In order for x to be feasible it has to be in the set

$$P_{\text{feasible}} = \{x : (x \in \mathbb{R}_+^{n_1}) \wedge (\exists y \in \mathbb{R}_+^{n_2}, Ax + Dy \leq b)\}.$$

We can obtain a different description of P_{feasible} using the Farkas' Lemma (Lemma 1.12) in the following way: observe that for a given x there exists y satisfying $Dy \leq b - Ax$ iff $(D^\intercal r \geq 0) \wedge (r \geq 0) \Rightarrow (b - Ax)^\intercal r \geq 0$. Now let R be the set of all extreme rays of the cone $C = \{r : (D^\intercal r \geq 0) \wedge (r \geq 0)\}$

We now need to ensure that

$$\forall r \in R, (b - Ax)^\intercal r \geq 0$$

because this implies for $\tilde{r} \in C$ where $\tilde{r} = \sum_{r \in R} \mu_r r, \mu \geq 0$ (by Theorem 1.4) that

$$(b - Ax)^\intercal \tilde{r} = \sum_{r \in R} (b - Ax)^\intercal \mu_r r \geq 0.$$

This in turn means that $x \in P_{\text{feasible}}$ can be replaced by the following set of linear inequalities:

$$\forall r \in R, (r^\intercal A)x \leq b^\intercal r.$$

Next we need to make sure that we also choose x optimally. First note that if $\alpha(x)$ is unbounded so is (3.13). As we have just taken care of feasibility we can now assume that $\alpha(x)$ has a bounded optimum. Observe that the objective of (3.13) can be restated as maximizing $c^\intercal x + \alpha(x)$. As $\alpha(x)$ is a linear program we can as well look at its dual

$$\alpha(x) = \min\{(b - Ax)^\intercal p : (D^\intercal p \geq d) \wedge (p \geq 0)\}.$$

Now let P be the set of all extreme points of $\{p : (D^\intercal p \geq d) \wedge (p \geq 0)\}$. As $\alpha(x)$ calculates a minimum over this polyhedron we know

$$\forall p \in P, \alpha(x) \leq (b - Ax)^\intercal p$$

and as the minimum is attained in at least one of the extreme points (by Theorem 1.11), these linear inequalities are also sufficient to characterize $\alpha(x)$. Putting all this together we can now express (3.13) only in terms of the x variables, the new inequalities we just derived and an additional variable α containing the objective value from the subproblem:

$$\begin{aligned} \max\ & c^\intercal x + \alpha & & \\ \text{s.t. } & (r^\intercal A)x & \leq b^\intercal r & \quad \forall r \in R \\ & (p^\intercal A)x + \alpha & \leq b^\intercal p & \quad \forall p \in P \\ & x & \geq 0. & \end{aligned} \tag{3.15}$$

Again the number of extreme points P and rays R may be too large to be handled explicitly by an LP solver. Instead the method of choice usually is to leave out those constraints and to generate them on demand as we did with the variables for the Dantzig-Wolfe decomposition. Again we assume that we have (potentially empty)

subsets of the extreme points and rays $\bar{P} \subseteq P$ and $\bar{R} \subseteq R$ for which we have solved the restricted master problem

$$\begin{aligned} \max\ & c^\intercal x + \alpha \\ \text{s.t. } & (r^\intercal A)x && \leq b^\intercal r \quad \forall r \in \bar{R} \\ & (p^\intercal A)x + \alpha && \leq b^\intercal p \quad \forall p \in \bar{P} \\ & x \in \mathbb{R}^{n_1}_+ \end{aligned} \tag{3.16}$$

with (x^*, α^*) being an optimal solution for this LP. Note that (3.16) might be unbounded (which will always be the case if $\bar{P} = \emptyset$ due to no bound on α). Therefore it might be advisable to add an artificial upper bound for the parameter α in practice. For many practical problems finding such an upper bound is possible given the knowledge of the specifics of the problem. The next step is to solve the subproblem (3.14) for x^*, i. e., solving:

$$\alpha(x^*) = \max\{d^\intercal y : (Dy \leq b - Ax^*) \wedge (y \in Y)\}.$$

We now need to consider the following cases:

1. $\alpha(x^*)$ is unbounded. As x^* was feasible for the subproblem the master problem must also be unbounded.
2. $\alpha(x^*)$ is infeasible. In this case we can derive $r \in R \setminus \bar{R}$ with $(r^\intercal A)x^* > b^\intercal r$ using the Farkas' Lemma as explained above. We add r to the set $\bar{R}$ and solve (3.16) again.
3. $\alpha(x^*) < \alpha^*$. In this case there exists a $p \in P \setminus \bar{P}$ such that $p^T Ax^* + \alpha > b^\intercal p$ using the dual formulation of (3.14). We add p to $\bar{P}$ and solve (3.16) again.
4. $\alpha(x^*) = \alpha^*$. As x^* is feasible and there is no gap between α^* and $\alpha(x^*)$ we found an optimal solution.

The Benders' decomposition algorithm can be seen as a dual to the Dantzig-Wolfe decomposition described earlier in this chapter. While for Dantzig-Wolfe it was necessary to add variables to the master problem, which were based upon the extreme points and rays of the subproblem, in Benders' decomposition one adds constraints to the master problem that are based upon the extreme points and rays of the subproblem's dual formulation.

Zero rows in the subproblem

It can easily happen that an entire row of D contains only zeros. In this case it is advisable to make a few modifications to the cutting plane algorithm stated above. So assume our original formulation is

$$\max\{c^\intercal x + d^\intercal y : (\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} x + \begin{pmatrix} 0 \\ D \end{pmatrix} y \leq \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}) \wedge ((x, y) \in X \times Y)\}$$

then in the subproblem we look for extreme rays from

$$\{\begin{pmatrix} r_1 \\ r_2 \end{pmatrix} : \begin{pmatrix} 0 \\ D \end{pmatrix}^\intercal \begin{pmatrix} r_1 \\ r_2 \end{pmatrix} \geq 0\}$$

and for extreme points in

$$\{\begin{pmatrix} e_1 \\ e_2 \end{pmatrix} : \begin{pmatrix} 0 \\ D \end{pmatrix}^\intercal \begin{pmatrix} e_1 \\ e_2 \end{pmatrix} \geq d\}.$$

The first thing to notice is that the first component r_1 or e_1 does not make any difference for the feasibility of a ray or point. Therefore in the separation routine we might end up generating a lot of rays or points that only differ in the r_1 or e_1 component. This can be avoided as follows: first we need to make sure that the r_1 or e_1 components can be safely discarded in the subproblem by modifying the restricted master problem to

$$\begin{aligned} \max\ & c^\intercal x + \alpha \\ \text{s.t. } & A_1 x && \leq b_1 \\ & (r_2^\intercal A_2) x && \leq b_2^\intercal r_2 \quad \forall r_2 \in \bar{R}_2 \\ & (p_2^T A_2) x + \alpha && \leq b_2^\intercal p_2 \quad \forall p_2 \in \bar{P}_2 \\ & x \in X. \end{aligned} \tag{3.17}$$

This ensures that for each $(r_1, r_2)^\intercal$ with $r_2 \in \bar{R}_2$ the inequality $r_1^\intercal A_1 x + r_2^\intercal A_2 x \leq b_1^\intercal r_1 + b_2^\intercal r_2$ is fulfilled as $A_1 x \leq b_1$ implies $r_1^\intercal A_1 x \leq b_1^\intercal r_1$.

The sets $\bar{R}_2$ and $\bar{P}_2$ contain the new extreme points and rays for the modified restricted master problem. The new cone to look for missing extreme rays is $\bar{R}_2$ is now given as

$$\{r_2 : (D^\intercal r_2 \geq 0) \wedge (r_2 \geq 0)\}$$

such that $r_2^\intercal A_2 x^* > b_2^\intercal r_2$. The vectors that need to be added to $\bar{P}_2$ are similarly the extreme points of

$$\{p_2 : (D^\intercal p_2 \geq d) \wedge (p_2 \geq 0)\}$$

where $p_2^\intercal A_2 x^* + \alpha > b_2^\intercal r_2$. Now the subproblem only handles the constraints that actually include some of the subproblem variables.

BENDERS' DECOMPOSITION FOR MILP

Similar to Section 3.3 we will now extend the Benders' decomposition to mixed integer programs. Not every mixed integer program can efficiently be tackled by these approaches. But for certain structures they prove to be very effective.

Mixed integer master problem

In the case where the domain of variables for (3.13) imposes integrality restrictions only for some of the variables in the master problem but not in the subproblem, i. e., $X = \mathbb{Z}_+^{n_{1,1}} \times \mathbb{R}_+^{n_{1,2}}$ and $Y = \mathbb{R}_+^{n_2}$, nothing needs to be done to adjust the procedure. As the subproblem is still a linear program, the reformulation using the duality theory from linear programming still applies. Therefore the only necessary change is to impose the integrality in the master problem (3.15) or the restricted master problem (3.16) respectively.

Solving MILPs of this form was the original goal of Benders' formulation as described by Benders (1962)[24]. Note that the bound of the LP relaxation in the MP will not be strengthened by this reformulation. But in some applications the size of the problem can be significantly reduced and thereby increase the solver performance. An example for such an application is explained in Chapter 4.

Binary master and mixed integer subproblem

When the subproblem is a mixed integer problem, it becomes more complex to properly relate the subproblem results back to the master problem. As there is no strong duality for the subproblem, blindly using the feasibility and optimality cuts from Section 3.4 is unlikely to produce the desired results. For example the subproblem might be infeasible over the integer domain, but its LP relaxation can be non empty. In this case one cannot simply produce a ray r to cut off the current master problem solution in the style of Farkas' Lemma. Similarly the optimality cuts might not be sufficiently tight and allow for α to take larger values than those of the corresponding subproblem.

Nevertheless it is possible to use a modified version of the basic concepts behind the Benders' decomposition. In the original version, we use duality theory in order to provide a proof that, e. g., a certain point cannot be feasible for the subproblem. This proof is then turned into a constraint for the master problem. While duality theory does not provide such proofs for combinatorial subproblems in general, there often exist other ways, to arrive at similar results. This concept was used by Hooker and Osorio (1999)[93] for various problems and then later generalized by Hooker and Ottosson (2003)[94] into a more general framework, denoted "logic-based Benders' decomposition". This was again improved upon and further generalized by Codato and Fischetti (2006)[47] which denoted their method as "combinatorial Benders' cuts".

The most generic variant of these methods considered in this thesis has the form of (3.13) and assumes the domain of the master problem to be binary and that of the subproblem to be mixed integral, i. e., $X = \{0,1\}^{n_1}$ and $Y = \mathbb{Z}_+^{n_{2,1}} \times \mathbb{R}_+^{n_{2,2}}$. The most generic (but also naïve) approach works as follows:

1. Start with the restricted master problem (rMP) $\max\{c^\intercal x + \alpha : x \in X\}$
2. Solve the rMP up to an optimal solution (x^*, α^*)
3. Solve the subproblem $\alpha(x^*) = \max\{d^\intercal y : (Dy \leq b - Ax^*) \wedge (y \in Y)\}$
   a) If the subproblem is infeasible, add the cut

   $$\sum_{i:x_i^*=0} x_i + \sum_{i:x_i^*=1} (1 - x_i) \geq 1$$

   to exclude exactly the current solution[47]. Go back to step 2 with the new rMP.
   b) If the subproblem is feasible and $\alpha(x^*) < \alpha^*$, add the cut

   $$\alpha \leq \alpha(x^*) + (M - \alpha(x^*))(\sum_{i:x_i^*=0} x_i + \sum_{i:x_i^*=1} (1 - x_i)))$$

where M is an upper bound on $\alpha(x)$. In this way, a bound for the objective is introduced in exactly the current master problem solution[97]. Go back to step 2 with the new rMP.

c) If the subproblem is feasible and $\alpha(x^*) = \alpha^*$ an optimal solution was found.

This general approach will likely not produce acceptable performance in practice, as each subproblem can only generate cuts that apply to exactly the current rMP solution and nowhere else. Depending on the problem structure, this method can be improved and result in an efficient algorithm.

In order to strengthen the cuts one can lookout for subsets of the variables that will result in an infeasible subproblem no matter what configuration is chosen for the rest.

Definition 3.1. A rMP solution $\hat{x}$ and a subset $\bar{I} \subseteq [n_1]$ of the rMP variable indices defines an infeasible subsystem (IS) if

$$\{y : (Dy \leq b - A_{\bar{I}}\hat{x}_{\bar{I}}) \wedge (y \in Y)\} = \emptyset.$$

An infeasible subsystem is called minimal (MIS) if no $\bar{I}' \subset \bar{I}$ is an IS.

Given an IS $\hat{x}$ and $\bar{I}$, it is possible to use the following stronger feasibility cut[47]:

$$\sum_{\substack{i \in \bar{I} \\ \hat{x}_i = 0}} x_i + \sum_{\substack{i \in \bar{I} \\ \hat{x}_i = 1}} (1 - x_i) \geq 1.$$

Given two index sets $\bar{I}_1$ and $\bar{I}_2$ both defining an infeasible subsystem but with $\bar{I}_1 \subset \bar{I}_2$. Then obviously the feasibility cut for $\bar{I}_1$ will dominate that for $\bar{I}_2$ as it can only separate more points. Therefore it is desirable to find a small IS, ideally an MIS.

Finding the MIS of minimum size is NP-complete for general IPs, hard to approximate and even NP-hard when D is totally unimodular and $b - A\hat{x}$ is integral[13]. Therefore searching for the smallest MIS in order to obtain a very strong feasibility cut is discouraged. In Codato and Fischetti (2006)[47] a method to quickly find some MIS is described.

A noteworthy special case are problems where $\hat{x}$ infeasible implies that $\hat{x}'$ with $\hat{x}' \geq \hat{x}$ is infeasible. Such a case will be investigated in Section 4.2. In these situations indices i with $\hat{x}_i = 0$ can always be dropped from an IS without destroying the infeasibility.

Theorem 3.2. *Given that for the subproblem $\hat{x}$ infeasible implies $\hat{x}'$ with $\hat{x}' \geq \hat{x}$ infeasible. Then for any IS $(\hat{x}, \bar{I})$, the subset $\bar{I}' = \{i \in \bar{I} : \hat{x}_i = 1\}$ also defines an IS $(\hat{x}, \bar{I}')$.*

Proof. Assume that $\bar{I}'$ does not define an IS, i. e., there exists x^* such that $x^*_{\bar{I}'} = \hat{x}_{\bar{I}'}$ but which is not infeasible for the subproblem. Let

$$J := \{i : x^*_i < \hat{x}_i\}.$$

$J \neq \emptyset$ as the feasibility of x^* implies $\hat{x} \not\leq x^*$. Furthermore $J \cap \bar{I} \neq \emptyset$ because $\tilde{x}$, with

$$\tilde{x}_i = \begin{cases} \hat{x}_i & \text{if } i \in \bar{I} \\ 0 & \text{otherwise} \end{cases}$$

must lead to an infeasible subproblem and $\tilde{x} \leq x^*$.

But on the one hand $J \cap (\bar{I} \setminus \bar{I}') = \emptyset$ as $\hat{x}_{\bar{I} \setminus \bar{I}'} = 0$ by definition of $\bar{I}'$. On the other hand $J \cap \bar{I}' = \emptyset$ as $x^*_{\bar{I}'} = \hat{x}_{\bar{I}'}$, which concludes the proof by contradiction to $J \cap \bar{I}' \neq \emptyset$. □

From Theorem 3.2 quickly follows that with the given preconditions on the subproblem structure, it suffices to use feasibility cuts of the form

$$\sum_{i \in \bar{I}} x_i \leq |\bar{I}| - 1$$

for certain infeasible subsystems $\bar{I} \subseteq [n_1]$.

4

MATCHING AS SUBPROBLEM FOR BENDERS' DECOMPOSITION

This chapter will explore cases where a problem can be decomposed using Benders' decomposition (see Chapter 3) such that the subproblem turns out to be some kind of matching (see Chapter 2). Such structures appear for example in timetabling applications as shown later in Chapter 6.

It will be shown that if the subproblem is a classical bipartite matching, the Benders' decomposition is strongly related to the separation algorithm of the partial transversal polytope. It extends the separation of the partial transversal polytope in the sense that also weighted matchings can be incorporated. The presentation will start with some original formulation which is solved with the cutting plane approach we already know from Section 2.4. In the second part of the chapter this methodology will be further extended to also deal with bipartite hypergraph matchings as subproblems.

BIPARTITE MATCHING AS SUBPROBLEM

In this section we will look at problems where part of the problem is to choose a partial transversal. As partial transversals are representatives for bipartite matchings (see Section 2.4), assume that for our problem there exists some bipartite graph $G = (U \cup V, E)$, and, when formulating the problem as an MILP, we will have a variable for each of the incidence vectors for the vertices U: $x^t \in \{0,1\}^{|U|}$. Other variables that are necessary for the non-matching part of the problem are denoted by $x^o \in X$ (with X being some mixed integer domain). In order to ensure that x^t is actually encoding a partial transversal, we need some variables to determine if there is a matching from the chosen vertices in U to the vertices in V. These will be denoted as $y \in \{0,1\}^{|E|}$. The y variables will determine the edges of a matching in G. The generic MILP we will be looking at now has the form

$$\begin{aligned} \max \ & \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}^\intercal \begin{pmatrix} x^o \\ x^t \end{pmatrix} - w^\intercal y \\ \text{s.t} \ & \begin{pmatrix} \bar{A} & \\ 0 & I \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x^o \\ x^t \end{pmatrix} + \begin{pmatrix} 0 \\ D_1 \\ D_2 \end{pmatrix} y \leq \begin{pmatrix} \bar{b} \\ 0 \\ 1 \end{pmatrix} \\ & x^o \in X, x^t \in \{0,1\}^{|U|}, y \in \{0,1\}^{|E|}. \end{aligned} \tag{4.1}$$

This MILP has $n_1 = \dim(X) + |U|$ variables $x = (x^o, x^t)^\intercal$ that will later be part of the master problem, and $|E|$ variables y which will be placed in the subproblem. The formulation consists of $m = m_1 + |U| + |V|$ constraints which are split up as follows: $\bar{A} \in \mathbb{Q}^{m_1 \times n_1}$ and $\bar{b} \in \mathbb{Q}^{m_1}$ define additional constraints of the problem unrelated to the partial transversal part (but possibly including the x^t variables).

The remaining rows define the matching substructure. The matrix I is the $|U| \times |U|$ dimensional identity matrix such that we get the two sets of constraints

$$Ix^t + D_1 y \leq 0 \tag{4.2}$$

and

$$D_2 y \leq 1. \tag{4.3}$$

The matrix $D_1 \in \{-1, 0\}^{|U| \times |E|}$ consists of entries $d^1_{u,e}$ where

$$d^1_{u,e} = \begin{cases} -1 & \text{if } u \in e \\ 0 & \text{otherwise.} \end{cases}$$

Constraints (4.2) can therefore be restated in the form

$$\forall u \in U, \qquad x^t_u - \sum_{e \ni u} y_e \leq 0$$

and ensure that for every vertex $u \in U$, where $x^t_u = 1$, there has to be an edge e connected with u such $y_e = 1$. If there exists an edge e with $w_e < 0$, it might be necessary to require equality in these constraints, to prevent that more edges are chosen than required. This will not be the case if $w \geq 0$ which will be the setting assumed from now onwards. In case equality is required one can easily change the model appropriately, but note that one can usually shift the weights such that their values become non negative by adding a constant term to all weights. Ensuring $w \geq 0$ basically means that the subproblem will be a minimum weight maximum matching problem (even though stated in form of a maximization problem to remain in line with the established notation) and will therefore be referred to as such.

Matrix $D_2 \in \{0, 1\}^{|V| \times |E|}$ consists of entries $d^2_{v,e}$ where

$$d^2_{v,e} = \begin{cases} 1 & \text{if } v \in e \\ 0 & \text{otherwise} \end{cases}$$

leading to constraints of the form

$$\forall v \in V, \qquad \sum_{e \ni v} y_e \leq 1.$$

These constraints ensure that for every vertex $v \in V$ there is at most one connected edge chosen by y.

The y variables encode a matching problem which is easy to solve if solved independently from the rest of the problem. They are therefore a canonical choice for the subproblem variables in a Benders' decomposition. Following the standard approach (see Section 3.4) we get the following cutting plane algorithm:

1. Solve the MILP

$$\begin{aligned} max\{ \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}^\intercal \begin{pmatrix} x^o \\ x^t \end{pmatrix} + \alpha : (\bar{A} \begin{pmatrix} x^o \\ x^t \end{pmatrix} \leq \bar{b}) \\ \wedge (x^o \in X) \wedge (x^t \in \{0,1\}^{|U|}) \}. \end{aligned} \tag{4.4}$$

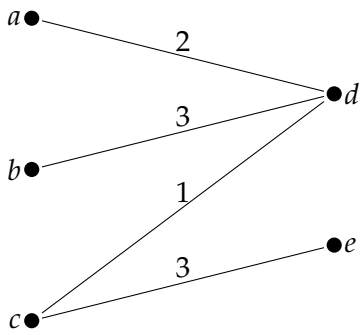


Figure 8: The bipartite matching graph $G = (U \cup V, E)$ from Example 4.1 with its edge weights

2. Let $(x^{o*}, x^{t*}, \alpha^*)^\intercal$ be an optimal solution for (4.4). Now solve

$$\alpha((x^{o*}, x^{t*})^\intercal) = \max\{-w^\intercal y : \begin{pmatrix} D_1 \\ D_2 \end{pmatrix} y \leq \begin{pmatrix} -Ix^{t*} \\ 1 \end{pmatrix}\}. \tag{4.5}$$

3. If (4.5) is infeasible, let $r = (r^1, r^2)^\intercal$ be a Farkas vector proving the infeasibility. Otherwise let $e = (e^1, e^2)^\intercal$ be an optimal dual solution for (4.5). If the subproblem is feasible and $\alpha((x^{o*}, x^{t*})^\intercal) = \alpha^*$, the algorithm terminates. In case of an infeasible subproblem, we add the constraint

$$r^{1\intercal} x^t \leq 1^\intercal r^2$$

for the dual extreme ray r. In case of a feasible subproblem with objective value greater than α^*, we retrieve the corresponding dual extreme point p and add the constraint

$$p^{1\intercal} x^t + \alpha \leq 1^\intercal p^2$$

to the master problem (4.4). Then go back to step 1 and calculate an optimum of the revised master problem.

As we assumed that $w \geq 0$, the subproblem will always result in non positive objective values. This implies that one can always impose the upper bound $\alpha \leq 0$ in the master problem, thereby avoiding unboundedness. If there are $w \ngeq 0$ the master problem may become unbounded.

The Benders' algorithm is closely related to the separation routine for the partial transversal polytope from Section 2.4, as in both cases matchability for an incidence vector shall be ensured. But unlike the routine in Section 2.4, the Benders' algorithm can also incorporate a weighted matching as a subproblem. Later some improvements for this algorithm are be developed which aim at combining the best traits from both worlds. But first the Benders' algorithm shall be illustrated by an example.

Example 4.1 (Matching with weak conflicts)**.** This example will walk through the iterations of the Benders' procedure for a problem with a matching substructure. The setting is as follows. We are given a bipartite graph $G = (U \cup V, E)$ where we may choose which elements from $U = \{a, b, c\}$ we want to use (each giving us a

value of 5). Each chosen vertex has to be matched to one of the vertices $V = \{d, e\}$. The matching generates some cost that is dependent on the connected vertices. Figure 8 shows G and the associated costs.

In addition to the cost from the matching, we penalize the parallel use of vertices a and c with a cost of 3. This problem can be formulated as an IP in the following way:

$$\begin{aligned}
\max\ & -3x' + 5x_a + 5x_b + 5x_c - 2y_{ad} - 3y_{bd} - y_{cd} - 3y_{ce} & \\
\text{s.t}\ & x_a + x_c - x' & \leq 1 \\
& x_a - y_{ad} & \leq 0 \\
& x_b - y_{bd} & \leq 0 \\
& x_c - y_{cd} - y_{ce} & \leq 0 \\
& y_{ad} + y_{bd} + y_{cd} & \leq 1 \\
& y_{ce} & \leq 1 \\
& x' \in \{0,1\}, x \in \{0,1\}^3, y \in \{0,1\}^4. &
\end{aligned}$$

In this IP x_a, x_b, x_c encode whether the respective vertex is used or not, x' encodes if we have to pay the penalty for using vertices a and c together. The y variables encode the matching between the chosen U and the V vertices. The optimal solution for this problem sets variables x_c and y_{cd} to 1, leaves all others at 0, and has an objective value of 4.

Applying the Benders' Decomposition in the manner described before leads us to the following master problem, based on the x variables:

$$\begin{aligned}
\max\ & -3x' + 5x_a + 5x_b + 5x_c + \alpha & \\
\text{s.t}\ & x_a + x_c - x' & \leq 1 \\
& \alpha & \leq 0 \\
& x' \in \{0,1\}, x \in \{0,1\}^3, \alpha \in \mathbb{R}. &
\end{aligned} \tag{4.6}$$

Note that we can set an upper bound of 0 for α as the matching subproblem can only result in negative objective values due to the convention that $w \geq 0$. This way we can avoid issues arising from an unbounded α. An optimal solution for (4.6) is $(x'^*, x_a^*, x_b^*, x_c^*, \alpha^*)^\intercal = (1,1,1,1,0)^\intercal$ with an objective value of 12. Using this we arrive at the following subproblem:

$$\begin{aligned}
\max\ & -2y_{ad} - 3y_{bd} - y_{cd} - 3y_{ce} & \\
\text{s.t}\ & -y_{ad} & \leq -1 \\
& -y_{bd} & \leq -1 \\
& -y_{cd} - y_{ce} & \leq -1 \\
& y_{ad} + y_{bd} + y_{cd} & \leq 1 \\
& y_{ce} & \leq 1 \\
& y \in \mathbb{R}^4_+. &
\end{aligned}$$

This subproblem is infeasible which can be proven using the dual ray $r = (1,1,0,1,0)^\intercal \geq 0$ fulfilling the conditions from the Farkas Lemma (see Lemma 1.12):

$$\begin{pmatrix} -1 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \end{pmatrix} r = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \geq 0, \quad \begin{pmatrix} -1 \\ -1 \\ -1 \\ 1 \\ 1 \end{pmatrix}^\intercal r = -1 < 0.$$

Using r we add a new constraint to the master problem (4.6):

$$x_a + x_b \leq 1.$$

Note that this constraint is the partial transversal polytope facet $x(\bar{U}) \leq |N(\bar{U})|$ from Theorem 2.10 for the subset $\bar{U} = \{a,b\}$. The last solution of the master problem is now infeasible with respect to the new constraint and a new solution of the rMP is

$$(x'^*, x_a^*, x_b^*, x_c^*, \alpha^*)^\intercal = (0,0,1,1,0)^\intercal$$

which has an objective value of 10.

For this solution the subproblem is

$$\begin{aligned} \max \; & -2y_{ad} - 3y_{bd} - y_{cd} - 3y_{ce} & \\ \text{s.t} \; & -y_{ad} & \leq 0 \\ & -y_{bd} & \leq -1 \\ & -y_{cd} - y_{ce} & \leq -1 \\ & y_{ad} + y_{bd} + y_{cd} & \leq 1 \\ & y_{ce} & \leq 1 \\ & y \in \mathbb{R}^4_+ & \end{aligned}$$

which is now feasible with an optimal solution

$$(y_{ad}^*, y_{bd}^*, y_{cd}^*, y_{ce}^*)^\intercal = (0,1,0,1)^\intercal$$

and dual solution $p = (0,5,3,2,0)^\intercal$. The objective value is -6 which is less than the current value of $\alpha^* = 0$. Therefore we add the constraint

$$5x_b + 3x_c + \alpha \leq 2$$

to the master problem and resolve it again. An updated solution is

$$(x'^*, x_a^*, x_b^*, x_c^*, \alpha^*)^\intercal = (1,1,0,1,-1)^\intercal$$

with an objective value of 6. Solving the corresponding subproblem results in the dual solution $p = (4,0,3,2,0)$ with the objective value of -5 leading to the following cut for the master problem:

$$4x_a + 3x_c + \alpha \leq 2.$$

Resolving leads to the solution

$$(x'^*, x_a^*, x_b^*, x_c^*, \alpha^*)^\intercal = (0, 0, 0, 1, -1)^\intercal$$

with the objective value of 4. This time the subproblem has an objective value of $-1 = \alpha^*$ which means that we can stop at this point as we have found an optimal solution.

Improved cuts

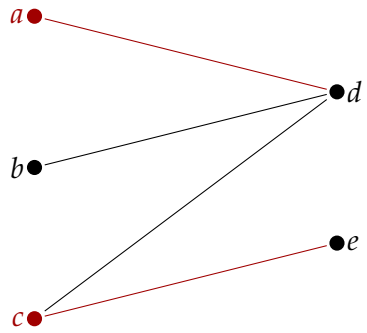
In Section 2.4 we have seen a separation routine for the partial transversal polytope which will generate cuts similar to the Benders' feasibility cuts. These cuts do not need to be facets of the partial transversal polytope and neither do the Benders' feasibility cuts. This will lead to slow convergence due to bad quality of the cuts. Therefore it is desirable to have a way to improve these cuts. Such an algorithm is presented now, based on the separation algorithm from Section 2.4. This will then be carried over to the Benders' optimality cuts, resulting in an improved version of the original Benders' method.

The basic ingredient for the improved cuts will be an algorithm, denoted as *RISS* (reachable infeasible subsystem search), which will improve the cuts at hand with little computational effort. Given a master solution $(\hat{x}^o, \hat{x}^t)$. Let $\hat{M}$ be a matching covering a maximum number of vertices u for which $x_u^t = 1$. For an infeasible subproblem, this matching will not cover all of these vertices u. Now RISS works as follows:

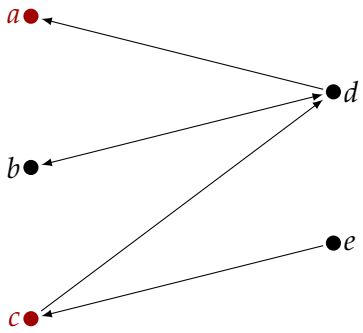
1. Construct a directed bipartite auxiliary graph $G' = (\hat{U} \cup V, E')$. The set $\hat{U} = \{u : x_u^t = 1\}$ is the partial transversal indicated by x^t. The edges are defined by the matching $\hat{M}$ as follows
   a) $\forall m \in \hat{M}, \forall u \in m \cap U, \forall v \in m \cap V$, let $(v, u) \in E'$
   b) $\forall m \in E \setminus \hat{M}, \forall u \in m \cap U, \forall v \in m \cap V$, let $(u, v) \in E'$
   c) $\forall u \in \hat{U}, \nexists e \in \hat{M}, u \in e, \forall v \in N(u)$ let $(v, u) \in E'$
2. Let $W = \{u \in \hat{U} : \nexists e \in \hat{M}, u \in e\}$ be the set of vertices chosen in the master problem that were not covered by $\hat{M}$.
3. For some $u \in W$ find the set $\mathcal{R}(u)$ of vertices reachable from u in G' (see also Section 1.3)
4. Return the cut $x(\mathcal{R}(u) \cap U) \leq |\mathcal{R}(u) \cap \{u' \in \hat{U} : \exists e \in \hat{M}, u' \in e\}| = \nu(\mathcal{R}(u) \cap U)$.

Note that $\mathcal{R}(u)$ can be calculated quickly by traversing the graph and marking the vertices accordingly (see Section 1.3) with a runtime complexity of $O(|E| + |\hat{U}| + |V|)$. For $W' = \{u \in \hat{U} : \nexists e \in \hat{M}, u \in e\}$ being the set of initially unmatched vertices, the reachable vertex set needs to be calculated at most once per vertex, leading to a runtime complexity of $O(|W'| \cdot (|E| + |\hat{U}| + |V|))$.

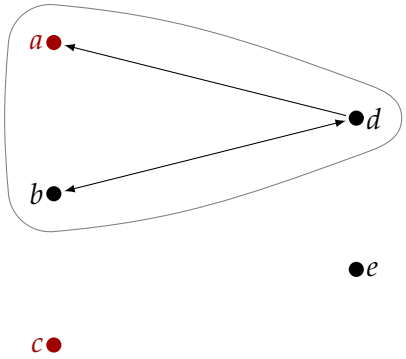
The basic intuition behind RISS is to start with the neighbors of an uncovered vertex, check by which edges from $\hat{M}$ they are "blocked", see why the corresponding U vertices are not assigned to some other edge, and so forth. Figure 9 shows an example for the RISS algorithm, based upon the example from 4.1. Note that in



(a) The original graph from Example 4.1 with a highlighted (red) maximum matching $\hat{M}$ and the unmatched node b



(b) The auxiliary graph G' derived from $\hat{M}$



(c) The nodes reachable from b in G'

Figure 9: An example for the RISS algorithm

this case the RISS algorithm computes a vertex set that translates to the same cut as the one produced by the Farkas' vector in the example.

In order to prove that RISS does indeed always yield a valid cut, it is necessary to show that the calculated $\nu(\mathcal{R}(u) \cap U)$ values are correct. Given that they are correct, one can easily see that $\nu(\mathcal{R}(u) \cap U) < |\mathcal{R}(u) \cap U|$ and therefore the generated cuts will actually separate the current master solution, as $\mathcal{R}(u) \cap \{u' \in \hat{U} : \exists e \in \hat{M}, u' \in e\}$ will exclude at least the unmatched vertex u and therefore be at least by one element smaller than $\mathcal{R}(u) \cap U$.

Theorem 4.2. *Given some set $\mathcal{R}(u) \cap U$ as found in the RISS algorithm. Let*

$$\xi(\mathcal{R}(u) \cap U) := |\mathcal{R}(u) \cap \{u' \in \hat{U} : \exists e \in \hat{M}, u' \in e\}|$$

then $\nu(\mathcal{R}(u) \cap U) = \xi(\mathcal{R}(u) \cap U)$.

Proof. As the matching $\hat{M}$ already covers $\xi(\mathcal{R}(u) \cap U)$ vertices from $\mathcal{R}(u) \cap U$ we immediately get $\nu(\mathcal{R}(u) \cap U) \geq \xi(\mathcal{R}(u) \cap U)$.

Now assume that $\nu(\mathcal{R}(u) \cap U) > \xi(\mathcal{R}(u) \cap U)$. In this case there exists a matching $\tilde{M}$ covering at least $\xi(\mathcal{R}(u) \cap U) + 1$ vertices of $\mathcal{R}(u) \cap U$. Now construct the matching

$$M' = \tilde{M} \cup \{e \in \hat{M} : e \cap \mathcal{R}(u) \cap U = \emptyset\}$$

which uses $\tilde{M}$ on $\mathcal{R}(u) \cap U$ and $\hat{M}$ otherwise.

To see that M' is feasible, consider an edge $e_1 \in \tilde{M}$ covering some vertex from $\mathcal{R}(u)$ and an edge e_2 covered by $\hat{M}$ with $e_2 \cap \mathcal{R}(u) \cap U = \emptyset$. As $e_1 \cap \mathcal{R}(u) \neq \emptyset$, there exists a path from u to each vertex in e_1 in the auxiliary graph G'. If $e_1 \cap e_2 \neq \emptyset$, then there would also be a path to the node $e_2 \cap U$ which is prohibited as $e_2 \cap \mathcal{R}(u) \cap U = \emptyset$.

By construction M' covers more vertices of $\mathcal{R}(u) \cap U$ than $\hat{M}$ did, but covers the same vertices outside of $\mathcal{R}(u) \cap U$. Therefore $|M'| > |\hat{M}|$ which is a contradiction to $\hat{M}$ being a maximum matching. □

The RISS algorithm can be seen as dividing the graph into components such that the vertices of a component are not affected by vertices outside of the component. A property of the resulting cuts is, that they are facets of the partial transversal polytope. This can be shown by using Theorem 1.9 for the facets of a polymatroid, as we only need to show that the returned subsets are ν-flat and ν-inseparable. The reader may refer to Section 2.4 for further explanations about the facets of the partial transversal polytope and its relation to polymatroids.

Theorem 4.3. *Let $\hat{G} = (\hat{U} \cup V, \{e \in E, e \cap \hat{U} \neq \emptyset\})$ be the graph restricted to the vertices chosen in the master problem. If $\hat{G}$ is a bipartite graph, then the subsets returned by RISS are ν-flat and ν-inseparable.*

Proof. Let $\mathcal{R}(u) \cap U$ be one of the subsets returned by RISS and let $\hat{M}' = \{e \in \hat{M} : e \cap \mathcal{R}(u) \cap U \neq \emptyset\}$. Assume that $\mathcal{R}(u) \cap U$ is not ν-flat, i. e., there exists $u' \in \hat{U} \setminus \mathcal{R}(u)$ with $\nu((\mathcal{R}(u) \cap U) \cup \{u'\}) = \nu(\mathcal{R}(u) \cap U)$. This implies that all nodes $N_G(u')$ must be already covered by $\hat{M}'$. But then there must be a path in G' from u to u' contradicting $u' \in \hat{U} \setminus \mathcal{R}(u)$.

Now assume that $\mathcal{R}(u) \cap U$ is not ν-inseparable and let $\tilde{U}_1, \tilde{U}_2$ be a partition of $\mathcal{R}(U) \cap U$ such that $\tilde{U}_1 \cup \tilde{U}_2 = \mathcal{R}(u) \cap U$, $\tilde{U}_1 \cap \tilde{U}_2 = \emptyset$, and $\nu(\mathcal{R}(u) \cap U) = \nu(\tilde{U}_1) + \nu(\tilde{U}_2)$. W.l.o.g. let $u \in \tilde{U}_1$. Let $\hat{M}_1 = \{e \in \hat{M} : e \cap \tilde{U}_1 \neq \emptyset\}$ be the matching restricted to $\tilde{U}_1$. By the properties of the partition we get that $|\hat{M}_1| = \nu(\tilde{U}_1)$. Next let p be a path from u to some node in $\tilde{U}_2$ in the auxiliary graph G' such that only one node in $\tilde{U}_2$ is visited by that path. There may be several nodes from $\tilde{U}_1$ upon p which are unmatched by $\hat{M}'$. From the last of these onwards p will be an alternating path up to its last edge. If the last node of p is covered by $\hat{M}$, then also the last edge is alternating and $\hat{M}_1$ can be increased using this alternating path. In this case $\nu(\tilde{U}_1) > |\hat{M}_1|$ which is a contradiction. If on the other hand the last node of p is not matched in $\hat{M}$, then $\hat{M}_2$ can be increased by the last edge of p which is also a contradiction. Therefore $\mathcal{R}(u) \cap U$ must be ν-inseparable □

Note that we are only guaranteed to get facets of the partial transversal polytope if we initially had a maximum matching at hand. As the Farkas' vector for the minimum weight maximum matching problem does not need to translate into proper ν values, using RISS here is not guaranteed to result in facets.

The next step will be to carry over these finding to the optimality cuts. One problem with the original Benders' algorithm is that the subproblem cost is represented by a single variable, which then needs to appear in each optimality cut, leaving no leeway for improving the cut. Therefore the first necessary step is to replace α appropriately by some vector $\hat{\alpha} \in \mathbb{R}^{|U|}$, which will be encoding the subproblem cost

for each vertex in U (meaning that $\hat{\alpha}$ will also enter the objective with cost of 1 each). The restricted master problem before adding any constraints now becomes

$$max\{ \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}^\intercal \begin{pmatrix} x^o \\ x^t \end{pmatrix} + \hat{\alpha}(U) : (\bar{A} \begin{pmatrix} x^o \\ x^t \end{pmatrix} \leq \bar{b}) \\ \wedge (x^o \in X) \wedge (x^t \in \{0,1\}^{|U|})\}.$$

The original α can now be seen as an aggregated version of the new vector: $\alpha = \sum_{u \in U} \hat{\alpha}_u$.

It will now be shown that there exists a collection of cuts which is feasible for the Benders' master problem. Such cut collections can be found using a simple strategy similar to the RISS algorithm used for the feasibility cuts. Each of these cuts will contain a subset of the original cuts variables. For some problems these cuts turn out to vastly improve the performance of a Benders' algorithm.

Theorem 4.4. *Given some subset $\bar{U} \subseteq U$ and a dual extreme point p of a minimum cost maximum matching problem covering at least all vertices from $\bar{U}$. Then the cut*

$$(\sum_{u \in \bar{U}} p_u x_u^t) + \hat{\alpha}(\bar{U}) \leq p(N_G(\bar{U}))$$

is valid for the Benders' master problem.

Proof. First note that the minimum weight maximum matching problem imposes some restrictions on the values that the p vector can take. The dual formulation of (4.5) has the constraints

$$\begin{pmatrix} D_1 \\ D_2 \end{pmatrix}^\intercal p \geq -w$$

implying

$$\forall \{u,v\} \in E, \qquad p_u - w_{\{u,v\}} \leq p_v.$$

In order for x^t and $\hat{\alpha}$ to be valid, they have to represent a partial transversal and the cost of a valid matching covering the partial transversal, i. e., there is some matching $\hat{M}$ covering all u for which $x^t = 1$ and such that

$$\forall \{u,v\} \in \hat{M}, \qquad \hat{\alpha}_u \leq -w_{\{u,v\}}.$$

Putting this together with the requirements for the dual extreme point p, we get

$$\forall \{u,v\} \in \hat{M}, \qquad p_u + \hat{\alpha}_u \leq p_v.$$

As $\{u,v\} \in \hat{M}$ implies $v \in N_G(u)$ and as each u for which $x_u^t = 1$ is covered by $\hat{M}$, it directly follows that the inequality

$$(\sum_{u \in \bar{U}} p_u x_u^t) + \hat{\alpha}(\bar{U}) \leq p(N_G(\bar{U}))$$

is valid. □

In order to find subsets that are guaranteed to actually separate the current master solution and improve on the original Benders' cut, connected components on a subgraph of G will be used. The subgraph needed for this will be called $\hat{G} = (\hat{U} \cup V, \hat{E})$ and will be the subgraph of G restricted to the vertices $\hat{U}$ which are active in the current master solution x^t, as well as the edges induced by this selection of vertices. Now one can easily find the set of all connected components $\mathcal{C}(\hat{G})$ within a time complexity of $O(|V| + |E|)$ by traversing the graph, starting from an unvisited vertex for each component (see Section 1.3).

Theorem 4.5. *Let $\mathcal{C}(\hat{G})$ be the set of connected components in $\hat{G}$, as defined previously. Then*

$$p^{1\intercal} x^t + 1^\intercal \hat{\alpha} > 1^\intercal p^2$$

implies

$$\exists C \in \mathcal{C}(\hat{G}), \qquad (\sum_{u \in C \cap U} p_u x_u^t) + \hat{\alpha}(C \cap U) > p(N_G(C \cap U))$$

Proof. The connected components form a complete partition of the vertices of subgraph $\hat{G}$. Therefore

$$\forall C \in \mathcal{C}(\hat{G}), \qquad (\sum_{u \in C \cap U} p_u x_u^t) + \hat{\alpha}(C \cap U) \leq p(C \cap V)$$

implies by summing up all these inequalities that

$$(\sum_{u \in \hat{U}} p_u x_u^t) + \hat{\alpha}(\hat{U}) \leq p(V).$$

This constraint must be violated if $p^{1\intercal} x^t + 1^\intercal \hat{\alpha} > 1^\intercal p^2$ and therefore there exists $C \in \mathcal{C}(\hat{G})$such that $(\sum_{u \in C \cap U} p_u x_u^t) + \hat{\alpha}(C \cap U) > p(C \cap V)$. As $N_G(C \cap U) = C \cap V$ for any connected component C, the theorem follows. □

Theorem 4.5 show that the original Benders' cuts can be replaced by our new optimality cuts which are derived from the connected components. Note that the sets resulting from the search for connected components in the undirected auxiliary graph can be larger than the reachable sets that the RISS algorithm is looking for. It turns out that the sets found by the RISS procedure are not sufficient for separating all relevant infeasible master solutions as demonstrated by the following example.

Figure 10 shows a graph where the minimum weight maximum matching $\hat{M} = \{\{a, d\}, \{b, e\}, \{c, f\}\}$ has a total weight of 2. A corresponding dual solution is

$$(p_a, p_b, p_c, p_d, p_e, p_f) = (1, 1, 1, 0, 1, 0).$$

In this setting the RISS algorithm would return the following subsets of $\{a, b, c\}$:

$$\bar{U}_1 = \{a, b\}, \quad \bar{U}_2 = \{b\}, \quad \bar{U}_3 = \{b, c\}$$

which lead to the inequalities

$$\begin{aligned} x_a^t + x_b^t + \hat{\alpha}_a + \hat{\alpha}_b &\leq 1 \\ x_b^t \qquad + \hat{\alpha}_b \qquad &\leq 1 \end{aligned}$$

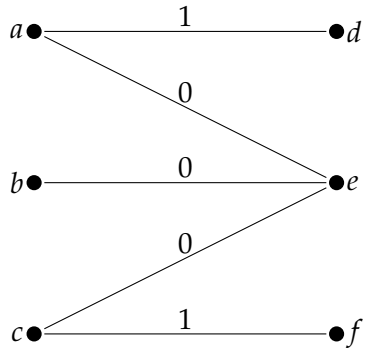


Figure 10: A graph where RISS does not produce sufficient optimality cuts

$$x_b^t + x_c^t + \hat{\alpha}_b + \hat{\alpha}_c \leq 1.$$

These inequalities are all satisfied by

$$(x_a^t, x_b^t, x_c^t, \hat{\alpha}_a, \hat{\alpha}_b, \hat{\alpha}_c) = (1, 1, 1, 0, -1, 0).$$

But this vector does violate the original Benders' cut

$$x_a^t + x_b^t + x_c^t + \hat{\alpha}_a + \hat{\alpha}_b + \hat{\alpha}_c \leq 1$$

which means that it is indeed necessary to use an undirected auxiliary graph for finding the new optimality cuts, as the directed version in the RISS procedure would not be sufficient to cut off all infeasible master solutions.

Note that the subproblem cost does not need to be distributed to the exact corresponding $\hat{\alpha}$ in the master problem. In the example a feasible master solution would be

$$(x_a^t, x_b^t, x_c^t, \hat{\alpha}_a, \hat{\alpha}_b, \hat{\alpha}_c) = (1, 1, 1, 0, -2, 0)$$

even though the costs in the subproblem are actually generated by the matching from the vertices a and c. Nevertheless the total subproblem cost is still equivalent.

The improvements of the Benders' cuts can have a significant impact on the solver performance (as shown in Chapter 6) and in the case of the optimality cuts it can even be crucial to use the improved cuts in order to arrive an algorithm that is sufficiently stable to not report wrong results. Note that these improved cuts require the current master problem solution to be integral (at least the x^t), while the classical Benders' cuts can also be used to separate fractional points.

BIPARTITE HYPERGRAPH MATCHING AS SUBPROBLEM

In MILP (4.1) defined in Section 4.1, the definition of matrix D ensured that the subproblem will always be a bipartite matching problem. The subproblem could be solved by linear programming as this structure ensures total unimodularity of D which in turn enabled us to perform the classical Benders' decomposition. If the underlying graph is a bipartite hypergraph, the matrix D does not need to be totally unimodular anymore. Therefore it is necessary to use the combinatorial Benders' methodology described in Section 3.5. Knowing that the subproblem is a bipartite hypergraph matching problem will allow us to improve the quality of the

generic feasibility cuts and do other improvements resulting in an efficient method for certain applications (e. g., university course timetabling as described in Chapter 6).

Assuming we again want to solve (4.1) but this time the definitions of D_1 and D_2 are based upon a bipartite hypergraph $G = (U \cup V, E)$ instead of a bipartite graph. We want to solve this model using the combinatorial Benders' procedure, placing the x variables in the master problem and the y variable in the subproblem. In order for the combinatorial Benders' method to work we require that the domain for x^o is also binary. Only the case where $d = 0$, i. e., the subproblem does not contribute to the objective function and is only needed to ensure feasibility, will be considered.

In Section 2.6 it was shown that this new subproblem is NP-complete in general. In our applications these problems are solved using a general purpose MILP solver, which have a sufficiently fast runtime in the examples considered. In certain applications it might be possible to create better performing combinatorial algorithms for these kinds of hypergraph matching problems.

As seen in Section 3.5 it is a desirable property when $\hat{x}$ leading to an infeasible subproblem implies that $\hat{x}'$ with $\hat{x}' \geq \hat{x}$ will also lead to an infeasible subproblem. For hypergraph matchings this property is trivially fulfilled, as if there is no matching covering all $\bar{U} \subseteq U$, adding an additional vertex to $\bar{U}$ will never be helpful (i. e., making the matching instance feasible again). This insight and Theorem 3.2 show that it would be possible to only use feasibility cuts of the form

$$x^t(\bar{U}) \leq |\bar{U}| - 1$$

for certain subsets $\bar{U} \subseteq U$ of the left hand side vertices. These $\bar{U}$ must satisfy $\nu(\bar{U}) < |\bar{U}|$ in order for the cuts to separate the current solution.

Thanks to the matching structure it is possible to still improve upon these cuts as explained in the following corollary.

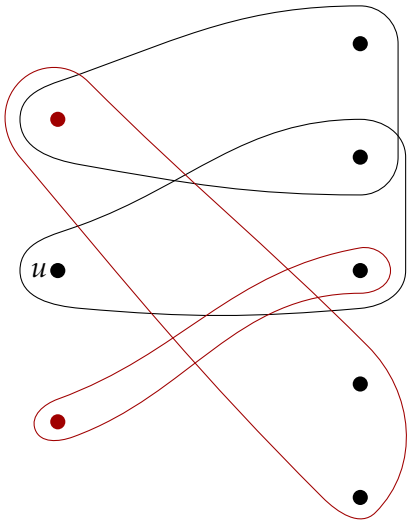
Corollary 4.6. *Given some subset $\bar{U} \subseteq U$ for which the feasibility cut $x^t(\bar{U}) \leq |\bar{U}| - 1$ should be added to the master problem. Then we can instead use the cut $x^t(\bar{U}) \leq \nu(\bar{U})$.*

Proof. Theorem 2.20 directly shows that these cuts will be valid and not separate feasible points. In fact it also grants that adding these constraints for all $\bar{U} \subseteq U$ will be sufficient to ensure feasibility of the subproblem in any case. Furthermore as one only needs to add a feasibility cut if $\nu(\bar{U}) \leq |\bar{U}| - 1$, the new cuts have to be at least as strong as the other ones. □

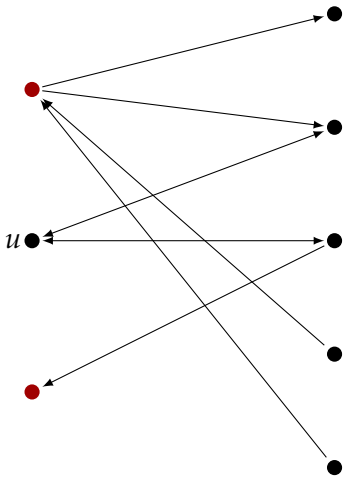
Note that the cuts from Corollary 4.6 are no more difficult to calculate than the more generic cuts. In order to know the correct subset $\bar{U} \subseteq U$, it is necessary to solve the subproblem to optimality and therefore the value $\nu(\bar{U})$ will already be calculated by the subproblem.

Currently the combinatorial Benders' algorithm for solving the MILP (4.1) looks as follows

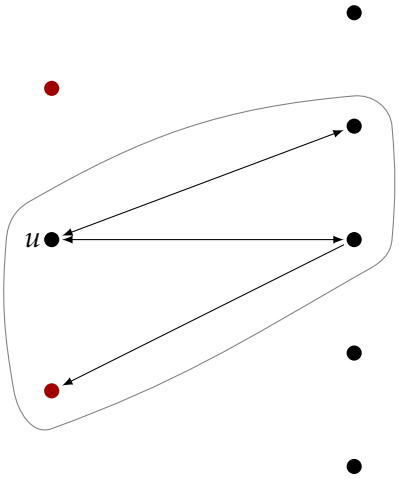
1. Initialize a set $\mathcal{U} = \emptyset$
2. Solve $\max\{c_1^T x^o + c_2^T x^t : (\bar{A}x^o \leq b) \wedge (\forall \bar{U} \in \mathcal{U}, x^t(\bar{U}) \leq \nu(\bar{U})) \wedge (x^o \in \{0,1\}^{n_1}) \wedge (x^t \in \{0,1\}^{|U|})\}$ to optimality and remember an optimal solution $(\hat{x}^o, \hat{x}^t)$ (if this problem is infeasible, the original problem was also infeasible).



(a) The original hypergraph with a highlighted (red) maximum matching $\hat{M}$ and the unmatched node u



(b) The auxiliary graph G' derived from $\hat{M}$



(c) The nodes reachable from u in G'

Figure 11: An example for the RISS algorithm on a hypergraph instance

3. Solve the maximum matching problem $\max\{1^\intercal y : (D_1 y \leq \hat{x}^t) \wedge (D_2 y \leq 1) \wedge (y \in \{0,1\}^{|E|})\}$ to optimality and remember an optimal solution $\hat{y}$.
4. If $1^\intercal \hat{y} < |\hat{U}|$ with $\hat{U} = \{u : \hat{x}^t_u = 1\}$, add $\hat{U}$ to $\mathcal{U}$ and go back to step 2.
5. Otherwise $(\hat{x}^o, \hat{x}^t, \hat{y})$ is an optimal solution to the original problem.

The maximum matching problem $\max\{1^\intercal y : (D_1 y \leq \hat{x}^t) \wedge (D_2 y \leq 1) \wedge (y \in \{0,1\}^{|E|})\}$ tries to cover as many vertices from U, that are represented by $\hat{x}^t$, as possible. The set of vertices chosen by $\hat{x}^t$ is denoted as $\hat{U}$. Each of these may be covered at most once and ideally we would like to see each one covered.

Improved feasibility cuts

We have arrived at a combinatorial Benders' algorithm that will result in an optimal solution (or report infeasibility) within finite time. But so far the cuts may affect

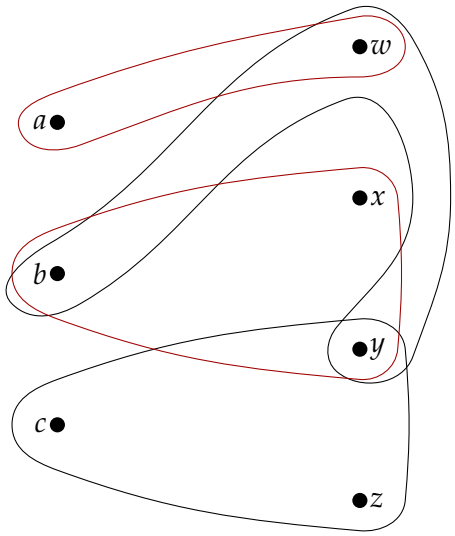


Figure 12: An instance where RISS fails to find an existing better division of vertices

much more vertices than those that were responsible for the maximum matching problem having no solution of sufficient size, similarly to the cuts we originally got for graphs. Therefore it would again be desirable to generate improved cuts, i. e., finding smaller infeasible subsystems for the combinatorial Benders' algorithm.

It turns out that the RISS algorithm from the previous section can be applied as well to the hypergraph setting. In fact neither in the definition of RISS, nor in its correctness proof (Theorem 4.3) it was required, that G is a graph. Therefore the subsets and their respective ν values will still be correct when the underlying graph is a hypergraph. Figure 11 shows RISS on an example hypergraph with a maximum matching of size 2. Using RISS only two of the vertices need to be involved in the feasibility cut.

Unlike in the graph case, RISS does not necessarily provide smallest infeasible subsystems. As an example, consider the graph G shown in Figure 12, defined as

$$
\begin{aligned}
G &= (U \cup V, E) \\
U &= \{a, b, c\} \\
V &= \{w, x, y, z\} \\
E &= \{\{a, w\}, \{b, w, y\}, \{b, x, y\}, \{c, y, z\}\}.
\end{aligned}
$$

Here, given the matching $\hat{M} = \{\{a, w\}, \{b, x, y\}\}$, one gets $\mathcal{R}(c) \cap U = \{a, b, c\}$. But as every edge, covering b and c, also covers y we get that $\nu(\{b, c\}) = 1$. As $\nu(\{a\}) = 1$ it would be a better choice to add the cuts for these subsets instead of adding that for $\{a, b, c\}$. It remains to be shown if finding such smallest possible subsets of the vertices is NP-complete or if there exists a better algorithm than RISS with polynomial complexity.

Even though RISS does not return the best possible infeasible subsystems, it can still vastly improve the quality of the added cuts and only takes a small amount of time to do so.

5

HETEROGENEOUS AGGREGATION FOR DANTZIG-WOLFE DECOMPOSITION

This chapter is based on a research project that the author of this thesis pursued together with his dear colleague Martin Bergner.

The heterogeneous aggregation scheme aims at exploiting certain symmetries when solving a Dantzig-Wolfe reformulation with a column generation procedure (see Sections 3.2 and 3.3). The method cleverly employs the theory around matchings and the partial transversal polytope (see Section 2.4). In this chapter the relevant theory will be established. To illustrate the concepts we will take the Multiple Knapsack Problem in several variations as an example. In Chapter 8 more applications of this methodology, as well as computational studies of its efficiency are shown.

AGGREGATION OF IDENTICAL SUBPROBLEMS

In Section 3.2 our goal was to solve a model of the form

$$\max\{c^\intercal x : (Ax \leq b) \wedge (Dx \leq d) \wedge (x \in X)\}.$$

The model may be some arbitrary MILP, so as usual $X = \mathbb{R}_+^{n_1} \times \mathbb{Z}_+^{n_2}$. We assume that D has block diagonal structure, leading to the model

$$\max\{\sum_{i\in[k]} {c^i}^\intercal x^i : (\sum_{i\in[k]} A_i x^i \leq b) \wedge (\forall i \in [k], D_i x^i \leq d^i) \wedge (x^i \in X_i)\}.$$

The $D_i d^i \leq b^i$ define our subproblem polyhedra

$$H^i_{\text{sub}} = \{x : (D_i x \leq d^i) \wedge (x \in X_i)\}.$$

For the rest of this chapter these are assumed to be bounded, i. e., there are no extreme rays. This is a crucial assumption as there is no generic way, known to the author of this thesis, to properly handle extreme rays in the aggregated master problem when it comes to performing integrality tests and deriving branching decisions.

As mentioned in Section 3.2, we assume that there is an easy way to find the extreme points of H^i_{sub} in any desired direction. Using the extreme points, we arrive at the master problem

$$\begin{aligned}
\max \quad & \sum_{i\in[k]} \sum_{p\in P^i} (c^{i\mathsf{T}} p)\lambda^i_p & & \\
\text{s.t} \quad & \sum_{i\in[k]} \sum_{p\in P^i} (A_i p)\lambda^i_p & \leq b & \\
& \sum_{p\in P^i} \lambda^i_p & = 1 & \forall i \in [k] \\
& \lambda & \in \mathbb{R}_+^{\sum_{i\in[k]}(|P^i|)}. &
\end{aligned}$$

If our problem turns out to have a symmetrical structure, where $c_i = c' \in \mathbb{Q}^{n'}$, $D_i = D' \in \mathbb{Q}^{m_D \times n'}$, $A_i = A' \in \mathbb{Q}^{m_A \times n'}$ and $X_i = X' = \mathbb{R}_+^{n'_1} \times Z_+^{n'_2}$ with $n' = n'_1 + n'_2$ for all $i \in [k]$ then we can significantly simplify this master problem. Instead of choosing one extreme point per subproblem, the goal is to select k extreme points from the single subproblem $H'_{\text{sub}} = \{x : (D'x \leq d') \wedge (x \in X')\}$, which can be done as the subproblems are all identical. Given that P' are the extreme points of H'_{sub} the master problem can now be aggregated by setting $\lambda_p = \sum_{i\in[k]} \lambda^i_p$. This leads to the following aggregated master problem (aMP):

$$\begin{aligned}
\max \quad & \sum_{p\in P'} (c'^{\mathsf{T}} p)\lambda_p & & \\
\text{s.t} \quad & \sum_{p\in P'} (A' p)\lambda_p & \leq b & \\
& \sum_{p\in P'} \lambda'_p & = k & \\
& \lambda & \in \mathbb{R}_+^{|P'|}. &
\end{aligned} \tag{5.1}$$

This way of aggregating identical subproblems is well known (see, e. g., chapter 13 of "50 years of integer programming"[97]). The aMP has several advantages over the original MP, as it is smaller and we only need to solve a single subproblem in the pricing step. Furthermore, as extreme points are valid for each subproblem, it is not necessary to add the same extreme point to each subproblem, reducing the number of variables by potentially a large factor. Also for the MP any solution can be permuted into $k!$ equivalent variations which can be a major issue in a branch & price algorithm. This problem is mitigated with the aMP as the decision which solution belongs to which subproblem is only made in the disaggregation step, which will be described after the following example.

Example

To illustrate aggregation, let us consider the following modified knapsack problem. Given a set I of items and a set K of knapsacks. Item $i \in I$ is described by its weight $a_i \geq 0$ and its profit $v_i \geq 0$. Knapsack $k \in K$ is described by its capacity c_k. The items can be placed in the knapsacks, as long as the total weight of the items does not exceed the knapsack's capacity. The goal is to find a packing of items that maximizes the total profit of the packed items. In order to have symmetrical subsystems in this example, the knapsack capacities will all be set equal as $\forall k \in K, c_k = c$.

We will use the following original formulation for this problem, where the binary variables $x_{i,k}$ encode whether item i is placed in knapsack k or not:

$$\begin{aligned}
\max \quad & \sum_{k\in K}\sum_{i\in I} v_i x_{i,k} & & & \\
\text{s.t.} \quad & \sum_{i\in I} a_i x_{i,k} & \leq c & \quad \forall k \in K & (5.2)\\
& \sum_{k\in K} x_{i,k} & \leq 1 & \quad \forall i \in I & (5.3)\\
& x & \in \{0,1\}^{|I|\cdot|K|}. & &
\end{aligned}$$

Here constraints (5.2) ensure that each knapsack's capacity is not exceeded. These constraints will form our subproblem. They make a good choice for subproblems, as each of them is simply a single knapsack constraint. Therefore the subproblems can be solved via a combinatorial algorithm for knapsack problems (e. g., via dynamic programming[57]). Constraints (5.3) ensure that each item is packed at most once. They will be put in the master problem.

When reformulating this original formulation as described in Section 3.2, each extreme point of the subproblems represents a feasible packing of items for the respective knapsack. As in this example all knapsacks have the same capacity, we can aggregate the master problem. As usual, the set P will contain all the subproblem's extreme points. This leads to the following aMP:

$$\begin{aligned}
\max \quad & \sum_{p\in P} v^\intercal p \lambda_p & & & \\
\text{s.t.} \quad & \sum_{\substack{p\in P\\ p_i=1}} \lambda_p & \leq 1 & \quad \forall i \in I & (5.4)\\
& \sum_{p\in P} \lambda_p & = |K| & & (5.5)\\
& \lambda & \in \mathbb{R}_+^{|P|}. & &
\end{aligned}$$

The constraints (5.4) are corresponding to constraints (5.3). There exists a variable per possible knapsack packing, making sure that each item is packed at most once and that exactly one packing is chosen per knapsack, which is enforced by constraint (5.5).

In a column generation procedure, the extreme points in P have to be generated on demand. Given that the dual values for constraints (5.4) are stored in vector π and that the dual value of (5.5) is stored in ρ, new extreme points can be found with the pricing problem:

$$\begin{aligned}
\max \quad & (\sum_{i\in I}(v_i - \pi_i)x_i) - \rho|K| & & \\
\text{s.t.} \quad & \sum_{i\in I} a_i x_i & \leq c + \rho & \\
& x & \in \{0,1\}^{|I|}. &
\end{aligned}$$

As already mentioned, these subproblems are simple knapsack problems, which can be solved very efficiently in practice. Note that if $c + \rho \leq 0$ the subproblem will be immediately infeasible.

So far we could solve the aMP using column generation, but we might end up with a fractional solution. In Section 3.3 we saw how to add branching constraints that are equivalent to branching decisions on the original variables. Such constraints may not be sufficient anymore in the aggregated case. It will therefore be necessary to ensure integrality in the aggregated master problem with an appropriate branching scheme, which will be described later.

Disaggregation

Given some solution of the aMP (or its LP relaxation), the challenge is to retrieve the values for the MPs variables λ_p^i from the solution. As the blocks are completely symmetrical, the solution can be distributed in any feasible way to the k blocks. It is desirable to have a disaggregation method that does preserve integrality, i. e., when all the λ_p of the aMP take integral values we do not want to distribute those in a way such that the λ_p^i are not integral anymore. This can easily be achieved by defining some order on the extreme points and then matching the aMP solution along this order to a corresponding MP solution. Such a method (with an additional requirement on the ordering) was described by Vanderbeck (2011)[160]. It works as follows.

First assume that there is a strict ordering $<$ of the extreme points in P'. Now some solution λ^* of the aMP can be disaggregated into a solution of the MP by the recursive rule

$$\lambda_p^{i\,*} = \min\{1, \lambda_p{}^* - \sum_{j\in[i-1]} \lambda_p^{j\,*}, (k - \sum_{p'<p} \lambda_p{}^*)^+\} \qquad \forall p \in P', i \in [k]. \tag{5.6}$$

The rule (5.6) ensures that

- no extreme point is assigned a value greater than 1.
- for each extreme point e no more than a total value of $\lambda_p{}^*$ is distributed among the blocks.
- within each block i a total value of at most 1 is distributed (in order to satisfy $\sum_{p\in P^i} \lambda_p^i = 1$).

This way any feasible solution of the aMP can be turned into a feasible counterpart of the MP. Note that any change in the ordering of the blocks might lead to a different feasible MP solution, which might also happen when changing the ordering $<$ of the extreme points.

If $\lambda^* \in \mathbb{Z}^{|P|}$ then this disaggregation method ensures that $\forall i \in [k], \lambda^{i*} \in \mathbb{Z}^{|P|}$ and therefore $x^i = \sum_{p\in P} \lambda_p^i p \in \mathbb{Z}^n$, i. e., the method preserves integrality of the aMP variables. But if the aMP solution is fractional, there may still exist some way to disaggregate the solution into an integral solution x^i of the original variables. In this case the suggested disaggregation method is not guaranteed to find integral solutions if they exist. It even turns out that trying to figure out such an integral disaggregation from a fractional aMP solution is in general an NP-complete task:

Theorem 5.1. *Given a set $P \subseteq \mathbb{Z}^n$ of extreme points, an integer number $k \geq 1$ of blocks, and values $\lambda_p \in \mathbb{R}_+$ for all $p \in P$, with $\sum_{p\in P} \lambda_p = k$. Deciding if the disaggregation problem has an integral solution, i. e., if there exists $\lambda_p^i \in \mathbb{R}_+$ for all $p \in P, i \in [k]$ such that $\forall p \in P, \sum_{i\in[k]} \lambda_p^i = \lambda_p$ and $\forall i \in [k], x^i = \sum_{p\in P} \lambda_p^i p \in \mathbb{Z}^n$, is NP-complete.*

Proof. The problem lies within NP, as one can easily verify whether a given distribution of the extreme points to the blocks leads to an integral solution.

The NP-hardness is shown via a reduction from the 3-dimensional matching problem. The 3-dimensional matching problem, as defined by Garey and Johnson (1979)[77], is given by three disjoint sets of vertices W, X, and Y of size $|W| = |X| = |Y| = k$ and a set $E \subseteq W \times X \times Y$ of hyperedges of cardinality 3 with one element from each set. The question for 3-dimensional matching is whether there exists a subset $M \subseteq E$ of pairwise disjoint edges with size $|E| = k$, i. e., covering all vertices. Answering this question is a NP-complete problem[100].

Given some 3-dimensional matching instance, where $V = W \cup X \cup Y$ is the set of all verties, let us construct a disaggregation problem with $|E|$ blocks and $2|V| + |E|$ extreme points of dimension $3|V| + |E| + 1$. Let $i(v) \in [|V|]$ be a unique index assigned to each vertex. Let $i(e) \in [|E|]$ be a unique index for each edge $e \in E$. In the following construction there will be an extreme point for each edge and two extreme points for each vertex. The first of each vertice's extreme points will be used to indicate chosen edges, the other one will be a dummy for the remaining edges. The construction will make sure that each edge extreme point must be paired with three vertex extreme points of the same type. One of the vertex extreme points will indicate the chosen edge and the other one will be a dummy for the remaining edges. The extreme points will be called p^e for each $e \in E$, p^v for each $v \in V$ and $\hat{p}^v$ as the dummy for $v \in V$. They are constructed as

$$p_i^e = \begin{cases} 1 & \text{if } \exists v \in e, i = i(v) \\ 2 & \text{if } i = 3|V| + 1 + i(e) \\ 0 & \text{otherwise} \end{cases}$$

$$p_i^v = \begin{cases} 3 & \text{if } i = i(v) \\ 2 & \text{if } i = |V| + 1 \\ 6 & \text{if } i = |V| + 1 + i(v) \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{p}_i^v = \begin{cases} 3 & \text{if } i = i(v) \\ 6 & \text{if } i = 2|V| + 1 + i(v) \\ 0 & \text{otherwise.} \end{cases}$$

Let P be the set containing all these points. When H_{sub} is chosen as the convex hull of P, then P will contain all extreme points of H_{sub}. Furthermore no $p \in P$ can be in the interior of H_{sub} as every point has one index with a non zero value that is unique to p. Therefore p cannot be a convex combination of the other points. While this does not give us further information about the form of H_{sub}, we know that there is some subproblem which will lead to the extreme points P.

The λ vector will be chosen as

$$\lambda_{p^e} = \frac{1}{2}, \lambda_{p^v} = \frac{1}{6}, \lambda_{\hat{p}^v} = \frac{|E(v)| - 1}{6}$$

where $E(v)$ is the set of edges incident to vertex $v \in V$. Note that these λ values satisfy the requirements of adding up to $|E|$, as

$$\frac{|V|}{6} + \sum_{v \in V} \frac{|E(v)| - 1}{6} = \frac{1}{6} \sum_{v \in V} |E(v)| = \frac{1}{2}|E|$$

where the last step follows from the fact that each edge contains exactly 3 vertices.

In order for the reduction to work it needs to be shown that a feasible solution of the 3-dimensional matching problem corresponds to an integral disaggregation of these extreme points. Let $M \subseteq E$ be a 3-dimensional matching. For the block corresponding to $e \in M$ the disaggregation of the extreme points will be given as

$$\lambda_p^{i(e)} = \begin{cases} \frac{1}{2} & \text{if } p = p^e \\ \frac{1}{6} & \text{if } \exists v \in e, p = p^v \\ 0 & \text{otherwise.} \end{cases}$$

I. e., these blocks receive the extreme points corresponding to their respective edges and the non dummy extreme points for the edges vertices.

Equivalently for $e \in E \setminus M$ the extreme points will be distributed in the following way

$$\lambda_p^{i(e)} = \begin{cases} \frac{1}{2} & \text{if } p = p^e \\ \frac{1}{6} & \text{if } \exists v \in e, p = \hat{p}^v \\ 0 & \text{otherwise.} \end{cases}$$

Note that there is always exactly one block for each edge which means that $\sum_{i \in [k]} \lambda^i_{p^e} = \lambda_{p^e}$ for all p^e. For p^v there is exactly one i such that $\lambda^i_{p^v} = \frac{1}{6}$, as every vertex is covered by exactly one edge $e \in M$. Equivalently each vertex v is covered by exactly $|E(v)| - 1$ edges from $E \setminus M$ which means that for $\hat{p}^v$ we get $\sum_{i \in [k]} \lambda^i_{\hat{p}^v} = (|E| - 1) \cdot \frac{1}{6} = \lambda_{\hat{p}^v}$.

Next we need to verify that this disaggregation is indeed integral. As each block corresponds to exactly one edge we can look at these individually. First we consider the edge $e \in E \setminus M$. For block $i(e)$, the disaggregated solution is $x^{i(e)} = \frac{1}{2}p^e + \sum_{v \ni e} \frac{1}{6}\hat{p}^v$. Now we can check the integrality of $x^{i(e)}$ for each dimension i. For $i = 3|V| + 1 + i(e)$ only p^e has a non zero entry (with value 2) therefore the corresponding position in $x^{i(e)}$ is $\frac{1}{2} \cdot 2$. Equivalently for entries $2|V| + 1 + 1$ till $2|V| + 1 + |V|$, each has one corresponding vertex v such that the relevant entry in $\hat{p}^v$ has a non zero value (of 6) if $v \in e$ and zero otherwise. Again this yields an integral entry for $x^{i(e)}$ of 1. The last range of indices where we have non zero values is 1 till $|V|$. If $v \notin e$ then $\lambda^{i(e)}_{\hat{p}^v} = 0$ and therefore $x^{i(e)}_{i(v)} = 0$. For $v \in e$ we get an entry from both p^e as well as $\hat{p}^v$, resulting in $x^{i(e)}_{i(v)} = \frac{1}{2} \cdot 1 + \frac{1}{6} \cdot 3 = 1$.

Now we need to verify that also the blocks for $e \in M$ are integral. Similar to before, the disaggregated solution for block $i(e)$ now is $x^{i(e)} = \frac{1}{2}p^e + \sum_{v \ni e} \frac{1}{6}p^v$. For indices $i = 3|V| + 1 + i(e)$ there is again only one edge and therefore only a single non zero entry with value 1. In the same way for indices of the form $|V| + 1 + i(v)$ we get an entry of $6 \cdot \frac{1}{6} = 1$ if $v \in e$ and zero otherwise. The last index we need to check is $|V| + 1$. As e covers exactly 3 vertices, we get $x^{i(e)}_{|V|+1} = 3 \cdot 2 \cdot \frac{1}{6} = 1$. Therefore the resulting solution is integral.

Finally it needs to be verified that an integral disaggregation of our disaggregation instance corresponds to a feasible solution of the 3-dimensional matching problem. First note that the extreme point components with indices starting at $|V|+2$ always occur in only one extreme point. They make sure that in an integral solution no less than a fraction of $\frac{1}{2}$ can be chosen for an edge extreme point and no less than $\frac{1}{6}$ for a vertex or a dummy extreme point.

Duplicate edges in the 3-dimensional matching instance can be removed without affecting the problem's complexity, therefore we can assume that no edge occurs twice. In an integral solution no block can be assigned two edge extreme points, because there must be at least one vertex not shared by the edges, for which the respective component $i(v)$ would end up with a non integral value of $\frac{1}{2}$. As there are a total of $|E|$ blocks, each block must be assigned exactly one edge extreme point with value $\frac{1}{2}$.

Next consider the distribution of the vertex extreme points p^v. Each block can host between one and three of them (taking into account the space already taken by the edge extreme points). If at least one p^v is assigned to a block, the component with index $|V|+1$ dictates that two more vertex extreme points must be assigned to this block in order to arrive at an integral solution. In order to achieve an integral entry in component $i(v)$, the vertex extreme point must be matched with an edge extreme point p^e such that $v \in e$.

Putting these findings together, a block in an integral solution will consist of one edge extreme point, either matched with three vertex extreme points or three dummy extreme points which fit to the corresponding edge. Those edges matched with the vertex extreme points form a solution to the 3-dimensional matching, which concludes the reduction. □

Integrality test

If some of the variables in the original formulation need to be integral, the aggregated master problem will need to be embedded in a branch & bound framework. Due to the aggregation, this can be even more challenging than for the non-aggregated master problem (see Section 3.3).

The first step is to check whether the current solution at hand is already integral. Note that the disaggregation scheme ensures that integral λ of the aMP translate to integral x of the original problem. But a fractional λ can still represent an integral solution of the original problem. We can stop if $x^{i^*} = \sum_{e \in P'} p \lambda_p^{i^*} \in X'$. Otherwise we need to perform branching which will be explained in the next section.

BRANCHING WITH IDENTICAL SUBPROBLEMS

This section will deal with the issue of adding branching decisions to the aMP in order to achieve integrality. It turns out that additional effort is required as a branching scheme based on the original variables might no longer be able to exclude all fractional solutions. Two branching schemes will be presented to handle this issue.

Feasibility of branching on original variables

Section 3.3 described how in a branch & price algorithm branching may be carried out such that it is equivalent to branching on the variables of the original formulation. When aggregation is used, this might no longer be sufficient to reach an integral solution. First note that due to the aggregation, the original variables simply do not have a direct correspondent in the aMP. If we aggregate the original variables as $\sum_{i\in[k]} x^i = y$ we get an aggregated equivalent of the original variables. In the aMP we could make branching decisions based on the y variables in a similar fashion as described in Section 3.3. Sadly it can happen, that y turns out to be integral in all components while the disaggregated x^i are not. This is especially the case when set partitioning constraints or other equalities are involved, which force y to be integral.

Even though branching on original variables does not guarantee an integral solution, it can still be a useful tool, especially as – depending on the problem structure – these branching decisions may be handled with greater computational efficiency.

As an example where branching on original variables helps in improving the dual bound, consider the following multiple knapsack problem. Given two knapsacks with capacity 4 each and four items with weights $a_1 = a_2 = a_3 = 2$ and $a_4 = 3$, as well as profits of 1 each. Optimally we can pack two of the first three items together and then choose one last item to pack in the second knapsack, leading to a total profit of 3. But a fractional aMP solution can do better. Using the subproblem extreme points $p_1 = (1,1,0,0)^\intercal$, $p_2 = (0,1,1,0)^\intercal$, $p_3 = (1,0,1,0)^\intercal$, and $p_4 = (0,0,0,1)$, a feasible aMP solution would be to set $\lambda_{p_1} = \lambda_{p_2} = \lambda_{p_3} = \lambda_{p_4} = \frac{1}{2}$. This way the first three items are chosen fully, while only half of the fourth item is taken. The aggregated original variables are $y = (1,1,1,0.5)^\intercal$. Branching on the fourth item yields the two branching constraints

$$\sum_{\substack{p\in P\\ p_4=1}} \lambda_p \leq 0 \text{ and } \sum_{\substack{p\in P\\ p_4=1}} \lambda_p \geq 1.$$

Now in each branch, the resulting solution will be integral, either choosing the first three items or two of the first three items and the fourth.

Discretization

When branching on (aggregated) original variables one can be sure not to cut off a feasible integral solution by accident. When turning to more advanced branching schemes, it gets more difficult to ensure this does not happen. As explained earlier, the disaggregation scheme we use is not guaranteed to find an integral disaggregation from a fractional aMP solution even if it exists and there is not much hope to find another method to do so in polynomial time (see Theorem 5.1). By cutting off this fractional aMP solution by the branching constraints, we might rid ourselves from the optimum.

A way to circumvent this problem is to ensure that for every integral original solution there exists an aMP solution with integer variables. In order to assert this property one can use the discretization approach (see Section 3.3) and make sure that the aMP has a variable for every subproblem solution, not just for each extreme point. Note that Theorem 5.1 means that it might make more sense to use

discretization than to try constructing a procedure to optimally disaggregate the aMP solution and is able to create branching decisions which exclude non integral aMP solutions but do not cut off optimal solutions by accident.

Ryan and Foster branching rule

One branching rule to handle aggregated problems with a certain structure was described by Ryan and Foster (1981)[139]. It requires the master problem to expose either a set packing or a set partitioning structure (see Section 1.8). Even though these restrictions are imposed on the problem, many important problem classes display the corresponding structure.

We require that the variables of the original problem are either fractional or binary, i. e., $x^i \in X' = \{0,1\}^{n'_1} \times \mathbb{R}_+^{n'_2}$. In branching we only need to enforce the integrality of the binary variables, therefore let B be the set of indices for which x^i should be binary and x^i_B the binary part of x^i.

Additionally we need the aMP to be of the form

$$\begin{aligned}
\max \quad & \sum_{p \in P} c'^{\intercal} p \lambda_p & & & \\
\text{s.t.} \quad & \sum_{\substack{p \in P \\ p_j = 1}} \lambda_p & [\leq \text{ or } =] 1 & \forall j \in B & (5.7) \\
& \sum_{p \in P} \lambda_p & = k & & \\
& \lambda & \in \mathbb{R}_+^{|P|} & &
\end{aligned}$$

where each of the constraints (5.7) is either a set packing or a set partitioning constraint. This structure enforces that each of the binary elements in the original variables is chosen either at most once or exactly once. For the aggregated original variables $y = \sum_{i \in [k]} x^i$ this implicates $y \leq 1$. In the case of set partitioning constraints the y will be forced to be integral. When dealing with set packing constraints, this needs to be ensured before using the branching rule by Ryan and Foster. This can be achieved by branching on the aggregated original variables, as described earlier.

When looking at two indices $j_1, j_2 \in B$, $j_1 \neq j_2$, with $y_{j_1} = y_{j_2} = 1$, there are now four distinct configurations for the corresponding original variables in each block: $x^i_{\{j_1,j_2\}} \in \{(0,0)^{\intercal}, (0,1)^{\intercal}, (1,0)^{\intercal}, (1,1)^{\intercal}\}$. We can count how often the $(1,1)^{\intercal}$ configuration is chosen, i. e., how often the two components appear together. To do so, we define the virtual variables

$$z_{j_1,j_2} = \sum_{\substack{p \in P \\ p_{j_1} = p_{j_2} = 1}} \lambda_p.$$

Assuming that all z are integral, we can fully disaggregate the binary parts of the original variables x^i_B to an integral solution. This works, e. g., by starting with setting $x^1_j = 1$ (where $y_j = 1$) for some $j \in B$, and then put $x^1_{j'} = z_{j,j'}$ for $j' \in B \setminus \{j\}$, repeating this for the other blocks and binary variables which were not yet distributed. Now the binary parts give a fixed disaggregation scheme for the extreme points, so the fractional parts can be disaggregated in a feasible way.

Therefore if we cannot disaggregate the aMP into an integral solution, there must be two indices $j_1, j_2 \in B, j_1 \neq j_2$ such that $z_{j_1,j_2} \notin \{0,1\}$. We can now branch on these virtual variables. Ryan and Foster (1981)[139] propose to handle the branching as follows.

In one branch, z_{j_1,j_2} will be forced to 1. This branch is called the "same branch", because any feasible solution enforces $x^i_{j_1} = x^i_{j_2}$. In this branch one now proceeds by setting the λ_p to 0 in the aMP if p is infeasible with respect to the branching decision, i. e., $p_{j_1} \neq p_{j_2}$. The subproblems need to be changed to

$$\max\{(c^\intercal - \pi^\intercal A')x : (D'x \leq d) \wedge (x_{j_1} = x_{j_2}) \wedge (x \in X')\}$$

to ensure that only new variables obeying the branching decision are created. In the "differ branch", where w_{j_1,j_2} is forced to 0, the subproblem accordingly is changed to

$$\max\{(c^\intercal - \pi^\intercal A')x : (D'x \leq d) \wedge (x_{j_1} \leq 1 - x_{j_2}) \wedge (x \in X')\}$$

to ensure that no extreme points are generated, where both variables are present. Again, in the aMP, we need to set $\lambda_p = 0$, when $p_{j_1} = p_{j_2} = 1$.

Note that these additional constraints in the master problem can significantly alter the subproblems structure. It can easily happen, that a previously employed combinatorial algorithm will cease to work on the new structure. In such cases other options to handle pricing are available, depending on the pricing structure[97].

Vanderbeck branching rule

The branching rule by Ryan and Foster can be extended to more general problems, as was shown by Vanderbeck (2011)[160]. For this to work, it is required that the ordering of the feasible points in the integrality test is the lexicographical ordering. Note that there exist multiple such orderings as one can just reorder the variables.

In the Ryan and Foster branching scheme we choose indices j_1 and j_2 for which we enforced that either a point with both set to 1 is in the solution or none. In the general case this is not sufficient, as on the one hand the components in a feasible solution may have values greater than 1 and on the other hand feasible points with the corresponding values may be used multiple times in the aMP.

In order to be more generic, the branching rule will be based upon two vectors $s \in \{-1,0,1\}^{n'}$ and $l \in \mathbb{Z}^{n'}$ of size corresponding to the subproblem size. Now the interesting feasible points are $F(s,l) = \{p : (p \in \bar{P}) \wedge (diag(s)p \geq l)\}$, where $\bar{P}$ are the points we have added to the master problem so far. Note that for $s_{j_1} = s_{j_2} = 1$, $\forall j \notin \{j_1, j_2\}, s_j = 0$ and $l = 1$, the set $P(s,l)$ will be exactly the set of feasible points where both components are set to at least one, as it was chosen in the Ryan and Foster rule.

Vanderbeck (2011)[160] explains that if the integrality test (based on the lexicographical ordering) fails, there always exist s and l such that

$$\sum_{p \in P(s,l)} \lambda_p = \alpha \notin \mathbb{Z}.$$

Based upon this, we can divide the problem alongside the branching constraints $\sum_{p \in F(s,l)} \lambda_p \geq \lceil \alpha \rceil$ and $\sum_{p \in F(s,l)} \lambda_p \leq \lfloor \alpha \rfloor$.

As in the Ryan and Foster branching rule, the subproblem needs to be adjusted based upon the branching decision taken. These adjustments are not trivial and can again significantly alter the structure of the subproblem. Furthermore precautions need to be taken on the choice of s and l in later branching decisions, to avoid an explosion in the complexity of the subproblem. For more details on how this can be handled in practice, please refer to Vanderbeck (2011)[160].

AGGREGATION WITH HETEROGENEOUS SUBPROBLEMS

In this section we will look at a more general case than we did previously, when aggregating identical subproblems. Again we start with a block diagonal structure

$$\max\{\sum_{i\in[k]} {c^i}^\intercal x^i : (\sum_{i\in[k]} A_i x^i \leq b) \wedge (\forall i \in [k], D_i x^i \leq d^i) \wedge (x^i \in X_i)\}.$$

But this time we only require that for all $i \in [k]$ we have $c^i = c'$, $A_i = A'$, and $X_i = X'$. This means we require that each block has the same structure in the master problem but can have a different subproblem. Again it is possible, that solutions can easily be swapped between blocks (even though not necessarily to all $k!$ permutations). Therefore it would be desirable to still have a way to aggregate problems with such structure.

As the subproblems now can have different sets of extreme points, we collect all of them in one set $P = \bigcup_{i\in[k]} P_i$. Lacking the distinction of blocks in the master problem means that, as long as only the master problem is concerned, one could just aggregate as before and arrive at the same aMP (5.1) as before. But now the problem arises that one of the solutions produced from this aMP cannot necessarily be disaggregated into a feasible original solution. Therefore additional work needs to be performed in the aMP to ensure feasibility.

Example

In the previous multiple knapsack example, each knapsack had the same capacity c. If we modify this setting such that each knapsack k has its own capacity c_k, our original formulation becomes

$$\begin{aligned}
\max \quad & \sum_{k\in K}\sum_{i\in I} v_i x_{i,k} & & \\
\text{s.t.} \quad & \sum_{i\in I} a_i x_{i,k} & \leq c_k & \quad \forall k \in K \\
& \sum_{k\in K} x_{i,k} & \leq 1 & \quad \forall i \in I \\
& x & \in \{0,1\}^{|I|\times|K|}. &
\end{aligned}$$

As already mentioned, it may often happen that the packings of two knapsacks can be easily exchanged if their capacities are similar, which would imply that many packings will be present multiple times for different knapsacks, leading to copies of variables carrying no additional information.

Disaggregation

When the subproblems are not identical, not every extreme point can be matched to any block anymore. Therefore disaggregation becomes more complicated. We can create a bipartite graph $G = (P \cup [k], E)$, where there is an edge $\{p, i\} \in E$ if $p \in H^i_{\text{sub}}$. This graph indicates which extreme points can be matched to which block. Assuming that the aMP solution is integral, disaggregation becomes equivalent to finding a matching in G between the $[k]$ vertices and those vertices from P where $\lambda_e \geq 1$. Note that to get a proper matching problem one needs to copy vertices $p \in P$ where $\lambda_p > 1$, a modification that does not add to the problems complexity. Also note that the resulting disaggregated solution will be integral.

The idea of a bipartite graph between extreme points and subproblems allows us the corresponding graph theoretical notation in cases where it is convenient, such as $N(i)$ and $N^{-1}(i)$ to refer to the different kinds of neighborhood of i.

If the aMP solution is not fractional, disaggregation can be achieved by solving a maximum flow problem on the graph $G_h = (P \cup [k] \cup \{s, t\}, E_h)$ with edges $E_h = E \cup \{(s, p) : p \in P\} \cup \{(i, t) : i \in [k]\}$, source s and sink t. The edge capacities are given as

$$\begin{aligned} w_{(p,i)} &= 1 && \forall i \in [k], p \in H^i_{\text{sub}} \\ w_{(s,p)} &= \lambda_p && \forall p \in P \\ w_{(i,t)} &= 1 && \forall i \in [k]. \end{aligned}$$

If the maximum flow on G_h has a total value of k, the aMP solution is valid and can be disaggregated. When $\phi_{(p,i)}$ is the corresponding flow on edge (p, i) for $p \in P$ and $i \in [k]$, the disaggregated solution is

$$\forall i \in [k], x^i = \sum_{p \in N(i)} \phi_{(p,i)} p.$$

This method of disaggregation preserves the integrality of the aMP solution if the underlying maximum flow algorithm finds integral flows in graphs with integral capacities (see Theorem 1.2). E. g., solving the maximum flow algorithm the Ford and Fulkerson algorithm will return an integral flow in this case[5].

As any way of disaggregating the aMP solution is equivalent to a flow in G_h, a maximum flow of value less than k is equivalent to an infeasible aMP solution. Therefore the maximum flow of size k is a necessary and sufficient condition for the disaggregatability of the aMP if the aMP solution is integral.

Ensuring feasibility in the aMP

As we have just seen, the disaggregation of the aMP is closely related to finding (fractional) matchings in a bipartite graph and the algorithm for disaggregation resembles the separation routine for the partial transversal polytope explained in Section 2.4. In fact an aMP solution only cannot be disaggregated, if it lies outside of the partial transversal polytope defined by the graph G from the previous paragraph. In order to ensure that the aMP solution can be disaggregated (and

is therefore feasible), we need to add the corresponding constraints to the aMP, which now becomes

$$\begin{aligned}
\max \quad & \sum_{p\in P} (c'^{\mathsf{T}} p)\lambda_p & & \\
\text{s.t} \quad & \sum_{p\in P} (A' p)\lambda_p & \leq b & \\
& \sum_{p\in P} \lambda_e & = k & \\
& \sum_{p\in N^{-1}(\bar{I})} \lambda_p & \leq |\bar{I}| \qquad & \forall \bar{I} \subseteq [k] \\
& \lambda & \in \mathbb{R}_+^{|P|}. &
\end{aligned} \tag{5.8}$$

With the partial transversal polytope constraints added, any solution of the aMP (5.8) can be disaggregated. As there are 2^k subsets $\bar{I} \subseteq [k]$, it may be intractable to add all of the partial transversal polytope constraints right away. Instead it is advisable to use a separation algorithm which adds those constraints on demand. Note that the maximum flow method used for disaggregating the aMP solution is equivalent to the minimum cut method that was used in Section 2.4 to separate the partial transversal polytope (see Theorem 1.3). Therefore if an aMP solution cannot be disaggregated, one immediately has a necessary cut at hand which needs to be added.

Special case: ordered subproblems

As seen in Theorem 2.17 the number of required constraints for the partial transversal polytope is small, if the matching graph has an ordered structure. Applied to the heterogeneous aggregation problem this means that the subproblems can be ordered such that a feasible solution for one subproblem is also feasible for all preceding subproblems, i. e., $\forall i \in [k], \forall x \in X', (D_i x \leq d_i \Rightarrow (\forall j \leq i, D_j x \leq d_j))$.

For such instances with ordered subproblems, the aMP can be formulated as

$$\begin{aligned}
\max \quad & \sum_{p\in P} (c'^{\mathsf{T}} p)\lambda_p & & \\
\text{s.t} \quad & \sum_{p\in P} (A' p)\lambda_p & \leq b & \\
& \sum_{p\in P} \lambda_p & = k & \\
& \sum_{p\in N^{-1}([i])} \lambda_p & \leq i \qquad & \forall i \in [k] \\
& \lambda & \in \mathbb{R}_+^{|P|}. &
\end{aligned}$$

and consequently the aMP can be solved without using a separation algorithm to generate the partial transversal polytope constraints on demand.

Example

The multiple knapsack problem is among the problems which have ordered subproblems, as a packing of a smaller knapsack will always fit into a larger one and

we can order the knapsacks such that $c_1 \geq c_2 \geq \ldots \geq c_k$. As a result only a small number of partial transversal polytope constraints are required. This means we can formulate the aMP without a separation routine as

$$\begin{aligned}
\max \ & \sum_{p \in P} v^\intercal p \lambda_p & & & \\
\text{s.t.} \ & \sum_{\substack{p \in P \\ p_i = 1}} \lambda_p & & \leq p & \forall i \in I \\
& \sum_{p \in P} \lambda_p & & = |K| & \\
& \sum_{\substack{p \in P \\ \forall k' \notin \bar{K}, a^\intercal p > c_{k'}}} \lambda_p & & \leq |\bar{K}| & \forall k \in K, \bar{K} = \{k' \in K : c_{k'} \geq c_k\} \\
& \lambda & & \in \mathbb{R}_+^{|P|}. &
\end{aligned}$$

Here the partial transversal polytope constraints are given by one constraint per knapsack. Each constraint ensures that for the group of the corresponding knapsack and the knapsacks of the same or larger capacity, not more packings are chosen, that do not fit into some smaller knapsack, than the number of knapsacks in the group.

A modification of the multiple knapsack problem without this property can be obtained by modifying the item weights such that each item can have different weights in different knapsacks, i. e., we now have weights $a_{i,k}$ for item i if it is to be placed in knapsack k.

BRANCHING WITH HETEROGENEOUS SUBPROBLEMS

As in the case of identical subproblems, branching decisions for heterogeneous subproblems need some additional work in order to arrive at integral solutions. Again branching on the original variables may help in some cases but is not guaranteed to result in an integral solution.

Ryan and Foster branching rule

The branching rule by Ryan and Foster[139] remains applicable for heterogeneously aggregated problems, as long as the underlying original formulation is a purely binary problem. In that case the virtual z variables completely determine which entries with value 1 occur together in one of the blocks, if the z turn out to be binary. Afterwards disaggregation is a simple bipartite matching problem.

To see why the branching rule might fail to result in an integral solution in problems with fractional variables, consider a problem with variable space $X' = \{0,1\}^2 \times \mathbb{R}$ and two blocks, where the first subproblem block has a constraint $(0,0,1)x \leq 1$ and the second one has a constraint $(0,0,-1)x \leq 2$. Now assume that the following extreme points for the subproblems exist.

$$p_1 = \begin{pmatrix} 1 \\ 1 \\ 0.5 \end{pmatrix}, p_2 = \begin{pmatrix} 1 \\ 1 \\ 2.5 \end{pmatrix}, p_3 = \begin{pmatrix} 0 \\ 0 \\ 0.5 \end{pmatrix}, p_4 = \begin{pmatrix} 0 \\ 0 \\ 2.5 \end{pmatrix}.$$

For a master solution $\lambda_{p_1} = \ldots = \lambda_{p_4} = 0.5$, the virtual variable $z_{1,2}$ would be 1, as index 1 and 2 only occur together. But the fractional (and not branched upon) part prevents an integral disaggregation of this solution.

Feasibility of Vanderbecks branching rule

For more general integer programming problems one would also like to use the Vanderbeck branching rule to enforce integrality of the master solution. Vanderbeck (2011)[160] needed the disaggregation to follow a strict scheme, where the extreme points were assigned to blocks according to a lexicographical ordering. In heterogeneous aggregation this is no longer possible, as we need to use a different disaggregation methodology. Therefore the original proof for the existence of branching set defining vectors l and s no longer applies and we cannot guarantee that the scheme will always arrive at an integral solution.

Note that in the binary case with only set packing and set partitioning constraints in the master problem, the Vanderbeck branching rule is equivalent to a combination of branching on original variables and the Ryan and Foster branching rule and therefore still works, as the Ryan and Foster rule is still applicable.

PRICING WITH HETEROGENEOUSLY AGGREGATED SUBPROBLEMS

When using heterogeneous aggregation, many (potentially an exponential amount of) constraints are added to the master problem in order to ensure that we can disaggregate a solution. Each of those constraints translates into a dual value that may affect the subproblem. Therefore it is important to ensure that the subproblem can still be solved efficiently and that these additions do not destroy structural properties that were important for efficiency. This section will show how it is possible to perform pricing in the presence of these obstacles.

The proposed pricing technique will use modified dual constraints for checking the optimality of the aMP. It will be shown that these modified constraints are equivalent to the original ones if the dual solution has a certain property that will be called σ-nested. Furthermore it will be shown that any dual solution of the aMP can be transformed into a σ-nested dual solution with the same objective value. This transformation does not alter the modified constraints mentioned before which will lead to the insight that the new constraints can also be used if the dual solution at hand has not the special property of being σ-nested.

Nested dual solutions

The first step is to look more closely at possible dual solutions of the aMP, especially the dual variables corresponding to the newly introduced constraints for the partial transversal polytope. Given that σ are the dual variables for the partial

transversal polytope constraints and ρ and π are those for the convexity and the remaining aMP constraints, the dual formulation of the aMP is

$$\begin{aligned} \min\ & b^\intercal \pi + k\rho + \sum_{\bar{I}\subseteq[k]} |\bar{I}|\sigma_{\bar{I}} \\ \text{s.t. } & (A'p)^\intercal \pi + \rho + \sum_{\bar{I}\supseteq N(p)} \sigma_{\bar{I}} \geq c'^\intercal p \qquad \forall p \in P \\ & (\pi, \rho, \sigma) \qquad\qquad \in \mathbb{R}_+^{m_A} \times \mathbb{R} \times \mathbb{R}_+^{2^k}. \end{aligned} \tag{5.9}$$

The $\sum_{I\supseteq N(p)} \sigma_I$ terms introduce significant changes into the dual formulation. In order to deal with these it will help if the σ variable have the following structure

Definition 5.2. An aMP dual solution is said to be σ-nested iff for $\bar{I}_1, \bar{I}_2 \subseteq [k]$ with $\sigma_{\bar{I}_1}, \sigma_{\bar{I}_2} > 0$ and $\bar{I}_1 \neq \bar{I}_2$ it holds that either $\bar{I}_1 \subset \bar{I}_2$ or $\bar{I}_2 \subset \bar{I}_1$.

The goal of this section will be to show that an aMPs dual solution can be expected to be σ-nested. The first step towards this goal is to establish that the dual variable values can be shifted away from non nested to nested sets.

Lemma 5.3. *Let (π, ρ, σ) be a solution to (5.9). Given $\bar{I}_1, \bar{I}_2 \subseteq [k]$, with $(\bar{I}_1 \not\subseteq \bar{I}_2) \wedge (\bar{I}_2 \not\subseteq \bar{I}_1) \wedge (\sigma_{\bar{I}_1}, \sigma_{\bar{I}_2} > 0)$, then (π, ρ, σ') with*

$$\sigma'_{\bar{I}} = \begin{cases} \sigma_I - \min\{\sigma_{\bar{I}_1}, \sigma_{\bar{I}_2}\} & \text{if } \bar{I} \in \{\bar{I}_1, \bar{I}_2\} \\ \sigma_I + \min\{\sigma_{\bar{I}_1}, \sigma_{\bar{I}_2}\} & \text{if } \bar{I} \in \{\bar{I}_1 \cup \bar{I}_2, \bar{I}_1 \cap \bar{I}_2\} \\ \sigma_{\bar{I}} & \text{otherwise} \end{cases}$$

is also a solution to (5.9) with the same objective value.

If (π, ρ, σ) was feasible then so is (π, ρ, σ').

Proof. In order to verify the equality of the objective values, we look at the four terms in the dual master problem that change due to the new σ' values, i. e., $|\bar{I}_1| \cdot \sigma'_{\bar{I}_1}$, $|\bar{I}_2| \cdot \sigma'_{\bar{I}_2}$, $|\bar{I}_1 \cup \bar{I}_2| \cdot \sigma'_{\bar{I}_1 \cup \bar{I}_2}$ and $|\bar{I}_1 \cap \bar{I}_2| \cdot \sigma'_{\bar{I}_1 \cap \bar{I}_2}$. On the one hand we have

$$\begin{aligned} & |\bar{I}_1| \cdot \sigma'_{\bar{I}_1} + |\bar{I}_2| \cdot \sigma'_{\bar{I}_2} \\ = {} & |\bar{I}_1| \cdot \sigma_{\bar{I}_1} + |\bar{I}_2| \cdot \sigma_{\bar{I}_2} - (|\bar{I}_1| + |\bar{I}_2|) \cdot \min(\sigma_{\bar{I}_1}, \sigma_{\bar{I}_2}) \end{aligned}$$

and on the other hand we get

$$\begin{aligned} & |\bar{I}_1 \cup \bar{I}_2| \cdot \sigma'_{\bar{I}_1 \cup \bar{I}_2} + |\bar{I}_1 \cap \bar{I}_2| \cdot \sigma'_{\bar{I}_1 \cap \bar{I}_2} \\ = {} & |\bar{I}_1 \cup \bar{I}_2| \cdot \sigma_{\bar{I}_1 \cup \bar{I}_2} + |\bar{I}_1 \cap \bar{I}_2| \cdot \sigma_{\bar{I}_1 \cap \bar{I}_2} + (|\bar{I}_1| + |\bar{I}_2|) \cdot \min(\sigma_{\bar{I}_1}, \sigma_{\bar{I}_2}) \end{aligned}$$

as $|\bar{I}_1| + |\bar{I}_2| = |\bar{I}_1 \cup \bar{I}_2| + |\bar{I}_1 \cap \bar{I}_2|$. Adding these two equations together, we get that there is no overall change in the objective value.

It remains to show that the new dual solution is feasible if (π, ρ, σ) was feasible. As there is no change in the π and ρ values, we need to ensure that $\forall p \in P, \sum_{\bar{I}\supseteq N(p)} \sigma_{\bar{I}} \leq \sum_{\bar{I}\supseteq N(p)} \sigma'_{\bar{I}}$. Of course this sum is again relevant only for the four changing terms of σ. In order to do so we distinguish four cases for the extreme points:

$N(p) \subseteq \bar{I}_1 \cap \bar{I}_2$: This case implies $(N(p) \subseteq \bar{I}_1) \wedge (N(p) \subseteq \bar{I}_2) \wedge (N(p) \subseteq \bar{I}_1 \cup \bar{I}_2)$, therefore all four changing σ terms are part of the corresponding constraint. From the construction of σ' it follows directly that $\sigma_{\bar{I}_1} + \sigma_{\bar{I}_2} + \sigma_{\bar{I}_1 \cup \bar{I}_2} + \sigma_{\bar{I}_1 \cap \bar{I}_2} = \sigma'_{\bar{I}_1} + \sigma'_{\bar{I}_2} + \sigma'_{\bar{I}_1 \cup \bar{I}_2} + \sigma'_{\bar{I}_1 \cap \bar{I}_2}$ which implies the feasibility for this extreme point's constraint.

$(N(p) \subseteq \bar{I}_1) \wedge (N(p) \not\subseteq \bar{I}_2)$: In this case we have that $N(p) \subseteq \bar{I}_1 \cup \bar{I}_2$ and $N(p) \not\subseteq \bar{I}_1 \cap \bar{I}_2$. With only two relevant terms in the corresponding constraint checking that $\sigma_{\bar{I}_1} + \sigma_{\bar{I}_1 \cup \bar{I}_2} = \sigma'_{\bar{I}_1} + \sigma'_{\bar{I}_1 \cup \bar{I}_2}$ (which holds as the min terms cancel each other) is sufficient to show feasibility.

$(N(p) \subseteq \bar{I}_2) \wedge (N(p) \not\subseteq \bar{I}_1)$: This case is equivalent to the previous one.

$(N(p) \not\subseteq \bar{I}_1) \wedge (N(p) \not\subseteq \bar{I}_2)$: In this case also $N(p) \not\subseteq \bar{I}_1 \cap \bar{I}_2$. Therefore only the component for $\bar{I}_1 \cup \bar{I}_2$ might be present in the constraint. Since $\sigma_{\bar{I}_1 \cup \bar{I}_2} \leq \sigma'_{\bar{I}_1 \cup \bar{I}_2}$ the constraint is again satisfied.

As the constraints of (5.9) will be satisfied for each extreme point $p \in P$, the newly constructed dual solution (π, ρ, σ') is again feasible, which concludes the lemma. □

Note that the procedure described in Lemma 5.3 not only retains dual feasibility for the aMP but might increase the left hand sides of the dual constraints in (5.9). Also an optimal dual solution will result in another optimal dual solution under the modifications of Lemma 5.3.

The next step will be to show that by using Lemma 5.3 it is possible to transform any dual solution of (5.9) into a corresponding σ-nested solution.

Lemma 5.4. *Given a solution (π, ρ, σ) of (5.9). Repeated application of Lemma 5.3 will result in a σ-nested dual solution within a finite number of steps.*

Proof. If (π, ρ, σ) is not already σ-nested, there exist two sets $\bar{I}_1^0, \bar{I}_2^0 \subseteq [k]$ such that $(\bar{I}_1^0 \not\subseteq \bar{I}_2^0) \wedge (\bar{I}_2^0 \not\subseteq \bar{I}_1^0) \wedge (\sigma_{\bar{I}_1^0}, \sigma_{\bar{I}_2^0} > 0)$. Let the dual solution resulting from application of Lemma 5.3 be denoted as (π, ρ, σ^1). By construction either $\sigma^1_{\bar{I}_1} = 0$ or $\sigma^1_{\bar{I}_2} = 0$. Given (π, ρ, σ^1) is still not σ-nested, the lemma can be applied repeatedly with (π, ρ, σ^i) being the solution in step i. The two sets used in step i will be denoted as $\bar{I}_1^i$ and $\bar{I}_2^i$.

Now assume that it is possible to create an infinite sequence $\sigma^1, \sigma^2, \ldots$ in this way. In this case there has to be $j > k$ such that $\sigma^j = \sigma^k$. Let $l' = \arg\max_{l \leq j} |\bar{I}_1^l \cup \bar{I}_2^l|$, i. e., $\bar{I}' = \bar{I}_1^{l'} \cup \bar{I}_2^{l'}$ is the subset for which some $\sigma^i_{\bar{I}'}$ was changed in the sequence up till j. This means that $\sigma^j_{\bar{I}'} > \sigma^k_{\bar{I}'}$ as the dual value in position $\bar{I}'$ was only increased so far but never decreased. Therefore the sequence cannot be infinite which concludes the lemma. □

Lemma 5.3 and 5.4 together ensure that each dual solution of the aMP can be turned into a σ-nested dual solution with the same objective value while preserving feasibility. The next step will be to show that with σ nested dual solutions it is possible to simplify the pricing problem and keep structural properties of the original pricing problem. Note that Lemma 5.4 does not claim that a σ-nested dual solution can be found efficiently. To arrive at a meaningful pricing procedure it will furthermore be necessary to show that one does not need to compute a σ-nested dual solution explicitly.

The modified pricing procedure

From the dual formulation of the aMP (5.9) it follows that in order to be LP optimal the dual values of the aMP must satisfy

$$\forall p \in P, (c'^{\intercal} - \pi^{\intercal} A')p - \sum_{\bar{I} \supseteq N(p)} \sigma_{\bar{I}} - \rho \leq 0. \tag{5.10}$$

While we assumed that it is possible to check all extreme points $p \in P$ implicitly via some unspecified algorithm, the σ terms can introduce additional complexity. Luckily it can be shown that one does not need to check all the possible combinations of blocks explicitly if the dual solution is σ-nested.

Lemma 5.5. *Given a σ-nested solution (π, ρ, σ) for (5.9). The solution (π, ρ, σ) is feasible if and only if*

$$\forall i \in [k], \forall p \in N(i), (c'^{\intercal} - \pi^{\intercal} A')p - \sum_{\substack{\bar{I} \subseteq [k] \\ i \in \bar{I}}} \sigma_{\bar{I}} - \rho \leq 0. \tag{5.11}$$

Furthermore if (5.11) is violated for (i, p), then (5.10) is violated for p.

Proof. The first step will be to show the "$\Rightarrow$" direction of the lemma. To do so it suffices to show that

$$\forall i \in [k], \forall p \in N(i), \sum_{\substack{\bar{I} \subseteq [k] \\ i \in \bar{I}}} \sigma_{\bar{I}} \geq \sum_{\bar{I} \supseteq N(p)} \sigma_{\bar{I}}.$$

As $p \in N(i) \Rightarrow i \in N(p)$ and therefore $(p \in N(i)) \wedge (\bar{I} \supseteq N(p)) \Rightarrow i \in \bar{I}$, every term on the right hand side will also appear on the left hand side, which shows that the inequality holds. Furthermore this asserts that a violation of (5.11) for some extreme point guarantees that (5.10) will be violated for the same extreme point.

In order to show the "$\Leftarrow$" direction, assume that all constraints (5.11) are fulfilled but an extreme point $\hat{p} \in P$ exists such that (5.10) is violated for $\hat{p}$. Let us define $g \in \mathbb{R}^k$ as

$$g_i = \sum_{\substack{\bar{I} \subseteq [k] \\ i \in \bar{I}}} \sigma_{\bar{I}} - \sum_{\bar{I} \supseteq N(\hat{p})} \sigma_{\bar{I}}.$$

In order for $\hat{p}$ to be violating (5.10), $g_i > 0$ for all $i \in N(\hat{p})$. It also follows that there has to be $\bar{I} \subseteq [k]$ such that $i \in \bar{I}$, $\bar{I} \not\supseteq N(\hat{p})$ and $\sigma_{\bar{I}} > 0$ for all $i \in N(\hat{p})$. Choose $\bar{I}_1$ as such a set of maximal size:

$$\bar{I}_1 = \arg\max_{\substack{\bar{I} \subseteq [k] \\ \bar{I} \not\supseteq N(\hat{p}) \\ \sigma_{\bar{I}} > 0}} |\bar{I}|.$$

Next choose any $\hat{i} \in N(\hat{p}) \setminus \bar{I}_1$. Such a $\hat{i}$ has to exist as $N(\hat{p}) \not\subseteq \bar{I}_1$. As $g_{\hat{i}} > 0$ there has to exist $\bar{I}_2$ with $N(\hat{p}) \not\subseteq \bar{I}_2$ and $\hat{i} \in \bar{I}_2$. Now $\bar{I}_2 \not\subseteq \bar{I}_1$ as $\hat{i} \in \bar{I}_2$ but $\hat{i} \notin \bar{I}_1$. As $|\bar{I}_1| \geq |\bar{I}_2|$ and $\bar{I}_1 \neq \bar{I}_2$ we also conclude $\bar{I}_1 \not\subseteq \bar{I}_2$. But as $\sigma_{\bar{I}_1}, \sigma_{\bar{I}_2} > 0$ this contradicts the assumption that (π, ρ, σ) was σ-nested. □

Using Lemma 5.5 one can see that for a σ-nested dual solution it suffices to perform pricing for each of the original blocks independently, where the σ terms add up to a constant that is fixed for each block and therefore do not change the subproblem's structure. As shown in Lemma 5.4 any aMP dual solution can be transformed to become σ-nested. The only drawback here is that the procedure in Lemma 5.4 might be very time consuming as we have not established a polynomial bound on the number of steps that need to be performed to arrive at a σ-nested dual solution. Therefore it is desirable to get along without computing a σ-nested solution explicitly. To do so the next lemma will show that contraints (5.11) remain unchanged under lemma 5.3.

Lemma 5.6. *Given a solution (π, ρ, σ) of (5.9) and let (π, ρ, σ') be the result of applying Lemma 5.3 to (π, ρ, σ). Then*

$$\forall i \in [k], \sum_{\substack{\bar{I} \subseteq [k] \\ i \in \bar{I}}} \sigma'_{\bar{I}} = \sum_{\substack{\bar{I} \subseteq [k] \\ i \in \bar{I}}} \sigma_{\bar{I}}.$$

Proof. To show this we need to verify that the procedure from Lemma 5.3 does not alter this sum in any way.

Given $\bar{I}_1$ and $\bar{I}_2$ as described in Lemma 5.3. Similar to the proof of Lemma 5.3 we need to check four different cases for the $i \in [k]$:

$i \in \bar{I}_1 \cap \bar{I}_2$: In this case also $\bar{I}_1$, $\bar{I}_2$ and $\bar{I}_1 \cup \bar{I}_2$ will be indices of the sum on both sides. Therefore the $\min\{\sigma_{\bar{I}_1}, \sigma_{\bar{I}_2}\}$ terms cancel each other out.

$i \in \bar{I}_1 \setminus \bar{I}_2$: Here only $\bar{I}_1$ and $\bar{I}_1 \cup \bar{I}_2$ are indices of the sum on both sides. Again the changing terms cancel out.

$i \in \bar{I}_2 \setminus \bar{I}_1$: This case is equivalent to the previous one.

$i \notin \bar{I}_1 \cup \bar{I}_2$: In this case no terms in the sum change at all.

Therefore none of the transformations done by Lemma 5.3 introduce changes to the relevant sums over the σ. □

With Lemma 5.6 we get that the contraints (5.11) do not differ between some non σ-nested dual solution and its σ-nested equivalent. Now all reuqired pieces are in place to show that it is sufficient for a valid pricing procedure to check constraints (5.11) for the dual solution of the aMP that is at hand without making transformations to it.

Theorem 5.7. *Given a dual solution (π, ρ, σ) for (5.9). If*

$$\forall i \in [k], \forall p \in N(i), (c'^{\intercal} - \pi^{\intercal} A')p - \sum_{\substack{\bar{I} \subseteq [k] \\ i \in \bar{I}}} \sigma_{\bar{I}} - \rho \leq 0 \tag{5.11}$$

then there exists a feasible dual solution for (5.9) with the same objective value as (π, ρ, σ). Otherwise if the constraints are violated for (i', p') then

$$(c'^{\intercal} - \pi^{\intercal} A')p' - \sum_{\bar{I} \supseteq N(p')} \sigma_{\bar{I}} - \rho > 0$$

.

Proof. Using Lemma 5.3 and 5.4 we know that there exists a σ-nested solution (π, ρ, σ') having the same objective value as (π, ρ, σ).

Assume that (5.11) are fulfilled for (π, ρ, σ). Then by Lemma 5.6 these constraints are also fulfilled for (π, ρ, σ'). Using Lemma 5.5 we can now conclude that (π, ρ, σ') is in fact a feasible dual solution for (5.9).

On the other hand assume that (5.11) is violated for (i', p'). Then by Lemma 5.6 we know that the same constraints will be violated for (π, ρ, σ'):

$$(c'^{\intercal} - \pi^{\intercal} A')p' - \sum_{\substack{\bar{I} \subseteq [k] \\ i' \in \bar{I}}} \sigma'_{\bar{I}} - \rho > 0.$$

Using Lemma 5.5 shows that (5.10) must then be violated by (π, ρ, σ') for p':

$$(c'^{\intercal} - \pi^{\intercal} A')p' - \sum_{\bar{I} \supseteq N(p')} \sigma'_{\bar{I}} - \rho > 0.$$

But if (π, ρ, σ') is infeasible so must (π, ρ, σ) by the construction in Lemma 5.3. This construction also guarantees that (5.10) will be violated for (π, ρ, σ) for the same extreme point p'. □

Given Theorem 5.7, we can use a pricing procedure that does not alter the structure of the subproblems. To do so we solve the original subproblem for each block $i \in [k]$:

$$\min\{(c'^{\intercal} - \pi^{\intercal} A')p : (D_i p \leq d^i) \wedge (x \in X')\}$$

and check whether for the resulting extreme point p holds

$$(c'^{\intercal} - \pi^{\intercal} A')p \leq \sum_{\substack{\bar{I} \subseteq [k] \\ i \in \bar{I}}} \sigma_{\bar{I}} + \rho.$$

If this constraint is violated, we have found a new variable to add to the restricted aMP. If for none of the blocks such a violated constraint is found, then Theorem 5.7 guarantees that there is some dual solution with the same objective value as the current one for which there is no violated dual constraint left. We can therefore stop the pricing routine for this node of the search tree. Note that it was not necessary to compute this solution explicitly, e. g., by repeated application of Lemma 5.3 but we can rely on the knowledge of its existence. Furthermore note that if we use a separation algorithm for the partial transversal polytope constraints in the master problem, all σ values for non separated constraints are zero. As long as the subset of separated constraints does not grow too large, which often is the case in practice, $\sum_{\substack{\bar{I} \subseteq [k] \\ i \in \bar{I}}} \sigma_{\bar{I}}$ can be easily calculated.

Part II

APPLICATION

6

TIMETABLING PROBLEMS

INTRODUCTION

Educational timetabling is a very common area of application for optimization methods. Such problems often arise in the administrative daily routine of many schools and universities. Wren (1996)[166] defines timetabling as "the allocation, subject to constraints, of given resources to objects being placed in space time, in such a way as to satisfy as nearly as possible a set of desirable objectives". The exact details can vary a lot depending on the use case and setting considered. Most timetabling problems have some kind of underlying graph coloring structure (which also renders almost all of them NP-hard[40,46,60,61,153]) in the sense that timeslots can be seen as colors that need to be assigned to courses and certain courses may not share the same color[60].

This chapter will consider certain timetabling problems as a particular application of the decomposition methods that were described in Chapter 4. Lach and Lübbecke (2008)[106,107] initially proposed such a decomposition for solving the university course timetabling problem. Matias Sørensen and the author of this thesis used and modified these methods to tackle a high school timetabling problem that arose in Danish high schools[149]. To do so we exploited the special structure of these problems to show that the size of the partial transversal polytope is small for the real world cases, and that the corresponding constraints can be enumerated. Furthermore we discovered a way to heuristically approximate the Benders' optimality cuts for matching subproblems without the need of a separation procedure.

Another modification to the methods by Lach and Lübbecke is to use hypergraph matchings in the Benders' subproblem in order to be able to solve certain timetabling problems with more complicated structures. Such problems arise at the timetabling problem that is currently solved at RWTH Aachen University within a project called "*carpe diem!*" which was conducted (among others) by G. Lach, M. Lübbecke and the author of this thesis. While certain methods are so far only theoretical considerations, they might also be applied within this project in the near future.

The timetabling problems explained in this chapter all deal with the assignment of courses to timeslots and rooms. Usually, the assignment to timeslots alone turns out to be NP-complete due to its relation to graph coloring. Such educational timetabling problems have been studied for a long time. Unsurprisingly, there exists a large body of research so far, which is surveyed in the next section. The methodologies by Lach and Lübbecke (2008)[106] and how they fit into the framework from Chapter 4 will be explained, alongside the modifications by Sørensen and Dahms (2014)[149] and the extensions for using hypergraph matchings in the subproblem. The last section will show the results of various computational experiments for the different methods.

LITERATURE REVIEW

A huge body of literature has been published so far about educational timetabling problems. Early introductions were published by Schmidt and Ströhlein (1980)[145] and de Werra (1985)[62]. A general survey about timetabling is given by Schaerf (1999)[143]. Surveys with a focus on university timetabling can be found by Carter and Laporte (1998)[46], Burke and Petrovic (2002)[43], Petrovic and Burke (2004)[131], and Lewis (2008)[111]. Note that there exist many different definitions and requirements for the problem formulations, so the results from different studies often cannot be compared directly. A part of this chapter will follow the definitions proposed by Bonutti et al. (2012)[38].

High school timetabling problems often vary dramatically between countries due to different legislations. Therefore many country specific studies have been conducted. Al-Yakoob and Sherali (2015)[7] give a list of several such country specific publications (many of them PhD theses). Sorted by country, there are for example:

AUSTRALIA: Merlot (2005)[119]

THE NETHERLANDS: de Haan et al. (2007)[59], Hartog (2007)[88], and Willemen (2002)[165]

DENMARK: Sørensen and Dahms (2014)[149] and Sørensen and Stidsen (2013)[150]

FINNLAND: Marte (2002)[115]

GERMANY: Jacobsen et al. (2006)[96] and Junginger (1986)[98]

GREECE: Beligiannis et al. (2008)[21], Beligiannis et al. (2009)[22], Birbas et al. (1997)[29], and Papoutsis et al. (2003)[130]

HONG KONG: Kwok et al. (1997)[105]

ITALY: Alvarez-Valdés et al. (2002)[11] and Avella et al. (2007)[15]

SOUTH AFRICA: Raghavjee and Pillay (2010)[137]

SPAIN: Alvarez-Valdés et al. (1995)[10], Alvarez-Valdés et al. (1996)[9], and Alvarez-Valdés et al. (2002)[12]

An archive of high school timetabling instances from eight different countries is presented by Post et al. (2014)[133].

The majority of the available studies focuses on the application of various (meta) heuristics to generate good feasible solutions. Sorted by type of heuristic there are for example:

ANT COLONY OPTIMIZATION: Socha et al. (2003)[148]

GENETIC ALGORITHM: Beligiannis et al. (2008)[21], Beligiannis et al. (2009)[22], Corne et al. (1994)[51], Fernández and Santos (2003)[70], Raghavjee and Pillay (2010)[137], and Ueda et al. (2004)[156]

LOCAL SEARCH: Avella et al. (2007)[15] and Schaerf (1999)[142]

NEURAL NETWORKS: Carrasco and Pato (2004)[44], Carrasco and Pato (2004)[45], and Gislén et al. (1989)[81]

SIMULATED ANNEALING: Abramson (1991)[3], Elmohamed et al. (1998)[68], Thompson and Dowsland (1996)[154], and Zhang et al. (2010)[167]

TABU SEARCH: Abdullah et al. (2007)[1], Aladag et al. (2009)[8], Alvarez-Valdés et al. (2002)[11], Alvarez-Valdés et al. (2002)[12], Hertz (1991)[91], Hertz (1992)[92], Jacobsen et al. (2006)[96], and Santos et al. (2005)[140]

Colorni et al. (1998)[49], Lewis (2008)[111], and Rossi-Doria et al. (2003)[138] provide comparisions of several metaheuristics.

As a non heuristic approach, several constraint logic programming approaches were studied by Azevedo and Barahona (1994)[16], Frangouli et al. (1995)[72], Guéret et al. (1996)[85], Henz and Würtz (1996)[90], Legierski (2003)[110], and Valouxis and Housos (2003)[158]

Integer programming formulations for timetabling have already been studied in the earliest days of integer programming by Gotlieb (1962)[83] and Lawrie (1969)[108]. Other MILP models for timetabling can be found by Al-Yakoob and Sherali (2006)[6], Birbas et al. (1997)[29], Burke et al. (2007)[41], Burke et al. (2008)[42], Daskalaki and Birbas (2005)[58], Harwood and Lawless (1975)[89], and Tripathy (1984)[155].

As already pointed out in Chapter 3 many hard problems can be efficiently solved using some kind of decomposition method. This chapter will focus on using Benders' decomposition, especially with matching as a subproblem. Solving university course timetabling problems using a similar approach was first described by Lach and Lübbecke (2008)[106] and Lach and Lübbecke (2008)[107]. Another application of this methodology was used for Danish high schools by Sørensen and Dahms (2014)[149]. The approach was applied for a broad spectrum of real life instances with up to 516 classes.

Chapter 3 also described the dual of the Benders' decomposition, the Dantzig-Wolfe decomposition, especially in conjunction with column generation (CG) as a method for solving these reformulations. Some studies exist using column generation for tackling timetabling problems. Papoutsis et al. (2003)[130] solve some very small (at most 9 classes, 21 teachers) Greek high school timetabling problems with CG. Santos et al. (2012)[141] generate lower bounds for slightly larger instances (20 classes, 33 teachers) by CG. A more recent use of CG in high school timetabling with a focus on Kuwait was studied by Al-Yakoob and Sherali (2015)[7] who handled instances with up to 34 classes and 65 teachers but also just provided bounds on the solution quality. It should be noted that CG approaches for high school timetabling so far were used on much smaller scale than the Benders' approach used by Sørensen and Dahms (2014)[149]. Even though this is not a thorough comparison (especially as the different countries enforce different requirements on the problem formulation) it indicates that high school timetabling is more suited for Benders' algorithms. CG for a university timetabling problem was studied by Qualizza and Serafini (2005)[136] but also only considered relatively small instances with up to 63 courses.

Another related problem is exam timetabling which is not studied in this thesis. A survey covering recent developments in this field is given by Qu et al. (2009)[135].

PROBLEM DEFINITION

Basic components

The concepts of this chapter are designed for creating weekly timetables, as they often appear in university and high school settings. Almost all of these timetabling problems have the following components in common:

- A set of lectures L. A lecture is a single unit, taught at most once per week over a consecutive time period, not changing rooms in between.
- A set of timeslots T. A timeslot represents the smallest, common period of time within which a lecture can be hold. Each timeslot appears once per week and all timeslots have the same duration. One objective of timetabling is to assign lectures to timeslots. It can be relevant whether two timeslots appear on the same day, therefore the timeslots can also be grouped to days $\bar{T} \subseteq T$ with D being the set of all days. The day of a timeslot will be denoted as $D(t)$.
- A set of rooms R. Each lecture shall be assigned to one or (in some cases) more rooms.
- A bipartite graph of feasible timeslots $G_{\text{time}} = (L \cup T, E_{\text{time}})$. Graph G_{time} defines which lectures may be assigned to which timeslots. This can be a hypergraph in which case a lecture may occupy multiple timeslots at the same time (e. g., if it spans more time than the duration of a single timeslot).
- A bipartite graph of feasible rooms $G_{\text{room}} = (L \cup R, E_{\text{room}})$. Graph G_{room} defines which lectures may be assigned to which rooms. This can also be a hypergraph, in which case a lecture may be hold in two rooms simultaneously.
- A conflict graph $G_{\text{conf}} = (L, E_{\text{conf}})$. Two neighboring lectures in G_{conf} may not share a timeslot.

These timetabling core components are often sufficient to describe the requirements of a feasible solution for many applications. A feasible solution of the timetabling problem are sets $M_{\text{time}} \subseteq E_{\text{time}}$ and $M_{\text{room}} \subseteq E_{\text{room}}$ such that each lecture is covered by one element of each set, neighboring lectures in G_{conf} do not share a timeslot and no two lectures sharing a timeslot are assigned the same room.

More formally, the requirement of covering each lecture once can be expressed as $\forall l \in L, (|\{e \in M_{\text{time}} : l \in e\}| = 1) \wedge (|\{e \in M_{\text{room}} : l \in e\}| = 1)$. Now we can use the shorthand notation $M_{\text{time}}(l)$ for a set containing the timeslot assigned to l and $M_{\text{room}}(l)$ for a set containing the room assigned to l. Then a feasible timetabling solution also needs to satisfy

$$\forall \{l_1, l_2\} \in E_{\text{conf}}, M_{\text{time}}(l_1) \cap M_{\text{time}}(l_2) = \emptyset$$

and with $I(M_{\text{time}}) = \{\{l_1, l_2\} : M_{\text{time}}(l_1) \cap M_{\text{time}}(l_2) \neq \emptyset\}$ being the set of intersecting lectures in M_{time} there also has to hold

$$\forall \{l_1, l_2\} \in I(M_{\text{time}}), M_{\text{room}}(l_1) \cap M_{\text{room}}(l_2) = \emptyset.$$

Sometimes it is not clear that such feasible solutions do exist, in which case one might want to maximize the size of M_{time} and M_{room} such that each lecture is covered at most by an edge in each of the two sets. In order to ensure that there are no infeasible instances, we will use the maximization formulation if not stated otherwise and penalize unassigned lectures with a default value of 100.

This thesis will not deal with multiple objectives, Pareto optimality and other issues stemming from having multiple, potentially conflicting criteria in an optimization problem. Therefore all soft requirements will need to be translated into a single currency that will be optimized in the objective.

Even in this most basic form the timetabling problem is already NP-complete, as the conflict graph G_{conf} can easily encode a graph coloring problem.

Theorem 6.1. *The basic timetabling problem is* NP*-complete*

Proof. The problem lies within NP as is easy to check whether two sets M_{time} and M_{room} are a feasible solution in polynomial time.

To show NP-hardness one can do a simple reduction from graph coloring. In graph coloring we are given some graph $G = (V, E)$ and a total of k colors. Then answering the question whether it is possible to assign each vertex a color such that neighboring vertices have different colors is NP-complete[100].

Given a graph coloring instance $G = (V, E)$ and k, a basic timetabling instance $L = V$, $T = [k]$, $R = V$ is constructed using the graphs

$$\begin{aligned} G_{\text{time}} &= (L \cup T, \{\{l, t\} : l \in L, t \in T\}) \\ G_{\text{room}} &= (L \cup R, \{\{v, v\} : v \in V\}) \\ G_{\text{conf}} &= (L, E). \end{aligned}$$

This way a lecture is created for each vertex and a timeslot for each color. Each lecture will be allowed to be scheduled in each timeslot. Furthermore every lecture gets its own room that can only be used by that specific lecture. This way the room assignment will always be trivial. The conflict graph will be exactly the coloring graph.

Now a solution of the coloring problem will directly translate into an assignment of lectures to timeslots and vice versa, which concludes the reduction. □

Restrictions on the structure of the conflict graph might render the problem easy but this usually is not the case. Instead often more requirements are needed for a specific application which even add complexity to the problem. All the problem instances presented here are NP-hard by the reduction from Theorem 6.1, as they do not impose sufficient restrictions on the structure of the conflict graph.

Depending on the problem, the edges in G_{time} and G_{room} may be weighted, modeling preferences for the time and room assignment. Here the weight of the chosen edges will be counted in the objective function.

The matching of lectures to rooms within the graph G_{room} will be used as a subproblem in a Benders' decomposition for the following applications. Therefore a major focus will be placed upon these matchings. Thus, if not specified otherwise, the graph G_{room} will be the default choice, e. g., for the neighborhood of a vertex (i. e., $N(l) = N_{G_{\text{room}}}(l)$).

For integer programming models, often certain constraints turn out to be stronger than others. For example coloring formulations can benefit greatly from using constraints based on maximal cliques in the graph, instead of constraints based on

edges, as the clique based constraints form some of the facets of the graph coloring polytope[48,129]. In the timetabling problem the conflict graph is usually defined by the constraints stemming from student curricula or teachers. These define cliques of lectures which may not intersect in the timetable and these cliques may very well be already maximal. Therefore it is a good idea to use this given information in the problem formulation. Such cliques can be modeled by introducing hyper edges into the conflict graph G_{conf}. In the following it will be assumed that such hyper edges are always allowed.

Lectio high school instances

Sørensen and Stidsen (2013)[150] and Sørensen and Dahms (2014)[149] use a dataset of timetabling problems taken from a wide range of high schools in Denmark. The problem formulation used in these papers reflect adjustments relevant for these specific instances. This problem formulation is also used in the commercial high school timetabling system Lectio. The used datasets stem from the users of the Lectio system.

Many additional soft constraints are incorporated in the Lectio model. Many requirements are based upon the preferences of students and teachers. The set A contains all these entities, where a student $a \in A$ may represent multiple real but equivalent students. Some of the requirements will only apply to students or teachers, represented as A^s and A^t. The lectures of entity $a \in A$ will be represented by $L(a)$

IDLE-TIMESLOTS: For students as well as teachers the number of idle timeslots between lectures on the same day shall be minimized. Each of these idle timeslots is penalized by an amount β_a that varies by the entity a (student, teacher).

DAYS-OFF: It is often required that teachers have at least one day off. Furthermore it is preferable for a teacher to have as many days off as possible. Therefore it is a hard requirement to have at least a certain number F_a of days off (depending on the teacher), but also each day off will be awarded an amount of γ_a in the objective. Conversely it is not preferred for students to have days off, so for students the γ_a will be negative.

DAY-CONFLICTS: The lectures are grouped to classes and from each class at most one lecture shall be scheduled each day, which is a hard constraint.

NEIGHBOR-DAYS-FOR-CLASSES: In order to allow more time for completion of homework, it is preferred to have lectures of the same class being scheduled over non-consecutive days (in addition for them not being on the same day). For each class $\bar{L}$ there is a specified number $N_{\bar{L}}$ of allowed neighboring days. Furthermore each of the occurring neighbor days shall be penalized by an amount of ζ.

TEACHER-DAILY-WORKLOAD: Each teacher shall have a maximum of p_a lectures scheduled for a certain day, which is to be modeled as a hard constraint. Also a teacher should have more than one lecture on each day (if it is not a day off). Days with exactly one lecture for a teacher are therefore penalized by an amount of η_a.

ROOM-STABILITY: For each class it is desirable to have all of its classes taking place in the same room. Each additional room used by a class is penalized by an amount of ϵ.

Most of the times only a schedule for one week is required which then repeats itself every week. In some cases it is desired to plan a total of two weeks (leading to twice the amount of timeslots) in order to allow for additional flexibility (like allowing an average of 1.5 days off for a certain teacher each week). In these cases the following two additional requirements are to be incorporated:

DAYS-OFF-STABILITY: The required days off shall be distributed evenly between weeks, i. e., when having 3 days off in total over the two weeks, each week shall contain at least one of these. This is to be modeled as a hard constraint.

CLASS-STABILLITY: Equivalently to the previous requirement, the lectures belonging to the same class shall be evenly distributed between the weeks. Each lecture that is out of balance shall be penalized by an amount of ι (e. g., if there shall be 7 classes in two weeks and they are distributed 2 to 5, there are 2 lectures out of balance).

Udine instances

There exists a couple of timetabling instances published by the University of Udines research group on scheduling and timetabling. These instances can be found at

http://tabu.diegm.uniud.it/ctt/index.php?page=instances

For the experiments of this chapter the following instance sets are used (and referred to as "Udine instances")

- ITC-2007: instances used for the international timetabling competition 2007.
- Erlangen: instances from the University of Erlangen (Germany)

All these instances exist in a unified XML format. Bonutti et al. (2012)[38] describe several modeling variants for timetabling problems based upon these data sets, including exact benchmarking instructions for evaluating solutions. As these models are not fully compatible with the requirements we had at RWTH Aachen, the solver methods developed in this thesis do not fit all parts of these models. Therefore not all components suggested by Bonutti et al. (2012)[38] are being implemented.

The Udine instances specify the requirements of the basic timetabling problem in the following way:

- The instances do not specify preferences of the lectures for timeslots. Therefore the edges of G_{time} will not have weights by default. Also the goal will be to maximize the number of scheduled lectures. Each unscheduled lecture will be penalized by an amount of 100.
- Each lecture shall be scheduled in a room with sufficient seats for its students. Each missing seat shall be penalized in the objective by an amount of 1. These penalties are modeled as weights on the edges of G_{room} with suitable rooms having weight 0 and smaller rooms a weight corresponding to the number of missing seats. Alternatively a variant will be tried where this component

will be modeled as a hard constraint, i. e., with the edges of weight greater than 0 removed.

- Each lecture may specify timeslots and rooms which are forbidden, i. e., the corresponding edges may not be present in G_{time} and G_{room}.
- Conflicts between lectures are defined by a set of curricula and by lectures sharing the same teacher.

Additionally the following requirements are modeled as soft constraints, affecting the optimization objective. The used weights will be the same for all instances:

MIN-WORKING-DAYS: Lectures are grouped as courses. The lectures of the same course shall be distributed over at least a certain number of days. Every day less than this minimum shall be penalized by an amount of 1.

STUDENT-MIN-MAX-LOAD: For each curriculum there is a minimum and maximum number of lectures from the curriculum that shall be scheduled on every day. Each lecture above or below these bounds shall be penalized by an amount of 1.

ISOLATED-LECTURES: For each curriculum isolated lectures (scheduled to a timeslot without another lecture of that curriculum scheduled to a neighboring timeslot) are penalized by an amount of 1.

Additional constraints, which are not included here, are those which affect the assignment of lectures to rooms. To the author's current state of knowledge these are not suitable for the Benders' algorithm developed here and would prevent proper comparisons.

RWTH Aachen instances

The timetabling problem that we encountered during the work on the "*carpe diem!*" project for establishing automated timetabling at RWTH Aachen university exhibit two special properties. On the one hand some rooms can be joined to form a single larger lecture hall. On the other hand it is very common that lectures span multiple timeslots, e. g., some tutorials have a duration of only one hour (thereby forcing a timeslot to be of a length of at most one hour) but most lectures have a length of at least two hours (sometimes even more) and it is unacceptable for such a lecture to be split up or to change lecture rooms in between timeslots. To cope with these requirements while maintaining a Benders' type decomposition as used for simpler timetabling problems, the combinatorial Benders' decomposition for subproblems with hypergraph structure was developed (see Section 4.2). The graphs G_{time} and G_{room} are therefore hypergraphs for these instances.

Two instances from the data set of the "*carpe diem!*" project are evaluated in this chapter – one for the scheduling of the summer term 2015 and one for the winter term 2015/2016. Anonymized versions of these data sets are available alongside the solver code in the following GitLab repository:

`https://gitlab.com/florian.dahms/TimetablingSolver`

These two instances are relatively large. Both instances work with 60 timeslots and 457 rooms. The summer term instance has 3708 lectures with 15307 conflicting

cliques of lectures, while the winter term instance has 3893 lectures with 21328 conflicting cliques of lectures. Note that the conflicts are stated in the form of cliques resulting from different lecturers and student groups for which the corresponding lectures shall not share a timeslot. In the notation this can be handled by allowing hyper edges in the graph G_{conf}. In the instances used here, all conflicts are treated equally as hard constraints in order to focus on the core concepts presented in this thesis. In a proper application many of them should be treated as weak constraints, while on the other hand additional requirements are necessary for applicable results.

No matter how one choses to treat the conflicts between lectures, the RWTH instances are on the one hand rather large (much larger than all the others considered here) and on the other hand there are only two of them. In order to test the algorithms more thoroughly, an instance generator was implemented to create additional instances with the appropriate hypergraph structure. The generated instances are smaller than the RWTH instances to allow for a shorter time limit (the solvers presented here need to be run over several days on the RWTH instances to reach meaningful results).

For the artificial instances four instance sets with respectively 80, 100, 200, and 500 lectures were created, each instance set containing 50 different instances, leading to a total of 200 different instances. The instances are furthermore described by the following properties (the last one will be explained later):

Number of lectures:	80	100	200	500
Number of rooms:	11	13	26	65
Number of conflict cliques:	80	100	200	500
$P(\text{“Lecture in conflict clique”})$	0.1875	0.15	0.075	0.03

For all instances the planning week is divided into 5 days with 12 timeslots each, leading to a total of 60 timeslots.

Each room is assigned a size which is equal to its index (i. e., for an instance with 11 rooms, each room will have an unique size between 1 and 11). In each instance there are two rooms which can be combined into one larger room. These two rooms are chosen uniformly at random from all rooms. The size of the combined rooms is the sum of the sizes of the two rooms it is made of.

Each lecture has a length of at least one and up to four timeslots. It is assigned a length of one timeslot with a probability of roughly 0.1786, a length of two timeslots with a probability of roughly 0.7143, a length of three timeslots with a probability of roughly 0.0714 and a length of four timeslots with a probability of roughly 0.0357. Now the lecture can be matched into the required number of timeslots, as long as these are on the same day and consecutive. Of these potentially possible hyper edges for the G_{time} graph, each is inserted into the graph with probability 0.5 (this procedure being repeated for the lecture, if no edge is chosen). Each of these edges is assigned a weight that is chosen uniformly at random from $\{0, 1, 2\}$.

Each lecture is furthermore assigned a size which is drawn uniformly at random from all available room sizes (without the virtual room resulting from the combination of two rooms). Each lecture can now be matched into all rooms with size at least the size of the lecture (including the combined room). Of these potentially possible edges (with a hyper edge for the combined room) for the G_{room}

graph, each is inserted into the graph with probability 0.5 (again the procedure is repeated for the lecture if no edge is chosen).

For the conflict graph for each conflicting clique of lectures, a lecture is inserted into the clique with a probability $P(\text{“Lecture in conflict clique”})$ which is different for each instance set and specified above. The probabilities are chosen such that the average clique size is 15.

OPTIMIZATION ALGORITHMS

The optimization algorithms considered here are based upon integer programming techniques. It is important to note that the formulation of the problem can have a major impact on the performance of the solver which, due to the complexity of today's solver technology, may not be easy to attribute to the factors under consideration. Even factors that seem unobtrusive at first, like, e. g., the column ordering, can massively alter the outcome of the solver. This effect is often denoted as performance variability[112]. While this should not prevent us from performing comparisons between model runtimes, these effects should always be kept in mind when comparing different ILP models.

Three indexed formulation

As a benchmark for the Benders' methods a canonical ILP formulation will be used. Usually it is stated with variables indexed by lecture, timeslot and room, indicating whether that lecture shall be matched into the specified timeslot and room. Due to these variables this formulation is often denoted as “three indexed formulation”. In order to generalize better to the hypergraph setting and to better conform with the notation of this thesis, here variables $x_{\{l,t\},\{l,r\}}$ with $\{l,t\} \in E_{\text{time}}$ and $\{l,r\} \in E_{\text{room}}$ will be used. Though these variables are only two indexed, this does neither change the number of variables nor the structure of the model in any way. Therefore this thesis will stick to calling the model “three indexed”.

For the basic model let $w_{\{l,t\}}$ for $\{l,t\} \in E_{\text{time}}$ be the weight of the time assignment edges and $w_{\{l,r\}}$ for $\{l,r\} \in E_{\text{room}}$ the weight for the room assignment. Note that the penalty for assigning too small rooms can be directly encoded in the $w_{\{l,r\}}$ weights.

Now the basic ILP formulation is:

$$\min \sum_{\substack{\{l,t\}\in E_{\text{time}} \\ \{l,r\}\in E_{\text{room}}}} (w_{(\{l,t\}} + w_{\{l,r\}} - 100)x_{\{l,t\},\{l,r\}} \tag{6.1}$$

$$\text{s.t.} \sum_{\substack{\{l',t\}\in E_{\text{time}} \\ \{l',r\}\in E_{\text{room}}}} x_{\{l',t\},\{l',r\}} \leq 1 \qquad \forall l' \in L \tag{6.2}$$

$$\sum_{\substack{\{l,t'\}\in E_{\text{time}} \\ \{l,r'\}\in E_{\text{room}}}} x_{\{l,t'\},\{l,r'\}} \leq 1 \qquad \forall t' \in T, r' \in R \tag{6.3}$$

$$\sum_{l \in e_{\text{conf}}} \sum_{\substack{\{l',t'\} \in E_{\text{time}} \\ \{l',r\} \in E_{\text{room}}}} x_{\{l',t'\},\{l',r\}} \leq 1 \quad \forall e_{\text{conf}} \in E_{\text{conf}}, t' \in T \tag{6.4}$$

$$x_{\{l,t\},\{l,r\}} \in \{0,1\} \quad \forall \substack{\{l,t\} \in E_{\text{time}} \\ \{l,r\} \in E_{\text{room}}}.$$

In the objective (6.1) each assigned lecture is awarded an amount of 100 (as stated in the requirements for the Udine instances) which is mathematically equivalent to penalizing a non assigned lecture with that respective amount. Constraint (6.2) ensures that each lecture is assigned at most once. Constraint (6.3) guarantees that no room is assigned to multiple lectures in any given timeslot. Finally, constraints (6.4) allow only one lecture of any conflict clique from E_{conf} at any given timeslot.

Basic Benders' formulation

As the assignment of lectures to rooms is a matching problem, the timetabling problem contains the necessary structure for the Benders' technique described in Chapter 4. The master problem will be very similar to the three indexed formulation but without the information which lecture will be assigned to which room. This reduces the number of involved variables by roughly an order of magnitude in most problem instances. Overall we arrive at the following reduced master problem for the basic timetabling problem (excluding the Benders' Cuts):

$$\begin{aligned}
\min \quad & \sum_{\{l,t\} \in E_{\text{time}}} (w_{\{l,t\}} - 100) x_{\{l,t\}} + \alpha \\
\text{s.t.} \quad & \sum_{\{l',t\} \in E_{\text{time}}} x_{\{l',t\}} && \leq 1 && \forall l' \in L \\
& \sum_{l' \in e_{\text{conf}}} \sum_{\{l',t'\} \in E_{\text{time}}} x_{\{l',t'\}} && \leq 1 && \forall e_{\text{conf}} \in E_{\text{conf}}, t' \in T \\
& x_{\{l,t\}} \in \{0,1\} && && \forall \{l,t\} \in E_{\text{time}} \\
& \alpha \in \mathbb{R}_+.
\end{aligned}$$

In order to ensure feasibility and optimality in the room assignment, several variations of the classical Benders' method can be applied to this problem as explained in Chapter 4. The experiments will feature comparisons between several of these.

The subproblem will need access to the result of the master problem. In these experiments the separation algorithm will only be run on integral solutions, therefore we do not need to consider fractional master solutions. The current master solution will be represented as the assignment M_{time} in the subproblem. Note that not every lecture needs to be present in this assignment, as not assigning a lecture is only penalized in the master problem.

In the unmodified version, the Benders' cuts are derived from the dual values of the problem

$$\begin{aligned} \min \quad & \sum_{\{l,r\}\in E_{\text{room}}} w_{\{l,r\}} y_{\{l,r\}} \\ \text{s.t.} \quad & \sum_{\{l,r\}\in E_{\text{room}}} y_{\{l,r\}} = 1 \quad \forall \{l,t\} \in M_{\text{time}} \\ & \sum_{\substack{\{l,r\}\in E_{\text{room}} \\ \{l,t\}\in M_{\text{time}}}} y_{\{l,r\}} \leq 1 \quad \forall r \in R, t \in T \\ & y \in \mathbb{R}_+^{|E_{\text{room}}|}. \end{aligned} \tag{6.5}$$

If (6.5) is infeasible, a Farkas vector $\pi \in \mathbb{R}^{|M_{\text{time}}|} \times \mathbb{R}_-^{|R|\cdot|T|}$ can be derived, proving the problem's infeasibility. Using this we can add the feasibility cut

$$\sum_{\{l,t\}\in M_{\text{time}}} \pi_{\{l,t\}} x_{\{l,t\}} \leq - \sum_{r\in R, t\in T} \pi_{r,t}$$

to the master problem and run the next iteration.

If the subproblem is feasible but its objective value is greater than the α value in the current master solution, instead of the Farkas vector, the dual solution of (6.5), denoted again as $\pi \in \mathbb{R}^{|L|} \times \mathbb{R}_-^{|R|\cdot|T|}$ is used to derive the optimality cut

$$\sum_{\{l,t\}\in M_{\text{time}}} \pi_{\{l,t\}} x_{\{l,t\}} - \alpha \leq - \sum_{r\in R, t\in T} \pi_{r,t}.$$

Note that the subproblem can be divided into a bipartite matching problem for each timeslot. As explained in Chapter 4, the matching structure allows us to improve the basic Benders' procedure in certain ways. For example the basic optimality cuts from the dual values tend to have slow convergence and lead to numerically unstable problems. Instead the single α value can be replaced by an $\alpha \in \mathbb{R}_+^{|L|}$ vector with a component for each lecture, that will then hold the subproblem cost for this lecture only. Now given a subproblem solution M_{room} we can use RISS to determine a group $\mathcal{S}$ of smaller subsets $S \subseteq M_{\text{time}}$, which do not affect each other in the matching. For these the following, more efficient, optimality cuts can be generated, given the subproblem's dual values $\pi \in \mathbb{R}^{|M_{\text{time}}|} \times \mathbb{R}_-^{|R|\cdot|T|}$:

$$\sum_{\{l,t\}\in S} (\pi_{\{l,t\}} x_{\{l,t\}} - \alpha_l) \leq - \sum_{\substack{\{l,t\}\in S \\ \{l,r\}\in M_{\text{room}}}} \pi_{r,t} \qquad \forall S \in \mathcal{S}.$$

Similarly, instead of the cuts from the Farkas vector, the separation routine for the partial transversal polytope can be used. Here it is implemented by solving the maximum matching problem

$$\begin{aligned} \max \quad & \sum_{\{l,r\}\in E_{\text{room}}} y_{\{l,r\}} \\ \text{s.t.} \quad & \sum_{\{l',r\}\in E_{\text{room}}} y_{\{l,r\}} \leq 1 \quad \forall \{l',t'\} \in M_{\text{time}} \\ & \sum_{\substack{\{l,r'\}\in E_{\text{room}} \\ \{l,t'\}\in M_{\text{time}}}} y_{\{l,r'\}} \leq 1 \quad \forall r' \in R, t' \in T \\ & y \in \mathbb{R}_+^{|E_{\text{room}}|}. \end{aligned} \tag{6.6}$$

If the solution of (6.6), denoted as M_{room}, is not covering all lectures assigned in the master problem, the RISS algorithm can be used to find a group of small infeasible subsystems $\mathcal{S} \subseteq 2^{M_{\text{time}}}$ such that each lecture without an assigned room is present in one of the subsystems. Now the following cuts can be added instead of the aforementioned feasibility cuts:

$$\sum_{\{l,t\} \in S} x_{l,t} \leq |\{\{l,r\} \in M_{\text{room}} : \exists \{l,t\} \in S\}| \qquad \forall S \in \mathcal{S}.$$

The modified feasibility cut method does not help to separate suboptimal solutions, so the other subproblem formulation (6.5) needs also to be solved to know whether to add optimality cuts. It turns out that the influence of the subproblem solver time is negligible, so this additional effort does not have relevant negative impact.

Modifications for Lectio high school instances

The algorithms used to solve these instances from high schools in Denmark are also described in detail by Sørensen and Dahms (2014)[149]. The methods used for the experiments on this data set are more tailored to the specific instances and therefore differ in many aspects from the more general methods used otherwise in this chapter. Note that the definitions and constraints described here slightly differ from the presentation by Sørensen and Dahms (2014)[149] in order to be more consistent with the rest of this thesis. Nevertheless the methods are mathematically equivalent.

A major modification to the Benders' algorithm used otherwise in this chapter is that one can manage to enumerate the facets of the partial transversal polytope for the room matching component in all of the involved instances. This makes it possible to get along without a cutting plane algorithm, which greatly improves the solver performance. A similar approach is taken by Lach and Lübbecke (2008)[107]. To achieve this, it is necessary to exploit the special structure present in the graph G_{room} for the high school instances at hand.

First note that lectures do not span multiple timeslots. Therefore each timeslot can be considered completely independently from the others when it comes to the room assignment. In the following $\{l,t\}$ will therefore always refer to the timeslot t that is currently being considered.

In the high school instances, there may be some events which can only fit into one particular room (these lectures are denoted as singleton lectures – with L^1 being the set containing all of these). Additionally some of the rooms have certain features (like being a room suited for chemistry classes) and some lectures may only fit into rooms exposing certain features. In this way the rooms can be grouped by their features into sets R^i, where all rooms in the same group are completely interchangeable, i. e., if the room is suitable for a certain lecture (not in L^1), all rooms from the same group will also be. Note that each room belongs to exactly one room group. Let RG be the set of all these room groups. It turns out that in the instances at hand there are always only a few of these groups (at most 12).

Using the theory set up in Section 2.4 the next step is to show that for a complete description of the partial transversal polytope only relatively few constraints are needed:

Corollary 6.2. *For the Lectio instances, the room matching polytope for a given timeslot t is fully described by the constraints $\sum_{l \in N^{-1}(\bar{R})} x_{\{l,t\}} \leq |\bar{R}|$ for the following sets $\bar{R} \subseteq R$:*

- $N(l) \qquad \forall l \in L^1$
- $\bigcup_{\bar{R}' \in \bar{RG}} \bar{R}' \qquad \forall \bar{RG} \subseteq RG$

Proof. Using the results from Lemma 2.14 and Corollary 2.15 it remains to be shown that $\bar{L} \neq N^{-1}(\bar{R})$, for all of the specified $\bar{R}$, cannot be ν-flat and ν-inseparable.

Assuming $\bar{L}$ is ν-flat and ν-inseparable. Then $\nu(\bar{L}) = |N(\bar{L})|$ by Lemma 2.13. Let $l \in \bar{L} \setminus L^1$. Given furthermore some $l' \notin \bar{L}$ but with $l' \in N^{-1}(N(l))$. Then $N(\bar{L}) = N(\bar{L} \cup \{l'\})$ contradicting the ν-flatness of $\bar{L}$. Therefore for $\bar{R} = N(\bar{L} \setminus L^1)$ we have $N^{-1}(\bar{R}) \subseteq \bar{L}$. Note that $\bar{R}$ must be the union of some set of room groups $\bar{RG} \subseteq RG$. Therefore let $L^{-1} \setminus N^{-1}(\bar{R}) = \bar{L^1} \neq \emptyset$. As all of the lectures in $\bar{L^1}$ only fit into their respective room in $N(\bar{L^1})$ it has to hold that $\nu(\bar{L^1}) = |N(\bar{L^1})|$. And as these rooms are disjoint from $\bar{R}$, we also get that $\nu(\bar{L} \setminus \bar{L^1}) = |N(\bar{L})| - |N(\bar{L^1})|$ which contradicts the ν-inseparability of $\bar{L}$. □

As the size of room groups RG is very limited, these constraints can easily be added a priori to the model (for 12 room groups there would be 4095 constraints per timeslot plus those for the singleton events).

In order to also circumvent the need for optimality cuts, a way to add a lower bound on the cost of the room matching problem with a relatively small set of constraints was developed by Sørensen and Dahms (2014)[149]. To establish this bound, we rely on the concept of graph deficiency.

Definition 6.3. For a bipartite graph $G = (L \cup R, E)$ the deficiency of a vertex set $\bar{L} \subseteq L$ is defined as

$$\mathrm{def}(\bar{L}) = |\bar{L}| - |N(\bar{L})|.$$

The deficiency of the entire graph is then defined as

$$\mathrm{def}(G) = \max\{\mathrm{def}(\bar{L}) : \bar{L} \subseteq L\}.$$

The deficiency is closely tied to the concept of maximum matchings and the König-Hall Theorem (see Theorem 2.6). For example the deficiency of the graph will be the number of vertices which cannot be matched:

Theorem 6.4. *For G we have $\nu(G) = |L| - \mathrm{def}(G)$*

Proof. See Lovász and Plummer (2009)[113]. □

Given the definition of the deficiency this makes it easy to determine the number of unmatched events within a linear program, already containing the corresponding constraints for the partial transversal polytope. Given a variable def_t, the constraints

$$\sum_{\substack{l \in N^{-1}(\bar{R}) \\ \{l,t\} \in E_{\text{time}}}} x_{\{l,t\}} - \mathrm{def}_t \leq |\bar{R}| \qquad \forall \bar{R} \subseteq R$$

can be used instead of the regular partial transversal polytope constraints in order to make sure that def_t will contain the number of unmatchable lectures in the considered timeslot t.

Next define the graph $G^{\leq w}_{\text{room}} = (L \cup R, E^{\leq w}_{\text{room}})$ with $E^{\leq w}_{\text{room}} = \{e \in E_{\text{room}} : w_e \leq w\}$ as the graph with edges of weight at most w. Next, let w_i for $0 < i \leq k^w$ be an ordering of all edge weights in G_{room} such that $w_i < w_j$ for $i < j$. Then define

$$a_{w_i} = \begin{cases} |L| - \text{def}(G^{\leq w_1}_{\text{room}}) & \text{if } i = 1 \\ \text{def}(G^{\leq w_{i-1}}_{\text{room}}) - \text{def}(G^{\leq w_i}_{\text{room}}) & \text{otherwise} \end{cases}$$

which measure the number of additionally matchable lectures when going from $G^{\leq w_{i-1}}_{\text{room}}$ to $G^{\leq w_i}_{\text{room}}$. Now it is possible to show the following lower bound on the minimum weight maximum matching in G_{room} for some subset of lectures:

Theorem 6.5. *Given the graph $\bar{G}_{room}(\bar{L} \cup R, E_{room})$ then*

$$\sum_{0<i\leq k^w} w_i a_{w_i}$$

is a lower bound on the minimum weight maximum matching problem in the given graph $\bar{G}_{room}$.

Proof. This theorem and its proof are also published in Sørensen and Dahms (2014)[149].

Assume for contradiction that there exists a maximum matching M with a lower weight, i. e., $\sum_{e\in M} w_e < \sum_{i>0} w_i a_{w_i}$. Let b_w denote the number of edges in M of weight lesser or equal w, i. e., $b_w = |\{e \in M : w_e \leq w\}|$. Now let k be the smallest number such that $b_{w_k} > \sum_{0<i\leq k} a_{w_i}$. Assume such a number did not exist. As M is a maximum matching we get $|M| = b_{k^w} = \sum_{0<i} a_{w_i}$. As there is no k with the desired property this implies $b_j = \sum_{0<i\leq j} a_{w_i}$ for any j. Therefore we would get $\sum_{e\in M} w_e = \sum_{i>0} w_i a_{w_i}$, which contradicts the assumption that M was cheaper than the bound.

As b_{w_k} cannot be larger than $\nu(\bar{G}^{\leq w_k}_{\text{room}})$ we get

$$\begin{aligned} b_{w_k} &\leq \nu(\bar{G}^{\leq w_k}_{\text{room}}) = |\bar{L}| - \text{def}(\bar{G}^{\leq w_k}_{\text{room}}) \\ &= |\bar{L}| \underbrace{- \text{def}(\bar{G}^{\leq w_1}_{\text{room}}) + \text{def}(\bar{G}^{\leq w_1}_{\text{room}}) - \text{def}(\bar{G}^{\leq w_2}_{\text{room}}) + \ldots + \text{def}(\bar{G}^{\leq w_{k-1}}_{\text{room}})}_{=0} \\ &\quad - \text{def}(\bar{G}^{\leq w_k}_{\text{room}}) \\ &= \sum_{0<i\leq k} a_{w_i} \end{aligned}$$

which is a contradiction. □

Putting all this together we can use this lower bound in our integer programming formulation if we track the deficiency for each of the subgraphs $G^{\leq w_i}_{\text{room}}$ in a variable $\text{def}_{t,\leq w_i}$. Their respective objective values should then be

$$c(\text{def}_{t,\leq w_i}) = \begin{cases} w_{i+1} - w_i & \text{if } i < k^w \\ -w_i & \text{if } i = k^w \end{cases}$$

in order to track the lower bound in the problem's objective function. Note that in our problem the lowest possible weight is $w_1 = 0$ and therefore we do not need to account for a factor $|\bar{L}| \cdot w_1$.

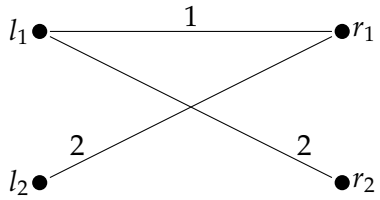


Figure 13: A bad instance for the approximation of the Benders' optimality cuts

Finally we need to make sure that the deficiencies can be tracked with acceptable effort. For the graph $G^{\leq w_k w}_{\text{room}}$ the Corollary 6.2 yields that we can track $\text{def}_{t,\leq w_k}$ with not too many constraints. Luckily the structure of the Lectio instances make sure that each $G^{\leq w_i}_{\text{room}}$ satisfies the conditions of Corollary 6.2, as similar rooms either have the same weight for a certain lecture, or exactly one room is preferred over all others, such that the lecture will be either connected to exactly that room (for small w) or the room will not be treated differently form the others in its group. Finally this means that the deficiencies can be tracked using the constraints

$$\sum_{\substack{l \in N^{-1}_{G^{\leq w_i}_{\text{room}}}(\bar{R}) \\ \{l,t\} \in E_{\text{time}}}} x_{\{l,t\}} - \text{def}_{t,\leq w_i} \leq |\bar{R}| \qquad \forall 0 < i \leq k^w, \forall \bar{R} \in \mathcal{R}^{\leq w_i}$$

with

$$\mathcal{R}^{\leq w_i} = \Big(\bigcup_{l \in L^{1,\leq w_i}} \{N_{G^{\leq w_i}_{\text{room}}}(l)\} \Big) \cup \Big(\bigcup_{\bar{RG} \subseteq RG} \{\cup_{\bar{R}' \in \bar{RG}} \bar{R}'\} \Big)$$

being the set of relevant room subsets according to Corollary 6.2. Note that the set of singleton lectures $L^{1,\leq w_i}$ depends on the current subgraph being considered.

These modifications make it possible to enforce feasibility in the subproblem and at the same time give a lower bound upon its objective value, without resorting to creating the Benders' cuts on the fly. Not having to separate constraints gives the solver more options for presolving and various other techniques to speed up the optimization. Therefore these modifications are quite desirable, even though the objective of the room assignment will not be exact in the master problem.

The method described above only yields an approximation of the true subproblem objective. The following instance is an example, where the lower bound fails to achieve an exact solution: $G = (\{l_1, l_2\} \cup \{r_1, r_2\}, \{\{l_1, r_1\}, \{l_1, r_2\}, \{l_2, r_1\}\})$ with edge weights $w_{\{l_1,r_1\}} = 1$ and $w_{\{l_1,r_2\}} = w_{\{l_2,r_1\}} = 2$. This graph is illustrated in Figure 13. Here a minimum weight maximum matching can only achieve an objective of 4. But for our lower bound we get $\text{def}(G^{\leq 1}) = 1$ and $\text{def}(G^{\leq 2}) = 0$, leading to a bound of $1 \cdot 1 + 2 \cdot 1 = 3$.

Furthermore, the additional requirements for the Lectio instances need to be incorporated. With the exception of the room-stability constraints this can be achieved by adding additional variables and constraints to the formulation such that they only affect the Benders' master problem. For a more compact presentation only the modifications for the Benders' master problem are stated explicitly. For the three indexed formulation they can also be added easily: as the room assignment is not directly affected, one simply needs to replace the assignment variables $x_{\{l,t\}}$ by the sums $\sum_{\{l,r\} \in E_{\text{room}}} x_{\{l,t\},\{l,r\}}$. The requirements only affecting the Benders' master problem can be modeled as follows:

IDLE-TIMESLOTS: In order to track the number of idle timeslots we require three additional types of variables: for each entity $a \in A$ and each day $\bar{T} \in D$ we add the non negative continuous variables $x^{\text{idle}\uparrow}_{a,\bar{T}}$ and $x^{\text{idle}\downarrow}_{a,\bar{T}}$, which shall give the ordinal index of the last and first timeslot used on that day, as well as $x^{\text{idle}}_{a,\bar{T}}$ holding the number of total idle timeslots. The latter variables will get an objective value of β_a, penalizing each idle timeslot. To properly build the constraints incorporating these variables, it is necessary to assign each timeslot in a day an ordinal number $\text{ord}(t)$ with the first timeslot taking number 1, the second number 2, and so forth.

Then a correct value of $x^{\text{idle}\downarrow}_{a,\bar{T}}$ can be ensured via the constraints

$$|\bar{T}| - (|\bar{T}| - \text{ord}(t)) \sum_{\substack{l \in L(a) \\ \{l,t\} \in E_{\text{time}}}} x_{\{l,t\}} \geq x^{\text{idle}\downarrow}_{a,\bar{T}} \qquad \forall a \in A, \bar{T} \in D, t \in \bar{T}.$$

Similarly the constraints to ensure the correct values for $x^{\text{idle}\uparrow}_{a,\bar{T}}$ are

$$\text{ord}(t) \sum_{\substack{l \in L(a) \\ \{l,t\} \in E_{\text{time}}}} x_{\{l,t\}} \leq x^{\text{idle}\uparrow}_{a,\bar{T}} \qquad \forall a \in A, \bar{T} \in D, t \in \bar{T}.$$

Now these variables can be used to ensure that the $x^{\text{idle}}_{a,\bar{T}}$ are tracking the total idle timeslots via the constraints

$$1 + x^{\text{idle}\uparrow}_{a,\bar{T}} - x^{\text{idle}\downarrow}_{a,\bar{T}} - \sum_{\substack{l \in L(a), t \in \bar{T} \\ \{l,t\} \in E_{\text{time}}}} x_{\{l,t\}} = x^{\text{idle}}_{a,\bar{T}} \qquad \forall a \in A, \bar{T} \in D.$$

DAYS-OFF: To count the number of days off, we use variables $x^{\text{d-off}}_{a,\bar{T}} \in \mathbb{R}_+$ counting whether day $\bar{T} \in D$ is used by entity $a \in A$ or not. They get objective values of $-\gamma_a$. Their correct values can be ensured with the constraints

$$\sum_{\substack{l \in L(a), t \in \bar{T} \\ \{l,t\} \in E_{\text{time}}}} x_{\{l,t\}} + x^{\text{d-off}}_{a,\bar{T}} \leq 1 \qquad \forall a \in A, \bar{T} \in D.$$

As some entities have a hard limit F_a of off-days, these shall be incorporated via the constraints

$$\sum_{\bar{T} \in D} x^{\text{d-off}}_{a,\bar{T}} \geq F_a \qquad \forall a \in A.$$

DAY-CONFLICTS: Let $\bar{L}$ be the set of lectures forming one class. Then one needs to add the constraints

$$\sum_{\substack{l \in \bar{L}, t \in \bar{T} \\ \{l,t\} \in E_{\text{room}}}} x_{\{l,t\}} \leq 1 \qquad \forall \bar{T} \in D$$

for each of the classes to ensure that at most one of its lectures is scheduled per day.

NEIGHBOR-DAYS-FOR-CLASSES: For these constraints we need to track the lectures of a class $\bar{L}$ over consecutive days, i. e., over the pairs Monday-Tuesday, Tuesday-Wednesday, and so forth. Let $D^{\text{cons}} \subseteq 2^T$ be the set of timeslots of consecutive days, such that $\bar{T} \in D^{\text{cons}}$ consists of the timeslots of two consecutive days. Now introduce variables $x^{\text{nday}}_{\bar{L},\bar{T}} \in \mathbb{R}_+$ tracking whether there are two lectures within the neighboring days $\bar{T} \in D^{\text{cons}}$ or not. They are assigned an objective value of ζ. To make sure that they take their proper values, add the constraints

$$\sum_{\substack{t\in\bar{T},l\in\bar{L}\\ \{l,t\}\in E_{\text{time}}}} x_{\{l,t\}} - x^{\text{nday}}_{\bar{L},\bar{T}} \leq 1 \qquad \forall \bar{T} \in D^{\text{cons}}.$$

The hard constraints on the amount of neighbor days can be included via constraints

$$\sum_{\bar{T}\in D^{\text{cons}}} x^{\text{nday}}_{\bar{L},\bar{T}} \leq N_{\bar{L}}$$

for each of the classes $\bar{L}$.

TEACHER-DAILY-WORKLOAD: The upper limits for lectures on each day can easily be modeled using constraints

$$\sum_{\substack{l\in L(a),t\in\bar{T}\\ \{l,t\}\in E_{\text{time}}}} x_{\{l,t\}} \leq p_a \qquad \forall a \in A^t, \bar{T} \in D.$$

In order to model the penalties for days with only a single lecture, introduce variables $x^{\text{single}}_{a,\bar{T}} \in \{0,1\}$ indicating whether there is only a single lecture on that day or not. They receive an objective value of η_a. To properly link them to the rest of the model we use the variables $x^{\text{d-off}}_{a,\bar{T}}$ from the *day-off* constraints, in addition to the time assignment variables:

$$2 - \sum_{\substack{l\in L(a),t\in\bar{T}\\ \{l,t\}\in E_{\text{time}}}} x_{\{t,l\}} - 2x^{\text{d-off}}_{a,\bar{T}} \leq x^{\text{single}}_{a,\bar{T}} \qquad \forall a \in A, \bar{T} \in D.$$

DAYS-OFF-STABILITY: To ensure the stability of off days over a biweekly schedule, first let $D^1 \subset D$ and $D^2 \subset D$ denote the set of days in the first and second week respectively. Then, using the variables counting the off days $x^{\text{d-off}}_{a,\bar{T}}$, add the constraints

$$\sum_{\bar{T}\in D^1} x^{\text{d-off}}_{a,\bar{T}} - \sum_{\bar{T}\in D^2} x^{\text{d-off}}_{a,\bar{T}} \leq 1 \qquad \forall a \in A$$

and

$$\sum_{\bar{T}\in D^2} x^{\text{d-off}}_{a,\bar{T}} - \sum_{\bar{T}\in D^1} x^{\text{d-off}}_{a,\bar{T}} \leq 1 \qquad \forall a \in A$$

to ensure a maximal imbalance of 1 between the weeks.

CLASS-STABILLITY: To ensure a low imbalance between weeks for the lectures $\bar{L}$ of some class, introduce a variable $x^{\text{stbl}}_{\bar{L}}$ with an objective value of ι, counting

the number of imbalanced lectures. Then let $\bar{T}^1 \subset T$ and $\bar{T}^2 \subset T$ be the set of timeslots in week one and two, respectively. Now the new variables will be connected to the rest of the model via the constraints

$$\sum_{\substack{l \in \bar{L}, t \in \bar{T}^1 \\ \{l,t\} \in E_{\text{time}}}} x_{\{l,t\}} - \sum_{\substack{l \in \bar{L}, t \in \bar{T}^2 \\ \{l,t\} \in E_{\text{time}}}} x_{\{l,t\}} - 1 \leq x^{\text{stbl}}_{\bar{L}}$$

and

$$\sum_{\substack{l \in \bar{L}, t \in \bar{T}^2 \\ \{l,t\} \in E_{\text{time}}}} x_{\{l,t\}} - \sum_{\substack{l \in \bar{L}, t \in \bar{T}^1 \\ \{l,t\} \in E_{\text{time}}}} x_{\{l,t\}} - 1 \leq x^{\text{stbl}}_{\bar{L}}.$$

The only part missing now are the *room-stability* constraints. As they affect the room assignment such that they destroy the matching structure, they cannot be easily incorporated into the Benders' master problem. As a pragmatic solution we decided to ignore them in the master problem and only add them into the subproblem. This way of course the decomposition will not necessarily yield an overall optimal solution. Also note that this way the subproblem will likely not be integral anymore, requiring integrality constraints to be imposed on the room assignment variables $y_{\{l,r\}}$.

The three indexed formulation on the other hand can easily incorporate the *room-stability* constraints. Again the constraints will be formulated only in terms of the Benders' subproblem variables $y_{\{l,r\}}$ but can easily be ported over to the three indexed formulation by instead using the sums $\sum_{\{l,r\} \in E_{\text{time}}} y_{\{l,t\},\{l,r\}}$.

ROOM-STABILITY: For each class $\bar{L}$ we will need variables $y^{\text{room-used}}_{\bar{L},r} \in \{0,1\}$ to track if room r was used by the class, as well as a variable $y^{\text{room-stbl}}_{\bar{L}} \in \mathbb{R}_+$ to count the number of violations of the room stability. The later variables will have an objective value of ϵ.

To make sure that the $y^{\text{room-used}}_{\bar{L},r}$ variables take correct values, the following constraints can be used

$$\sum_{\substack{l \in \bar{L} \\ \{l,r\} \in E_{\text{room}}}} y_{\{l,r\}} \leq |\bar{L}| y^{\text{room-used}}_{\bar{L},r} \qquad \forall r \in R.$$

A stronger LP relaxation can be achieved with constraints

$$y_{\{l,r\}} \leq y^{\text{room-used}}_{\bar{L},r} \qquad \forall l \in \bar{L}, \forall \{l,r\} \in E_{\text{room}}.$$

The algorithm used for the experiments uses the former variant, as it was described in Sørensen and Dahms (2014)[149]. As the subproblem will turn out to not be the bottleneck in the computations this choice does not affect overall performance in a relevant way.

Finally the following constraint needs to be added for each class to correctly count the number of room stability violations:

$$\sum_{r \in R} y^{\text{room-used}}_{\bar{L},r} - 1 \leq y^{\text{room-stbl}}_{\bar{L}}.$$

Modifications for the Udine instances

The additional requirements of the Udine instances need to be included in the model. As with most of the Lectio requirements, they only affect the matching of lectures to timeslots so they can be part of the Benders' master problem. Again only the modifications for the Benders' master problem are stated explicitly and the variants for the three indexed formulation follow from replacing the assignment variables $x_{\{l,t\}}$ by the sums $\sum_{\{l,r\}\in E_{\text{room}}} x_{\{l,t\},\{l,r\}}$. Each requirement can be formulated using additional variables and constraints as follows.

MIN-WORKING-DAYS: Given a set of lectures $\bar{L} \subseteq L$ that form a course and should be spread over at least u days. For each of these courses the following additions need to be made to the model: in order to capture the number of days that were used less than u, add a variable $x^{\text{mwd1}} \in \mathbb{R}_+$. This variable gets an objective value of 1. Also add variables $x^{\text{mwd2}}_{\bar{T}} \in \{0,1\}$ for each set of timeslots $\bar{T} \in D$ belonging to the same day, which model whether that day is used by one of the course's lectures or not. Now these are linked to the rest of the master problem via the constraints

$$x^{\text{mwd2}}_{\bar{T}} \leq \sum_{\substack{l\in\bar{L},t\in\bar{T}\\ \{l,t\}\in E_{\text{time}}}} x_{\{l,t\}} \qquad \forall \bar{T} \in D$$

which ensure that $x^{\text{mwd2}}_{\bar{T}}$ can only be set to 1 if there is at least one lecture of the respective curriculum scheduled to one of the timeslots $\bar{T}$. Finally these variables are then linked with

$$u - \sum_{\bar{T}\in D} x^{\text{mwd2}}_{\bar{T}} \leq x^{\text{mwd1}}_{\bar{T}}$$

to the $x^{\text{mwd1}}_{\bar{T}}$ variable to assert that the penalty is paid for each missing day.

STUDENT-MIN-MAX-LOAD: Let $\bar{L} \subseteq L$ be the set of lectures forming a curriculum and p be the minimum as well as q the maximum number of lectures that should be scheduled on any day from $\bar{L}$. The penalty here can be captured using a similar mechanism as the one for the MinWorkingDays. For each curriculum we will need two variables $x^{\text{above}}_{\bar{T}} \in \mathbb{R}_+$ and $x^{\text{below}}_{\bar{T}} \in \mathbb{R}_+$ to capture for each day the number lectures above or below the respective limit. These will have an objective value of 1. To link them to the other variables, add the constraints

$$p - \sum_{\substack{l\in\bar{L},t\in\bar{T}\\ \{l,t\}\in E_{\text{time}}}} x_{\{l,t\}} \leq x^{\text{below}}_{\bar{T}} \qquad \forall \bar{T} \in D$$

and

$$\sum_{\substack{l\in\bar{L},t\in\bar{T}\\ \{l,t\}\in E_{\text{time}}}} x_{\{l,t\}} - q \leq x^{\text{above}}_{\bar{T}} \qquad \forall \bar{T} \in D$$

ISOLATED-LECTURES: Let again $\bar{L} \subseteq L$ be the set of lectures forming a curriculum. For each curriculum, add variables $x^{\text{iso}}_t \in \mathbb{R}_+$ for each timeslot $t \in T$ representing whether there is an isolated lecture in the respective timeslot or

not. Each of those gets an objective value of 1. To link them to the assignment variables, let $SP(t)$ be the set of directly preceding or succeeding timeslots of t. This way we need to add the constraints

$$\sum_{\substack{l \in L \\ \{l,t'\} \in E_{\text{time}}}} x_{\{l,t'\}} - \sum_{\substack{l \in L, t \in SP(t') \\ \{l,t\} \in E_{\text{time}}}} x_{\{l,t\}} \leq x^{\text{iso}}_{t'} \qquad \forall t' \in T.$$

These additional components add some complexity to the master problem and are exemplary for a multitude of possible other modifications that could be done, depending on the actual problem at hand.

Additionally to these modifications on the master problem and the improvements of the Basic Benders' algorithm explained in Chapter 4, a method to pre-generate some of the feasibility cuts was constructed. This was motivated by the success that the heuristic optimality cuts showed for the Lectio high school instances (see later in Section 6.5). As it cannot be expected to be generally feasible to pre-generate all feasibility cuts it is important to find a relevant sample of them. Here, the following cuts were used:

$$\sum_{\substack{l' \in N^{-1}(N(l)) \\ \{l',t\} \in E_{\text{time}}}} x_{\{l',t\}} \leq |N(l)| \qquad \forall \{l,t\} \in E_{\text{time}}.$$

These cuts have the property that they are already sufficient, if the underlying room matching graph has an ordered structure (where, e. g., each lecture fits into all rooms of sufficient size and no additional requirements on the room are imposed). This condition is usually not satisfied completely but can be assumed to hold for at least a large part of the lectures, making these cuts a reasonable choice for pre-generation.

Modifications for RWTH Aachen University

In order to focus on the hypergraph structure of the subproblem, no further requirements were added to the problem (as was done for the Udine and the Lectio instance). For all instances, those from RWTH Aachen as well as the generated ones, three different solvers were used: the three indexed formulation, as well as two combinatorial Benders' algorithms, of which one used pre-generated Benders' cut (which were generated in the same fashion as for the Udine instances before).

The three indexed formulation is very similar to the one defined earlier in this section. Due to the hypergraph notation only a few notational changes are required:

$$\begin{aligned}
\min \quad & \sum_{\substack{e_{\text{time}} \in E_{\text{time}} \\ e_{\text{room}} \in E_{\text{room}}}} (w_{e_{\text{time}}} - 100) x_{e_{\text{time}}, e_{\text{room}}} & & \\
\text{s.t.} \quad & \sum_{\substack{l \in e_{\text{time}} \in E_{\text{time}} \\ l \in e_{\text{room}} \in E_{\text{room}}}} x_{e_{\text{time}}, e_{\text{room}}} & \leq 1 & \qquad \forall l \in L \\
& \sum_{\substack{t \in e_{\text{time}} \in E_{\text{time}} \\ r \in e_{\text{room}} \in E_{\text{room}}}} x_{e_{\text{time}}, e_{\text{room}}} & \leq 1 & \qquad \forall t \in T, \forall r \in R \\
& \sum_{l \in e_{\text{conf}}} \sum_{\substack{l \in e_{\text{time}} \in E_{\text{time}} \\ t \in e_{\text{time}} \\ l \in e_{\text{room}} \in E_{\text{room}}}} x_{e_{\text{time}}, e_{\text{room}}} & \leq 1 & \qquad \forall e_{\text{conf}} \in E_{\text{conf}}, \forall t \in T
\end{aligned}$$

$$x_{e_{\text{time}},e_{\text{room}}} \in \{0,1\} \qquad \forall l \in L, \substack{\forall e_{\text{time}} \in E_{\text{time}}, l \in e_{\text{time}} \\ \forall e_{\text{room}} \in E_{\text{room}}, l \in e_{\text{room}}}.$$

Note that no weights for the room matching are specified in the objective. This shall keep the model comparable to the combinatorial Benders' algorithms, for which no optimality cuts have been developed within this thesis.

The Benders' algorithms have the master problem formulation

$$\begin{aligned}
\min \quad & \sum_{e_{\text{time}} \in E_{\text{time}}} (w_{e_{\text{time}}} - 100) x_{e_{\text{time}}} \\
\text{s.t.} \quad & \sum_{l \in e_{\text{time}} \in E_{\text{time}}} x_{e_{\text{time}}} && \leq 1 && \forall l \in L \\
& \sum_{l \in e_{\text{conf}}} \sum_{\substack{l \in e_{\text{time}} \in E_{\text{time}} \\ t \in e_{\text{time}}}} x_{e_{\text{time}}} && \leq 1 && \forall e_{\text{conf}} \in E_{\text{conf}}, \forall t \in T \\
& x \in \{0,1\}^{|E_{\text{time}}|}.
\end{aligned}$$

and the subproblem is calculated as a maximum hypergraph matching using the formulation

$$\begin{aligned}
\max \quad & \sum_{e_{\text{room}} \in E_{\text{room}}} y_{e_{\text{room}}} \\
\text{s.t.} \quad & \sum_{l \in e_{\text{room}} \in E_{\text{room}}} y_{e_{\text{room}}} && \leq 1 && \forall e_{\text{time}} \in M_{\text{time}}, \forall l \in e_{\text{time}} \\
& \sum_{\substack{r \in e_{\text{room}} \in E_{\text{room}} \\ t \in e_{\text{time}} \in M_{\text{time}} \\ \exists l, l \in e_{\text{time}} \wedge l \in e_{\text{room}}}} y_{e_{\text{room}}} && \leq 1 && \forall r \in R, t \in T \\
& y \in \{0,1\}_{+}^{|E_{\text{room}}|}.
\end{aligned}$$

where M_{time} is the current master problem solution. If an optimal subproblem solution does not cover all lectures from M_{time}, feasibility cuts are separated as described in Section 4.2, using the RISS algorithm.

For all hypergraph instances also the feasibility cut pre-generation, that was introduced previously for the Udine instances, was tried out. Note that these cuts can be used in a similar way as before (making sure that the hypergraph structure is treated correctly):

$$\sum_{\substack{l' \in N^{-1}(N(l)) \\ t \in e_{\text{time}} \in E_{\text{time}} \\ l' \in e_{\text{time}}}} x_{(l',\bar{T})} \leq |N(l)| \qquad \forall e_{\text{time}} \in E_{\text{time}}, l \in e_{\text{time}}, \forall t \in e_{\text{time}}.$$

EXPERIMENTS

Lectio high school instances

The results presented here are those run for the previous publication by Sørensen and Dahms (2014)[149]. For solving the integer programming formulations Gurobi 5.0.1 was used. The experiments were run on a machine with an Intel Core i7 930@2.80 GHz CPU and 12GB of RAM, running Windows 8, 64 bit. Gurobi was accessed using its C# interface (employing C# 4.5) and it was run with default parameter settings.

The following optimization algorithms were compared:

	ALNS	3 indexed	TSD	TSD^{RoomLB}
Solution found	100	99	100	100
Best solution	77	2	8	18
Bound found	-	46	79	80
Best bound	-	13	19	49

Table 1: Summary of algorithm objectives on the Lectio instances

3 INDEXED: The three indexed formulation. This variant was allowed a time limit of 7200 seconds per instance. This is the only exact method for the entire problem.

TSD^{RoomLB}: This variant uses the Benders' decomposition (two stage decomposition) with adding all relevant feasibility cuts beforehand as well as the constraints for the lower bound on the subproblem objective. Note that by the construction of the formulation it is necessary to run both the master and the subproblem only once. The master problem is constrained by a time limit of 6480 seconds and the subproblem by 720 seconds.

TSD: This variant is equivalent to TSD^{RoomLB} except for it not using the lower bound. The time limits here are the same as for TSD^{RoomLB}.

ALNS: This is the adaptive large neighborhood search heuristic currently used by Lectio for serving its customers. The heuristic is run 10 times, each time with a time limit of 240 seconds. The best of the 10 results is finally taken.

The results of the experiments are summarized in Table 1. The table reports for each algorithm the number of times it found a solution and bound, as well as the times the found solution / bound was the best of the algorithm ensemble (which does not necessarily mean optimal). For more detailed result tables please refer to Sørensen and Dahms (2014)[149].

From a practitioners perspective it quickly becomes clear that the ALNS heuristic does the best job in providing good solutions and is therefore well suited for most applications. From the integer programming based methods TSD^{RoomLB} clearly outperforms the others, with the three indexed formulation providing the poorest results. This indicates that our "static decomposition" approach might be a good direction for future research into exact optimization algorithms for timetabling problems.

In the future it will be interesting to find out whether it is possible to have a way to model the subproblem cost exactly in the master problem without using a cutting plane method. Furthermore, finding a way of including the room stability in a similar way would be very desirable. Also finding ways to improve the performance of the master problem of TSD^{RoomLB} would be helpful, as in the experiments Gurobi here only achieved an average optimality gap of 44.3% and only reported optimality for 4 (small) instances.

Udine instances

For evaluating the different optimization methods on the Udine instances, Gurobi 5.6.3 was used. The separation algorithms for the Benders' cuts as well as the model generating code were implemented in Scala (using Scala version 2.11.4) and accessed Gurobi via its Java interface. This code is available on GitLab at

`https://gitlab.com/florian.dahms/TimetablingSolver`

The experiments were run on machines with Intel Core i7-2600 CPUs with 16GB of RAM. The machines run on openSUSE 13.1 (x86_64) and the 3.11.10 Linux kernel. Gurobi was set to single threaded mode in all experiments. All experiments were run with a time limit of 3600 seconds. Gurobi was configured to stop when reaching an optimality gap of 1%.

The Udine instances were used in two different variations. In one the room capacities are modeled as hard constraints, i. e., there are no edges between lectures and rooms with insufficient size in G_{room}. Here no edge weights are given for the room assignment and therefore there is no need for optimality cuts in the Benders' algorithms. In the other variant the capacities are modeled using edge weights in G_{room} corresponding to the amount of missing seats in the respective lecture room, resulting in turning the capacity requirements into soft constraints. In both variants, rooms which are explicitly forbidden for a certain lecture were not connected by an edge in G_{room} (i. e., the graph cannot be expected to be a complete bipartite graph). The results on the two variations as well as on the instance subsets "ITC" and "Erlangen" are reported separately, resulting in a total of four sets of experiments (abbreviated as "ITC (hard)", "ITC (soft)", "Erlangen (hard)", and "Erlangen (soft)").

The Benders' decomposition is evaluated in three different variants. In one setting, the feasibility cuts are generated by calculating a maximum matching in the room assignment and, if infeasibility is detected, using the RISS algorithm for calculating stronger cuts. In the second setting the feasibility cuts are created using the classical Benders' algorithm by employing the subproblems Farkas vector in case of infeasibility. Thirdly a variant including pre-generated cuts was tested. This variant was used together with the feasibility cuts from the RISS algorithm. In the soft room assignment cases, the optimality cuts are generated by using the subproblems dual values and the connected components algorithm to arrive at smaller optimality cuts, using an α value for each lecture (as each lecture is scheduled only once, there is no need to have α values for each combination of lecture and timeslot). The classical Benders' algorithm with a single α value is not used as such an implementation turned out to be too numerically unstable for providing meaningful results. For further details on these algorithms the reader may refer to Chapter 4.

The Benders' decomposition is only run on integral solutions of the master problem, not on fractional ones, as the addition of "lazy cuts" (removing integral points) at solver stages other than the discovery of a new integral solution seems to be unsupported by Gurobi.

For the three indexed formulation, Gurobi was run with default settings (apart from the already mentioned common settings for all algorithms). For the Benders' decompositions, Gurobi's "MIPFocus" parameter was set to 1, in order to provide more master problem solutions for discovering necessary Benders' cuts.

Instances	3 ind.	B.s' (RISS)	B.s' (Farkas)	B.s' (PreGen)
ITC (hard)	26.23	10.02	19.69	4.37
ITC (soft)	50.54	6.35	8.15	3.82
Erlangen (hard)	131.75	693.57	2503.25	1355.66
Erlangen (soft)	248.17	3453.06	3600.00	3600.00

Table 2: Median running times in seconds on the Udine instances

Instances	3 ind.	B.s' (RISS)	B.s' (Farkas)	B.s' (PreGen)
ITC (hard)	1	4	0	16
ITC (soft)	1	2	9	9
Erlangen (hard)	5	1	0	0
Erlangen (soft)	5	0	1	0

Table 3: "Best algorithm" statistics on the Udine instances

In the following, some core statistics on the performance of the different algorithms will be examined. More detailed information (such as the explicit run times on each instance) is given in Appendix A.

Table 2 shows the median running times in seconds for the three indexed formulation, the Benders' algorithm with maximum matching and RISS, the Benders' algorithm with Farkas vector based feasibility cuts, and the Benders' algorithm with pre-generated cuts. Note that median running times are better suited to account for outliers (such as unsolved instances, hitting the 3600 seconds time limit) than mean running times would be, as the choice of the time limit would heavily impact the means.

Table 3 displays how often each algorithm had the best performance of the three. Best performance is measured as the lowest runtime, and in case no algorithm finished within the time limit, as the lowest remaining optimality gap.

Figures 14 and 15 show runtime profiles of the algorithms on the "ITC" and "Erlangen" instances. In each profile the x-axis is used for the solver runtime and the y-axis to depict the number of instances solved in at most the corresponding amount of time. Note that the x-axis uses a logarithmic scale to facilitate a better resolution on the small time spans while showing the entire 3600 seconds period.

The statistics indicate that, using the stated measures, the Benders' methods outperform the three indexed formulation on the "ITC" instances, while performing badly on the "Erlangen" instances. The good performance on the "ITC" instances might be explained by the much smaller master problem when compared with the three indexed formulation. The solver can solve the LP relaxations much more quickly and work through more nodes of the branch & bound tree within the same amount of time than with the larger model. To a certain regard the three indexed formulation can compensate this disadvantage by using more sophisticated techniques of the MILP solver (such as better preprocessing, heuristics, and cutting planes), some of which are not available in the Benders' formulation due to the subproblem being a black box to the solver. But these advantages seem to be insufficient to really compete with the master problem being much smaller.

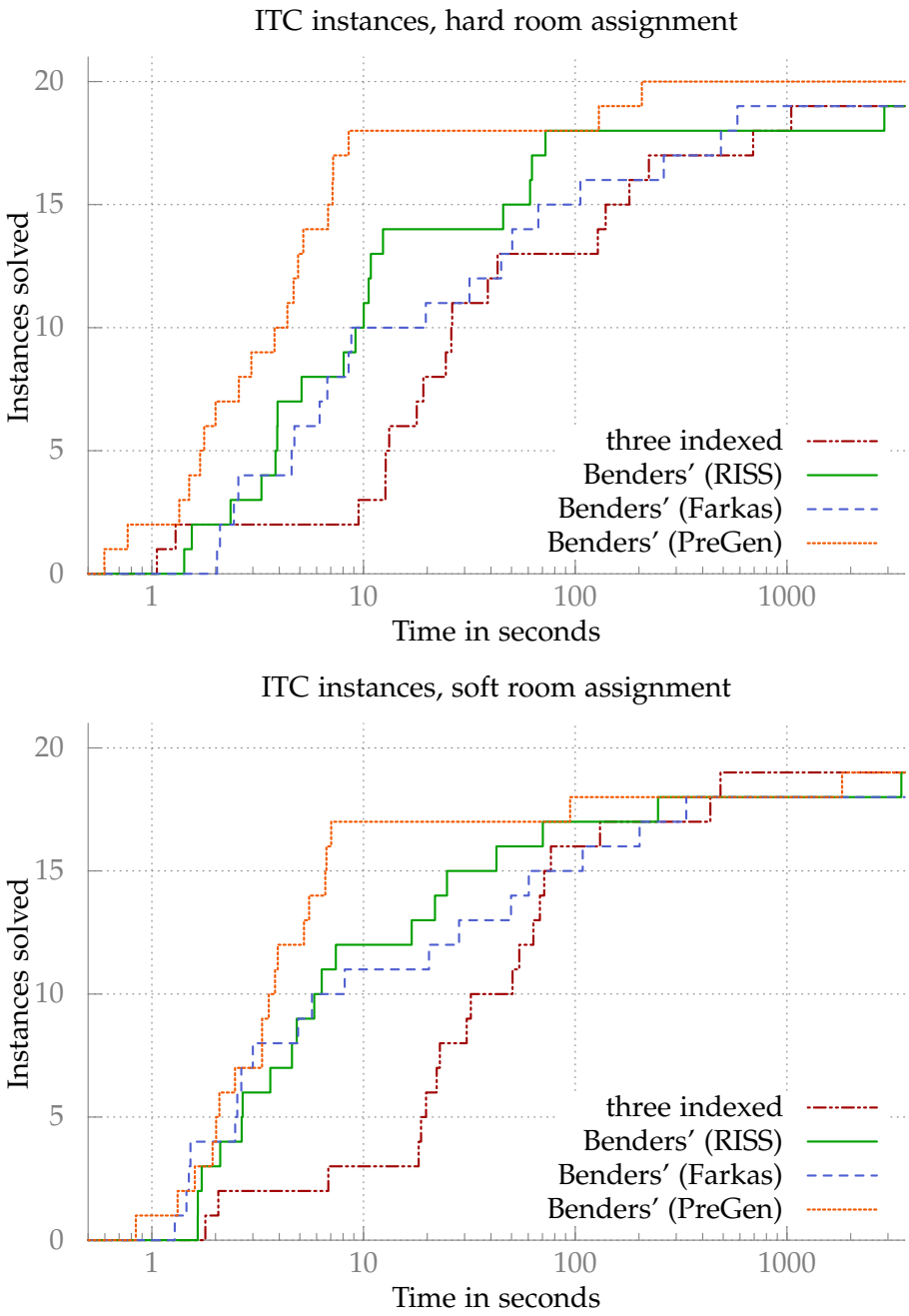


Figure 14: Runtime profile on "ITC" instances

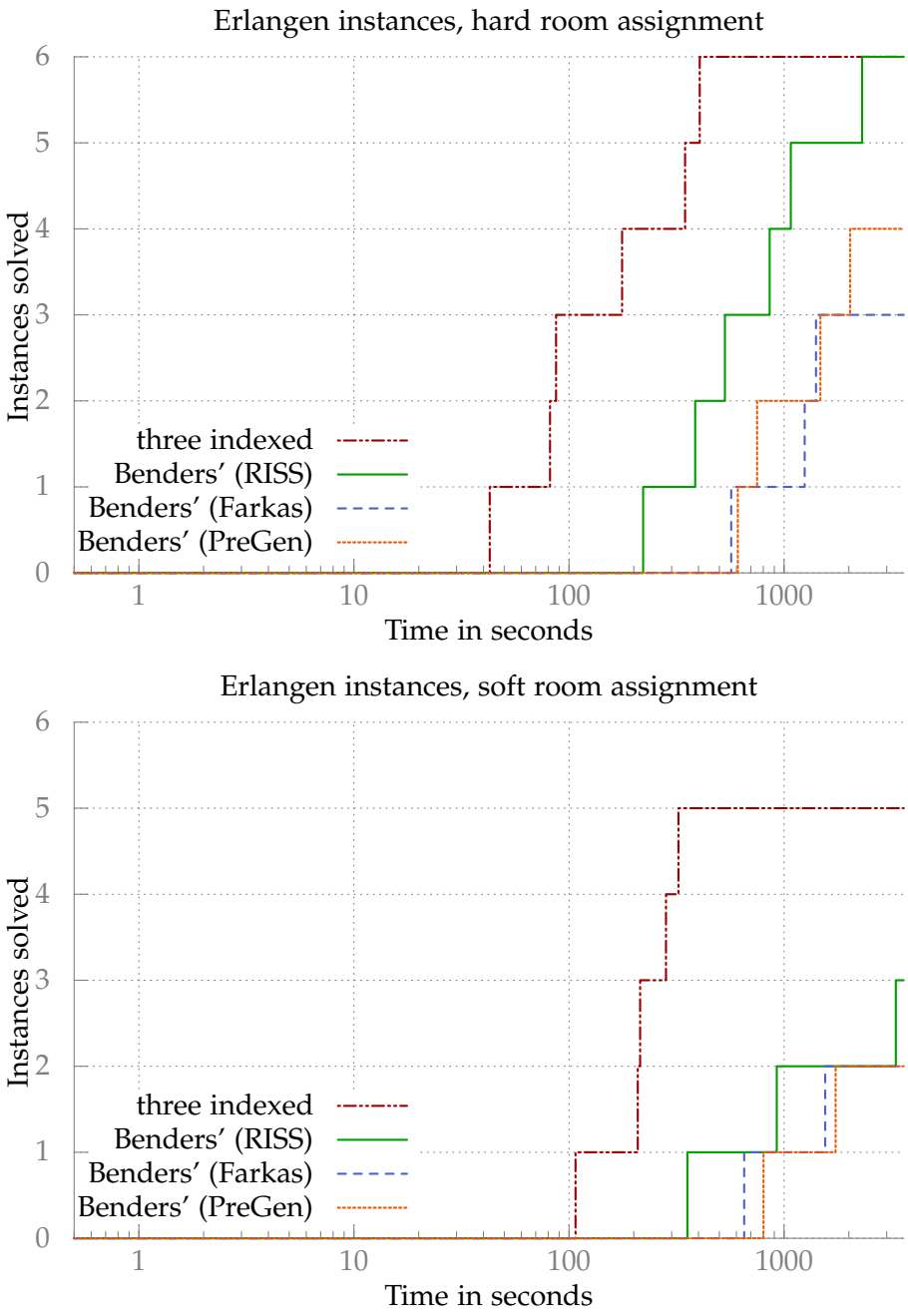


Figure 15: Runtime profiles on "Erlangen" instances

When comparing the variations of the Benders' algorithm, it seems that using the maximum matching with RISS for generating the feasibility cuts seems to yield a slight advantage over the Farkas vector variant, when running on the instances without a subproblem objective while being roughly comparable when the soft room constraints were used.

When looking more closely at the "Erlangen" instances, one discovers that here each lecture only has a very restricted set of possible rooms available in G_{room}. This massively reduces the advantage of the Benders' algorithm, as now its size will be closer to the three indexed formulation, which might be an explanation of the worse performance on these instances.

RWTH Aachen instances

The experimental setting here is the same as for the Udine instances: machines with Intel Core i7-2600 CPUs with 16GB of RAM, openSUSE 13.1 (x86_64), and the 3.11.10 Linux kernel. Again Gurobi was set to single threaded mode in every run. The compared algorithms are the three indexed formulation, a combinatorial Benders' algorithm with the hypergraph room matching as the subproblem and the same Benders' algorithm, but with pre-generated feasibility cuts in the master problem. These pre-generated cuts are generated in the same way as for the Udine instances. The implementation resides within the same code base as the solver for the Udine instances and can be found in the same repository. An optimality gap of 1% was configured to be sufficient for Gurobi to report optimality.

For the artificially generated instances a time limit of 3600 seconds was set. As the RWTH instances are significantly larger, they were run with larger time limits. The summer term instance was run with a time limit of 259200 seconds (three days) and the winter term instance (which is larger than the summer term instance) had a time limit of 518400 seconds (six days).

In Table 4 the results on the two RWTH instances are shown. The table contains the total time each solver ran, the remaining optimality gap at termination (note that the solvers terminated at a gap of less than 1%), the time needed for solving the root node and the number of nodes explored in the branch & bound tree. For the winter term instances no solver was capable of proving a proper optimality gap, despite the large running time. The three indexed formulation did produce a non trivial heuristic solution but did not finish solving the root node. The other two solvers where able to solve the LP relaxation but only found the trivial solution.

Finally the results for the artificial hypergraph timetabling instances are given. Table 5 shows the median remaining optimality gap (note that the solvers did not terminate for most instances, so the median runtimes are always 3600 seconds) for the different instance sizes. Table 6 show a comparision, how often one of the algorithms outperformed the others in terms of runtime or remaining optimality gap (if the time limit was reached for all algorithms). Figures 16 and 17 show graphs indicating how many instances could be solved with up to a certain remaining optimality gap.

	3 ind.	B.s′ (RISS)	B.s′ (PreGen)
summer term instance			
Time	259200	194635	187377
Gap	0.0127	0.0097	0.0060
Time (root node)	4462	4710	1893
Nodes explored	0	1702	1302
winter term instance			
Time	518400	518400	518400
Gap	∞	∞	∞
Time (root node)	518400	314113	362225
Nodes explored	0	49	33

Table 4: Results on the RWTH instances

Instances	3 ind.	B.s′ (RISS)	B.s′ (PreGen)
80 lectures	0.0322	0.0332	0.0287
100 lectures	0.3142	0.1335	0.1259
200 lectures	0.5633	0.5036	0.5095
500 lectures	0.7283	0.6569	0.6467

Table 5: Median remaining optimality gap in percent on the artificial hypergraph timetabling instances

Instances	3 ind.	B.s′ (RISS)	B.s′ (PreGen)
80 lectures	14	7	29
100 lectures	0	20	30
200 lectures	0	22	28
500 lectures	0	28	22

Table 6: "Best algorithm" statistics on the artificial hypergraph timetabling instances

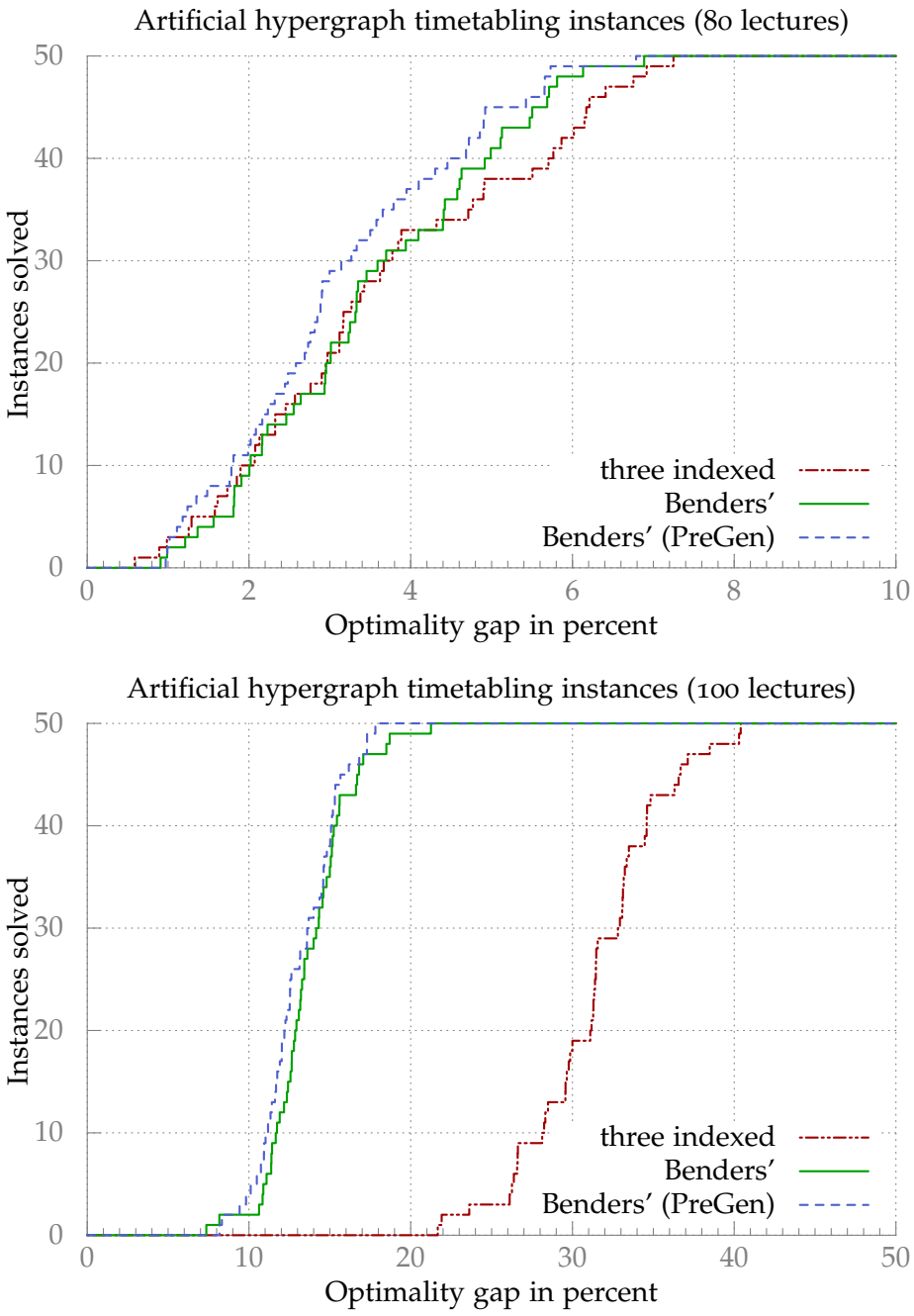


Figure 16: Remaining optimality gap on artificial hypergraph timetabling instances with 80 and 100 lectures

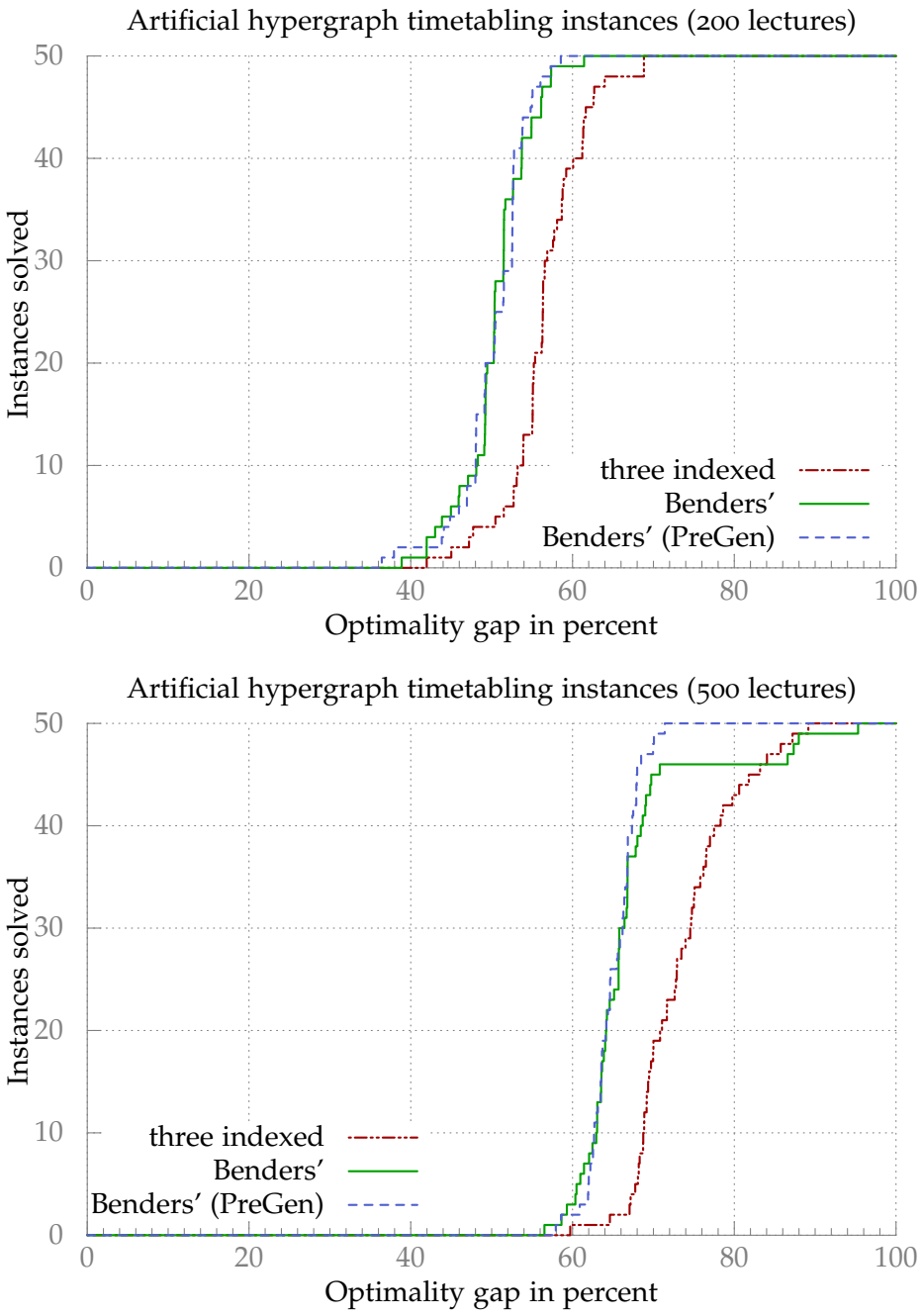


Figure 17: Remaining optimality gap on artificial hypergraph timetabling instances with 200 and 500 lectures

CONCLUSIONS

This chapter has shown how the theoretical results given in Chapter 4 about Benders' type decompositions with matchings as subproblems can be successfully applied to improve integer programming models in educational timetabling problems. The decomposition can result in an improved algorithm performance, if the resulting Benders' master problem is significantly smaller than the original formulation (which, e. g., is not the case for the "Udine – Erlangen" instances). Improvements of the classical Benders' decomposition, like strengthened or pre-generated cuts, can also have a positive impact on the algorithm performance. The method can yield better performance in both cases, where the subproblem is a matching on a regular or on a hypergraph (which need to be bipartite in both cases). Also a method to approximate the optimality cuts, without the need for a separation algorithm, was presented for a specific high school timetabling problem, which turned out to be an improvement upon the previous algorithm lacking this feature.

These results indicate that for timetabling applications Benders' decomposition can be a good method of choice to create better optimization algorithms. For the future there exist several directions into which an interested researcher might want to look:

- A major drawback of implementing the Benders' decomposition with a separation algorithm is, that the underlying MILP solver can not properly anticipate the overall structure of the problem (given that the separation algorithm is provided to it as a black box method, as is usually the case with the major MILP solvers). Therefore the solvers heuristics can have some serious problems with finding good solutions for the master problem. This drawback could be circumvented with tailoring existing heuristics to the problem structure or inventing new, specialized heuristics for this purpose. While meta heuristics for timetabling problems are well studied, an interesting direction would be to construct heuristics that are generically suitable for Benders' decompositions exhibiting certain structures (such as matching).

- For the hypergraph instances it turns out that the pre-generated cuts do not contribute a significant improvement in the algorithm performance, which might be attributed to the fact that the cuts are not strong enough if the underlying structure is based upon a hypergraph matching. To circumvent these drawbacks one can search for cuts which can be easily pre-generated and turn out to be stronger in the hypergraph setting.

- If the subproblem is a hypergraph, the Benders' optimality cuts cannot easily be carried over to the new, nonlinear subproblem. Finding a proper way to do so would be a major contribution and likely have further impact on other combinatorial Benders' algorithms, where such a method is desired. Even some kind of approximation, akin to the one developed for the Lectio high school instances, would mean a significant step forward.

- The timetabling applications are a narrow set of problems where a matching structure can be found. As matchings are a very general concept it is likely that similar structures can be found in other applications. It would be a relevant contribution to identify such problem classes, or even construct a method to automatically detect the relevant structures within a given integer

program. Also the algorithm implementations used for the experiments are all tailored towards the respective timetabling problems. When enough similar problems can be found a more generic implementation of these algorithm will be desirable in order to apply to more problems.

7

THE MULTI-STAGE TRAIN FORMATION PROBLEM

An important step in railway operations is the formation of trains. In this chapter we will look at a particular train formation problem, called the multi-stage train formation problem (MSTF), that is inspired by the operations currently performed at the Hallsberg Rangerbangård hump yard. The MSTF was the original inspiration for the heterogeneous aggregation concept (see Chapter 5) and is now a good showcase for a successful application of this technique. First an introduction into the problem will be given. Then we develop a column generation formulation for the problem that is capable of solving the problem in practice. Next we will reverse engineer the column generation formulation into a compact formulation. Finally it will be shown how the column generation formulation is applicable for heterogeneous aggregation. Unlike the other application chapters, this chapter will not feature computational experiments. Experimental results are available in the publications this chapter is based upon[34,36]. Note that till now no computational study has been performed regarding the application of heterogeneous aggregation towards the MSTF.

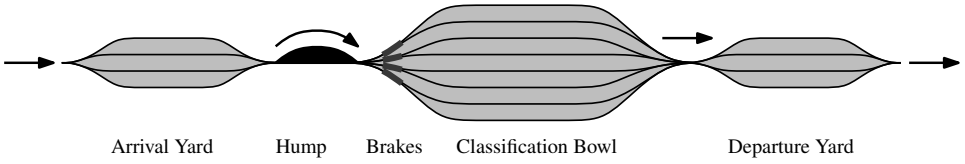
The MSTF was originally introduced by Bohlin et al. in 2011[32,33]. Some results in this chapter have already been published by the author of this thesis and the corresponding co-authors Markus Bohlin, Holger Flier, Sarah Gestrelius and Matúš Mihalák[34–36].

INTRODUCTION

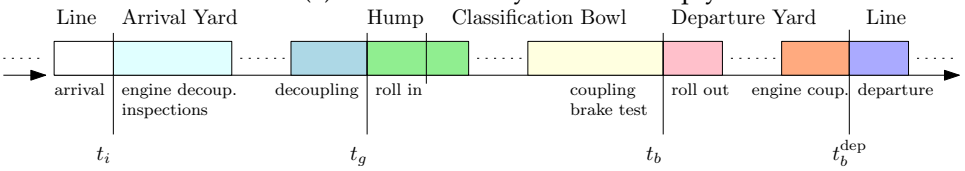
When thinking about railway freight operations one often imagines a train being loaded at its origin and then going all the way to its destination where the freight is cleared. This operational mode, called full train load transportation, represents one of the two main services which are usually provided by rail freight companies. The other one is full car load transportation. Here trains are formed of cars with different origins and destinations. Each car has to go through a network of different railway lines. Here the trains need to be reformed on multiple occasions. This is done in classification yards (also known as marshalling or shunting yards). Most classification yards are so called hump yards.

A hump yard usually consists of four parts: an arrival yard, a hump, a classification bowl and a departure yard (see Figure 18a). The arrival and departure yard, as well as the classification bowl, each consist again of several tracks (arrival, departure, and classification tracks).

The operations explained next are taken from the Hallsberg Rangerbangård hump yard but they are applicable to other yards as the classification procedures are the same or similar for many yards. Trains coming to the yard arrive at the arrival yard, where the cars and their engine are decoupled and inspections are performed. The cars wait there until it is their time to be pushed over the hump, which is an elevated part of the hump yard. From there the cars roll into the classification bowl, guided by automated switches, onto their respective classification track. Here the cars are coupled into outbound trains and a brake test is performed.



(a) A schematic layout of a hump yard.



(b) Activities performed on a single car from its arrival at the yard with a certain train via roll-in to the classification bowl, roll-out, and finally departure as part of a new train

Figure 18: Typical layout of a hump yard, and activities performed on each car passing through the yard (the pictures were originally published by Bohlin et al. (2015)[36])

Note that on each classification track only one train can be formed at any given point of time. After the last car of an outbound train has arrived at the classification track, it can be pulled out into the departure yard, where an engine is coupled to it. Again the cars wait here until their designated departure time. Note that several cars might already be grouped by having the same incoming and outbound train. To avoid overhead we can treat this car group as a single joined car. Therefore the terms car and car group will be used interchangeably.

If cars arrive long before the departure time of their outbound train, they will spent a long time in the yard. Therefore it occurs that cars of more outbound trains need to be present in the classification bowl than there are classification tracks. To accommodate this issue, one or more classification tracks are reserved for cars of trains that are currently not being formed. From these – so called mixing tracks – the cars are periodically pulled out and rolled in over the hump again, in order for the cars to reach their classification track once the formation of their train has started. Of course a car might be placed on the mixing tracks several times, depending on the schedule. The classification tracks that are not used as mixing tracks will be denoted as formation tracks.

Our focus will be on the procedures in the classification bowl, thus the assumption is that the schedules for the arrival and departure yard are already fixed. Also the times when cars are pulled out from the mixing tracks (these times are called periods) are predetermined. Now the free parameters are:

- Which outbound train should be formed on which formation track?
- Should a car be sent to the track of its outbound train or to a mixing track?
- If a car is placed on a mixing track, and the mixing track is pulled out, shall it be sent back to the mixing track?

While making these decisions it must be ensured, that

- two cars of different outbound trains do not share a formation track
- an outbound train is only formed on a track that has sufficient length (as the lengths might differ)

- the lengths of the cars on a mixing track do not exceed the length of the mixing track
- an outbound train is fully formed at the time it is scheduled to leave for the departure yard.

Each time a car is rolled in over the hump, the involved equipment like switches and brakes is worn down a little. Furthermore this operation consumes time as the cars need to be coupled and decoupled again. Our goal is to keep these costs as small as possible, which means minimizing the number of cars that need to be pulled back from a mixing track and rolled back in. All other roll-in operations cannot be avoided as every car needs to be rolled in at least once.

Solving MSTF turns out to be NP-complete by a reduction from the list-coloring problem on interval graphs[32,33]. The list-coloring in turn is a modification of the classical graph coloring problem where each vertex may only receive a color from a predetermined set of colors that may vary for each vertex. This problem is NP-complete even if restricted to interval graphs and even if the lists are nested (i. e., each list must contain all lists of smaller cardinality)[30,37].

Now that the basic idea of MSTF is outlined it is time to introduce the proper notation to describe an instance of this optimization problem and to derive formal definitions of the constraints and objectives. But beforehand a brief overview of related literature will be given.

RELATED WORK

Apart from the previous publications[32–36] upon which parts of this chapter are based, there exists a considerable amount of research regarding railway related problems. Some of these are very early applications of operations research techniques. A good overviews about railway operations in general is given by Nemani and Ahuja (2011)[124]. Good shunting specific overviews are given by Gatto et al. (2009)[78] and Boysen et al. (2012)[39].

In the following part a brief overview on the literature regarding related problems shall be given. These publications do not cover the exact problem definition studied in this chapter but are in some way related.

Blocking problems

In the blocking problem, optimization happens on a more macroscopic level. The goal is to determine transportation plans for the entire railway network. Consequently one does not focus heavily (if at all) on the internal operations of the shunting yards. These problems can be seen as the "bigger picture" into which other shunting problems and the MSTF are embedded. Approaches for solving variants of the blocking problem have been studied by

BODIN ET AL. (1980)[31]
: An early description and mathematical model for the blocking problem.

ASSAD (1993)[14]
: A dynamic programming approach.

VAN DYKE (1986)[159]
A successive shortest path method.

KEATON (1992)[101]
A mixed integer programming approach alongside a Lagrangean relaxation method.

HUNTLEY ET AL. (1995)[95]
A simulated annealing meta heuristic.

GORMAN (1998)[82]
A genetic algorithm and tabu search meta heuristic.

NEWTON ET AL. (1998)[126]
An integer programming approach based on column generation.

BARNHART ET AL. (2000)[19]
Another column generation approach where a Lagrangean relaxation technique is used to solve the subproblems.

Single-stage shunting problems

The single-stage shunting problems do not allow pulling back cars over the hump. Therefore all cars are rolled in only once and will be formed directly afterwards on the formation track they were placed upon. Such settings are quite restrictive and can often only provide limited value for practical applications. On the other hand looking at simplified problem formulations often helps in deriving valuable theoretical lessons about the problem structure.

DAHLHAUS ET AL. (2000)[53]
Introduction into the so called train marshalling problem. This is a single-stage shunting problem where multiple trains may be formed on the same formation track. The track capacity is assumed to be infinite and the objective is to use the minimal amount of formation tracks. The problem is shown to be NP-complete by reduction from numerical matching with target sums.

BEYGANG ET AL. (2010)[28]
The online version of the train marshalling problem is analyzed, where the destinations of the incoming trains are disclosed only when they arrive at the yard. The report shows that there exists a lower bound of 2 for the competitive ratio of this online problem and presents a greedy type algorithm that achieves this ratio. Related to this problem is the diploma thesis of the author of this thesis[54].

DEFINITONS

An instance of the MSTF consists of

a set of car groups	$\mathcal{G}$
a set of outbound trains	$\mathcal{B}$

a set of periods	$\mathcal{P}$
a set of tracks	$\mathcal{A}$
the arrival time of car (group) $g \in \mathcal{G}$ at the hump	t_g
the time train $b \in \mathcal{B}$ departs from its formation track	t_b
the starting time of a period $p \in \mathcal{P}$	t_p
the length of a car (group) $g \in \mathcal{G}$	l_g
the length of a track $a \in \mathcal{A}$	l_a
the total length of the mixing tracks	l^{mix}
the number of cars in a car group $g \in \mathcal{G}$	n_g
the outbound train a car (group) $g \in \mathcal{G}$ belongs to	$b(g)$
the cars of an outbound train $b \in \mathcal{B}$	$\mathcal{G}(b)$
the time needed for roll-out of train	t^{roll}
the time needed for mixing track pull back	t^{pull}

Note that technical set up times like t^{pull} and t^{roll} are given as constants and are independent of the number of cars they are applied to. They are provided by the yard operator and are not subject to optimization as they also include some safety buffer. The arriving trains are not present in the data as it turn out that the only relevant information are the arrival times of the individual car groups, and not the way they were coupled when arriving at the yard. Therefore when the following text refers to a train, always an outbound train will be meant.

Further note that only the total length of the mixing tracks l^{mix} is considered. This simplification is justified as the length of a car compared to the length of a track is small and as the stated track length leaves some safety buffer which gives leeway when distributing the cars on the mixing track.

Some further detail of the operations can be factored into the data above and do not need to be treated independently. When a car arrives after a period is started, but before the time that is planned for it to be finished, its arrival time has to be delayed till the period is finished (which is done by parking the car in the arrival yard). Such information can be directly included in the cars arrival time.

FEASIBLE SCHEDULES

We want to define what constitutes a feasible schedule for the hump yard. A feasible schedule will be defined in terms of track sequences. To understand these we first take a closer look at the kind of decisions we are allowed to make during the shunting process. Assume we have already decided which trains shall be formed on which classification tracks. Naturally the formation of a train b_1 can only start once the respective track is cleared, i. e., a train b_2 that was build previously on this track must have left. Up till that moment all cars of b_1 must be sent to the mixing track (possibly multiple times). When b_2 has left the yard, the formation of b_1 can immediately start. From now on there is no reason to sent one of its cars (back) to the mixing track, as this would result in avoidable roll-ins. This means that the assignment of trains to classification tracks is sufficient to fully determine the operational schedule for the classification tracks.

It may happen that two trains cannot be assembled on the same classification track. If their departure times are so close that there is no period in between, it will not be possible to retrieve cars from the mixing tracks for the train that is departing second. Therefore only certain combinations will be allowed to be build upon the same track. This can be represented via a strict partial order $\prec$ over the outbound trains. For two trains $b_1, b_2 \in \mathcal{B}$ the relation $b_1 \prec b_2$ holds iff

$$(t_{b_1} \leq t_{b_2} - t^{\text{roll}}) \wedge ((\exists g \in \mathcal{G}(b_1), t_g < t_{b_2}) \Rightarrow (\exists p \in \mathcal{P}, t_{b_1} < t_p \leq t_{b_2} - t^{\text{pull}} - t^{\text{roll}})).$$

This basically states that for two trains to be comparable according to the strict partial order, one of the trains needs to leave before the other one (taking into account some buffer for technical set up) and, if there exists a car of the later train arriving before the first one has left the yard, there has to be a period between the two train departures (again taking set up times into account).

To simplify notation let us introduce two virtual trains u and v which shall exist for each track. For these shall hold that for each $b \in \mathcal{B}$ we have $u \prec b \prec v$, i. e., u may precede and v may succeed any train. The virtual trains are designed to act as the first and last train upon each track.

A train sequence $s = (u, b_1, b_2, \ldots, b_k, v)$ will consist of trains such that for $1 \leq i < k$ it holds that $b_i \prec b_{i+1}$, i. e., the trains of a sequence shall be ordered according to the strict partial order. For a given train sequence s the notation $(b, b') \in s$ shall denote that trains b and b' appear in direct succession within the sequence (i. e., there is no other train in between within the sequence). If a train b appears in the sequence this will also be denoted by $b \in s$. The set of all sequences is denoted as $\mathcal{S}$.

Not every possible train sequence will be feasible for each track as the track and train lengths vary. The set of all trains that fit on a given track $a \in \mathcal{A}$ is given by $\mathcal{B}(a) := \{b : (b \in \mathcal{B}) \wedge (l_b \leq l_a)\}$. The set of all sequences that are feasible for a given track $a \in \mathcal{A}$ will then be given as $\mathcal{S}(a)$, i. e., this set will be defined as

$$\mathcal{S}(a) := \{s : (s \in \mathcal{S}) \wedge ((b \in s \wedge v \neq b \neq u) \Rightarrow (l_b \leq l_a))\}.$$

Finally a feasible schedule for an instance consists of a train sequence for each formation track such that every train appears in exactly one sequence, no train of a track's sequence exceeds the length of the track, and such that mixing capacity is never overused. In order to calculate the mixing track usage, first remember that a car needs to be mixed in a certain period if the preceding train on the classification track has not yet left. For period $p \in \mathcal{P}$ let

$$\mathcal{G}_p(b_1, b_2) = \begin{cases} \{g : (g \in \mathcal{G}(b_2)) \wedge (t_g < t_p)\} & \text{if } t_p < t_{b_1} \\ \emptyset & \text{otherwise} \end{cases}$$

be the set of cars that need to be mixed from train b_2 if it is to be formed directly after b_1 upon the same track. Note that mixing only occurs in periods before b_1 has left. In those periods all cars need to be mixed that arrived before the respective period. Furthermore note that for the virtual trains we will have $\mathcal{G}_p(u, b) = \mathcal{G}_p(b, v) = \emptyset$.

For two trains $b_1 \prec b_2$ the required mixing track length in period p for cars of b_1 is

$$l_p(b_1, b_2) = \sum_{g \in \mathcal{G}_p(b_1, b_2)} l_g.$$

Correspondingly the required mixing track length for an entire sequence is

$$l_p(s) = \sum_{(b_1,b_2)\in s} l_p(b_1, b_2).$$

In order to be feasible, a schedule, consisting of a sequence s_a for each track $a \in \mathcal{A}$, therefore has to satisfy

$$\forall p \in \mathcal{P}, \quad \sum_{a\in\mathcal{A}} l_p(s_a) \leq l^{\text{mix}}.$$

In a similar fashion it is possible to calculate the objective of a schedule. For two trains $b_1 \prec b_2$ the number of required extra roll-ins for cars of b_1 is

$$c(b_1, b_2) = \sum_{p\in\mathcal{P}} \sum_{g\in\mathcal{G}_p(b_1,b_2)} n_g.$$

For an entire sequence the total number of required extra roll-ins now is

$$c(s) = \sum_{(b_1,b_2)\in s} c(b_1, b_2).$$

This results in an overall objective of $\sum_{a\in\mathcal{A}} c(s_a)$ for the entire schedule.

Note that the number of pulled out cars for a sequence $c(s)$ and the length of the mixed cars for a certain period $l_p(s)$ are related but not the same. It could, for example, be beneficial to overuse the mixing track capacity for a certain period in order to minimize overall roll-in operations. Still it can be expected that the objective will in general help in keeping the mixing track usage low.

COLUMN GENERATION FORMULATION

The column generation formulation was the first optimization model for the MSTF that was capable of solving instances of practical size (multiple days) within a reasonable time frame (less than 20 minutes). It was first published by Bohlin et al. (2012)[34] and constituted a major improvement over the model that was used beforehand[32,33]. The basic idea of the model is to use a variable per possible sequence for each track. For each such variable it is then easy to calculate the required quantities (mixing track usage and number of cars rolled in due to mixing) as shown in Section 7.4.

The model is now given by

$$\min \sum_{a\in\mathcal{A}} \sum_{s\in\mathcal{S}(a)} c(s) \cdot x_{s,a} \tag{7.1}$$

$$\text{s.t.} \sum_{a\in\mathcal{A}} y_{b,a} \geq 1 \qquad \forall b \in \mathcal{B} \tag{7.2}$$

$$\sum_{\substack{s\in\mathcal{S}(a)\\ s\ni b}} x_{s,a} \geq y_{b,a} \qquad \forall a \in \mathcal{A}, \forall b \in \mathcal{B} \tag{7.3}$$

$$\sum_{s\in\mathcal{S}(a)} x_{s,a} \leq 1 \qquad \forall a \in \mathcal{A} \tag{7.4}$$

$$\sum_{a\in\mathcal{A}} \sum_{s\in\mathcal{S}(a)} l_p(s) \cdot x_{s,a} \leq l^{\text{mix}} \qquad \forall p \in \mathcal{P} \tag{7.5}$$

$$x_{s,a} \in \{0, 1\} \qquad \forall a \in \mathcal{A}, \forall s \in \mathcal{S}(a)$$

$$y_{b,a} \in \{0,1\} \qquad \forall a \in \mathcal{A}, \forall b \in \mathcal{B}(a).$$

In this model the variables $y_{b,a}$ encode whether a certain train shall be assembled upon a certain track. As explained in Section 7.4, this information is sufficient to know the entire schedule. These variables will be used to facilitate branching decisions. The IP solver will be configured to only branch upon the y variables, not on the x variables. Note that in a solution where all y variables are integral, the assignment of trains to tracks will be fully determined and therefore the x variables will need to be integral as well. Branching on the y variables instead of the x variables makes sense as the number of x variables is much larger and branching an x variables to 0 results in very little information gain while the corresponding 1-branch is comparatively strong, leading to an unbalanced branch & bound tree. This is usually undesirable (see also Section 3.3 for more information on branching rules for column generation). By constraints (7.2) it is ensured that one of the variables must be set to 1 for each train, i. e., each train will appear in the resulting schedule.

The variables $x_{s,a}$ track the information which sequence is used on which track. In Section 7.4 was explained that for entire sequences it is easy to compute relevant properties, in particular the number of required extra roll-ins, and the required mixing track length in each period. The quantities are then used to keep track of the objective value (7.1) and to ensure that the total mixing capacity is not overused which is done by constraints (7.5). The constraints (7.4) forbid that more than one sequence is chosen for any track.

The x and y variables are linked together using constraints (7.3) which ensure that if a train is required to appear on a certain track, then a sequence that contains this train has to be selected for the specified track. Due to the minimization objective it is sufficient to use greater or equal constraints here and for constraints (7.2), instead of equality constraints. Not using equality constraints will later help in the formulation of the pricing problem.

Pricing

The number of possible sequences can grow exponentially in the number of trains. Therefore the number of x variables is likely too large to be used explicitly when solving the model. Instead it makes sense to generate these variables only when they are required. This procedure, denoted column generation, is also described in Sections 3.2 and 3.3, where it is used to solve Dantzig-Wolfe decompositions.

The algorithm will always work with a (possibly empty) subset of the x variables and add variables on demand. To figure out which variables are missing, one can use the problem's dual formulation and search for violated constraints (note that each original variable corresponds to a dual constraint – a violated dual constraint corresponds to a primal variable that might lead to an improvement in the objective function). The dual formulation of the MSTF is

$$\max \sum_{b \in \mathcal{B}} \alpha_b + \sum_{a \in \mathcal{A}} \gamma_a + \sum_{p \in \mathcal{P}} l^{\text{mix}} \cdot \delta_{\mathcal{P}}$$

$$\text{s.t.} \sum_{b \in \mathcal{P}} \beta_{b,a} + \gamma_a + \sum_{p \in \mathcal{P}} l_p(s) \cdot \delta_p \leq c(s) \qquad \forall a \in \mathcal{A}, \forall s \in \mathcal{S}(a) \qquad (7.6)$$

$$\alpha_b \leq \beta_{b,a} \qquad \forall a \in \mathcal{A}, \forall b \in \mathcal{B}(a) \qquad (7.7)$$

$$\begin{aligned}
&\alpha_b \in \mathbb{R}_+ && \forall b \in \mathcal{B} \\
&\beta_{b,a} \in \mathbb{R}_+ && \forall a \in \mathcal{A}, \forall b \in \mathcal{B}(a) \\
&\gamma_a \in \mathbb{R}_- && \forall a \in \mathcal{A} \\
&\delta_p \in \mathbb{R}_- && \forall p \in \mathcal{P}.
\end{aligned}$$

Here the α variables correspond to the constraints (7.2), the β variables correspond to the constraints (7.3), the γ variables correspond to the constraints (7.4), and the δ variables correspond to the constraints (7.5). The dual constraints (7.6) correspond to the primal x variables and the dual constraints (7.7) to the primal y variables.

As our interest lies in finding missing x variables, we need to look for constraints in (7.6) that are violated in the current solution. For a given dual solution one can find such a violation by solving the optimization problem

$$\max_{s \in \mathcal{S}(a)} \{(\sum_{b \in s} \beta_{b,a}) + (\sum_{p \in \mathcal{P}} l_p(s) \cdot \delta_p) - c(s)\} \tag{7.8}$$

for each track $a \in \mathcal{A}$. If the resulting objective value does exceed $-\gamma_a$ then the corresponding constraint is violated. Otherwise no constraint is violated and the corresponding primal problem therefore is optimal.

In order to solve (7.8) without enumerating all sequences, we look more closely at the structure of its objective function. The first term $\sum_{b \in s} \beta_{b,a}$ depends on each train in the sequence separately but is independent of the combination of the trains. The other two terms are dependent on tuples of trains, more precisely on whether a certain train succeeds another one (as explained in Section 7.4). Now given a sequence ends with train b_1 (ignoring the virtual train v), adding another train b_2 to the end of the sequence will add the following term to the objective function:

$$\beta_{b_2,a} + \sum_{p \in \mathcal{P}} \sum_{g \in \mathcal{G}_p(b_1,b_2)} (l_g \cdot \delta_p - n_g).$$

This structure of the subproblem costs makes it possible to model the subproblem as a longest path problem on a directed acyclic graph $G_a = (V_a, E_a)$, where there is a vertex $V_a = \{b : (b \in \mathcal{B}) \wedge (l_b \leq l_a)\}$ for each train that fits on the track corresponding to the current subproblem (including the virtual trains u and v). There is a directed edge (b_1, b_2) for vertices $b_1, b_2 \in V$ if these trains can be scheduled in direct succession on the same track (i. e., $b_1 \prec b_2$). These edges are weighted as follows:

$$w_{(b_1,b_2)} = \begin{cases} 0 & \text{if } b_2 = v \\ \beta_{b_2,a} + \sum_{p \in \mathcal{P}} \sum_{g \in \mathcal{G}_p(b_1,b_2)} (l_g \cdot \delta_p - n_g) & \text{otherwise.} \end{cases}$$

The choice of the edges E_a ensures that every path from u to v corresponds to a feasible schedule for track a. On the other hand every feasible schedule for a will have a corresponding path in G_a, as all possible successions of trains are contained in the graphs edges. The weight of the path will match the subproblem objective of the corresponding schedule. Therefore finding a path maximizing these weights (i. e., a longest path) results in an optimal solution for the subproblem. An example graph for this subproblem is shown in Figure 19.

As G_a is acyclic, solving the longest path problem can be achieved by a polynomial combinatorial algorithm (in general the longest path problem turns out

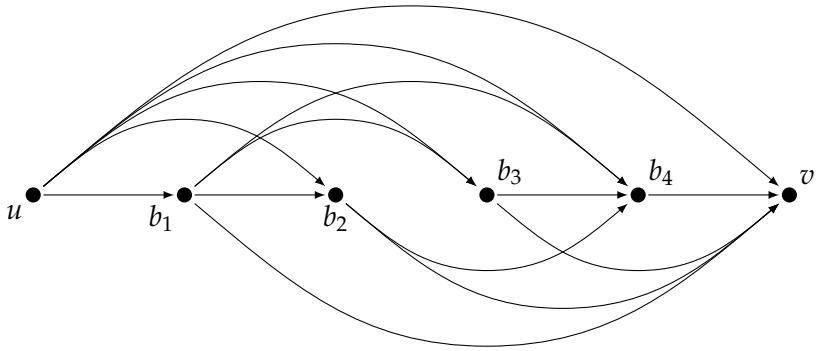


Figure 19: An example for the longest path subproblem of the MSTF CG model

to be NP-complete[77]). Instead of solving the longest path problem on G_a with edge weights w one can solve the equivalent shortest path problem by using the negative edge weights $-w$. As the resulting graph does not contain cycles of negative total weight one can find such a shortest path, e. g., using the Bellman-Ford algorithm[23].

Branching decisions

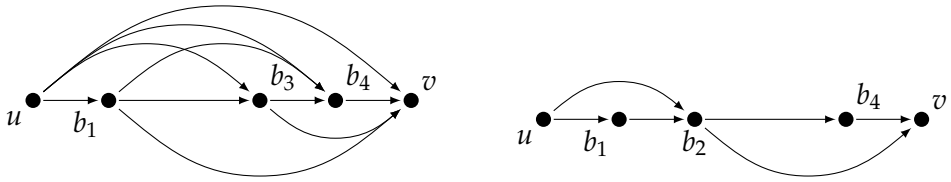
When using column generation it often turns out that the IP solvers standard branching decisions are insufficient to yield good results (see also Section 3.3). In our formulation we introduced variables $y_{b,a}$, encoding whether a certain train is to be scheduled upon a certain track. If these variables are integral, the entire solution will be. Therefore it is sufficient to perform branching decisions upon the y variables. In this way we avoid having to perform branching decisions on the generated variables which would increase the problems complexity. To ensure that the IP solver does not perform branching on x variables they can be declared fractional (or implied integer if such a setting is supported by the solver).

Though the branching itself can now be handled by the underlying IP solver, the pricing problem still requires adjustments to prevent that sequences are generated which are in conflict with the branching decisions of the current branch & bound node. Each branching decision has two possible branches. Either $y_{b,a}$ is set to 0 or to 1 for some combination of train and track. Both of these cases will now be considered separately:

$y_{b,a} = 0$: In this case train b needs to be removed from vertex set V_a. Also all edges ending or starting in b must be removed from E_a. Now no sequence containing b can be generated for track a, while all other sequences are still possible.

$y_{b,a} = 1$: In this case it needs to be guaranteed that every u-v-path in G_a traverses b. To do so, consider all combinations of trains $b', b'' \in V$ such that $b' \prec b''$ and where $t_{b'} \leq t_b \leq t_{b''}$. To ensure that b is not skipped by edge (b', b''), it needs to be removed from E_a. Furthermore we can additionally reduce the graphs size by removing vertices which are now either unreachable or result in a dead end (i. e., vertices $b' \in V$ such that neither $b' \prec b$ nor $b \prec b'$).

The modifications based upon the example in Figure 19 for a branching decision made on train b_2 are shown in Figure 20.



(a) Modifications for $y_{b_2,a} = 0$ (b) Modifications for $y_{b_2,a} = 1$

Figure 20: Branching decisions on train b_2 based upon the example from Figure 19

COMPACT FORMULATION

The column generation (CG) formulation from preceding Section 7.5 can solve reasonably large instances of the MSTF efficiently[36] but the correct implementation of this algorithm requires the consideration of many complex details (such as handling the pricing and branching decisions properly). Therefore it is desirable to find a compact IP formulation that could be used instead. In fact it is possible to formulate the problem in a compact way such that the linear programming relaxation turns out to yield the same bound as the relaxed master problem of the CG formulation. This formulation was first introduced by Güçlü (2012)[84] as a Master's thesis. Later Bohlin et al. (2013)[35] improved this model with slight modifications and provided a rigorous proof of the equivalence of the LP relaxation of this formulation and the CG model.

The compact model exhibits many similarities to the CG formulation as it also centers around the concept of feasible sequences. The basic building block are three indexed variables $x_{b_1,b_2,a}$ which encode whether train b_1 shall be scheduled directly preceding train b_2 upon track a or not. As also used in Section 7.5, the information which train directly precedes another one contains sufficient information to model the associated costs as well as the mixing track usage.

The compact formulation is now given by

$$\min \sum_{a\in\mathcal{A}} \sum_{\substack{b_1,b_2\in\mathcal{B}\\ b_1\prec b_2}} c(b_1,b_2)\cdot x_{b_1,b_2,a} \tag{7.9}$$

$$\text{s.t.} \sum_{\substack{a\in\mathcal{A}\\ b_2\in\mathcal{B}(a)}} \sum_{\substack{b_1\in\mathcal{B}(a)\\ b_1\prec b_2}} x_{b_1,b_2,a} \geq 1 \qquad \forall b_2\in\mathcal{B} \tag{7.10}$$

$$\sum_{b_2\in\mathcal{B}} x_{u,b_2,a} \leq 1 \qquad \forall a\in\mathcal{A} \tag{7.11}$$

$$\sum_{a\in\mathcal{A}} \sum_{\substack{b_1,b_2\in\mathcal{B}(a)\\ b_1\prec b_2}} l_p(b_1,b_2)\cdot x_{b_1,b_2,a} \leq l^{\text{mix}} \qquad \forall p\in\mathcal{P} \tag{7.12}$$

$$\sum_{\substack{b_1\in\mathcal{B}(a)\\ b_1\prec b_2}} x_{b_1,b_2,a} - \sum_{\substack{b_1\in\mathcal{B}(a)\\ b_2\prec b_1}} x_{b_2,b_1,a} = 0 \qquad \forall a\in\mathcal{A}, \forall b_2\in\mathcal{B}(a)\setminus\{u,v\} \tag{7.13}$$

$$x_{b_1,b_2,a} \in \{0,1\} \qquad \forall a\in\mathcal{A}, {}^{\forall b_1,b_2\in\mathcal{B}(a)}_{b_1\prec b_2}.$$

The objective (7.9) is calculated by counting the costs incurred by the chosen succeeding trains for each track. In constraints (7.10) it is guaranteed that each train does appear in one of the variables (and is therefore part of the schedule). With constraints (7.11) only one train is allowed to succeed the virtual train u on each

track, therefore only one sequence can be build upon each track. Constraints (7.12) ensure that the total mixing capacity is not exceeded in any period. Constraints (7.13) are a kind of "flow conservation", enforcing that a train having a predecessor on a certain track must have a successor on the same track (except for the virtual trains u and v).

A useful property of this compact formulation is that its LP relaxation has the same optimal objective value as the relaxed master problem of the column generation formulation:

Theorem 7.1. *The LP relaxation of the compact formulation and the relaxed master problem of the column generation formulation have the same objective value for the same problem.*

Proof. This proof is adapted from Bohlin et al. (2013)[35].

To establish the result, it is enough to show that any feasible solution to the LP relaxation of one problem can be translated into a feasible solution of the other one, having the same objective value.

Note that the y variables in the column generation formulation exist for the purpose of facilitating branching decisions and do not carry information about the solution which is not also present in the x variables. They can be removed from the formulation by joining constraints (7.2), (7.3) and (7.4) to

$$\sum_{a \in \mathcal{A}} \sum_{\substack{s \in \mathcal{S}(a): \\ b \in s}} x_{s,a} \geq 1 \qquad b \in \mathcal{B}. \tag{7.14}$$

Now let x^* be a feasible solution to the LP relaxation of the column generation formulation. A feasible solution $\tilde{x}$ for the compact formulation can then be constructed as

$$\tilde{x}_{b_1,b_2,a} = \sum_{\substack{s \in \mathcal{S}(a): \\ (b_1,b_2) \in s}} x^*_{s,a} \qquad \forall b_1, b_2 \in \mathcal{B}, a \in \mathcal{A}$$

The first step is to show that constraints (7.13) are fulfilled given the construction of x^*. First let $a \in \mathcal{A}$ and $b \in \mathcal{B}(a) \setminus \{u, v\}$. Then for some sequence

$$s' \in \{s \in \mathcal{S}(a) : \exists b' \in \mathcal{B}(a), (b' \prec b) \wedge ((b', b) \in s)\}$$

there has to be some train $b' \in \mathcal{B}(a)$ such that $b \prec b'$ and $(b, b') \in s'$ as $b \neq v$. Similarly the other direction holds as well, as $b \neq u$, which means that

$$\begin{aligned} &\{s \in \mathcal{S}(a) : \exists b' \in \mathcal{B}(a), (b' \prec b) \wedge ((b', b) \in s)\} \\ =&\{s \in \mathcal{S}(a) : \exists b' \in \mathcal{B}(a), (b \prec b') \wedge ((b, b') \in s)\}. \end{aligned}$$

Using this it can now be shown that

$$\begin{aligned} \sum_{\substack{b' \in \mathcal{B}(a) \\ b' \prec b}} \tilde{x}_{b',b,a} - \sum_{\substack{b' \in \mathcal{B}(a) \\ b \prec b'}} \tilde{x}_{b,b',a} &= \sum_{\substack{b' \in \mathcal{B}(a) \\ b' \prec b}} \sum_{\substack{s \in \mathcal{S}(a): \\ (b_1,b_2) \in s}} x^*_{s,a} - \sum_{\substack{b' \in \mathcal{B}(a) \\ b \prec b'}} \sum_{\substack{s \in \mathcal{S}(a): \\ (b_1,b_2) \in s}} x^*_{s,a} \\ &= 0 \end{aligned}$$

and therefore (7.13) is fulfilled. For the other constraints and the objective the proof will work in a similar way.

The objective stays the same in both cases. This can be shown using the definition of the cost $c(s)$:

$$\begin{aligned}\sum_{a\in\mathcal{A}}\sum_{\substack{b_1,b_2\in\mathcal{B}\\ b_1\prec b_2}} c(b_1,b_2)\cdot\tilde{x}_{b_1,b_2,a} &= \sum_{a\in\mathcal{A}}\sum_{\substack{b_1,b_2\in\mathcal{B}\\ b_1\prec b_2}} c(b_1,b_2)\cdot\sum_{\substack{s\in\mathcal{S}(a):\\ (b_1,b_2)\in s}} x^*_{s,a}\\ &= \sum_{a\in\mathcal{A}}\sum_{s\in\mathcal{S}(a)} x^*_{s,a}\cdot\sum_{(b_1,b_2)\in s} c(b_1,b_2)\\ &= \sum_{a\in\mathcal{A}}\sum_{s\in\mathcal{S}(a)} c(s)\cdot x^*_{s,a}.\end{aligned}$$

In a similar way it is shown that the left hand side for (7.5) is the same as in (7.12):

$$\begin{aligned}\sum_{a\in\mathcal{A}}\sum_{\substack{b_1,b_2\in\mathcal{B}(a)\\ b_1\prec b_2}} l_p(b_1,b_2)\cdot\tilde{x}_{b_1,b_2,a} &= \sum_{a\in\mathcal{A}}\sum_{s\in\mathcal{S}(a)} l_p(s)\cdot\sum_{\substack{s\in\mathcal{S}(a):\\ (b_1,b_2)\in s}} x^*_{s,a}\\ &= \sum_{a\in\mathcal{A}}\sum_{s\in\mathcal{S}(a)} x^*_{s,a}\cdot\sum_{(b_1,b_2)\in s} l_p(b_1,b_2)\\ &= \sum_{a\in\mathcal{A}}\sum_{s\in\mathcal{S}(a)} l_p(s)\cdot x^*_{s,a}.\end{aligned}$$

The equivalence of the left hand sides for (7.4) and (7.11) is again shown in a similar fashion:

$$\sum_{b_2\in\mathcal{B}}\tilde{x}_{u,b_2,a} = \sum_{b_2\in\mathcal{B}}\sum_{\substack{s\in\mathcal{S}(a):\\ (b_1,b_2)\in s}} x^*_{s,a} = \sum_{s\in\mathcal{S}(a)} x^*_{s,a}.$$

Also the equivalence of the left hand sides for (7.14) and (7.10) works similarly:

$$\sum_{\substack{a\in\mathcal{A}\\ b_2\in\mathcal{B}(a)}}\sum_{\substack{b_1\in\mathcal{B}(a)\\ b_1\prec b_2}}\tilde{x}_{b_1,b_2,a} = \sum_{\substack{a\in\mathcal{A}\\ b_2\in\mathcal{B}(a)}}\sum_{\substack{b_1\in\mathcal{B}(a)\\ b_1\prec b_2}}\sum_{\substack{s\in\mathcal{S}(a):\\ (b_1,b_2)\in s}} x^*_{s,a} = \sum_{a\in\mathcal{A}}\sum_{\substack{s\in\mathcal{S}(a):\\ b_2\in s}} x^*_{s,a}.$$

The final step of the proof is to show that a corresponding solution x^* can be constructed given a valid $\tilde{x}$. The flow preservation constraints (7.13) ensure that $\tilde{x}$ can be interpreted as a flow between trains for each track, where u is the source and v the sink node. Due to the partial order $\prec$ there can be no cycles in this flow and therefore the flow can be decomposed into simple paths $s^a_1,\ldots,s^a_{n_a}\in\mathcal{S}(a)$ for each track $a\in\mathcal{A}$ (see Ahuja et al. (1993)[5], Theorem 3.5, pp. 89, for a proof of the existence of this decomposition). Let $x^*_{s,a}$ be the flow sent over path s in this decomposition. Due to the nature of the decomposition we get that

$$\tilde{x}_{b_1,b_2,a} = \sum_{\substack{s\in\mathcal{S}(a):\\ (b_1,b_2)\in s}} x^*_{s,a} \qquad \forall b_1,b_2\in\mathcal{B}, a\in\mathcal{A}$$

which concludes the proof, as this implies equality for the objective and the left hand sides of the relevant constraints. □

Theorem 7.1 basically states that in terms of the relaxation the CG and the compact formulation are equivalent. While the column generation formulation has the advantage that the subproblems can be solved very efficiently, the compact formulation can be implemented with less effort. Furthermore it directly profits more from advances made in the technology of the used IP solver, while a CG formulation often requires additional effort to implement new improved solver techniques.

AGGREGATED COLUMN GENERATION FORMULATION

When analyzing the structure of the CG formulation more closely it turns out that most sequences are feasible for multiple tracks. In fact, the real instances studied by Bohlin et al. (2015)[36] exhibit classification tracks which are mostly very similar in length. This leads to many equivalent variables being generated for multiple tracks. This overhead can be circumvented. If we already know which train sequences shall be used, finding the final schedules turns into a bipartite matching problem where sequences need to be matched to tracks of sufficient length.

Column generation problems that exhibit such a partially symmetric structure have already been studied in a more generic setting in Chapter 5. The same techniques can be applied to the CG formulation of the MSTF presented in Section 7.5. This can be achieved by aggregating the subproblems for the different tracks into a single pooled subproblem. The variables will no longer be tied to a certain track but will only encode whether a specific sequence is chosen or not.

As each sequence fitting on some track will also fit on all larger tracks this problem has a nested structure and therefore only $|\mathcal{A}|$ many partial transversal polytope constraints are required to ensure matchability of the chosen sequences. Therefore no cutting plane procedure is required.

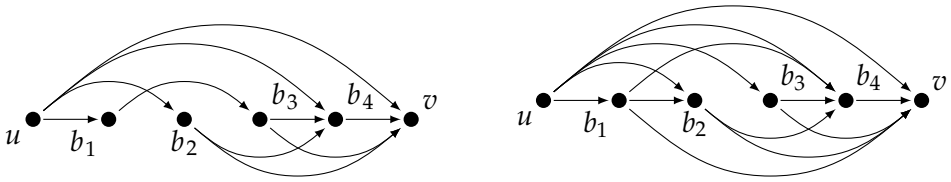
Applying the heterogeneous aggregation technique to the CG formulation results in the following aggregated CG formulation:

$$
\begin{aligned}
\min \quad & \sum_{s \in \mathcal{S}(a)} c(s) \cdot x_s & & \\
\text{s.t.} \quad & \sum_{s \ni b} x_s \geq 1 & & \forall b \in \mathcal{B} \\
& \sum_{s \in \mathcal{S}} l_p(s) \cdot x_s \leq l^{\text{mix}} & & \forall p \in \mathcal{P} \\
& \sum_{\substack{s \in \mathcal{S} \\ \forall a' \in \{a'' \in \mathcal{A} : l_{a''} < l_a\}, s \notin \mathcal{S}(a')}} x_s \leq |\{a' : (a' \in \mathcal{A}) \wedge (l_{a'} \geq l_a)\}| & & \forall a \in \mathcal{A} \\
& x_s \in \{0, 1\} & & \forall s \in \mathcal{S}.
\end{aligned}
$$

Branching

Note that the aggregated CG formulation does not have the y variables used for branching in the CG formulation. Due to the aggregation the old branching scheme is not feasible anymore. As described in Chapter 5 a different approach is required. In this case we are facing a master problem with set covering structure as each train needs to be included in one of the used sequences (note that the $\geq$ in theory allows a train to appear in multiple sequences but such a schedule can always be replaced by a schedule where this train is removed from all but one of these sequences). Therefore the Ryan-Foster branching scheme is applicable in this case. To better fit into the longest path algorithm of the subproblem, the scheme will be slightly modified.

The two possible branching decisions for a tuple of trains (b_1, b_2) will be whether they are to be scheduled in direct succession or not (directly translating the scheme of Ryan and Foster would be to distinguish between them being in the same sequence or not). In each case the pricing problem graph needs to be adjusted in order to prevent forbidden sequences to be generated.



(a) b_1 directly preceding b_3 (b) b_1 not directly preceding b_3

Figure 21: Branching on the train pair (b_1, b_3) for the heterogeneously aggregated column generation formulation based upon the example from Figure 19

b_1 DIRECTLY SUCCEEDED BY b_2: In this case all edges (b', b_2) as well as (b_1, b') for $b' \in \mathcal{B} \setminus \{b_1, b_2\}$ need to be removed from E_a to ensure that b_2 can only be reached from b_1 and choosing b_1 means choosing b_2 next.

b_1 NOT DIRECTLY SUCCEEDED BY b_2: In this case only the edge (b_1, b_2) has to be removed from E_a.

Figure 21 shows how the resulting graphs look like for the example from Figure 19 based upon the train pair (b_1, b_3).

CONCLUSION

This chapter presented the multi-stage train formation problem and three integer programming models for solving it. One of these is based upon variables for entire train sequences for each classification track. This formulation is to be solved using column generation and insights into the implementational details (such as the pricing problem and the branching decisions) are given. As the model exhibits significant symmetries between the subproblems for different tracks, heterogeneous aggregation turns out to be an applicable technique for this problem class. The corresponding formulation as well as the required modifications for the branching decision are presented.

Furthermore a compact model is presented that does not require the use of column generation, and which turns out to be equivalent to the CG model. It is stated that the corresponding linear programming relaxations achieve the very same objective.

The column generation formulation is suitable for heterogeneous aggregation (see chapter 5). So far no experiments have been performed with regard to this method, taking into account the special structure of the problem and comparing the performance with the plain column generation and the compact formulation. This will be an interesting area for future research.

8

VARIOUS APPLICATIONS OF HETEROGENEOUS AGGREGATION

This chapter will present several computational experiments for the concept of heterogeneous variable aggregation that was developed previously in Chapter 5. The optimization problems which are eligible for this methodology and which will be studied here are the list coloring problem, the machine scheduling problem with variable machine speeds and the multiple knapsack problem, which was already used to illustrate the theory in Chapter 5.

First the problems and their definitions will be explained. Hopefully this will help the reader to discover problems with a similar structure where a heterogeneous aggregation algorithm might be of benefit. For the multiple knapsack problem a tailor made implementation will be compared against a generic branch & price method. Together with my colleague Martin Bergner, we implemented a generic implementation of the heterogeneous aggregation method which can be used on arbitrary problem instances exposing the corresponding structure. This implementation will be compared on various instances of the three different optimization problems against the results of the generic column generation framework GCG[75].

Together with the findings in Chapter 5, the experiments in this section where a collaborative effort with Martin Bergner.

PROBLEM DEFINTIONS

This section will briefly explain the optimization problems used for the experiments. The structure will be examined and it will be shown how it is applicable for the heterogeneous aggregation technique. The appropriate integer programming formulation will be stated. Also the sources for the various problem instances will be stated.

The multiple knapsack problem

This problem was already described in Chapter 5. For an instance of the multiple knapsack problem we are given the following sets and parameters:

- a set I of items
- a set K of knapsacks
- a weight a_i for each item $i \in I$
- a value v_i for each item $i \in I$
- a capacity c_k for each knapsack $k \in K$

Each item can be placed in exactly one knapsack or not be packed at all. The sum of the weights of the items within a knapsack may not exceed the knapsack's capacity. The objective is to maximize the total value of the packed items. This problem

is on the one hand closely related to the classical knapsack problem, where only one knapsack would be in use, i. e., $|K| = 1$. On the other hand it generalizes the bin packing problem in the sense that when all knapsacks have the same size we can search for the minimal amount of required knapsacks for packing all items via bisection search. Unlike the classical knapsack problem, the multiple knapsack problem is therefore NP-hard in the strong sense.

A common integer programming formulation for the classical knapsack problem is

$$\max\{\sum_{i\in I} v_i x_i : (\sum_{i\in I} a_i x_i \leq c) \wedge (x \in \{0,1\}^{|I|})\}$$

where each x_i variable encodes whether the item is packed into the knapsack (which has a capacity of c) or not. For the multiple knapsack problem this can easily be extended to

$$\begin{aligned}
\max \quad & \sum_{k\in K}\sum_{i\in I} p_i x_{i,k} & & & \\
\text{s.t.} \quad & \sum_{i\in I} a_i x_{i,k} & \leq c_k & \forall k \in K & (8.1)\\
& \sum_{k\in K} x_{i,k} & \leq 1 & \forall i \in I & (8.2)\\
& x & \in \{0,1\}^{|I|\cdot|K|} & &
\end{aligned}$$

with variables $x_{i,k}$ indicating whether item $i \in I$ will be placed in knapsack $k \in K$ or not.

A typical Dantzig-Wolfe reformulation of the multiple knapsack problem places the constraints (8.1) in the subproblem and the constraints (8.2) in the master problem. The subproblem will in this way have block diagonal structure, with a block corresponding to each knapsack (and each block consisting of a single knapsack constraint). In this way we arrive at the master problem

$$\begin{aligned}
\max \quad & \sum_{k\in K}\sum_{p\in P^k} v^\intercal p \lambda_p^k & & \\
\text{s.t.} \quad & \sum_{k\in K}\sum_{\substack{p\in P^k\\ p_i=1}} \lambda_p^k & \leq 1 & \forall i \in I\\
& \sum_{p\in P^k} \lambda_p^k & = 1 & \forall k \in K\\
& \lambda^k & \in \{0,1\}^{|P^k|} & \forall k \in K
\end{aligned}$$

where each extreme point represents a packing pattern (containing one value for each packed item) for a single knapsack and the sets P^k contain all possible packings for the respective knapsack. The corresponding aMP, using heterogeneous aggregation reads

$$\begin{aligned}
\max \quad & \sum_{p\in P} v^\intercal p \lambda_p & & \\
\text{s.t.} \quad & \sum_{\substack{p\in P\\ p_i=1}} \lambda_p & \leq 1 & \forall i \in I
\end{aligned}$$

$$\begin{aligned}
\sum_{p\in P} \lambda_p & & = |K| & \\
\sum_{\substack{p\in P:\\ \forall k'\notin \bar{K}, a^\intercal p > c_{k'}}} \lambda_p & & \leq |\bar{K}| & \qquad \forall k \in K, \bar{K} = \{k' \in K : c_{k'} \geq c_k\} \qquad (8.3)\\
\lambda & & \in \{0,1\}^{|P|}. &
\end{aligned}$$

Note that the multiple knapsack problem belongs to the class of problems with ordered subproblems (see Section 5.4). Therefore the constraints (8.3) are sufficient to ensure the matchability of the aMP solution to the original blocks.

The instances used for the experiments are based upon those also used by Martello and Toth (1990)[116], Pisinger (1999)[132] and Fukunaga (2011)[73]. A generator for these instances can be found on David Pisinger's homepage:

http://www.diku.dk/~pisinger/codes.html

Of the instance subclasses available from the generator we used the following three:

UNCORRELATED: The item values v_i and the weights a_i are independent and identically distributed (i.i.d.) using a uniform distribution on the interval $[10, 1000]$.

WEAKLY CORRELATED: The weights a_i are i.i.d. with a uniform distribution on the interval $[10, 1000]$. The corresponding values v_i are then drawn uniformly from the interval $[\max\{1, a_i - 99\}, a_i + 99]$ based upon the respective item's weight a_i

STRONGLY CORRELATED: The weights a_i are i.i.d. with a uniform distribution on the interval $[10, 1000]$. The value v_i is then simply assigned as $v_i = a_i + 99$.

For the knapsack capacities c_k there are two different variants:

SIMILAR CAPACITIES: For the first $|K| - 1$ knapsacks the capacities are drawn independent and identically from the uniform distribution on the interval

$$\left[\frac{0.4}{|K|}\sum_{i\in I} a_i, \frac{0.6}{|K|}\sum_{i\in I} a_i\right]$$

DISSIMILAR CAPACITIES: Here the knapsack capacities are dependent upon previously generated capacities. Given the knapsacks are ordered as $k_1, \ldots, k_{|K|}$, then the capacity of knapsack k_j with $1 \leq j \leq |K| - 1$ is drawn from

$$\left[0, \frac{1}{2}\left(\sum_{i\in I} a_i - \sum_{1\leq j'\leq j-1} c_{j'}\right)\right]$$

In both cases, the capacity of the last knapsack $k_{|K|}$ is then assigned as $c_{k_{|K|}} = \frac{1}{2}\sum_{i\in I} a_i - \sum_{k\in K\setminus\{k_{|K|}\}} c_k$ such that the total knapsack capacity is always exactly half of the sum of the knapsack weights.

For the multiple knapsack problem two different sets of implementations are examined. On the one hand there is a dedicated implementation suitable for only the multiple knapsack problem. On the other hand a generic implementation, also suitable for the other heterogeneously aggregatable problems is run for this problem class.

For the dedicated solver, instances of the following sizes were generated: 45 items and 15 knapsacks, 48 items and 12 knapsacks, 60 items and 10 knapsacks, as well as 75 items and 15 knapsacks. For each combination of settings and sizes, 20 different instances were created, leading to a total of 480 individual instances.

For the generic implementation, instances with 2, 5, 10, 20, and 40 knapsacks were created, each of these once with 200 and once with 500 items. For each combination of these sizes, as well as the other instance settings, a total of 10 different instances were created, leading to overall 600 individual instances.

The list coloring problem

The list coloring problem is an extension of the classical vertex coloring problem and was introduced by Vizing (1976)[162]. A list coloring instance consists of

- a regular graph $G = (V, E)$
- a set C of colors
- for each vertex $v \in V$ a list $C(v) \subseteq C$ of colors

A function $f : V \to C$ is called a proper list coloring of G if each vertex is assigned a color from it's list and two adjacent vertices do not share the same color, i. e., $\forall v \in V, f(v) \in C(v)$ and $\forall \{v_1, v_2\} \in E, f(v_1) \neq f(v_2)$. As an extension of the classical vertex coloring problem, the problem of finding a proper list coloring is also NP-hard in the strong sense.

In the optimization variant of the problem, the goal is to find a proper vertex coloring f such that the total amount of used colors $|\bigcup_{v \in V} \{f(v)\}|$ is minimal. An MILP formulation for this is

$$
\begin{aligned}
\min \quad & \sum_{c \in C} y_c & & & \\
\text{s.t.} \quad & \sum_{c \in C(v)} x_{v,c} & \geq 1 & \forall v \in V & (8.4)\\
& x_{v_1,c} + x_{v_2,c} & \leq 1 & \forall \{v_1, v_2\} \in E, \forall c \in C(v_1) \cap C(v_2) & (8.5)\\
& x_{v,c} & \leq y_c & \forall v \in V, \forall c \in C(v) & (8.6)\\
& x_{v,c} & \in \{0,1\} & \forall v \in V, \forall c \in C(v) & \\
& y & \in \{0,1\}^{|C|}. & &
\end{aligned}
$$

Here the variables $x_{v,c}$ encode whether vertex v gets assigned color c or not and the variables y_c track the usage of color c. This formulation can be reformulated in the Dantzig-Wolfe sense by placing the constraints (8.4) in the master problem and relegating constraints (8.5) and (8.6) to the subproblem. This leads to the master problem

$$
\begin{aligned}
\min \quad & \sum_{c \in C} \sum_{p \in P^c} d_p \lambda_p^c & & \\
\text{s.t.} \quad & \sum_{c \in C(v)} \sum_{\substack{p \in P^c \\ p_v = 1}} \lambda_p^c & \geq 1 & \forall v \in V\\
& \lambda^c & \in \{0,1\}^{|P^c|} & \forall c \in C.
\end{aligned}
$$

Each extreme point $p \in P^c$ here encodes an independent set in the subgraph of G containing the vertices which can use color c. Note that independent sets are exactly the groups of vertices which can have the same color. The cost d_p of an extreme point will be zero for the independent set without any vertices and one otherwise.

In each subproblem we need to solve a weighted independent set problem. This column generation formulation using independent sets as columns was initially proposed by Mehrotra and Trick (1996)[118] as an efficient algorithm for the classical vertex coloring problem.

As the master problem treats each subproblem completely equal, heterogeneous aggregation can be applied for list coloring. Let $C(p) = \bigcap_{v \in V : p_v = 1} C(v)$ be the set of all colors suitable for extreme point $p \in P$. Then we arrive at the aMP

$$\begin{aligned}
\min \quad & \sum_{p \in P} d_p \lambda_p & & & \\
\text{s.t.} \quad & \sum_{\substack{p \in P \\ p_v = 1}} \lambda_p & \geq 1 & \quad \forall v \in V & \\
& \sum_{\substack{p \in P \\ C(p) \subseteq \bar{C}}} \lambda_p & \leq |\bar{C}| & \quad \forall \bar{C} \subseteq C & (8.7) \\
& \lambda & \in \{0,1\}^{|P|}. & &
\end{aligned}$$

Note that the number of partial transversal polytope constraints (8.7) might be small if the total number of colors is, which can be the case for many coloring problems. Here one might consider to include the constraints a priori instead of using a separation algorithm.

The list coloring instances used for the experiments are based upon a collection of regular graph coloring instances, published by Michael Trick on

http://mat.gsia.cmu.edu/COLOR/instances.html

These instances are then transformed into list coloring instances as follows. The list of totally available colors is based upon the already known chromatic number (or the best known upper bound for this). We chose about 1.25 times the chromatic number as the number of available colors in most instances, but these numbers were adjusted by hand for some instances in order to yield sufficiently difficult instances if necessary. For each vertex and each color, the color is added to the list of the vertex with probability $1 - p$, where we tried several possible values for p. Instances were generated for $p \in \{0.05, 0.1, 0.2, 0.3, 0.5\}$. If this procedure results in an empty list for some vertex, its list is generated anew, possibly repeating this until achieving a non empty list. Note that this leaves a slight possibility for generating infeasible instances (e. g., having two neighboring vertices with the same color as the only possibility in each of their lists) which was not checked on instance generation and which is a case that did not occur in any of the instances. In total 46 graphs from the library were used, resulting in 230 different instances for the 5 different probability settings.

The machine scheduling problem with variable machine speeds

An instance of the machine scheduling problem with variable machine speeds is defined by the following parameters:

- a set of machines M
- a set of jobs J
- for each job $j \in J$ a weight $w_j \in \mathbb{R}_+$
- for each job $j \in J$ a due date $d_j \in \mathbb{Z}_+$
- for each machine $m \in M$ and each job $j \in J$ a processing time $q_{j,m} \in \mathbb{Z}_+$ indicating the time machine m needs to finish job j

Each job $j \in J$ needs to be assigned to exactly one machine $m \in M$ and a point in time, denoted as $s(j)$, when the job shall start processing. From the assigned time the job will take $q_{j,m}$ time units to be finished. During that time interval no other job may be assigned to the respective machine. The job may not be interrupted or reassigned to another machine but has to reside on its original machine till completion. The job will finish at time $s(j) + q_{j,m}$ and its tardiness will be defined as $a(j) = \max\{0, s(j) + q_{j,m} - d_j\}$, i. e., the time by which the due date was missed.

Our goal will be to minimize the total weighted tardiness, i. e., $\sum_{j \in J} w_j a(j)$. Many different integer programming models exist for this problem (and the numerous variants thereof). Unlu and Mason (2010)[157] list multiple MILP formulations and compare their performance with respect to CPLEX 10.1. They report best performances for time indexed formulations, of which a modified variant shall also be used in this chapter.

Note that all time related properties are assumed to be integral, making it easy to discretize time. For the remainder of the chapter assume that we know an upper bound on the total processing time (which can, e. g., be derived from a heuristic solution). This allows us to use a discrete set T of possible timeslots to which the jobs can be assigned. For a timeslot $t \in T$ the next following timeslot shall be denoted as $t + 1$. Now the following integer programming formulation is suitable to solve the minimum weighted tardiness problem:

$$
\begin{aligned}
\min \quad & \sum_{m \in M} \sum_{j \in J} w_j y_{j,m} & & & \\
\text{s.t.} \quad & \sum_{m \in M} \sum_{t \in T} x_{j,m,t} & = 1 & \forall j \in J & (8.8)\\
& \sum_{j \in J} \sum_{\substack{t' \in T, t' \le t \\ t' \ge t - q_{j,m} + 1}} x_{j,m,t'} & \le 1 & \forall t \in T, \forall m \in M & (8.9)\\
& \Big(\sum_{t \in T} (t + q_{j,m}) x_{j,m,t}\Big) - d_j & \le y_{j,m} & \forall j \in J, m \in M & (8.10)\\
& x & \in \{0,1\}^{|J| \cdot |M| \cdot |T|} & & \\
& y & \in \mathbb{Z}_+^{|J| \cdot |M|}. & &
\end{aligned}
$$

where variables $x_{j,m,t}$ indicate whether job j starts on machine m at time t or not. The variable $y_{j,m}$ counts the tardiness that job j generates on machine m. Note that one can reduce the number of variables by replacing the y variables by an

aggregated version $y'_j = \sum_{m \in M} y_{j,m}$ counting the total tardiness of a job regardless of the machine. While this does not improve the LP relaxation of the formulation, replacing the y variables in this way would later destroy the structure needed to aggregate the master problem variables. Therefore it is desirable to use the aforementioned model.

One way to formulate a Dantzig-Wolfe decomposition is to place constraints (8.8) in the master problem and constraints (8.9) and (8.10) in the subproblem. In this way we get a subproblem for each machine and each column will represent the schedule for its respective machine. P^m will be the set of all feasible schedules for machine m. A schedule will be fully determined by the contained jobs and their completion times. The notation $j \in p$ for some job j and a schedule p will denote that job j is present in schedule p. $f(p,j)$ will be the time when job j is finished according to schedule p. The tardiness for a schedule p is then given as

$$c_p = \sum_{j \in p} \max\{0, f(p,j) - d_j\}$$

The resulting master problem is

$$\begin{aligned}
\min \quad & \sum_{m \in M} \sum_{p \in P^m} c_p \lambda_p^m & & \\
\text{s.t.} \quad & \sum_{\substack{m \in M \\ p \in P^m \\ j \in p}} \lambda_p^m & = 1 & \forall j \in J \\
& \sum_{p \in P^m} \lambda_p^m & = 1 & \forall m \in M \\
& \lambda^m & \in \{0,1\}^{|P^m|} & \forall m \in M.
\end{aligned}$$

As a schedule can be used for any machine that completes the contained jobs sufficiently fast, the following aggregated master problem can be used in order to reduce potential for symmetric solutions. Here $P = \bigcup_{m_i n M} P^m$ is the set of all schedules feasible for any machine and $M(p)$ denotes the set of machines for which p is a feasible schedule.

$$\begin{aligned}
\min \quad & \sum_{p \in P} c_p \lambda_p & & & \\
\text{s.t.} \quad & \sum_{\substack{p \in P \\ j \in p}} \lambda_p & = 1 & \forall j \in J & \\
& \sum_{\substack{p \in P \\ M(p) \subseteq \bar{M}}} \lambda_p & \leq |\bar{M}| & \forall \bar{M} \subseteq M & (8.11) \\
& \lambda & \in \{0,1\}^{|P|}. & &
\end{aligned}$$

Similar to the list coloring problem, the partial transversal polytope constraints (8.11) can be easily enumerated if the number of machines is low (with more than 15 machines the number here quickly grows too large). Also in practice it can easily happen that processing times are ordered in the sense that a faster machine will be generally faster on all jobs, leading to a nested ordering of the subproblems, making it trivial to enumerate the necessary partial transversal polytope constraints.

The instances used for the experiments were created using a random instance generator. The generator and its settings were designed according to recommenda-

tions from Hall and Posner (2001)[86]. As no reason for a more complicated procedure was found, the instance basic parameters are drawn independently from uniform distributions.

First in order to generate the jobs J, a base processing time q_j^{base} for each job is drawn i.i.d from the uniform distribution over the discrete set $\{q^{\min}, \ldots, q^{\max}\}$. In the same fashion the job weights are drawn i.i.d from the uniform distribution over $\{w^{\min}, \ldots, w^{\max}\}$ and the jobs due dates are drawn i.i.d from the uniform distribution over $\{d^{\min}, \ldots, d^{\max}\}$. For each of the machines $m \in M$ a base speed s_m is drawn i.i.d. from the uniform distribution $\{s^{\min}, \ldots, s^{\max}\}$. Now the processing times are assigned as $q_{j,m} = \lceil \frac{q_j^{\text{base}}}{s_m} \rceil$.

In order to generate the time indexed formulation presented earlier, an upper bound on the number of required timeslots is needed. In our case this is estimated as

$$2 \left\lceil \frac{\sum_{j \in J} q_j^{\text{base}}}{\sum_{m \in M} s_m} \right\rceil$$

i. e., the most optimistic make span is taken and multiplied by a factor of 2. For all generated instances this bound leads to a feasible model.

The number of machines was set to 8 and the number of jobs to 40. For the job weights and due dates the values $w^{\min} = 1$, $w^{\max} = 5$, $d^{\min} = 5$, and $d^{\max} = 15$ were chosen for all instances. For the other generator parameters the following five scenarios were tried:

	Scenario				
	A	**B**	**C**	**D**	**E**
$q^{\min}$	1	1	10	10	10
$q^{\max}$	10	15	100	150	150
$s^{\min}$	1	1	10	10	10
$s^{\max}$	2	3	20	30	15

For each scenario 50 different instances were created, resulting in a total of 250 different instances.

Other eligible problems

In general, a problem will likely be suitable for the heterogeneous variable aggregation technique if there is a Dantzig-Wolfe decomposition where branch & price has proven to be an efficient method, and where there are multiple subproblems which are very similar (having the same impact on the objective and the linking constraints) but not necessarily identical. For example the following problems have variants which also might be suitable for heterogeneous aggregation:

VEHICLE ROUTING: In the vehicle routing problem (VRP) a set of vehicles shall be routed through a graph in order to visit certain vertices, minimizing total travel cost and potentially subject to certain constraints, like vehicle capacity. One can formulate a column generation algorithm where each variable represents one complete route for a single vehicle. These variables can easily be aggregated, if one does not care which route is associated with which vehicle.

As a variant of the classical VRP, many applications have to deal with differing vehicle specifics, leading to the variable fleet VRP. Here the matching of routes to vehicles might get more complicated, resulting in an appropriate setting for heterogeneous aggregation.

CUTTING STOCK: The cutting stock problem is a close relative to the bin packing (and the multiple knapsack) problem. The objective is to cut several large rolls (e. g., of paper) of fixed width into several smaller pieces such that the amount of left overs is minimized. For cutting stock one of the earliest column generation procedures is described in Gilmore and Gomory (1961)[80]. Each variable here represents a cutting pattern for a single roll. In the cutting stock problem with variable stock lengths, the rolls have slightly different sizes, leading to the required formulation for the heterogeneous aggregation method.

CREW SCHEDULING: Crew scheduling deals with the task of assigning crew members to certain tasks (usually the operation of aircrafts, trains, etc.) where the order of the tasks plays an important role (e. g., due to the transportation systems schedule). For many variants of this problem, column generation approaches have proven to be efficient for dealing with them[20,64]. Often each variable represents the schedule of a single crew member. As crew members are likely not able to perform every task, this might be again suitable case for heterogeneous aggregation.

EXPERIMENTS – MKP SPECIFIC

For the multiple knapsack instances, two dedicated solvers were implemented based upon the SCIP[4] framework. SCIP's binpacking example was used as the basis for both implementations. In one variant, no aggregation was employed, in the other one the heterogeneous aggregation was implemented. For the heterogeneous aggregation, the partial transversal polytope constraints were added initially in the master problem, as multiple knapsack has ordered subproblems. The branching decisions in the branch & bound tree are either based upon the original variables or performed using the branching rule by Ryan and Foster (see Chapter 5 and Ryan and Foster (1981)[139]). The implementation's source code is available on GitLab:

`https://gitlab.com/florian.dahms/multiple-knapsack`

The experiments were run on machines with Intel Core i7-2600 CPUs with 16GB of RAM. The machines ran on openSUSE 13.1 (x86_64) and the 3.11.10 Linux kernel. The solvers did only employ a single thread each. All experiments were run with a time limit of 600 seconds. For a few instances the experimental code does break due to software bugs, which can be expected in experimental software on this level of complexity. The corresponding instances are treated as if they hit the time limit and their optimality gap is set to ∞

Table 8 shows statistics of the two solvers performance, aggregated by the knapsack similarities, Table 9 shows the same statistics, but aggregated by the correlation between the item sizes and values. Each table shows median running times in seconds, the number of times each solver was able to prove optimality within the time limit and the number of times the solver had the best overall performance –

	Similar capacities		Dissimilar capacities	
	no agg.	agg.	no agg.	agg.
Median time	33.02	26.62	16.27	15.54
Optimal solution found	154	158	217	217
Best performance	127	151	106	139

Table 8: Statistics for dedicated multiple knapsack solvers, aggregated by knapsack similarities

	UnCor		WeakCor		StrongCor	
	no agg.	agg.	no agg.	agg.	no agg.	agg.
Median time	13.44	13.57	18.93	19.73	37.91	32.17
Optimal solution found	150	155	125	121	96	99
Best performance	69	93	83	90	81	107

Table 9: Statistics for dedicated multiple knapsack solvers, aggregated by correlation between item size and value

measured as the lowest running time or the lowest remaining gap (if both did not prove optimality within the time limit).

EXPERIMENTS – GENERIC GCG IMPLEMENTATION

In order to test the heterogeneous aggregation technique on a wider variety of instances, we implemented a generic version of it within the generic column generation framework GCG[75] in version 2.0.0. More information regarding GCG can be found on its homepage:

http://www.or.rwth-aachen.de/gcg/

The implementation is compared to a plain version of GCG 2.0.0 and to a variant of GCG with settings adjusted in order to be more comparable to the heterogeneous aggregation method, as the new technique requires some advanced solver features to be disabled.

The experiments were performed on the RWTH Aachen HPC Cluster using Intel Xeon X5675 processors running at 3.06 GHz. Each job ran non-exclusively on nodes with up to 8 concurrent processes. As concurrent jobs might affect the solver performance and in order to counter the effects of performance variability[112] each instance was run with 3 random permutations of its rows and columns. Each job was given a time limit of 1 hour. For a few instances the experimental code does break due to software bugs, which can be expected in experimental software on this level of complexity. The corresponding instances are treated as if they hit the time limit and their optimality gap is set to ∞.

Tables 10, 11 and 12 show some core statistics regarding the performance of the tested algorithms. For each algorithm is listed how often it

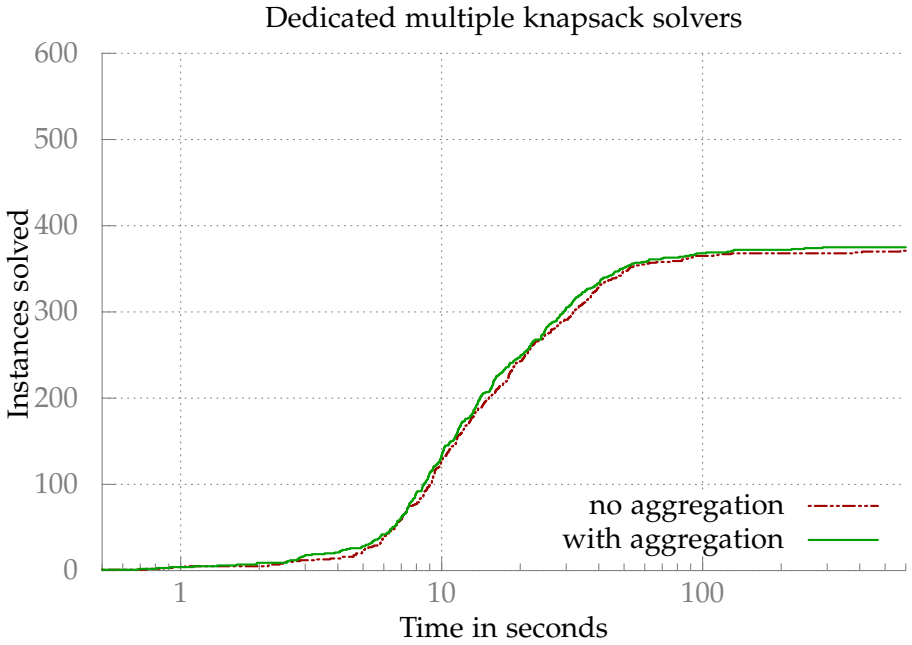


Figure 22: Solver runtimes for dedicated multiple knapsack solvers on all instances

	GCG (plain)	GCG (comparable)	GCG (het. aggr.)
Best performance	1696	1164	108
Best dual bound	1737	1696	970
Best primal bound	1708	1184	103
Best gap	1698	1164	108

Table 10: Overview results for "multiple knapsack" instances

- had the overall best performance, which is measured as the lowest run time (if one of the algorithm terminated within the time limit) or the lowest gap (if each algorithm hit the time limit)
- found the best dual bound from the linear programming relaxation among the three algorithms
- found the best primal bound, i. e., the best solution, among the three algorithms
- had the overall lowest optimality gap among the three algorithms

Draws between the algorithms were counted. Note that each instance is run in three permutations, therefore there are a total of 1800 multiple knapsack, 690 list coloring, and 750 machine scheduling instances.

CONCLUSION

The experimental results show that the heterogeneous aggregation methodology may lead to improved dual bounds in the branch & bound process depending on the problem class and on the implementation and framework used for the

	GCG (plain)	GCG (comparable)	GCG (het. aggr.)
Best performance	445	338	278
Best dual bound	567	563	599
Best primal bound	635	557	469
Best gap	606	528	479

Table 11: Overview results for "list coloring" instances

	GCG (plain)	GCG (comparable)	GCG (het. aggr.)
Best performance	436	329	53
Best dual bound	586	555	360
Best primal bound	664	596	276
Best gap	623	559	297

Table 12: Overview results for "machine scheduling" instances

method. With the dedicated multiple knapsack solvers some improvement could be observed, when the knapsack capacities were similar. This does fit the intuition that in these cases more can be gained by reusing a certain solution for other – similar – subproblems. But on the broad majority of instances only a minor improvement could be observed. While the multiple knapsack problem seems to be not exceptionally suited for the heterogeneous aggregation technique, it would be interesting to try out dedicated algorithms for other suitable problems. For example the list coloring problem might turn out to be a good candidate as the generic GCG implementation obtained a decent dual bound improvement on these.

For the generic GCG implementation one quickly sees that the new method prevents GCG's heuristics from finding good primal solutions, leading to overall worse performance. To make the implementation suitable for a general purpose solver it would therefore be of great importance to design new heuristics specially designed for dealing with the modified problem structure. These – and likely other – implementational improvements are needed before this method can be used as the method of choice in some generic column generation solver framework.

9

SUMMARY AND OUTLOOK

This thesis studied combinatorial optimization problems containing matching structures that can be exploited in order to decompose the larger problem into smaller parts. In many cases such a decomposition can be solved more efficiently. Algorithms that can hold up to this promise are presented for several variations of such decompositions and compared on various instances – some taken from real world examples and some artificial.

Chapter 2 presents results from matching theory which form the foundation for the methods of the later chapters. One of the most fundamental concepts here is the partial transversal polytope, representing the matchable subsets of vertices from one side of a bipartite graph. Multiple equivalent representations of this polytope are shown. It is shown how the partial transversal polytope can be separated and how polymatroid theory can be used to derive the facets of this polytope. These theoretical insights turn out to be applicable in many practical settings. In addition to the theory about the partial transversal polytope it is shown that these results can not be easily adapted for popular matchings, making them less appealing as a subproblem in a larger application. Furthermore several NP-hardness results regarding matching problems in bipartite hypergraphs are presented. Even in very restricted settings these problems turn out to be NP-hard.

In Chapter 4 it is shown that a Benders' decomposition where the subproblem is a bipartite matching problem, can be seen as an algorithm that separates the partial transversal polytope. This knowledge is then used to create an algorithmic framework that is more efficient than a direct application of the Benders' algorithm. It turns out that these findings can be transferred to the case where the subproblem is a bipartite hypergraph matching problem. The methods presented here are rather generic and the reader may find similar structures in her problems. Using the given theory may enable her to fit these algorithms to many other applications, even if their structure does not exactly match those presented in this thesis.

Chapter 5 deals with column generation formulations having multiple similar subproblems. If these subproblems are not fully symmetric but solutions for one subproblem are likely to be feasible for other subproblems it is shown how the subproblems can be aggregated such that the aggregated problem can still be solved to optimality. The key ingredient here is again the partial transversal polytope that is used to ensure that subproblem solutions can be matched to their respective subproblems. Some difficulties here arise regarding changes in the pricing problem. A major theoretical contribution here is given by theorem 5.7 which grants that the pricing problem does not need to increase in complexity when using the proposed heterogeneous aggregation.

The theoretical results from Chapter 4 are applied to various timetabling problems in Chapter 6. The computational results show that in some settings using Benders' decomposition can yield superior algorithms as opposed to a more direct integer programming formulation. It is furthermore demonstrated that the improvements that could be derived from the theory about partial transversals turn out to be beneficial in terms of algorithm performance.

Example applications for the heterogeneous aggregation from Chapter 5 are demonstrated in Chapters 7 and 8. The multi-stage train formation problem from Chapter 7 demonstrates a complex real world application that can be modeled well using a column generation formulation. For this formulation it turns out that one can either use a compact formulation that is equivalent to the column generation formulation or one can solve the column generation formulation directly or using heterogeneous aggregation. Chapter 8 gives more examples for combinatorial optimization problems – the multiple knapsack problem, the list coloring, problem and a machine scheduling problem with variable machine speeds – for which heterogeneous aggregation can be applied. Experiments using a special purpose solver for the multiple knapsack problem as well as a generic implementation for heterogeneous aggregation are performed on various instances. It turns out that these implementations are in general not sufficient to consistently outperform the unaggregated column generation algorithm that was used for comparison. For some instances slight performance benefits could be observed, indicating that heterogeneous aggregation might be a fruitful direction for future research.

The research shown in this thesis opens up multiple directions for future investigations. Here the partial transversal polytope was used to deal with matchings as subproblems of larger integer programs. An open question now is whether these findings can be transfered to other kinds of subproblems. The maximum flow problem for example generalizes the bipartite maximum matching and also has an integral linear programming formulation. It would be interesting to see if there are practical applications where a subproblem turns out to be maximum flow problem and that can be decomposed in a way similar to, e. g., the timetabling problems from Chapter 6.

The heterogeneous aggregation methodology did not perform overly well in the experiments of Chapter 8. At least partially this can be attributed to the fact that implementations used for the presented experiments are not as mature as the algorithms they were compared to. One particular direction in which our implementations may likely benefit from advances are primal heuristics that are tailored towards heterogeneous aggregation. Furthermore other structural properties might exist that allow for a more efficient implementation.

In a similar way the Benders' type algorithms used for the timetabling problems in Chapter 6 suffered from not finding good enough primal solutions. Primal heuristics that can exploit the matching structure of the subproblem might result in even better algorithm performance.

Part III

APPENDIX

A

EXPERIMENTAL RESULTS ON THE TIMETABLING INSTANCES

This appendix contains more detailed solver statistics with regard to each instance used for the experiments of chapter 6. The results on the Lectio instances are omitted as they were already published by Sørensen and Dahms (2014)[149].

UDINE INSTANCES

The Udine dataset is split into two groups, the "ITC" instances and the "Erlangen" instances. All instances were evaluated using the room capacity condition either as hard constraints (thereby removing the need for optimality cuts in the Benders' algorithms) or as soft constraints. The three indexed formulation is compared to the Benders' formulation, which in turn is tried once with the feasibility cuts from using the maximum matching subproblem together with the RISS algorithm and once with the feasibility cuts resulting from the subproblems Farkas vector.

All algorithms were run with a time limit of 3600 seconds and the solver was set to terminate once the optimality gap reached a value of less than 1%. Here Gurobi uses for the optimality gap the definition

$$\frac{|lb - obj|}{|obj|}$$

where "lb" is the best lower bound (as the problems were formulated with a minimization objective) and "obj" the objective value of the best feasible solution found so far.

For each instance and each model, the total solve run time (in wall clock seconds) and the number of branch & bound nodes is reported. For the Benders' algorithms also the number of created cuts (feasibility as well as optimality cuts) is given. For more details on the compared algorithms see Chapter 6.

Tables 13 and 14 show the statistics from solving the "ITC" instances either with hard constrained or soft constrained room matching. Tables 15 and 16 show the respective statistics for the "Erlangen" instances.

Instance	3 ind. model		Benders' (RISS)			Benders' (Farkas)		
	Time	Nodes	Time	Nodes	Cuts	Time	Nodes	Cuts
Fis0506-1	1.29	0	3600.01	168198	65599	3600.01	103530	9937
Fis0506-2	1.06	0	10.02	1005	2063	583.49	71699	11959
Ing0203-1	139.18	256	2.35	0	17	2.56	0	6
Ing0203-2	180.19	121	5.10	0	33	6.20	0	20
Ing0203-3	12.73	0	1.42	0	24	2.44	0	12
Ing0304-1	25.97	0	45.65	602	131	50.41	602	61
Ing0304-2	38.66	0	8.05	0	34	8.76	0	20
Ing0304-3	26.23	0	1.54	0	21	2.03	0	4
Ing0405-1	19.17	0	10.57	284	171	44.71	2424	472
Ing0405-2	127.83	0	12.36	570	377	19.69	715	302
Ing0405-3	9.47	0	3.91	347	153	2.10	16	15
Ing0506-1	42.96	0	62.46	3650	2734	31.67	1233	303
Ing0506-2	691.91	524	72.37	4419	3513	487.78	44395	3869
Ing0506-3	13.22	0	3.93	272	126	4.72	348	65
Ing0607-1	24.48	0	10.82	94	73	66.92	603	110
Ing0607-2	1048.50	629	61.19	3814	2802	105.75	7976	1440
Ing0607-3	12.68	0	3.30	220	92	6.74	578	169
Ing0708-1	222.82	27	9.16	143	82	4.57	0	16
Let0304-1	17.84	0	3.84	0	16	8.50	19	10
Let0405-1	3600.00	9342	3600.00	47800	1284	3600.00	48600	745
Let0506-2	3604.39	2443	2883.88	5902	924	261.99	1159	155

Table 13: Solver statistics on the Udine "ITC" instances with hard constrained room matching

Instance	3 ind. model		Benders' (RISS)			Benders' (Farkas)		
	Time	Nodes	Time	Nodes	Cuts	Time	Nodes	Cuts
Fis0506-1	1.79	0	16.89	1716	3445	201.35	19183	6423
Fis0506-2	2.06	0	3.63	507	761	49.73	7570	3565
Ing0203-1	485.20	282	2.68	0	12	2.53	0	8
Ing0203-2	130.87	0	7.40	0	41	2.64	0	7
Ing0203-3	19.74	0	1.72	0	32	1.50	0	12
Ing0304-1	30.71	0	3600.00	118000	575	3600.00	127000	319
Ing0304-2	68.10	0	4.83	0	27	4.91	0	14
Ing0304-3	22.94	0	1.65	0	7	1.46	0	1
Ing0405-1	63.44	0	21.75	484	382	60.34	953	241
Ing0405-2	54.41	0	42.43	1861	2223	28.25	1238	297
Ing0405-3	18.70	0	1.65	0	33	8.15	305	107
Ing0506-1	50.54	0	246.03	3728	4744	20.39	858	325
Ing0506-2	76.75	0	24.77	883	1019	108.47	6757	1767
Ing0506-3	18.21	0	2.10	0	45	1.52	0	50
Ing0607-1	32.12	0	6.35	0	53	3.00	0	12
Ing0607-2	434.92	0	70.28	3013	3113	334.59	11543	2603
Ing0607-3	22.17	0	4.59	328	116	1.28	0	20
Ing0708-1	71.35	0	2.66	0	41	2.48	0	30
Let0304-1	6.82	0	5.85	0	16	5.71	0	12
Let0405-1	3600.01	5016	3600.00	26985	1953	3600.00	23500	780
Let0506-2	3600.01	3798	3470.67	7022	748	3600.00	6700	801

Table 14: Solver statistics on the Udine "ITC" instances with soft constrained room matching

Instance	3 ind. model		Benders' (RISS)			Benders' (Farkas)		
	Time	Nodes	Time	Nodes	Cuts	Time	Nodes	Cuts
erlangen2011_2	405.33	815	385.98	2465	1257	3600.00	53107	952
erlangen2012_1	42.77	0	856.57	1365	3744	3600.00	8567	3509
erlangen2012_2	81.56	0	2305.48	1202	4159	3600.00	2593	1254
erlangen2013_1	346.37	331	530.56	604	649	1246.73	1969	519
erlangen2013_2	87.01	0	1073.92	1002	2163	1406.51	2602	1685
erlangen2014_1	176.49	0	221.09	315	344	567.35	1002	267

Table 15: Solver statistics on the Udine "Erlangen" instances with hard constrained room matching

Instance	**3 ind. model**		**Benders′ (RISS)**			**Benders′ (Farkas)**		
	Time	Nodes	Time	Nodes	Cuts	Time	Nodes	Cuts
erlangen2011_2	282.29	487	3600.01	66418	1099	3600.00	53845	917
erlangen2012_1	214.05	0	3600.00	12042	5904	3600.00	8815	3295
erlangen2012_2	107.23	0	3306.11	2776	4139	3600.00	4210	1370
erlangen2013_1	3600.47	1133	3600.00	10619	712	3600.00	9778	575
erlangen2013_2	208.74	0	924.62	902	2274	1552.73	2702	1667
erlangen2014_1	322.58	0	354.83	602	486	652.19	1281	324

Table 16: Solver statistics on the Udine "Erlangen" instances with soft constrained room matching

B

EXPERIMENTAL RESULTS ON AGGREGATION INSTANCES

DEDICATED MULTIPLE KNAPSACK SOLVER

Table 17 shows the results that were achieved by the dedicated multiple knapsack algorithms on several instances (the details about the instances and the algorithms can be found in Chapter 8). Each instance set consisted of 20 instances. The table shows

- median run times in seconds
- how often each algorithm found an optimal solution
- how often each algorithm had the best overall performance (either measured in terms of runtime, or of the remaining gap if no algorithm managed to find an optimal solution)
- how often each algorithm found the best dual bound of the two
- how often each algorithm found the best primal bound of the two
- how often each algorithm terminated with the best optimality gap of the two

In statistics where a "winner" was determined, draws were counted towards both algorithms.

	UnCor		WeakCor		StrongCor	
	no agg.	agg.	no agg.	agg.	no agg.	agg.
Similar Capacities						
45 items, 15 bins						
Median time	7.54	9.69	9.47	7.22	600.00	600.00
Optimal solution found	17	19	17	17	7	9
Best performance	10	11	8	15	12	17
Best dual bound	18	20	20	20	17	18
Best primal bound	18	20	20	20	17	18
Best gap	18	20	20	20	17	18
48 items, 12 bins						
Median time	6.52	6.89	27.91	20.00	600.00	600.00
Optimal solution found	20	19	12	13	4	4
Best performance	11	9	11	12	13	18
Best dual bound	20	19	17	18	16	18
Best primal bound	20	19	17	17	17	19
Best gap	20	19	16	17	15	19

	UnCor		WeakCor		StrongCor	
	no agg.	agg.	no agg.	agg.	no agg.	agg.
60 items, 10 bins						
Median time	17.16	15.46	11.90	11.52	600.00	600.00
Optimal solution found	20	20	16	16	8	8
Best performance	8	12	10	12	10	14
Best dual bound	20	20	18	18	15	15
Best primal bound	20	20	18	19	13	16
Best gap	20	20	18	18	13	15
75 items, 15 bins						
Median time	48.13	46.67	34.56	31.59	600.03	600.07
Optimal solution found	16	18	15	15	2	0
Best performance	11	10	12	8	11	13
Best dual bound	17	19	16	15	19	12
Best primal bound	18	19	17	15	11	14
Best gap	17	19	16	15	11	13
Dissimilar Capacities						
45 items, 15 bins						
Median time	9.50	9.09	12.93	16.03	14.36	10.99
Optimal solution found	19	20	17	14	17	19
Best performance	6	14	14	7	11	9
Best dual bound	19	20	18	16	17	19
Best primal bound	19	20	18	15	17	19
Best gap	19	20	18	15	17	19
48 items, 12 bins						
Median time	9.38	9.87	11.32	9.06	13.50	10.08
Optimal solution found	19	20	16	16	18	19
Best performance	10	10	9	13	6	14
Best dual bound	19	20	19	18	18	19
Best primal bound	19	20	18	19	18	19
Best gap	19	20	18	19	18	19
60 items, 10 bins						
Median time	13.95	13.00	20.11	18.96	11.41	10.50
Optimal solution found	20	20	17	17	20	20
Best performance	7	13	11	9	10	10
Best dual bound	20	20	17	17	20	20
Best primal bound	20	20	18	17	20	20
Best gap	20	20	17	17	20	20

	UnCor		WeakCor		StrongCor	
	no agg.	agg.	no agg.	agg.	no agg.	agg.
75 items, 15 bins						
Median time	32.61	30.53	44.53	31.82	31.09	27.11
Optimal solution found	19	19	15	13	20	20
Best performance	6	14	8	14	8	12
Best dual bound	19	19	18	17	20	20
Best primal bound	19	19	17	17	20	20
Best gap	19	19	17	17	20	20

Table 17: Results of the dedicated multiple knapsack solvers (without and with heterogeneous aggregation)

GENERIC IMPLEMENTATION

	GCG (plain)	GCG (comparable)	GCG (het. aggr.)
Multiple Knapsack (200 items, DissCap, StrongCor)			
Median gap	0.15	0.29	5.92
Best performance	136	71	16
Best dual bound	149	148	111
Best primal bound	137	75	15
Best gap	136	71	16
Multiple Knapsack (200 items, SimCap, StrongCor)			
Median gap	0.20	0.28	9.44
Best performance	136	75	15
Best dual bound	141	138	82
Best primal bound	139	79	12
Best gap	136	75	15
Multiple Knapsack (200 items, DissCap, UnCor)			
Median gap	0.10	0.23	4.12
Best performance	144	83	12
Best dual bound	145	137	78
Best primal bound	146	85	11
Best gap	144	83	12
Multiple Knapsack (200 items, SimCap, UnCor)			
Median gap	0.11	0.20	3.65
Best performance	145	86	4
Best dual bound	139	139	97
Best primal bound	145	86	4
Best gap	145	86	4

	GCG (plain)	GCG (comparable)	GCG (het. aggr.)
Multiple Knapsack (200 items, DissCap, WeakCor)			
Median gap	0.03	0.49	4.11
Best performance	136	46	17
Best dual bound	146	137	85
Best primal bound	137	49	18
Best gap	136	46	17
Multiple Knapsack (200 items, SimCap, WeakCor)			
Median gap	0.03	0.49	3.67
Best performance	133	45	19
Best dual bound	143	135	86
Best primal bound	134	48	20
Best gap	133	45	19
Multiple Knapsack (500 items, DissCap, StrongCor)			
Median gap	0.17	0.15	3495.15
Best performance	135	141	2
Best dual bound	137	142	59
Best primal bound	136	142	1
Best gap	135	141	2
Multiple Knapsack (500 items, SimCap, StrongCor)			
Median gap	0.22	0.20	7715.96
Best performance	138	130	14
Best dual bound	138	137	46
Best primal bound	139	133	13
Best gap	138	130	14
Multiple Knapsack (500 items, DissCap, UnCor)			
Median gap	0.11	0.13	2701.89
Best performance	147	142	3
Best dual bound	150	147	40
Best primal bound	147	142	3
Best gap	147	142	3
Multiple Knapsack (500 items, SimCap, UnCor)			
Median gap	0.08	0.11	4.52
Best performance	149	138	0
Best dual bound	149	145	83
Best primal bound	149	138	0
Best gap	149	138	0

	GCG (plain)	GCG (comparable)	GCG (het. aggr.)
Multiple Knapsack (500 items, DissCap, WeakCor)			
Median gap	0.22	0.28	4.12
Best performance	148	103	3
Best dual bound	150	146	103
Best primal bound	149	103	3
Best gap	149	103	3
Multiple Knapsack (500 items, SimCap, WeakCor)			
Median gap	0.22	0.28	4.51
Best performance	149	104	3
Best dual bound	150	145	100
Best primal bound	150	104	3
Best gap	150	104	3
List coloring (0.05)			
Median gap	0.00	30.26	49.93
Best performance	86	61	68
Best dual bound	115	108	121
Best primal bound	124	108	111
Best gap	117	100	110
List coloring (0.1)			
Median gap	6.00	21.71	30.26
Best performance	87	73	65
Best dual bound	115	113	125
Best primal bound	130	120	97
Best gap	122	113	99
List coloring (0.2)			
Median gap	15.21	14.00	31.51
Best performance	91	73	54
Best dual bound	113	117	118
Best primal bound	125	116	91
Best gap	118	109	92
List coloring (0.5)			
Median gap	0.00	2.04	9.68
Best performance	86	66	40
Best dual bound	111	114	127
Best primal bound	126	104	88
Best gap	123	103	94

	GCG (plain)	GCG (comparable)	GCG (het. aggr.)
Machine scheduling (A)			
Median gap	0.00	0.00	4.11
Best performance	103	54	22
Best dual bound	119	118	79
Best primal bound	141	112	60
Best gap	130	107	62
Machine scheduling (B)			
Median gap	0.00	0.70	10.59
Best performance	95	53	19
Best dual bound	117	98	70
Best primal bound	127	100	58
Best gap	114	89	62
Machine scheduling (C)			
Median gap	0.00	0.00	∞
Best performance	86	70	3
Best dual bound	120	109	62
Best primal bound	143	120	41
Best gap	137	113	47
Machine scheduling (D)			
Median gap	0.00	0.00	∞
Best performance	84	71	5
Best dual bound	113	110	68
Best primal bound	125	129	48
Best gap	123	125	55
Machine scheduling (E)			
Median gap	0.00	0.00	0.28
Best performance	68	81	4
Best dual bound	117	120	81
Best primal bound	128	135	69
Best gap	119	125	71

Table 18: Solver quality statistics on various instances, applicable for heterogeneous aggregation

BIBLIOGRAPHY

[1] Abdullah, S., Ahmadi, S., Burke, E. K., Dror, M., and McCollum, B. A tabu-based large neighbourhood search methodology for the capacitated examination timetabling problem. *Journal of the Operational Research Society*, 58(11): 1494–1502, 2007. (Cited on page 93.)

[2] Abraham, D. J., Irving, R. W., Kavitha, T., and Mehlhorn, K. Popular matchings. *SIAM Journal on Computing*, 37(4):1030–1045, 2007. (Cited on pages 36, 37, and 38.)

[3] Abramson, D. Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management science*, 37(1):98–113, 1991. (Cited on page 93.)

[4] Achterberg, T. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. (Cited on page 149.)

[5] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., New Jersey, 1993. ISBN 0-13-617549-X. (Cited on pages 10, 80, and 137.)

[6] Al-Yakoob, S. M. and Sherali, H. D. Mathematical programming models and algorithms for a class–faculty assignment problem. *European Journal of Operational Research*, 173(2):488–507, 2006. (Cited on page 93.)

[7] Al-Yakoob, S. M. and Sherali, H. D. Mathematical models and algorithms for a high school timetabling problem. *Computers & Operations Research*, 2015. (Cited on pages 92 and 93.)

[8] Aladag, C. H., Hocaoglu, G., and Basaran, M. A. The effect of neighborhood structures on tabu search algorithm in solving course timetabling problem. *Expert Systems with Applications*, 36(10):12349–12356, 2009. (Cited on page 93.)

[9] Alvarez-Valdés, R., Martin, G., and Tamarit, J. Constructing good solutions for the spanish school timetabling problem. *Journal of the Operational Research Society*, pages 1203–1215, 1996. (Cited on page 92.)

[10] Alvarez-Valdés, R., Martin, G., and Tamarit, J. M. Hores: A timetabling system for spanish secondary schools. *Top*, 3(1):137–144, 1995. (Cited on page 92.)

[11] Alvarez-Valdés, R., Crespo, E., and Tamarit, J. M. Design and implementation of a course scheduling system using tabu search. *European Journal of Operational Research*, 137(3):512–523, 2002. (Cited on pages 92 and 93.)

[12] Alvarez-Valdés, R., Parreño, F., and Tamarit, J. M. A tabu search algorithm for assigning teachers to courses. *Top*, 10(2):239–259, 2002. (Cited on pages 92 and 93.)

[13] Amaldi, E., Pfetsch, M. E., and Trotter, L. E., Jr. On the maximum feasible subsystem problem, IISs and IIS-hypergraphs. *Mathematical Programming*, 95 (3):533–554, 2003. (Cited on page 53.)

[14] Assad, A. A. Analysis of rail classification policies. *Infor*, 21(4):293–314, 1983. (Cited on page 127.)

[15] Avella, P., Bernardo, D., Salerno, S., and Vasil'ev, I. A computational study of local search algorithms for italian high-school timetabling. *Journal of Heuristics*, 13(6):543–556, 2007. (Cited on page 92.)

[16] Azevedo, F. and Barahona, P. Timetabling in constraint logic programming. In Liebowitz, J., editor, *Proceedings of the 2nd World Congress on Expert Systems*. Cognizant Communication Corp., Elmsford, NY, 1994. (Cited on page 93.)

[17] Balas, E. and Pulleyblank, W. The perfectly matchable subgraph polytope of a bipartite graph. *Networks*, 13(4):495–516, 1983. (Cited on pages 31 and 32.)

[18] Balas, E. and Pulleyblank, W. R. The perfectly matchable subgraph polytope of an arbitrary graph. *Combinatorica*, 9(4):321–337, 1989. (Cited on pages 31 and 32.)

[19] Barnhart, C., Jin, H., and Vance, P. H. Railroad blocking: A network design application. *Operations Research*, 48(4):603–614, 2000. (Cited on page 128.)

[20] Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., and Vance, P. H. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998. (Cited on page 149.)

[21] Beligiannis, G. N., Moschopoulos, C. N., Kaperonis, G. P., and Likothanassis, S. D. Applying evolutionary computation to the school timetabling problem: The greek case. *Computers & Operations Research*, 35(4):1265–1280, 2008. (Cited on page 92.)

[22] Beligiannis, G. N., Moschopoulos, C., and Likothanassis, S. D. A genetic algorithm approach to school timetabling. *Journal of the Operational Research Society*, 60(1):23–42, 2009. (Cited on page 92.)

[23] Bellman, R. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958. (Cited on page 134.)

[24] Benders, J. F. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962. (Cited on pages 41 and 52.)

[25] Berthold, T. Heuristics of the branch-cut-and-price-framework SCIP. In Kalcsics, J. and Nickel, S., editors, *Operations Research Proceedings 2007*, pages 31–36. Springer, Berlin, Heidelberg, 2008. (Cited on page 17.)

[26] Bertsimas, D. and Tsitsiklis, J. N. *Introduction to linear optimization*, volume 6. Athena Scientific, Belmont, MA, 1997. (Cited on page 15.)

[27] Bertsimas, D. and Weismantel, R. *Optimization over integers*, volume 13. Dynamic Ideas, Belmont, MA, 2005. (Cited on pages 19 and 20.)

[28] Beygang, K., Krumke, S. O., and Dahms, F. *Train marshalling problem – algorithms and bounds*. Technische Universität Kaiserslautern, Fachbereich Mathematik, 2010. (Cited on page 128.)

[29] Birbas, T., Daskalaki, S., and Housos, E. Timetabling for greek high schools. *Journal of the Operational Research Society*, 48(12):1191–1200, 1997. (Cited on pages 92 and 93.)

[30] Biró, M., Hujter, M., and Tuza, Z. Precoloring extension. I. Interval graphs. *Discrete Mathematics*, 100(1-3):267–279, 1992. (Cited on pages 35 and 127.)

[31] Bodin, L. D., Golden, B. L., Schuster, A. D., and Romig, W. A model for the blocking of trains. *Transportation Research Part B: Methodological*, 14(1-2): 115–120, 1980. (Cited on page 127.)

[32] Bohlin, M., Flier, H., Maue, J., and Mihalák, M. Hump yard track allocation with temporary car storage. In *Proc. 4th Internat. Seminar on Railway Oper. Modelling and Analysis*, pages 38–51, 2011. (Cited on pages 125, 127, and 131.)

[33] Bohlin, M., Flier, H., Maue, J., and Mihalák, M. Track Allocation in Freight-Train Classification with Mixed Tracks. In *Proc. 11th Workshop on Algorithmic Approaches for Transportation Modelling, Optim., and Systems (ATMOS 2011)*, volume 20, pages 38–51, Dagstuhl, Germany, September 2011. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. (Cited on pages 125, 127, and 131.)

[34] Bohlin, M., Dahms, F., Flier, H., and Gestrelius, S. Optimal Freight Train Classification using Column Generation. In *Proc. 12th Workshop on Algorithmic Approaches for Transportation Modelling, Optim., and Systems (ATMOS 2012)*, volume 25, pages 10–22, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. (Cited on pages v, 125, and 131.)

[35] Bohlin, M., Gestrelius, S., Dahms, F., Mihalák, M., and Flier, H. Optimized shunting with mixed-usage tracks. 2013. Technical Report T2013:06. (Cited on pages v, 135, and 136.)

[36] Bohlin, M., Gestrelius, S., Dahms, F., Mihalák, M., and Flier, H. Optimization methods for multistage freight train formation. *Transportation Science*, 2015. (Cited on pages v, xi, 125, 126, 127, 135, and 138.)

[37] Bonomo, F. and Cecowski, M. Between coloring and list-coloring: μ-coloring. *Electronic Notes in Discrete Mathematics*, 19:117–123, 2005. (Cited on page 127.)

[38] Bonutti, A., De Cesco, F., Di Gaspero, L., and Schaerf, A. Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. *Annals of Operations Research*, 194(1): 59–70, 2012. (Cited on pages 92 and 97.)

[39] Boysen, N., Fliedner, M., Jaehn, F., and Pesch, E. Shunting yard operations: Theoretical aspects and applications. *EJOR*, 220(1):1–14, 2012. (Cited on page 127.)

[40] Burke, E. and Ross, P., editors. *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture notes in computer science*. Springer, Berlin, Heidelberg, New York, 1996. (Cited on page 91.)

[41] Burke, E. K., Marecek, J., Parkes, A. J., and Rudová, H. On a clique-based integer programming formulation of vertex colouring with applications in course timetabling. *arXiv preprint arXiv:0710.3603*, 2007. (Cited on page 93.)

[42] Burke, E. K., Mareček, J., Parkes, A. J., and Rudová, H. Penalising patterns in timetables: Novel integer programming formulations. Operations Research Proceedings 2007, pages 409–414. Springer, Berlin, Heidelberg, 2008. (Cited on page 93.)

[43] Burke, E. K. and Petrovic, S. Recent research directions in automated timetabling. *European Journal of Operational Research*, 140(2):266–280, 2002. (Cited on page 92.)

[44] Carrasco, M. P. and Pato, M. V. A potts neural network heuristic for the class/teacher timetabling problem. In Resende, M. G. C. and Pinho de Sousa, J., editors, *Metaheuristics: Computer Decision-Making*, pages 173–186. Springer, Berlin, Heidelberg, 2004. (Cited on page 93.)

[45] Carrasco, M. P. and Pato, M. V. A comparison of discrete and continuous neural network approaches to solve the class/teacher timetabling problem. *European Journal of Operational Research*, 153(1):65–79, 2004. (Cited on page 93.)

[46] Carter, M. W. and Laporte, G. Recent developments in practical course timetabling. In *Practice and Theory of Automated Timetabling II*, pages 3–19. Springer, 1998. (Cited on pages 91 and 92.)

[47] Codato, G. and Fischetti, M. Combinatorial benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766, 2006. (Cited on pages 52 and 53.)

[48] Coll, P., Marenco, J., Díaz, I. M., and Zabala, P. Facets of the graph coloring polytope. *Annals of Operations Research*, 116(1-4):79–90, 2002. (Cited on page 96.)

[49] Colorni, A., Dorigo, M., and Maniezzo, V. Metaheuristics for high school timetabling. *Computational optimization and applications*, 9(3):275–298, 1998. (Cited on page 93.)

[50] Cook, S. A. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158. ACM, New York, 1971. (Cited on page 6.)

[51] Corne, D., Ross, P., and Fang, H.-L. Evolutionary timetabling: Practice, prospects and work in progress. In *Proceedings of the UK Planning and Scheduling SIG Workshop*. Strahclyde, 1994. (Cited on page 92.)

[52] Cramer, G. *Introduction à l'analyse des lignes courbes algébriques*. Europeana: Bibliothèque de Genève, 1750. (Cited on page 19.)

[53] Dahlhaus, E., Horak, P., Miller, M., and Ryan, J. F. The train marshalling problem. *Discrete Applied Mathematics*, 103(1):41–54, 2000. (Cited on page 128.)

[54] Dahms, F. Train marshalling problems – algorithms and complexity. Diploma thesis, TU Kaiserslautern, 2010. (Cited on page 128.)

[55] Dantzig, G. B. Linear programming. *Operations Research*, 50(1):42–47, 2002. (Cited on page 15.)

[56] Dantzig, G. B. and Wolfe, P. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960. (Cited on page 41.)

[57] Dasgupta, S., Papadimitriou, C. H., and Vazirani, U. *Algorithms*. McGraw-Hill, Inc., New York, 2006. (Cited on page 71.)

[58] Daskalaki, S. and Birbas, T. Efficient solutions for a university timetabling problem through integer programming. *European Journal of Operational Research*, 160(1):106–120, 2005. (Cited on page 93.)

[59] de Haan, P., Landman, R., Post, G., and Ruizenaar, H. A case study for timetabling in a dutch secondary school. In Burke, E. and Rudová, H., editors, *Practice and Theory of Automated Timetabling VI*, volume 3867 of *Theoretical Computer Science and General Issues*, pages 267–279. Springer, Berlin, Heidelberg, 2007. (Cited on page 92.)

[60] de Werra, D. The combinatorics of timetabling. *European Journal of Operational Research*, 96(3):504–513, 1997. (Cited on page 91.)

[61] de Werra, D., Asratian, A. S., and Durand, S. Complexity of some special types of timetabling problems. *Journal of Scheduling*, 5(2):171–183, 2002. (Cited on page 91.)

[62] de Werra, D. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, 1985. (Cited on page 92.)

[63] Desaulniers, G., Desrosiers, J., and Solomon, M. M., editors. *Column Generation*. Springer, US, 1st edition, 2005. (Cited on page 41.)

[64] Desrochers, M. and Soumis, F. A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23(1):1–13, 1989. (Cited on page 149.)

[65] Edmonds, J. Submodular functions, matroids, and certain polyhedra. *Combinatorial Structures and Their Applications*, pages 69–87, 1970. (Cited on pages 12 and 13.)

[66] Edmonds, J. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3): 449–467, 1965. (Cited on pages 22 and 24.)

[67] Edmonds, J. and Fulkerson, D. R. Transversals and matroid partition. *Journal of Research of the National Bureau of Standards, Section B: Mathematics and Mathematical Physics*, 69B(3), 1965. (Cited on page 28.)

[68] Elmohamed, M. S., Coddington, P., and Fox, G. A comparison of annealing techniques for academic course scheduling. In Burke, E. and Carter, M., editors, *Practice and Theory of Automated Timetabling II*, pages 92–112. Springer, Berlin, Heidelberg, 1998. (Cited on page 93.)

[69] Farkas, J. Theorie der einfachen Ungleichungen. *Journal für die reine und angewandte Mathematik*, 124:1–27, 1902. (Cited on pages 11 and 15.)

[70] Fernández, C. and Santos, M. A non-standard genetic algorithm approach to solve constrained school timetabling problems. In Moreno Diaz, R. and Pichler, F., editors, *Computer Aided Systems Theory – EUROCAST 2003*, volume 2809 of *Lecture Notes in Computer Science*, pages 26–37. Springer, Berlin, Heidelberg, 2003. (Cited on page 92.)

[71] Ford, L. R. and Fulkerson, D. R. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956. (Cited on page 10.)

[72] Frangouli, H., Harmandas, V., and Stamatopoulos, P. UTSE: Construction of optimum timetables for university courses – a clp based approach. In Marien, A. and Roth, A., editors, *Proceedings of the Third International Conference on the Practical Applications of Prolog*. Alinmead Software Ltd, 1995. (Cited on page 93.)

[73] Fukunaga, A. S. A branch-and-bound algorithm for hard multiple knapsack problems. *Annals of Operations Research*, 184(1):97–119, 2011. (Cited on page 143.)

[74] Gale, D., Kuhn, H. W., and Tucker, A. W. Linear programming and the theory of games. *Activity analysis of production and allocation*, 13:317–335, 1951. (Cited on page 14.)

[75] Gamrath, G. and Lübbecke, M. E. Experiments with a generic Dantzig-Wolfe decomposition for integer programs. In Festa, P., editor, *Experimental algorithms*, volume 6049 of *Theoretical Computer Science and General Issues*, pages 239–252. Springer, Berlin, Heidelberg, 2010. (Cited on pages 141 and 150.)

[76] Gärdenfors, P. Match making: assignments based on bilateral preferences. *Behavioral Science*, 20(3):166–173, 1975. (Cited on page 36.)

[77] Garey, M. R. and Johnson, D. S. *Computer and intractability*. A Series of Books in the Mathematical Sciences. W. H. Freeman and Company, New York, 1979. (Cited on pages 5, 6, 24, 35, 73, and 134.)

[78] Gatto, M., Maue, J., Mihalák, M., and Widmayer, P. Shunting for dummies: An introductory algorithmic survey. In *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 310–337. Springer, Berlin Heidelberg, 2009. (Cited on page 127.)

[79] Ghouila-Houri, A. Caractérisation des matrices totalement unimodulaires. *Comptes Rendus de l'Académie des Sciences Paris*, 254:1192–1194, 1962. (Cited on page 20.)

[80] Gilmore, P. C. and Gomory, R. E. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961. (Cited on page 149.)

[81] Gislén, L., Peterson, C., and Söderberg, B. "teachers and classes" with neural networks. *International Journal of Neural Systems*, 1(02):167–176, 1989. (Cited on page 93.)

[82] Gorman, M. F. An application of genetic and tabu searches to the freight railroad operating plan problem. *Annals of operations research*, 78:51–69, 1998. (Cited on page 128.)

[83] Gotlieb, C. C. The construction of class-teacher time-tables. In Popplewell, C. M., editor, *IFIP Congress*, pages 73–77. North-Holland, Amsterdam, 1962. (Cited on page 93.)

[84] Güçlü, T. Ein Spaltengenerierungsansatz für die Zuordnung von Güterzügen. Master's thesis, RWTH Aachen University, 2012. (Cited on page 135.)

[85] Guéret, C., Jussien, N., Boizumault, P., and Prins, C. Building university timetables using constraint logic programming. In Burke, E. and Ross, P., editors, *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 130–145. Springer, Berlin, Heidelberg, 1996. (Cited on page 93.)

[86] Hall, N. G. and Posner, M. E. Generating experimental data for computational testing with machine scheduling applications. *Operations Research*, 49 (6):854–865, 2001. (Cited on page 148.)

[87] Hall, P. On the representation of subsets. *Journal of the London Mathematical Society*, 10:26–30, 1935. (Cited on page 26.)

[88] Hartog, J. *Timetabling on Dutch high schools*. PhD thesis, TU Delft, 2007. (Cited on page 92.)

[89] Harwood, G. B. and Lawless, R. W. Optimizing organizational goals in assigning faculty teaching schedules. *Decision sciences*, 6(3):513–524, 1975. (Cited on page 93.)

[90] Henz, M. and Würtz, J. Using Oz for college timetabling. In Burke, E. and Ross, P., editors, *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 162–177. Springer, Berlin, Heidelberg, 1996. (Cited on page 93.)

[91] Hertz, A. Tabu search for large scale timetabling problems. *European Journal of Operational Research*, 54(1):39–47, 1991. (Cited on page 93.)

[92] Hertz, A. Finding a feasible course schedule using tabu search. *Discrete Applied Mathematics*, 35(3):255–270, 1992. (Cited on page 93.)

[93] Hooker, J. N. and Osorio, M. A. Mixed logical-linear programming. *Discrete Applied Mathematics*, 96:395–442, 1999. (Cited on page 52.)

[94] Hooker, J. N. and Ottosson, G. Logic-based benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003. (Cited on page 52.)

[95] Huntley, C. L., Brown, D. E., Sappington, D. E., and Markowicz, B. P. Freight routing and scheduling at CSX transportation. *Interfaces*, 25(3):58–71, 1995. (Cited on page 128.)

[96] Jacobsen, F., Bortfeldt, A., and Gehring, H. Timetabling at German secondary schools: tabu search versus constraint programming. In Burke, E. and Rudová, H., editors, *International conference on the practise and theory of automated timetabling*, volume 3867 of *Theoretical Computer Science and General Issues*. Springer, Berlin, Heidelberg, 2006. (Cited on pages 92 and 93.)

[97] Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., Rinaldi, G., and Wolsey, L. A., editors. *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-art*. Springer, Berlin, Heidelberg, 1st edition, 2009. (Cited on pages 42, 53, 70, and 78.)

[98] Junginger, W. Timetabling in Germany – a survey. *Interfaces*, 16(4):66–74, 1986. (Cited on page 92.)

[99] Karmarkar, N. A new polynomial-time algorithm for linear programming. In DeMillo, R., editor, *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, New York, 1984. (Cited on page 16.)

[100] Karp, R. M. *Reducibility among combinatorial problems*. Springer, US, 1972. (Cited on pages 33, 73, and 95.)

[101] Keaton, M. H. Designing railroad operating plans: A dual adjustment method for implementing lagrangian relaxation. *Transportation science*, 26 (4):263–279, 1992. (Cited on page 128.)

[102] Khachiyan, L. G. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980. (Cited on page 16.)

[103] Klee, V. and Minty, G. J. How good is the simplex method. *Inequalities-III*, pages 159–175, 1972. (Cited on page 15.)

[104] Kőnig, D. Theorie der endlichen und unendlichen Graphen. *Akademische Verlagsgesellschaft, Leipzig*, 1936, 1936. (Cited on pages 24 and 26.)

[105] Kwok, L.-F., Kong, S.-C., and Kam, Y.-Y. Timetabling in Hong Kong secondary schools. *Computers & Education*, 28(3):173–183, 1997. (Cited on page 92.)

[106] Lach, G. and Lübbecke, M. Curriculum based course timetabling: Optimal solutions to the udine benchmark instances. In Burke, E. and Gendreau, M., editors, *Proceedings of the 7th PATAT Conference*, volume 194 of *Annals of Operations Research*, 2008. (Cited on pages 91 and 93.)

[107] Lach, G. and Lübbecke, M. E. Optimal university course timetables and the partial transversal polytope. In McGeoch, C. C., editor, *Experimental Algorithms*, volume 5038 of *Theoretical Computer Science and General Issues*, pages 235–248. Springer, Berlin, Heidelberg, 2008. (Cited on pages 91, 93, and 103.)

[108] Lawrie, N. L. An integer linear programming model of a school timetabling problem. *The Computer Journal*, 12(4):307–316, 1969. (Cited on page 93.)

[109] Le Gall, F. Powers of tensors and fast matrix multiplication. In Nabeshima, K., editor, *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, pages 296–303. ACM, New York, 2014. (Cited on page 22.)

[110] Legierski, W. Search strategy for constraint-based class–teacher timetabling. In Burke, E. and De Causmaecker, P., editors, *Practice and Theory of Automated Timetabling IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 247–261. Springer, Berlin, Heidelberg, 2003. (Cited on page 93.)

[111] Lewis, R. A survey of metaheuristic-based techniques for university timetabling problems. *OR spectrum*, 30(1):167–190, 2008. (Cited on pages 92 and 93.)

[112] Lodi, A. and Tramontani, A. Performance variability in mixed-integer programming. *TutORials in Operations Research: Theory Driven by Influential Applications*, pages 1–12, 2013. (Cited on pages 100 and 150.)

[113] Lovász, L. and Plummer, M. *Matching theory*, volume 367. AMS, Chelsea Publishing, 2009. (Cited on pages 23 and 104.)

[114] Marchand, H., Martin, A., Weismantel, R., and Wolsey, L. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123 (1):397–446, 2002. (Cited on page 17.)

[115] Marte, M. *Models and algorithms for school timetabling – a constraint programming approach*. PhD thesis, LMU München, 2002. (Cited on page 92.)

[116] Martello, S. and Toth, P. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., New Jersey, 1990. (Cited on page 143.)

[117] Megiddo, N. On finding primal-and dual-optimal bases. *ORSA Journal on Computing*, 3(1):63–65, 1991. (Cited on page 16.)

[118] Mehrotra, A. and Trick, M. A. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996. (Cited on page 145.)

[119] Merlot, L. *Techniques for academic timetabling*. PhD thesis, University of Melbourne, 2005. (Cited on page 92.)

[120] Micali, S. and Vazirani, V. V. An $O(\sqrt{|v|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science*, pages 17–27. IEEE, New Jersey, 1980. (Cited on page 22.)

[121] Minkowski, H. *Geometrie der Zahlen*. Teubner, Leipzig, 1910. (Cited on pages 11 and 12.)

[122] Motzkin, T. S. *Beiträge zur Theorie der linearen Ungleichungen*. Azriel, Jerusalem, 1936. (Cited on page 11.)

[123] Mucha, M. and Sankowski, P. Maximum matchings via gaussian elimination. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 248–255. IEEE, Washington, DC, 2004. (Cited on page 22.)

[124] Nemani, A. K. and Ahuja, R. K. OR models in freight railroad industry. In Cochran, J. J., Cox, L. A., Keskinocak, P., Kharoufeh, J. P., and Smith, J. C., editors, *Wiley Encyclopedia of Oper. Res. and Management Sci.* John Wiley & Sons, Inc., 2011. (Cited on page 127.)

[125] Nemhauser, G. L. and Wolsey, L. A. *Integer and combinatorial optimization*, volume 18. Wiley, New York, 1988. (Cited on pages 11 and 15.)

[126] Newton, H. N., Barnhart, C., and Vance, P. H. Constructing railroad blocking plans to minimize handling costs. *Transportation Science*, 32(4):330–345, 1998. (Cited on page 128.)

[127] Orlin, J. B. Max flows in $o(nm)$ time, or better. In Boneh, D., Roughgarden, T., and Feigenbaum, J., editors, *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 765–774. ACM, New York, 2013. (Cited on pages 10 and 29.)

[128] Oxley, J. G. *Matroid theory*, volume 1997. Oxford University Press, Oxford, 1992. (Cited on page 12.)

[129] Palubeckis, G. On the graph coloring polytope. *Information technology and control*, 37:7–11, 2008. (Cited on page 96.)

[130] Papoutsis, K., Valouxis, C., and Housos, E. A column generation approach for the timetabling problem of Greek high schools. *Journal of the Operational Research Society*, 54(3):230–238, 2003. (Cited on pages 92 and 93.)

[131] Petrovic, S. and Burke, E. K. University timetabling. *Handbook of scheduling: algorithms, models, and performance analysis*, 45:1–23, 2004. (Cited on page 92.)

[132] Pisinger, D. An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114(3):528–541, 1999. (Cited on page 143.)

[133] Post, G., Kingston, J. H., Ahmadi, S., Daskalaki, S., Gogos, C., Kyngas, J., Nurmi, C., Musliu, N., Pillay, N., Santos, H., et al. XHSTT: an XML archive for high school timetabling problems in different countries. *Annals of Operations Research*, 218(1):295–301, 2014. (Cited on page 92.)

[134] Qi, N. On separation and adjacency problems for perfectly matchable subgraph polytopes of a graph. *Operations research letters*, 6(5):239–241, 1987. (Cited on page 29.)

[135] Qu, R., Burke, E. K., McCollum, B., Merlot, L. T., and Lee, S. Y. A survey of search methodologies and automated system development for examination timetabling. *Journal of scheduling*, 12(1):55–89, 2009. (Cited on page 93.)

[136] Qualizza, A. and Serafini, P. A column generation scheme for faculty timetabling. In Burke, E. and Trick, M., editors, *Practice and Theory of Automated Timetabling V*, volume 3616 of *Lecture Notes in Computer Science*, pages 161–173. Springer, Berlin, Heidelberg, 2005. (Cited on page 93.)

[137] Raghavjee, R. and Pillay, N. An informed genetic algorithm for the high school timetabling problem. In Kotzé, P., van der Merwe, A., and Gerber, A., editors, *Proceedings of the 2010 Annual Research Conference of the South African*

Institute of Computer Scientists and Information Technologists, pages 408–412. ACM, New York, 2010. (Cited on page 92.)

[138] Rossi-Doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L. M., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B., Paquete, L., and Stützle, T. A comparison of the performance of different metaheuristics on the timetabling problem. In Burke, E. and De Causmaecker, P., editors, *Practice and Theory of Automated Timetabling IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 329–351. Springer, Berlin, Heidelberg, 2003. (Cited on page 93.)

[139] Ryan, D. M. and Foster, B. A. An integer programming approach to scheduling. *Computer scheduling of public transport urban passenger vehicle and crew scheduling*, pages 269–280, 1981. (Cited on pages 77, 78, 82, and 149.)

[140] Santos, H. G., Ochi, L. S., and Souza, M. J. A tabu search heuristic with efficient diversification strategies for the class/teacher timetabling problem. *Journal of Experimental Algorithmics (JEA)*, 10:2–9, 2005. (Cited on page 93.)

[141] Santos, H. G., Uchoa, E., Ochi, L. S., and Maculan, N. Strong bounds with cut and column generation for class-teacher timetabling. *Annals of Operations Research*, 194(1):399–412, 2012. (Cited on page 93.)

[142] Schaerf, A. Local search techniques for large high school timetabling problems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 29(4):368–377, 1999. (Cited on page 92.)

[143] Schaerf, A. A survey of automated timetabling. *Artificial intelligence review*, 13(2):87–127, 1999. (Cited on page 92.)

[144] Scheel, O. Structure and characterization of popular matchings. Bachelor's thesis, RWTH Aachen University, 2014. (Cited on pages 36, 38, and 39.)

[145] Schmidt, G. and Ströhlein, T. Timetable construction–an annotated bibliography. *The Computer Journal*, 23(4):307–316, 1980. (Cited on page 92.)

[146] Schrijver, A. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester, 1986. (Cited on pages 10, 15, and 20.)

[147] Schrijver, A. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24. Springer, Berlin, Heidelberg, 1st edition, 2003. (Cited on pages 12, 13, 24, 25, 26, and 27.)

[148] Socha, K., Sampels, M., and Manfrin, M. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In Esparcia-Alcázar, A. I., editor, *Applications of Evolutionary Computation*, volume 7835 of *Lecture Notes in Computer Science*, pages 334–345. Springer, 2003. (Cited on page 92.)

[149] Sørensen, M. and Dahms, F. H. W. A two-stage decomposition of high school timetabling applied to cases in Denmark. *Computers & Operations Research*, 43:36–49, 2014. (Cited on pages v, 91, 92, 93, 96, 103, 104, 105, 109, 112, 113, and 157.)

[150] Sørensen, M. and Stidsen, T. R. Comparing solution approaches for a complete model of high school timetabling. Technical report, Department of Management Engineering, Technical University of Denmark, 2013. (Cited on pages 92 and 96.)

[151] Steinitz, E. Bedingt konvergente Reihen und konvexe Systeme.(Schluß.). *Journal für die reine und angewandte Mathematik*, 146:1–52, 1916. (Cited on page 12.)

[152] Tarjan, R. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972. (Cited on page 9.)

[153] ten Eikelder, H. M. and Willemen, R. Some complexity aspects of secondary school timetabling problems. In Burke, E. and Erben, W., editors, *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pages 18–27. Springer, Berlin, Heidelberg, 2001. (Cited on page 91.)

[154] Thompson, J. and Dowsland, K. A. General cooling schedules for a simulated annealing based timetabling system. In Burke, E. and Ross, P., editors, *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 345–363. Springer, Berlin, Heidelberg, 1996. (Cited on page 93.)

[155] Tripathy, A. School timetabling – a case in large binary integer linear programming. *Management Science*, 30(12):1473–1489, 1984. (Cited on page 93.)

[156] Ueda, H., Ouchi, D., Takahashi, K., and Miyahara, T. Comparisons of genetic algorithms for timetabling problems. *Systems and Computers in Japan*, 35(7): 1–12, 2004. (Cited on page 92.)

[157] Unlu, Y. and Mason, S. J. Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. *Computers & Industrial Engineering*, 58(4):785–800, 2010. (Cited on page 146.)

[158] Valouxis, C. and Housos, E. Constraint programming approach for school timetabling. *Computers & Operations Research*, 30(10):1555–1572, 2003. (Cited on page 93.)

[159] Van Dyke, C. D. The automated blocking model: A practical approach to freight railroad blocking plan development. In *Transportation Research Forum*, volume 27, pages 116–121, 1986. (Cited on page 128.)

[160] Vanderbeck, F. Branching in branch-and-price: a generic scheme. *Mathematical Programming*, 130(2):249–294, 2011. (Cited on pages 72, 78, 79, and 83.)

[161] Vanderbeck, F. and Savelsbergh, M. W. A generic view of Dantzig–Wolfe decomposition in mixed integer programming. *Operations Research Letters*, 34(3):296–306, 2006. (Cited on page 47.)

[162] Vizing, V. G. Vertex colorings with given colors. *Metody Diskretnogo Analiza*, 29:3–10, 1976. (Cited on page 144.)

[163] Weyl, H. Elementare Theorie der konvexen Polyeder. *Commentarii Mathematici Helvetici*, 7(1):290–306, 1934. (Cited on pages 11 and 12.)

[164] Whitney, H. On the abstract properties of linear dependence. *American Journal of Mathematics*, pages 509–533, 1935. (Cited on page 12.)

[165] Willemen, R. *School timetable construction: algorithms and comlexity*. PhD thesis, TU Eindhoven, 2002. (Cited on page 92.)

[166] Wren, A. Scheduling, timetabling and rostering – a special relationship? In Burke, E. and Ross, P., editors, *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 46–75. Springer, Berlin, Heidelberg, 1996. (Cited on page 91.)

[167] Zhang, D., Liu, Y., M'Hallah, R., and Leung, S. C. A simulated annealing with a new neighborhood structure based algorithm for high school timetabling problems. *European Journal of Operational Research*, 203(3):550–558, 2010. (Cited on page 93.)

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*. `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis