Markó Horváth

Integer programming approaches for solving routing and scheduling problems

PhD thesis

Supervisor: Tamás Kis, DSc

scientific advisor, Institute for Computer Science and Control
assistant professor, Department of Operations Research

Doctoral School of Mathematics

director: Professor Tibor Jordán, DSc

Doctoral Program of Applied Mathematics

director: Professor János Karátson, DSc

Department of Operations Research
Faculty of Science, Eötvös Loránd University

2019

"Egyszerű élet ez, de mégsem nyugodt, csendes, mint némely gondolná:
folytonos küzdés, melyben én voltam a gyengébb fél."
Arany János

Acknowledgment

First of all, I would like to express my gratitude to my supervisor Tamás Kis. Tamás was my supervisor also in MSc thesis, and he invited me to work together at SZTAKI. I am grateful to him for his time, patience, advice and guidance in our work.

I am thankful to my teachers at the university for treating me as a colleague from the beginning, to the Research Laboratory on Engineering & Management Intelligence for providing me with everything I need at a workplace, and to my colleagues for making workdays colorful.

Last but not least, I cannot be grateful enough to my family and friends for all their help and support.

My research was supported by the GINOP-2.3.2-15-2016-00002 grant on an "Industry 4.0 research and innovation center of excellence", and by the Young Researcher Scholarship of the Hungarian Academy of Sciences.

Contents

Chapter 1

Introduction

Even if we do not realize, almost all of us are facing combinatorial optimization problems in our daily life, i.e., problems where there are several possible solutions and we have to choose the best or a fair enough solution in some respects. For example, when we want to get somewhere using public transport we may have several possible routes but we want to choose the fastest one (or the one with the fewest transfers, or the one with the least walking, etc.). The public transport company also faces combinatorial optimization problems. For example, on the operational planning phase they have to create (i) daily schedules for the vehicles of their fleet to perform the timetabled trips, minimizing some assets and operational costs; (ii) daily shifts and (iii) rosters over a longer planning period (e.g., weeks or months) for the drivers, satisfying a wide variety of federal regulations, minimizing the labor costs. Although our planning problem can be easily handled and there are also several applications to support our decisions, the problems arising in the public transport company require complex systems containing suitable mathematical models and efficient algorithms to make them computationally tractable.

Today's industries are also teeming with combinatorial optimization problems. For example, machine scheduling problems permanently arise in a company producing or assembling some products. Such a problem involves sequencing a set of jobs (e.g., manufacturing operations of a given product) to be processed in a set of resources. Without a claim to completeness, we mention that these problems have different flow patterns (e.g., single machine, parallel machines, shop models), constraints (e.g., precedence constraints, setup times, release dates, no-wait operations) and objectives (minimizing costs, penalties, makespan, or increasing production throughput) based on the particularity of the companies and the products. Again, it requires adaptable models and algorithms to take more and more details of the production process into consideration while keeping the problem computationally tractable.

Exact algorithms or heuristics?

Facing a hard problem, we can use two types of solution approaches: exact algorithms and heuristics. Exact algorithms are guaranteed to find an optimal solution, however, it may require plenty of time. In contrast, heuristics does not guarantee optimality, however, finds a fair enough solution in a shorter execution time. So there is a trade-off between these approaches we need to deal with. Although everybody desires an optimal solution to their problem, since it minimizes the costs or maximizes the profit, we may not insist on such a solution within a reasonable running time due to the complexity of the problem. For example, in case of vehicle and crew scheduling problems the exact solution of an instance taken from a middle-sized city (e.g., there are 2763 timetabled trips on an average workday in Szeged, Hungary) is not possible (Árgilán et al., 2010).

Despite all the difficulties mentioned so far, for all the problems investigated in this thesis we provide exact algorithms. In addition, for one of these problems we also present an approximation algorithm, i.e., an algorithm which guarantees that its returned solution is within a multiplicative factor of the optimal solution. In all these exact algorithms, integer programming approaches play a key role.

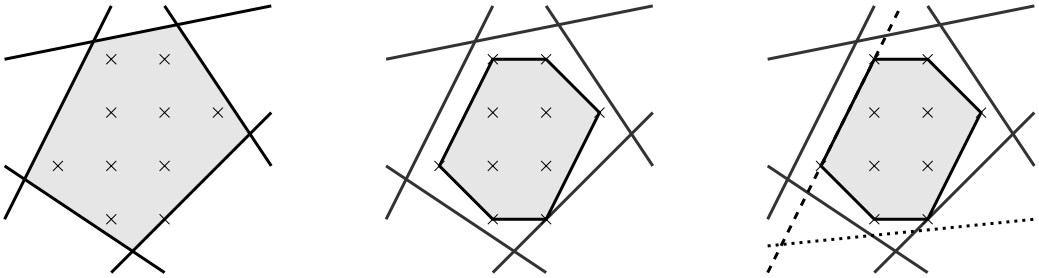
Integer programming approaches

A well-established technique to create an exact solution approach is to formulate the given problem as an integer linear program (ILP) and solve it with a branch-and-bound type algorithm. Such a formulation consists of integer variables (i.e., variables that should take integer values in the solution), linear constraints (or inequalities) and a linear objective function such that the feasible solutions of the formulation represent the feasible solutions of the problem. To solve an ILP problem one can adapt a branch-and-bound procedure which splits up the problem into smaller subproblems and discards those that are not promising an optimal solution for the original problem.

Although several softwares have been developed to solve ILP problems, these are usually struggling on hard problems that we consider in this thesis. One of the main reasons for this is that these are general solvers and have no insight into the problem except the initial formulation. Thus, in order to make these problems manageable, we need to understand the structure of the problem and utilize this knowledge to improve either the initial ILP formulation or the solution procedure. The former task leads us to polyhedral investigations.

Polyhedral investigations

The linear inequalities of an ILP formulation determine a polyhedron in a multi-dimensional space, called the LP-relaxed polyhedron, and the integer points (i.e., points with integer coordinates) in this polyhedron represent the feasible solutions of the problem, see Fig. 1.1a. These integer points (under certain conditions) also determine a polyhedron, called the polytope of



(a) The polyhedron determined by linear constraints and its integer points

(b) The polytope of the integer points

(c) A valid (dotted) and a facet-defining (dashed) inequality for the polytope

Figure 1.1: 2-dimensional example for the polyhedron associated with an integer linear programming formulation

feasible solutions, see Fig. 1.1b, and basically the tighter an LP-relaxed polyhedron is the better the corresponding ILP formulation for the problem is. So, our purpose is to find constraints that make the formulation tighter, called valid inequalities for the polytope of feasible solutions, see Fig. 1.1c. The greatest achievement would be finding the complete description of that polytope, i.e., its determining (so-called facet-defining) inequalities, see Fig. 1.1c, since in that case we had the tightest ILP formulation for the problem. The bad news, besides that finding this description is generally a challenging task, is that such a description could be exponential size which makes again the problem computationally intractable. The good news is that if we can identify a class of valid inequalities we can improve a branch-and-bound procedure by adding these inequalities gradually to the formulation, if needed.

Problems investigated in the thesis

In this thesis we deal with three different combinatorial optimization problems presenting our theoretical results, and our solution approaches along with the computational results of our thorough experiments.

Resource constrained shortest path problem. In Chapters 3 and 4 we consider the resource constrained shortest path problem where the goal is to find the shortest path between two points in a given network such that certain resource consumptions along the path do not exceed the given limits. This problem and its variants have several direct real-world applications, e.g., finding a sequence of treatment processes that reduces the concentrations of the pollutants to acceptable levels (Elimam and Kohler, 1997), designing a thermally efficient composite wall or roof structure (Elimam and Kohler, 1997), calculating an aircraft's optimal risk trajectory (Zabarankin et al., 2002), locating a pipeline between two locations with safety valves (Laporte et al., 2012). The problem and its variants also arise in other optimization problems as a subproblem to be solved repeatedly, e.g., crew scheduling and rostering problems (Desrochers

and Soumis, 1989; Gamache et al., 1999), various vehicle routing problems (Chabrier, 2006; Derigs et al., 2009), maximizing profit over the life cycle of a family of products by prescribing the content and timing of upgrades (Wilhelm et al., 2003), optimizing the placement operations of a dual-head placement machine (Wilhelm et al., 2007). In Chapter 3 we investigate approximation schemes for the problem, where the resource limits can be slightly violated. In Chapter 4 we propose linear programming based exact solution algorithms for the problem.

Position-based scheduling on a single machine. As we mentioned before, machine scheduling has a great practical relevance, and thus the problem has a large number of variants. Hence, we do not attempt to enumerate any application of these problems, however, for a tertiary analysis we refer to (Abedinnia et al., 2017). In Chapter 5 we consider and give an exact solution approach to a machine scheduling problem where the set of jobs has to be sequenced on a single machine.

Multiple-depot integrated vehicle and crew scheduling problem. As we mentioned before, the vehicle scheduling and the crew scheduling problems are two main planning problems that arise in public transportation. Several solution approaches have been proposed to solve the variants of these problems which are either tested on real-life instances or integrated to an information system, e.g., the decision support system (GIST) of Portuguese transport companies (Dias et al., 2002), the public transport company (Connexxion) in the Netherlands (Huisman et al., 2004), the public transport company (Ljubljanski potniški promet) in Ljubljana, Republic of Slovenia (Békési et al., 2009), the bus company (Tisza Volán) in Szeged, Hungary (Békési et al., 2009; Árgilán et al., 2010), the mass transit system (MIO) in Cali, Colombia (Baldoquin and Rengifo-Campo, 2018). In Chapter 6 we consider and give an exact solution approach to the integrated vehicle and crew scheduling problem, where the vehicles and the crew are scheduled simultaneously, which makes the problem more complex.

Chapter 2

Terminology, notation

2.1 Basics

We denote with $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$ the set of integer, rational, and real numbers, respectively. For a set $S \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$ of numbers, the set of positive and non-negative elements of S are denoted with $S_{>0} := \{e \in S : e > 0\}$ and $S_{\geq 0} := \{e \in S : e \geq 0\}$, respectively. Analogously, we denote with $\mathbb{Z}^d$, $\mathbb{Q}^d$, and $\mathbb{R}^d$ the set of d-dimensional vectors ($d \in \mathbb{Z}_{>0}$) with integer, rational, and real coefficients, respectively, and for a set $S \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$ of numbers, the set of d-dimensional vectors with positive and non-negative coefficients are denote with $S^d_{>0}$ and $S^d_{\geq 0}$, respectively.

2.2 Graphs

2.2.1 Undirected graphs

An *undirected graph* is an ordered pair (V, E) where V is a set of elements, called *nodes*, and E is a set of unordered pairs of nodes, called *edges*. If $\{u, v\} \in E$, we say that nodes u and v are *adjacent*, and for an edge $e = \{u, v\} \in E$ we say that e is incident to node u and node v.

A subset of nodes is a *vertex cover* if each edge of the graph is incident to at least one node of this subset. A subset of nodes is a *clique* if every two distinct nodes of this subset are adjacent. A subset of nodes is *independent* if no two nodes of this subset are adjacent.

2.2.2 Directed graphs

A *directed graph* is an ordered pair (V, A) where V is a set of elements, called *nodes*, and A is a set of ordered pairs of nodes, called *directed edges* or *arcs*. For an arc $e = (u, v)$, u is called the *tail* of e and v is called the *head* of e, and denoted with $\text{tail}(e)$ and $\text{head}(e)$, respectively.

For a node v let $\delta^{in}(v) := \{e \in A : \text{head}(e) = v\} = \{(v_1, v_2) \in A : v_2 = v\}$ be the set of *incoming arcs of* v, and $\delta^{out}(v) := \{e \in A : \text{tail}(e) = v\} = \{(v_1, v_2) \in A : v_1 = v\}$ be the set of *outgoing arcs of* v. For a non-empty set of nodes $S \subset V$ let $\delta^{out}(S) := \{(u, v) \in A : u \in S \text{ and } v \notin S\}$ be the set of *outgoing arcs of* S, and $\delta^{in}(S) := \{(u, v) \in A : u \notin S \text{ and } v \in S\}$ be

the set of *incoming arcs of* S. For a set of nodes $S \subseteq V$ let $\gamma(S) := \{(u,v) \in A : u \in S \text{ and } v \in S\}$ denote the set of arcs *spanned by* S. Sometimes, to avoid ambiguity, we use notation $\delta^{in}_D(v)$, $\delta^{out}_D(v)$, $\delta^{in}_D(S)$, $\delta^{out}_D(S)$, and $\gamma_D(S)$ to indicate the underlying graph $D = (V, A)$.

A sequence $(e_1, \dots, e_\ell)$ of arcs is called a *walk*, if $\text{head}(e_i) = \text{tail}(e_{i+1})$ for all $i = 1, \dots, \ell - 1$. A walk $(e_1, \dots, e_\ell)$ is a *path*, if nodes $\text{tail}(e_1), \text{head}(e_1), \text{head}(e_2), \dots, \text{head}(e_\ell)$ are pairwise distinct. A walk $(e_1, \dots, e_\ell)$ is a *cycle*, if nodes $\text{head}(e_1), \text{head}(e_2), \dots, \text{head}(e_\ell)$ are pairwise distinct and $\text{head}(e_\ell) = \text{tail}(e_1)$. A directed graph without any cycle is called *acyclic*. For distinct nodes $s, t \in V$, a path $(e_1, \dots, e_\ell)$ is called an *s-t path*, if $s = \text{tail}(e_1)$ and $t = \text{head}(e_\ell)$. The set of all *s-t* paths in a given graph is denoted with $\mathcal{P}_{st}$.

A node u is *reachable from node* v if D contains a *v-u* path. The set of nodes reachable from node v is denoted with $\rho^{out}(v)$, and symmetrically, let $\rho^{in}(v)$ be the set of nodes from which node v can be reached. Note that $v \in \rho^{out}(v)$ and $v \in \rho^{in}(v)$ for each node v. For a node $v \in V$, let $D[v] := (\rho^{in}(v) \cup \rho^{out}(v), \gamma(\rho^{in}(v)) \cup \gamma(\rho^{out}(v)))$. Similarly, for an arc $e = (u,v) \in A$, let $D[e] := (\rho^{in}(u) \cup \rho^{out}(v), \gamma(\rho^{in}(u)) \cup \{e\} \cup \gamma(\rho^{out}(v)))$. Clearly, π is an *s-t* path in D containing node v (arc e) only if π is an *s-t* path in $D[v]$ (in $D[e]$).

A *cut* is a partition of the nodes into two disjoint subsets X and $\bar{X}$, where $\emptyset \neq X \subset V$ and $\bar{X} = V \setminus X$. For distinct nodes $s, t \in V$ a cut $(X, \bar{X})$ is called an *s-t cut* if $s \in X$ and $t \in \bar{X}$.

For a directed graph $D = (V, A)$ and arc $e \in A$, let $D \setminus e$ denote the directed graph obtained from D by deleting arc e, i.e., $D \setminus e := (V, A \setminus \{e\})$. We call an arc $e = (u,v)$ *transitive* in D, if there exists an *u-v* path in $D \setminus e$. A directed, acyclic graph without any transitive arc is called a *precedence graph*.

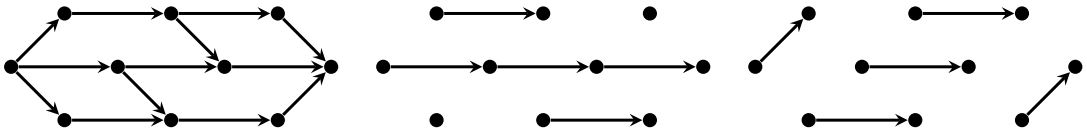
2.3 Machine scheduling

A set $\{J_1, \dots, J_n\}$ of *jobs* has to be processed on a set $\{M_1, \dots, M_m\}$ of *machines*. Each machine can process at most one job at a time and each job can be processed on at most one machine at a time. To reflect various job, machine and scheduling characteristics we use the 3-field notation $\alpha \,|\, \beta \,|\, \gamma$ introduced by Graham et al. (1979), where the first field, α, specifies the machine environment; the second field, β, indicates additional restrictions (e.g., processing time, precedence constraints, etc.); and the third field, γ, indicates the optimality criterion.

Unless otherwise stated, we assume that each job has to be processed on exactly one machine. A single-machine problem is denoted by $\alpha = 1$, i.e., we write $1 \,|\, \beta \,|\, \gamma$. For multiple-machine problems $\alpha = P$ refers to case of identical parallel machines, where the processing time of a job (i.e., the amount of time required to process the job) does not depend on the machines. In the case of $\alpha = Pm$ the number of machines, m, is given a priori.

For each job J_j we denote with p_{ji} its processing time related to machine M_i. In the case of a single machine or parallel identical machines we could simply use p_j. The notation $\alpha \,|\, p_j = 1 \,|\, \gamma$ refers to the case of *unit-time jobs*, that is, for each job J_j we have $p_j = 1$.

We denote with *prec* the presence of *precedence constraints*, that is, when a precedence



(a) arbitrary precedences (b) chain precedences (c) 2-chain precedences

Figure 2.1: Examples for precedence constraints

relation between the jobs is specified. Such a relation is derived from a precedence graph with nodes corresponding to jobs; if there is a directed path from J_i to J_j ($J_i \neq J_j$) then job J_j can be started only if job J_i is already processed. In this case, we write $J_i \prec\prec J_j$, and job J_i is called a *predecessor* of job J_j and job J_j is called a *successor* of job J_i. Moreover, if (J_i, J_j) is an arc in the precedence graph, we say that job J_i is an *immediate predecessor* of job J_j and job J_j is an *immediate successor* of job J_i, and we write $J_i \prec J_j$. We illustrate some special cases of precedence constraints in Fig. 2.1. In the case of *chain precedence constraints* (denoted by *chains*) each job has at most one immediate predecessor and at most one immediate successor, moreover, if each job has either exactly one immediate predecessor or exactly one immediate successor but not both (i.e., each chain consists of exactly two jobs) we have a *2-chain precedence* constraint (denoted by 2-*chains*).

For a given schedule S let C_j^S denote the completion time of job J_j. The *makespan* of some schedule S is the maximum of the job completion times, i.e., $C_{\max}^S := \max_j C_j^S$. If a due-date d_j is given for each job J_j, then the *tardiness* of the job is $T_j^S := \max\{0, C_j^S - d_j\}$, while U_j^S indicates if the job is late, i.e., $U_j^S = 1$, if $C_j^S > d_j$, and 0 otherwise. The jobs may also have some non-negative weight w_j. The optimality criterion for minimizing the makespan, the sum of completion times, the weighted sum of completion times, the total tardiness and the number of tardy jobs is denoted by $C_{\max}$, $\sum C_j$, $\sum w_j C_j$, $\sum T_j$, and $\sum U_j$, respectively.

2.4 Combinatorial optimization problems

An instance of the *combinatorial optimization problem* is given by a set $\mathcal{S}$ of feasible solutions along with an objective function $c : \mathcal{S} \to \mathbb{R}$, and the problem is the following:

$$\text{minimize (or maximize) } c(S) \text{ subject to } S \in \mathcal{S}. \tag{2.1}$$

The budgeted version of a combinatorial optimization problem (2.1) has an additional set of k weight functions on the feasible solutions, that is, $\mathbf{w} : \mathcal{S} \to \mathbb{Q}^k$, and there is a limit $\mathbf{L} \in \mathbb{Q}^k$ on the total budget allowed. The *k-budgeted optimization problem* is formulated as

$$\text{minimize (or maximize) } c(S) \text{ subject to } S \in \mathcal{S}, \text{ and } \mathbf{w}_i(S) \leq \mathbf{L}_i \text{ for all } i = 1, \ldots, k. \tag{2.2}$$

Note that problem (2.2) is not more general then problem (2.1), since it can be written in the form: $\min / \max\{c(S) : S \in \mathcal{S}'\}$, where $\mathcal{S}' = \{S \in \mathcal{S} : \mathbf{w}_i(S) \leq \mathbf{L}_i \text{ for all } i = 1, \ldots, k\}$.

However, the form of (2.2) will be useful for us, because in some cases we will relax the budget limits.

2.4.1 Approximation

An *α-approximation algorithm*, $\alpha > 1$, for an optimization problem Π is a polynomial time algorithm which finds an α-approximate solution S for Π, that is, $c(S) \geq c(S_{OPT})/\alpha$ if Π is a maximization problem, and $c(S) \leq \alpha c(S_{OPT})$ if Π is a minimization one, where S_{OPT} is an optimal solution. A *polynomial time approximation scheme* (PTAS) for an optimization problem Π is a family of approximation algorithms $\{A_\varepsilon\}_{\varepsilon>0}$ such that A_ε is an $(1+\varepsilon)$-approximation algorithm for Π for any $\varepsilon > 0$. A *fully polynomial time approximation scheme* (FPTAS) for Π is a PTAS with $\{A_\varepsilon\}_{\varepsilon>0}$ such that A_ε runs in polynomial time in $1/\varepsilon$ as well.

2.4.2 Examples

In the following we introduce some cases of the combinatorial optimization problem, and provide some results which will be useful for the rest of this thesis. Note that machine scheduling problems proposed in Section 2.3 are also combinatorial optimization problems where the set of feasible solutions consists of the schedules satisfying the corresponding restrictions, and the cost of such a schedule is given by the corresponding optimality criterion.

In the following examples given a set U of elements along with costs $c : U \to \mathbb{R}$, the feasible solutions are certain subsets of the elements, i.e., $\mathcal{S} \subseteq 2^U$, and the cost of a feasible solution $S \in \mathcal{S}$ is defined by $c(S) := \sum_{e\in S} c(e)$. In the case of budgeted version, there are k weight functions on the elements, $\mathbf{w} : U \to \mathbb{Q}^k$, and the weight of a feasible solution $S \in \mathcal{S}$ is defined by $\mathbf{w}_i(S) := \sum_{e\in S} \mathbf{w}_i(e)$ for all $i = 1, \ldots, k$.

Shortest Path Problem. In the Shortest Path Problem (SPP) given a directed graph with two designated nodes, each arc has a length, and a path with minimum total length is sought between the designated nodes, that is, SPP refers to the minimization version of problem (2.1), where $\mathcal{S} = \mathcal{P}_{st}$, i.e., the set of all s-t paths of a directed graph $D = (V, U)$ with designated nodes $s, t \in V$.

Resource Constrained Shortest Path Problem. The Resource Constrained Shortest Path Problem (RCSPP) corresponds to the budgeted version of SPP. Dror (1994) shows that RCSPP is NP-hard hard in the strong sense for graphs containing negative cost cycles, however, the problem remains NP-hard even if the graph is acyclic, there is a single weight function, and all the arc weights and costs are non-negative (cf. (Garey and Johnson, 1979, problem ND30)).

Matroid Basis Problem. The Matroid Basis Problem refers to the problem (2.1), where $\mathcal{S}$ is the set of bases of a given matroid with ground set U. Specially, if $\mathcal{S}$ is the set of all spanning

trees of a given undirected graph $G = (V, U)$ then problem (2.1) refers to the Spanning Tree Problem.

Matching problems. In the case of (Bipartite) (Perfect) Matching Problem the solution set $\mathcal{S}$ consists of the (perfect) matchings of an undirected (bipartite) graph $G = (V, U)$.

Vertex Cover Problem. In the Vertex Cover Problem (VCP) a vertex cover with minimum cardinality is sought, that is, VCP refers to the minimization version of problem (2.1), where $\mathcal{S}$ is the set of all vertex covers of an undirected graph $G = (U, E)$, and $c(v) = 1$ for all $v \in U$. Dinur and Safra (2005) show that it is NP-hard to approximate VCP to within any factor smaller than $10\sqrt{5} - 21 \approx 1.3606$.

Clique Problem. In the Clique Problem (ClP) a clique with maximum cardinality is sought for a given graph, that is, ClP refers to the maximization version of problem (2.1), where $\mathcal{S}$ is the set of all cliques of an undirected graph $G = (U, E)$, and $c(v) = 1$ for all $v \in U$. Hastad (1996) shows that ClP is not approximable within $|V|^{1/2-\varepsilon}$ for any $\varepsilon > 0$.

Independent Set Problem. In the Independent Set Problem (ISP) an independent set with maximum cardinality is sought for a given graph, that is, ISP refers to the maximization version of problem (2.1), where $\mathcal{S}$ is the set of all independent sets of an undirected graph $G = (U, E)$, and $c(v) = 1$ for all $v \in U$. Clearly, an independent set S in G is a clique in the complement graph $\bar{G}$, and vice versa, thus the result of Hastad (1996) for ClP is also valid for ISP.

Knapsack problems. In the Binary Knapsack Problem (KP) we have a set of items, each with a weight and a value, and a subset of item with maximum total value is sought such that the total weight of these items does not exceed a given capacity. The Multi-Dimensional Knapsack Problem (MDKP) is similar to KP, however, in this case each item has k weight values. KP and MDKP refer to problem (2.2) with $k = 1$ and $k \geq 2$, respectively, where $\mathcal{S} = 2^U$ for a set U of items.

2.5 Polyhedral theory

Consider a set of finitely many points $S = \{x^1, \ldots, x^k\} \subseteq \mathbb{R}^n$, and scalars $\lambda_1, \ldots, \lambda_k \in \mathbb{R}^n$. The vector $x := \sum_{i=1}^k \lambda_i x^i$ is called a *linear combination of the vectors* $x^1, \ldots, x^k$. If in addition $\sum_{i=1}^k \lambda_i = 1$ holds, then x is called an *affine combination of the vectors* $x^1, \ldots, x^k$. Moreover, if $0 \leq \lambda_i$ for all $i = 1, \ldots, k$, then x is called a *convex combination of the vectors* $x^1, \ldots, x^k$. The set of all linear (affine, convex) combinations of the points in S, denoted by $\text{lin}(S)$ ($\text{aff}(S)$, $\text{conv}(S)$), is called the *linear (affine, convex) hull of* S.

A set of points $x^1, \dots, x^k \in \mathbb{R}^n$ is *linearly independent* if $\sum_{i=1}^k \lambda_i x^i = 0$ implies $\lambda_i = 0$ for all $i = 1, \dots, k$; otherwise, the points are *linearly dependent*. The *rank of a set* $S \subseteq \mathbb{R}^n$, denoted by $\operatorname{rank}(S)$, is the cardinality of the largest linearly independent subset of S. The *rank of a matrix* $A \in \mathbb{R}^{m \times n}$, denoted by $\operatorname{rank}(A)$, is the rank of the set of its row vector (which is the same as the rank of the set of its column vectors). A set of points $x^1, \dots, x^k \in \mathbb{R}^n$ is *affinely independent* if $\sum_{i=1}^k \lambda_i x^i = 0$ and $\sum_{i=1}^k \lambda_i = 0$ imply $\lambda_i = 0$ for all $i = 1, \dots, k$; otherwise, the points are *affinely dependent*.

A *polyhedron* $P \subseteq \mathbb{R}^n$ is the set of points that satisfy a finite number of linear inequalities; that is, $P = \{x \in \mathbb{R}^n : Ax \leq b\}$, where (A, b) is an $m \times (n+1)$ matrix. Throughout the text we assume that (A, b) has rational coefficients. Let $[m] := \{1, \dots, m\}$, and for $i \in [m]$ let (a^i, b_i) denote the ith row of matrix (A, b). A polyhedron P is of *dimension* k, denoted by $\dim(P) = k$, if the maximum number of affinely independent points in P is $k+1$. Let $M^= := \{i \in [m] : a^i x = b_i \text{ for all } x \in P\}$ and $M^\leq := [m] \setminus M^= = \{i \in [m] : a^i x < b_i \text{ for some } x \in P\}$, and let $(A^=, b^=)$ and $(A^\leq, b^\leq)$ be the corresponding rows of (A, b), respectively. $(E, f) \subseteq m' \times (n+1)$ is called an *equation system for* P, if $\operatorname{aff}(A^=, b^=) = \{x \in \mathbb{R}^n : Ex = f\}$, moreover, it is *minimal system for* P, if $\operatorname{rank}(E, f) = \operatorname{rank}(A^=, b^=)$ also holds.

Proposition 2.1. *Let* (E, f) *be an equation system for* $P \subseteq \mathbb{R}^n$, *then* $\dim(P) + \operatorname{rank}(E, f) = n$. *Specially,* $\dim(P) + \operatorname{rank}(A^=, b^=) = n$.

Note that the result of Proposition 2.1 is still valid with the convention that if $P = \emptyset$, then $\dim(P) = -1$.

The inequality $\alpha x \leq \beta$ is called a *valid inequality* (or *cutting plane*) for P, if it is satisfied by all points in P. If $\alpha x \leq \beta$ is a valid inequality for P, then $F = \{x \in P : \alpha x = \beta\}$ is called a *face of* P, and we say that $\alpha x \leq \beta$ *represents* F. A face F of P is a *facet of* P, if $\dim(F) = \dim(P) - 1$. We say that an inequality is a *facet-defining inequality for* P, if it represents a facet of P.

2.6 Mixed-integer linear programming

Consider the *mixed-integer linear programming (MILP) problem* of the following form:

$$\text{minimize} \left\{ cx + dy : Ax + By \geq b, x \in \mathbb{Z}^n_{\geq 0}, y \in \mathbb{R}^p_{\geq 0} \right\}, \tag{2.3}$$

where $A \in \mathbb{Q}^{m \times n}$, $B \in \mathbb{Q}^{m \times p}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{R}^n$, and $d \in \mathbb{R}^p$. The set $S = \{(x, y) \in \mathbb{Z}^n_{\geq 0} \times \mathbb{R}^p_{\geq 0} : Ax + By \geq b\}$ is called the *set of feasible solutions of* (2.3) and $P := \operatorname{conv}(S)$ is called the *polyhedron of feasible solutions of* (2.3) (or the *polytope of feasible solutions of* (2.3), if it is bounded). The problem

$$\text{minimize} \left\{ cx + dy : Ax + By \geq b, x \in \mathbb{R}^n_{\geq 0}, y \in \mathbb{R}^p_{\geq 0} \right\} \tag{2.4}$$

is called the LP-*relaxation of* (2.3). The *(pure) integer linear programming (ILP) problem*

$$\text{minimize} \left\{ cx : Ax \geq b, x \in \mathbb{Z}^n_{\geq 0} \right\} \tag{2.5}$$

is the special case of MILP (2.3) in which there are no continuous variables, and the *linear programming (LP) problem*

$$\text{minimize}\left\{dy : By \geq b, y \in \mathbb{R}^p_{\geq 0}\right\} \tag{2.6}$$

is the special case of MILP (2.3) in which there are no integer variables. The *dual problem of* (2.6) is the problem

$$\text{maximize}\left\{\pi b : \pi B \leq d, \pi \in \mathbb{R}^m_{\geq 0}\right\}. \tag{2.7}$$

2.6.1 Column generation

Linear programs can be solved efficiently e.g., with the simplex algorithm or interior point methods, however, in some cases all the variables cannot be considered explicitly (e.g., due to the huge number of variables). In a column generation there is an initial system consisting of a subset of the columns only, which is gradually augmented by the missing columns, if needed. The approach is based on the observation that most of the columns will be nonbasic in an optimal solution, and have their corresponding variable equal to zero, thus, the majority of the columns are irrelevant for solving the problem optimally.

Consider the linear programming problem of the form (2.6), called *master problem*, and the *restricted master problem*

$$\text{minimize}\left\{\tilde{d}y : \tilde{B}y \geq b, y \in \mathbb{R}^q_{\geq 0}\right\} \tag{2.8}$$

which is obtained from the master problem (2.6) by excluding a set of columns, and its dual problem

$$\text{maximize}\left\{\pi b : \pi\tilde{B} \leq \tilde{d}, \pi \in \mathbb{R}^m_{\geq 0}\right\}. \tag{2.9}$$

That is, $\tilde{d}$ and $\tilde{B}$ is the vector and matrix obtained from d and B, respectively, by excluding $p - q$ columns. Solving the restricted master problem (2.8) we get an optimal (primal) solution $\bar{y} \in \mathbb{R}^q_{\geq 0}$ for (2.8) with optimal (dual) solution $\bar{\pi} \in \mathbb{R}^q_{\geq 0}$ for (2.9). Clearly, the vector $\bar{y}^0 \in \mathbb{R}^p_{\geq 0}$ obtained from $\bar{y}$ by completing with zero values is a feasible solution for the original problem (2.6), moreover, it is an optimal solution, if the dual solution $\bar{\pi}$ is feasible for the original dual problem (2.7), i.e., $\bar{\pi}B^i \leq d_i$ holds for $i = 1, \ldots, p$. The subproblem

$$\text{minimize}\left\{d_i - \bar{\pi}B^i : i \in \{1, \ldots, p\}\right\} \tag{2.10}$$

to find a violating column (if any) is called the *pricing problem*. If its optimum is greater than or equal to zero, then $\bar{\pi}$ is optimal for the original dual problem; otherwise there exists a column B^i ($i \in \{1, \ldots, p\} \setminus I$) with negative *reduced cost* $d_i - \bar{\pi}B^i$.

The sketch of a column generation procedure for (2.6) is the following: (i) we obtain a restricted master problem (2.8), then (ii) we solve the restricted mater problem to yield primal-dual solution $(\bar{y}, \bar{\pi})$, and finally (iii) we solve pricing problem (2.10). If its optimum is greater than or equal to zero, we stop since $\bar{y}^0$ is an optimal solution for the master problem (2.6); otherwise, we add the violating column for the restricted master problem and repeat the procedure.

2.6.2 Branch-and-bound procedure

The branch-and-bound procedure is generally used to solve combinatorial optimization problems. The basic idea of the procedure is to split the original problem into smaller ones, such that the set of feasible solutions of the original problem is the (usually disjunct) union of the set of feasible solutions of the subproblems. The subproblems can be divided into other subproblems, and the investigated subproblems can be represented with a rooted tree, called *enumeration tree*, where the root represents the original problem, and the children of a node represent the corresponding subproblems. We use 'subproblem' and '(enumeration tree) node' as synonyms for each other. For each subproblem a lower bound on its optimal solution is calculated, and a subproblem can be pruned from the tree, if its lower bound greater than an upper bound on the original problem.

In the case of an LP-*based branch-and-bound procedure* the lower bounds are obtained from the LP-relaxation, and the branching is taken by adding linear inequalities to the subproblems. In the following we describe the sketch of the LP-based branch-and-bound procedure to solve problem (2.5). Note that the procedure is very similar for solving problem (2.3).

Let Π_0 denote the original problem. During the procedure we maintain a lower bound LB and an upper bound UB for the optimum value of Π_0, and a list $\mathcal{L}$ of subproblems. In the beginning $LB = -\infty$, $UB = +\infty$ and $\mathcal{L}$ contains only Π_0. The sketch of the procedure is the following: (i) *Node selection:* In a general step we choose a subproblem Π from list $\mathcal{L}$, and find the optimal solution $x^\star$ of its LP-relaxation. (ii) (a) *Pruning:* If $x^\star$ does not exists (since Π is infeasible) or $UB \leq cx^\star$, then we prune the corresponding node from the tree, since the sub-tree rooted at that node does not contain better solution than the actual one. (b) *Bounding:* If $cx^\star < UB$ and $x^\star$ is integral, we set UB to $cx^\star$. (c) *Branching:* If $cx^\star < UB$ but $x^\star$ is not integral, we create two subproblems, i.e., two child nodes in the tree. Let i be an index such that $x_i^\star$ is not integral, and let Π_{down} and Π_{up} be the subproblems obtained from Π by adding inequalities $x_i \leq \lfloor x_i^\star \rfloor$ and $\lceil x_i^\star \rceil \leq x_i$, respectively. We add Π_{down} and Π_{up} to the list $\mathcal{L}$. (iii) We remove Π from list $\mathcal{L}$ and we select a new one, if $\mathcal{L}$ is not empty.

Above we described the default branching strategy for an LP-based branch-and-bound procedure (which is called the 0-1 branching in the case of problems with binary variables only). Note that in this branching strategy each subproblem is divided into exactly two subproblems by considering a single variable, however, one can create arbitrary number of subproblems, and take multiple variables into consideration to perform branch.

2.6.3 Branch-and-cut procedure

The branch-and-cut procedure is the extension of the branch-and-bound procedure where in the nodes of the enumeration tree the corresponding LP-relaxation is strengthened by valid inequalities. That is, if the optimal solution $x^\star$ for the LP-relaxation is not integral, then we add inequality $\alpha x \geq \beta$ to the problem such that $\alpha x \geq \beta$ is valid for the polytope of the feasible

solutions, but $x^\star$ violates it, i.e., $\alpha x^\star < \beta$. The procedure to find such an inequality is called *separation procedure*.

2.6.4 Branch-and-price procedure

The branch-and-price procedure is the combination of the branch-and-bound procedure with column generation. Note that to solve problem (2.5) optimally it is not sufficient in general to apply column generation for the LP-relaxation of the problem and to solve the integer programming problem with the obtained column set. In the case of a branch-and-price procedure in each node of the enumeration tree a column generation procedure is applied. If cutting planes are also separated, the method is known as branch-price-and-cut procedure.

2.6.5 Improvement techniques

Several techniques are developed to improve a branch-and-bound procedure. For example, we already mentioned the use of cutting planes that yielded the branch-and-cut procedure. In the following we discuss some other techniques.

Presolving

In most cases, the MILP formulation can be simplified (e.g., by removing redundant inequalities) and strengthened (e.g., by exploiting integrality information).

Branching strategies

Based on the relying problem one can create problem-specific branching strategies. Typically, in a branch-and-price method the default branching strategy could be weak. Consider for example a problem with set partitioning structure, i.e., of the form $\min\{cx : Ax = \mathbb{1},\ x \in \{0,1\}^n\}$ where A has 0-1 columns representing some subsets of a set. Fixing a variable to 1 on a branch reduces the search space on the corresponding enumeration subtree, however, fixing a variable to 0 on the other branch has a meager affect. Ryan and Foster (1981) suggest a branching strategy based on the observation that in every fractional solution of the LP-relaxation, there exists a pair of rows $(A_{i\cdot}, A_{j\cdot})$ with $0 < \sum_{c \in C(i,j)} x_c < 1$, where $C(i,j)$ is the set of columns covering both constraints $A_{i\cdot}$ and $A_{j\cdot}$, i.e., $C(i,j) = \{c \in \{1,\dots,n\} : A_{ic} = A_{jc} = 1\}$. Their branching strategy creates two branches: one forcing to cover rows $A_{i\cdot}$ and $A_{j\cdot}$ by the same column, and another one forcing to cover the two rows by different columns.

Bound tightening, variable fixing

Bound tightening can be applied in an enumeration tree node prior to solving the corresponding node-LP. The aim of the procedure is to tighten the bounds of some variables locally, i.e., in the enumeration subtree rooted in the corresponding node. In the case of binary linear

programs we refer to this procedure as variable fixing, since tightening bounds of a binary variable is fixing the variable to 0 or 1.

Primal heuristics

Primal heuristics aim to find feasible solutions during the global search. Some simple heuristics which do not require a starting solution can be applied before the initial LP-relaxation of the MILP is solved. Other heuristics (e.g., simple rounding heuristics, diving heuristics) can be applied in enumeration tree nodes after the corresponding node-LP is solved. Local search heuristics can be applied after a feasible solution for the MILP is found.

2.6.6 Solvers

Several commercial and non-commercial solvers have been developed for solving MILPs. CPLEX Optimizer, FICO Xpress Solver, and Gurobi Optimizer are the most well-known commercial solvers, and all of them provide a C++ callable library for users to implement their own plugins via callback functions (cutting plane separators, branching rules, primal heuristics, etc.). Note that none of these solvers support branch-and-price, that is, they allow to add columns to the problem only at the root node. SCIP Optimization Suite is a non-commercial, open-source solver (implemented in C along with C++ wrapper) for constraint integer programming including MILP. It is also a framework for branch-price-and-cut and provides total control of the solution process.

Chapter 3

Multi-criteria approximation scheme for the resource constrained shortest path problem

In Section 2.4.2 we introduced the Resource Constrained Shortest Path Problem (RCSPP) as the budgeted version of the Shortest Path Problem. In this chapter we investigate multi-criteria approximation schemes for RCSPP, where the budget limits can be slightly violated. The motivation for our study is the paper by Grandoni et al. (2014) in which the budgeted versions of a number of combinatorial optimization problems are discussed. Our positive and negative results, along with the observations of Grandoni et al. (2014) give a complete picture on the approximability of RCSPP in terms of approximation schemes.

3.1 Introduction

3.1.1 Problem definition

In this chapter we consider budgeted combinatorial optimization problems of the following form. Given a set U of elements along with a cost function $c : U \to \mathbb{Q}$, a finite set $\mathcal{S} \subseteq 2^U$ of feasible solutions, and a set of k weight functions $\mathbf{w} : U \to \mathbb{Q}^k_{\geq 0}$ on the elements along with budget limits $\mathbf{L} \in \mathbb{Q}^k_{>0}$. The k-budgeted combinatorial optimization problem is formalized as

$$\text{minimize (or maximize) } c(S) \text{ subject to } S \in \mathcal{S}, \text{ and } \mathbf{w}_i(S) \leq \mathbf{L}_i \text{ for all } i = 1, \ldots, k$$

with $c(S) := \sum_{e \in S} c(e)$ and $\mathbf{w}_i(S) := \sum_{e \in S} \mathbf{w}_i(e)$ for all $i = 1, \ldots, k$.

3.1.2 Preliminaries

In Section 2.4.1 we introduced the concepts of approximation, PTAS and FPTAS for optimization problems, and now we define similar concepts for the budgeted version. A *multi-criteria*

$(\alpha_0; \alpha_1, \ldots, \alpha_k)$-*approximation algorithm*, $\alpha_i \geq 1$, for a k-budgeted optimization problem is an algorithm which finds an α_0-approximate solution S for the non-budgeted problem such that $\mathbf{w}_i(S) \leq \alpha_i \mathbf{L}_i$ for all $i = 1, \ldots, k$. A *multi-criteria approximation scheme* for a k-budgeted optimization problem Π contains a $(1+\varepsilon; 1+\varepsilon, \ldots, 1+\varepsilon)$-approximation algorithm for Π for any $\varepsilon > 0$.

3.1.3 Related work

In the case of a single budget (i.e., $k = 1$), Hassin (1992) propose the first $(1+\varepsilon; 1)$-FPTAS for RCSPP on acyclic graphs with time complexity $O(m(n^2/\varepsilon)\log(n/\varepsilon))$, where m and n denote the number of arcs and nodes of the given directed graph, respectively. Ergun et al. (2002) give another $(1+\varepsilon; 1)$-FPTAS with improved running time of $O(mn/\varepsilon)$. For general graphs Lorenz and Raz (2001) propose an $(1+\varepsilon; 1)$-FPTAS with time complexity of $O(mn(\log\log n + 1/\varepsilon))$. Goel et al. (2001) give an $(1; 1+\varepsilon)$-FPTAS with running time $O((m + n\log n)n/\varepsilon)$.

In the case of $2 \leq k = O(1)$, one can obtain a multi-criteria $(1+\varepsilon; 1+\varepsilon, \ldots, 1+\varepsilon)$-FPTAS for the k-BUDGETED s-t PATH PROBLEM (including RCSPP) and the k-BUDGETED SPANNING TREE PROBLEM based on the general technique of Papadimitriou and Yannakakis (2000), however, there is no $(\alpha_0; \alpha_1, \ldots, \alpha_k)$-approximation algorithm with two or more α_i's equal to 1 for these problems, unless P=NP (see (Grandoni et al., 2014)). Further on, Grandoni et al. (2014) describe $(1; 1+\varepsilon, \ldots, 1+\varepsilon)$-PTASs for the k-BUDGETED SPANNING TREE PROBLEM and the k-BUDGETED MATROID BASIS PROBLEM, however, they do not provide such an algorithm for RCSPP, which is one of the motivations for our work. They also provide a $(1+\varepsilon; 1+\varepsilon, \ldots, 1+\varepsilon)$-PTAS for the k-BUDGETED BIPARTITE MATCHING PROBLEM. Another motivation is that the method of Papadimitriou and Yannakakis (2000) and the results of Grandoni et al. (2014) work only if the number of weight functions, k, is a constant.

Song and Sahni (2006) describe ε-approximation algorithms for k-MCP. For an overview of exact and approximation algorithms for k-MCP and its variants we refer to (Garroppo et al., 2010).

3.1.4 Our contribution

Firstly, we show that RCSPP admits no multi-criteria PTAS if the number of weight functions, k, is part of the input (i.e., not a constant), unless P = NP. This statement is also true for some other k-budgeted optimization problems (Theorem 3.1). On the positive side, we provide a $(1; 1+\varepsilon, \ldots, 1+\varepsilon)$-FPTAS for the case of $k = O(1)$ (Theorem 3.2). Notice that a direct application of the method of Papadimitriou and Yannakakis (2000) would give only an $(1+\varepsilon; 1+\varepsilon, \ldots, 1+\varepsilon)$-FPTAS. Our multi-criteria FPTAS is a dynamic programming algorithm, similar to the SPPP algorithm of Lorenz and Raz (2001), with a combination of the rounding technique of Song and Sahni (2006). The time complexity of our multi-criteria approximation scheme is $O(m(n/\varepsilon)^k)$, which for $k = 1$ matches that of Ergun et al. (2002), who provided an $(1+\varepsilon; 1)$-FPTAS for RCSPP restricted to acyclic graphs.

Our positive and negative results, along with the observations of Grandoni et al. (2014) give a complete picture on the approximability of RCSPP in terms of approximation schemes.

Theorem 3.1. *If* P $\neq$ NP, *and the number of weight functions, k, is not a constant (i.e., part of the input), then there is no polynomial time multi-criteria approximation scheme either for the minimization or for the maximization version of the k-*budgeted *s-t* Path Problem, *the k-*budgeted Spanning Tree Problem, *the k-*budgeted Matroid Basis Problem, *and the k-*budgeted Bipartite Perfect Matching Problem.

Note that RCSPP refers to the minimization version of the *k-*budgeted *s-t* Path Problem.

Theorem 3.2. *If the number of weight functions, k, is a constant (i.e., not part of the input), then there exists a fully polynomial time $(1; 1+\varepsilon, \dots, 1+\varepsilon)$-approximation scheme for the* Resource Constrained Shortest Path Problem.

We emphasize that our result is valid for general graphs with nonnegative weights and arbitrary costs, however, cycles with negative total cost are not allowed.

3.2 Proof of Theorem 3.1

In this section we assume that the number of weight functions, k, is part of the input (not a constant). First, we prove that unless P = NP, there exists no polynomial time multi-criteria approximation scheme for RCSPP (Theorem 3.3). Based on the proof, it is a routine to show that there is no multi-criteria PTAS for the minimization version of the *k-*budgeted Spanning Tree Problem (thus for the *k-*budgeted Matroid Basis Problem) and the *k-*budgeted Bipartite Perfect Matching Problem.

Theorem 3.3. *If* P $\neq$ NP, *and the number of weight functions, k, is part of the input (i.e., not a constant), then there is no polynomial time multi-criteria approximation scheme for the* Resource Constrained Shortest Path Problem.

Proof. We give a PTAS-preserving reduction from the Vertex Cover Problem (VCP) to RCSPP to show that there is no polynomial time $(1+\varepsilon; 2-\varepsilon, \dots, 2-\varepsilon)$-approximation algorithm for RCSPP with $0 < \varepsilon < 0.3606$, unless P = NP. Recall that an instance of VCP is given by an undirected graph G, and a minimum size subset of nodes $C \subseteq V(G)$ is sought such that for each edge $(u,v) \in E(G)$, $1 \leq |\{u,v\} \cap C|$. Given a VCP instance, we create an instance of RCSPP with $k = |E(G)|$ and with directed graph D as follows. For each node $v_i \in V(G)$, $i = 1, \dots, n$, we add two distinct nodes x_i and y_i, and also node x_{n+1} to $V(D)$, and three arcs (x_i, x_{i+1}), (x_i, y_i) and (y_i, x_{i+1}) to $A(D)$. Let $s = x_1$, $t = x_{n+1}$. We create a cost function $c : A(D) \to \mathbb{Q}_{\geq 0}$ such that $c(x_i, x_{i+1}) = 1$ and $c(x_i, y_i) = c(y_i, x_{i+1}) = 0$ for all $i = 1, \dots, n$. We create weights $\mathbf{w} : A(D) \to \mathbb{Q}_{\geq 0}^{|E(G)|}$ such that

$$\mathbf{w}_{ij}(a) = \begin{cases} 1 & \text{if } a = (x_i, y_i) \text{ or } a = (x_j, y_j), \\ 0 & \text{otherwise} \end{cases}$$

for each $(v_i, v_j) \in E(G)$. We set the corresponding budget limit $\mathbf{L}_{ij}$ to 1.

Consider the one-to-one correspondence between the node sets of G and the s-t paths in D, such that to a node set $C \subseteq V(G)$ we assign the s-t path $\pi[C]$ in D consisting of arcs $\{(x_i, x_{i+1}) : v_i \in C\} \cup \{(x_i, y_i), (y_i, x_{i+1}) : v_i \notin C\}$. It is clear that for a node set $C \subseteq V(G)$ and the corresponding path $\pi[C]$, $c(\pi[C]) = |C|$, moreover, we claim that C is a vertex cover if and only if $P[C]$ satisfies the budget limits. If C is a vertex cover, then for each edge $(v_i, v_j) \in E(G)$ we have $1 \leq |\{v_i, v_j\} \cap C|$, thus $|\{(x_i, y_i), (x_j, y_j)\} \cap \pi[C]| \leq 1$, therefore $\mathbf{w}_{ij}(\pi[C]) \leq \mathbf{L}_{ij}$. The opposite direction can be shown similarly.

Assume that we have an $(1+\varepsilon; 2-\varepsilon, \ldots, 2-\varepsilon)$-approximation algorithm for RCSPP with $0 < \varepsilon < 0.3606$. Applying this algorithm for RCSPP instance, we can find in polynomial time an s-t path π such that $c(\pi) \leq (1+\varepsilon)c(\pi_{OPT})$ and $\mathbf{w}_{ij}(\pi) \leq (2-\varepsilon)\mathbf{L}_{ij}$, for all $(v_i, v_j) \in E(G)$, where π_{OPT} is an optimal solution for RCSPP. On the one hand, $c(\pi) \leq (1+\varepsilon)c(\pi_{OPT}) = (1+\varepsilon)|C_{OPT}|$, where C_{OPT} is an optimal solution for VCP. On the other hand, $\mathbf{w}_{ij}(\pi) \leq 2-\varepsilon < 2$, i.e., $\mathbf{w}_{ij}(\pi) \leq 1$ holds for all $(v_i, v_j) \in E(G)$, thus the set $C \subseteq V(G)$ corresponding to π is a vertex cover. To sum up, by applying the approximation algorithm for RCSPP instance, we can find in polynomial time a vertex cover C in G such that $|C| \leq (1+\varepsilon)|C_{OPT}|$ which is impossible for $\varepsilon < 0.3606$, unless P = NP (see (Dinur and Safra, 2005)). □

Now, we prove that the Multi-Dimensional Knapsack Problem (MDKP) does not admit a polynomial time multi-criteria approximation scheme if the number of dimensions, k, is part of the input (Theorem 3.4). Recall that an instance of MDKP is given by a set of items U, where each item $u \in U$ has a cost $c(u)$ and a weight $\mathbf{w}(u) \in \mathbb{Q}^k_{\geq 0}$, and there is a weight limit $\mathbf{L} \in \mathbb{Q}^k_{\geq 0}$. A subset of items $S \subseteq U$ of maximum $c(S)$ value is sought such that $\mathbf{w}(S) \leq \mathbf{L}$. Apparently, it is the budgeted version of a trivial maximization problem over all subsets of U.

By using similar techniques it is easy to prove that there exists no multi-criteria PTAS for the maximization version of the k-budgeted s-t Path Problem, k-budgeted Spanning Tree Problem, the k-budgeted Matroid Basis Problem and the k-budgeted Bipartite Perfect Matching Problem, if the number of weight functions, k, is part of the input (i.e., not a constant), unless P = NP.

Theorem 3.4. *If* P ≠ NP, *and the dimension k of* MDKP *is part of the input (i.e., not a constant), then there is no polynomial time multi-criteria approximation scheme for* MDKP.

Proof. We give a PTAS-preserving reduction from the Independent Set Problem (ISP) problem to MDKP. An instance of ISP is given by an undirected graph $G = (V, E)$, and a maximum size subset of nodes $S \subseteq V$ is sought such that for each edge $(u, v) \in E$, $|\{u, v\} \cap S| \leq 1$. Given an ISP instance, we create an instance of MDKP with $k = |E|$ as follows. We create a set of items $U = \{u_1, \ldots, u_n\}$ where item u_i corresponds to node $v_i \in V$ and it has a cost $c(u_i) = 1$. We create weights $\mathbf{w}_{ij} : U \to \mathbb{Q}^{|E|}_{\geq 0}$ such that

$$\mathbf{w}_{ij}(u) = \begin{cases} 1 & \text{if } u \text{ corresponds to } v_i \text{ or } v_j, \\ 0 & \text{otherwise} \end{cases}$$

for each $(v_i, v_j) \in E$. We set the corresponding budget limit $\mathbf{L}_{ij}$ to 1.

Consider a node set $I = \{v_{i_1}, \dots, v_{i_p}\} \subseteq V$ and the corresponding set of items $S = \{u_{i_1}, \dots, u_{i_p}\}$. If I is independent, then for each edge $(v_i, v_j) \in E$ we have $|\{v_i, v_j\} \cap I| \leq 1$, thus $|\{u_i, u_j\} \cap S| \leq 1$, therefore $\mathbf{w}_{ij}(S) \leq \mathbf{L}_{ij}$, that is, S satisfies the budget limits, moreover $c(S) = |I|$. The opposite direction (that is, the node set corresponding to a set of items that satisfies the budget limits is independent) can be shown similarly.

Similarly to the previous proof, if we had a $(1+\varepsilon; 2-\varepsilon, \dots, 2-\varepsilon)$-approximation algorithm for MDKP with $0 < \varepsilon < 1$, we could find in polynomial time an independent set I in G such that $|I| \geq |I_{OPT}|/(1+\varepsilon)$ which is impossible, unless P = NP (see (Hastad, 1996)). □

3.3 Proof of Theorem 3.2

We give a $(1; 1+\varepsilon, \dots, 1+\varepsilon)$-FPTAS for RCSPP, where the number of weight functions, k, is a constant. Recall that this problem can be formulated as

$$\min_{\pi \in \mathcal{P}_{st}} \{c(\pi) : \ \mathbf{w}_i(\pi) \leq \mathbf{L}_i \text{ for all } i = 1, \dots, k\}. \tag{3.1}$$

For a given an $\varepsilon > 0$ we scale and round the weights, that is, we define a scale vector $\mathbf{\Delta} \in \mathbb{Q}^k_{>0}$ and scaled weights $\bar{\mathbf{w}} \in \mathbb{Q}^k_{>0}$ as follows: for all $i = 1, \dots, k$ let $\mathbf{\Delta}_i := \varepsilon \mathbf{L}_i/(n-1)$, and for each arc $a \in A$ let $\bar{\mathbf{w}}_i(a) := d_i(a)\mathbf{\Delta}_i$, where $d_i(a) = 1$ if $\mathbf{w}_i(a) = 0$, otherwise $d_i(a)$ is a positive integer such that $(d_i(a)-1)\mathbf{\Delta}_i < \mathbf{w}_i(a) \leq d_i(a)\mathbf{\Delta}_i$ holds. Note that $0 < \bar{\mathbf{w}}_i(a)$ holds for each arc a and $i = 1, \dots, k$. Consider the following, scaled problem:

$$\min_{\pi \in \mathcal{P}_{st}} \{c(\pi) : \ \bar{\mathbf{w}}_i(\pi) \leq (1+\varepsilon)\mathbf{L}_i \text{ for all } i = 1, \dots, k\}. \tag{3.2}$$

For any s-t path π and $i = 1, \dots, k$ we have

$$\bar{\mathbf{w}}_i(\pi) = \sum_{a \in \pi} \bar{\mathbf{w}}_i(a) \leq \sum_{a \in \pi} (\mathbf{w}_i(a) + \mathbf{\Delta}_i) \leq \sum_{a \in \pi} \mathbf{w}_i(a) + (n-1)\mathbf{\Delta}_i = \mathbf{w}_i(\pi) + \varepsilon \mathbf{L}_i,$$

since π consists of at most $n-1$ arcs. Clearly, if $\mathbf{w}_i(\pi) \leq \mathbf{L}_i$ holds for some $i = 1, \dots, k$, then $\bar{\mathbf{w}}_i(\pi) \leq (1+\varepsilon)\mathbf{L}_i$ also holds. That is, if π is a feasible solution for the original problem (3.1), then π is feasible for the scaled problem (3.2) as well. Moreover, since by definition $\mathbf{w}_i(a) \leq \bar{\mathbf{w}}_i(a)$ holds for all arcs a and $i = 1, \dots, k$, thus for each feasible solution π for the scaled problem (3.2) we have $\mathbf{w}_i(\pi) \leq \bar{\mathbf{w}}_i(\pi) \leq (1+\varepsilon)\mathbf{L}_i$, for all $i = 1, \dots, k$. These imply the following proposition.

Proposition 3.1. *If problem* (3.1) *has a feasible solution, then any optimal solution for problem* (3.2) *is a* $(1; 1+\varepsilon, \dots, 1+\varepsilon)$*-approximate solution for* (3.1).

3.3.1 Dynamic programming algorithm

In the following we use element-wise operations for vectors. That is, the Hadamard product of vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Q}^k$ is the vector $\mathbf{a} \circ \mathbf{b} \in \mathbb{Q}^k$ with $(\mathbf{a} \circ \mathbf{b})_i = \mathbf{a}_i \mathbf{b}_i$ for all $i = 1, \dots, k$. The

inverse of vector $\mathbf{a} \in \mathbb{Q}^k_{>0}$ is the vector $\mathbf{a}^{-1} \in \mathbb{Q}^k_{>0}$ with $(\mathbf{a}^{-1})_i = 1/\mathbf{a}_i$ for all $i = 1, \ldots, k$. For k-dimension vectors $\mathbf{a}$ and $\mathbf{b}$ we write $\mathbf{a} \leq \mathbf{b}$ ($\mathbf{a} < \mathbf{b}$) if $\mathbf{a}_i \leq \mathbf{b}_i$ ($\mathbf{a}_i < \mathbf{b}_i$) holds for all $i = 1, \ldots, k$.

A *pattern* is a vector $\eta = (\eta_1, \ldots, \eta_k)$ where η_i ($i = 1, \ldots, k$) is a non-negative integer. Note that for any path π, there is a pattern η with nonzero elements such that $\bar{\mathbf{w}}(\pi) = \eta \circ \mathbf{\Delta}$. A pattern η is *feasible* if $\eta \circ \mathbf{\Delta} \leq (1+\varepsilon)\mathbf{L}$ holds.

Proposition 3.2. *The number of feasible patterns is* $O((n/\varepsilon)^k)$.

Proof. For any feasible pattern η, $0 \leq \eta_i \leq (1+\varepsilon)\mathbf{L}_i/\mathbf{\Delta}_i = (n-1)(1+1/\varepsilon)$ holds for all $i = 1, \ldots, k$, thus the number of feasible patterns is at most $((n-1)(1+1/\varepsilon)+1)^k = O((n/\varepsilon)^k)$. □

For a node v and pattern η let $\chi(v, \eta)$ denote the cost of the minimum cost s-v path π such that $\bar{\mathbf{w}}(\pi) \leq \eta \circ \mathbf{\Delta}$, if any. By this, $\chi(t, \lfloor (1+\varepsilon)\mathbf{L} \circ \mathbf{\Delta}^{-1} \rfloor)$ is the optimal solution value of (3.2). Let $H = (\eta^1, \eta^2, \ldots, \eta^{|H|})$ denote the set of the feasible patterns, where patterns are partially ordered by the element-wise comparison, that is, if $\eta^p \leq \eta^q$ holds for patterns η^p and η^q, then $p \leq q$. Clearly $\eta^1 = (0, \ldots, 0)$.

The sketch of the algorithm can be seen in Algorithm 1. In the initialization phase for each pattern η we set $\chi(s, \eta)$ to zero and $\chi(v, \eta)$ to infinity ($v \neq s$). We iterate over the partially ordered set of feasible patterns (note that according to the initialization, we can skip pattern $\eta^1 = (0, \ldots, 0)$), and in each iteration we visit each node in the graph. For a given pattern η and node v we examine the incoming arcs of v. Let (u, v) be an impending arc. If $\bar{\mathbf{w}}_i(u, v) > \eta_i \mathbf{\Delta}_i$ holds for some $1 \leq i \leq k$, then there is no s-v path containing arc (u, v) such that $\bar{\mathbf{w}}(\pi) \leq \eta \circ \mathbf{\Delta}$, thus we cannot update $\chi(v, \eta)$. Otherwise $\bar{\mathbf{w}}(u, v) \circ \mathbf{\Delta}^{-1} \leq \eta$, and by definition $0 < \bar{\mathbf{w}}(u, v) \circ \mathbf{\Delta}^{-1}$, thus $0 \leq \eta - \bar{\mathbf{w}}(u, v) \circ \mathbf{\Delta}^{-1} < \eta$, i.e., pattern $\eta - \bar{\mathbf{w}}(u, v) \circ \mathbf{\Delta}^{-1}$ was already examined in a former iteration, that is, $\chi(u, \eta - \bar{\mathbf{w}}(u, v) \circ \mathbf{\Delta}^{-1})$ is already computed (and valid), so we can update $\chi(v, \eta)$. Finally, we return $\chi(t, \lfloor (1+\varepsilon)\mathbf{L} \circ \mathbf{\Delta}^{-1} \rfloor)$ which is the optimal solution value of (3.2). In Theorem 3.5 we prove the correctness of the algorithm.

Algorithm 1 Dynamic programming algorithm for (3.2)

```
χ(s, η) ← 0 (η ∈ H)
χ(v, η) ← ∞ (v ≠ s, η ∈ H)
for η = η^2, η^3, ..., η^|H| do
    for v ∈ V do
        for a ∈ {(u, v) ∈ A : w̄(u, v) ≤ η ∘ Δ} do
            χ(v, η) ← min{χ(v, η), χ(u, η − w̄(a) ∘ Δ^{-1}) + c(a)}
return χ(t, (⌊(n − 1)(1 + 1/ε)⌋, ..., ⌊(n − 1)(1 + 1/ε)⌋))
```

Theorem 3.5. *After Algorithm 1 terminates, for each node $v \in V$ and for each pattern η, $\chi(v, \eta)$ is equal to the cost of the minimum cost s-v path π such that $\bar{\mathbf{w}}(\pi) \leq \eta \circ \mathbf{\Delta}$.*

Proof. Basically, we prove that after the algorithm terminates $\chi(v,\eta)$ is equal to the cost of the minimum cost s-v walk π (i.e., vertices may be repeated) such that $\bar{\mathbf{w}}(\pi) \leq \eta \circ \boldsymbol{\Delta}$. However, according to our assumptions the graph does not contain cycles with negative total cost or negative total weight, therefore a minimum cost s-t walk always comprises an s-t path of the same cost. To prove the former statement, it is sufficient to show that after a pattern η is examined (i.e., the corresponding iteration is performed):

a) for each node v, if $\chi(v,\eta)$ is not infinity, then it is equal to the cost of an s-v walk π such that $\bar{\mathbf{w}}(\pi) \leq \eta \circ \boldsymbol{\Delta}$.

b) for each node v, if there is an s-v walk π with $\bar{\mathbf{w}}(\pi) \leq \eta \circ \boldsymbol{\Delta}$, then $\chi(v,\eta) \leq c(\pi)$ holds.

We prove these statements by induction. Clearly, statements a) and b) are satisfied before the first iteration is performed (i.e., after the initialization).

To prove statement a) consider a moment when $\chi(v,\eta)$ is updated (line 6), that is $\chi(v,\eta) = \chi(\eta - \bar{\mathbf{w}}(u,v) \circ \boldsymbol{\Delta}^{-1}) + c(u,v)$ for some $(u,v) \in A$ with $\bar{\mathbf{w}}(u,v) \leq \eta \circ \boldsymbol{\Delta}$. Since $0 < \bar{\mathbf{w}}(u,v)$ holds by definition, thus $0 \leq \eta - \bar{\mathbf{w}}(u,v) \circ \boldsymbol{\Delta}^{-1} < \eta$, i.e., pattern $\eta - \bar{\mathbf{w}}(u,v) \circ \boldsymbol{\Delta}^{-1}$ was already examined in a former iteration, that is, $\chi(u, \eta - \bar{\mathbf{w}}(u,v) \circ \boldsymbol{\Delta}^{-1})$ is already computed. By inductive assumption, $\chi(u, \eta - \bar{\mathbf{w}}(u,v) \circ \boldsymbol{\Delta}^{-1})$ is equal to the cost of an s-u walk π with $\bar{\mathbf{w}}(\pi) \leq \eta \circ \boldsymbol{\Delta} - \bar{\mathbf{w}}(u,v)$, thus $\chi(v,\eta)$ is equal to the cost of the walk $\pi' = \pi \cup \{(u,v)\}$ with $\bar{\mathbf{w}}(\pi') \leq \eta \circ \boldsymbol{\Delta}$.

To prove statement b) consider the shortest s-v path π such that $\bar{\mathbf{w}}(\pi) \leq \eta \circ \boldsymbol{\Delta}$ holds, and let (u,v) be its last arc, i.e., $\pi = \pi' \cup (u,v)$ for an s-u path π'. Clearly, π' is the minimum cost s-u walk such that $\bar{\mathbf{w}}(\pi') \leq \eta \circ \boldsymbol{\Delta} - \bar{\mathbf{w}}(u,v)$ holds. On the one hand, by inductive assumption, we have $\chi(v, \eta - \bar{\mathbf{w}}(u,v) \circ \boldsymbol{\Delta}^{-1}) \leq c(\pi')$, and on the other hand we compared $\chi(v,\eta)$ and $\chi(u, \eta - \bar{\mathbf{w}}(u,v) \circ \boldsymbol{\Delta}^{-1}) + c(u,v)$ in the iteration of pattern η (line 6), therefore we have $\chi(v,\eta) \leq \chi(u, \eta - \bar{\mathbf{w}}(u,v) \circ \boldsymbol{\Delta}^{-1}) + c(u,v) \leq c(\pi') + c(u,v) = c(\pi)$. □

According to Proposition 3.2 we have at most $O((n/\varepsilon)^k)$ iterations, and in each iteration each arc is examined once, thus the running time of the algorithm is $O(m(n/\varepsilon)^k)$.

Chapter 4

LP-based methods for the resource constrained shortest path problem

In Section 2.4.2 we introduced the Resource Constrained Shortest Path Problem (RCSPP) as the budgeted version of the Shortest Path Problem, and in Chapter 3 we investigated approximation schemes for the problem. In this chapter we turn our attention to exact solution approaches. Although several types of solution approaches are proposed to solve variants of RCSPP, only a few papers deal with LP-based methods, which was the main motivation for our study. We present new primal heuristics and a variable fixing method to improve a branch-and-bound procedure, new cutting planes along with separation procedures that can be used in a branch-and-cut procedure, and we also present the results of our thorough computational experiments.

4.1 Introduction

4.1.1 Problem definition

Recall that an instance of RCSPP is given by a directed graph $D = (V, A)$ with designated nodes $s, t \in V$, a cost function $c : A \to \mathbb{Q}$ on the arcs, and a set of k weight functions $\mathbf{w} : A \to \mathbb{Q}^k$ on the arcs (also called *resource functions*) along with limits $\mathbf{L} \in \mathbb{Q}^k$ (also called *resource limits*); and a minimal cost s-t path π is sought such that the resource limits on the path are not violated, that is, $\sum_{e \in \pi} \mathbf{w}_i(e) \leq \mathbf{L}_i$ holds for all $i = 1, \ldots, k$. Note that in contrast to the definition in Section 2.4.2 we allow negative weights and thus non-positive limits as well.

Assumptions

Throughout this chapter we consider an instance of RCSPP with the following assumptions.

Assumption 4.1. *For the underlying directed graph D we have $\rho^{out}(s) = \rho^{in}(t) = V$ and $\delta^{in}(s) = \delta^{out}(t) = 0$.*

Assumption 4.2. *The underlying directed graph D contains no directed cycle of negative total cost, or of negative total resource consumptions for any of the resources.*

Note that Assumption 4.1 is not a restriction, since each node $v \in (V \setminus \rho^{out}(s)) \cup ((V \setminus \rho^{in}(t))$ and each arc $e \in \delta^{in}(s) \cup \delta^{out}(t)$ can be removed from the graph, because they cannot appear in any s-t path.

4.1.2 Problem formulation

Let x_e be the binary variable indicating whether the path sought goes through the arc $e \in A$ or not. We formulate RCSPP as:

$$\min \sum_{e \in A} c(e) x_e \tag{4.1}$$

$$\sum_{e \in \delta^{out}(v)} x_e - \sum_{e \in \delta^{in}(v)} x_e = \begin{cases} 1 & \text{if } v = s \\ 0 & \text{if } v \in V \setminus \{s,t\} \\ -1 & \text{if } v = t \end{cases} \qquad \text{for all } v \in V \tag{4.2}$$

$$x(\gamma(S)) \leq |S| - 1 \qquad \text{for all } S \subset V \text{ with } 2 \leq |S| \tag{4.3}$$

$$\sum_{e \in A} \mathbf{w}_i(e) x_e \leq \mathbf{L}_i \qquad \text{for all } i = 1, \ldots, k \tag{4.4}$$

$$x_e \in \{0,1\} \qquad \text{for all } e \in A. \tag{4.5}$$

Flow conservation equations (4.2) along with cycle elimination inequalities (4.3) and binary conditions (4.5) ensure that feasible solutions constitute s-t paths. The objective function (4.1) expresses the total cost of the path, while the resource consumptions of the path are bounded by inequalities (4.4). Let $S^{RCSPP} := \{x \in \{0,1\}^A : x \text{ satisfies } (4.2) - (4.4)\}$ be the set of feasible solutions of problem (4.2)–(4.5), and $P^{RCSPP} := \text{conv}(S^{RCSPP})$ be the polytope of feasible solutions of problem (4.2)–(4.5), called *the* RCSPP-*polytope*.

Proposition 4.1. *S^{RCSPP} is the set of incidence vectors corresponding to resource feasible s-t paths.*

Note that cycle elimination inequalities (4.3) are only necessary if negative cost cycles are present, which is not our case due to Assumption 4.2, thus we do not use these inequalities. By this, cycles with zero cost can be present in an optimal solution, however, such an optimal solution always comprises a feasible s-t path with the same costs, that is, one can remove zero cost cycles by postprocessing to obtain an optimal s-t path.

4.1.3 Related work

Several types of exact solution approaches have been proposed to solve variants of RCSPP, e.g., path ranking-based methods, dynamic programming strategies, label-setting/correcting algorithms, methods based on branch-and-bound procedure. For an overview we refer to e.g., (Pugliese and Guerriero, 2013b), and in the following we only discuss papers related to branch-and-cut procedure or polyhedral investigation of the RCSPP-polytope.

Avella et al. (2004) present a generalization of RCSPP where numerical non-totalizable attributes are also considered. The authors use a similar formulation that of (4.1)–(4.5), and propose a branch-and-cut approach where certain cycle elimination inequalities are separated. Jepsen et al. (2008) consider a problem similar to RCSPP where weights are associated with nodes rather than arcs. The authors also propose a branch-and-cut procedure where generalized subtour elimination inequalities and two other classes of valid inequalities are separated. Garcia (2009) provides several class of valid inequalities for the RCSPP-polytope in the acyclic and generic case as well. The author also presents a branch-and-cut procedure along with preprocessing procedures, branching strategies, and primal heuristics.

Dahl and Realfsen (2000) and Dahl and Gouveia (2004) study a special case of RCSPP with a single resource function and unit weights (i.e., only the length of the path in terms of arcs is restricted) on acyclic graphs. Dahl and Realfsen (2000) show that the polytope associated with the LP-relaxation of their problem formulation is integral in some special cases, and thus can be solved as a linear programming problem. Dahl and Gouveia (2004) give a complete linear characterization of the polytope when the resource limit is equal to 2 or 3.

4.1.4 Our contribution

We investigate LP-based branch-and-bound solution approaches. We propose new primal heuristics and a variable fixing procedure to improve the branch-and-bound procedure, and we also generalize some inequalities of Garcia (2009) and give exact and heuristic separation procedures in order to use them in a branch-and-cut procedure. We present the results of our thorough computational experiments in which (i) we examined the effectiveness of the components mentioned above, and (ii) we compared our LP-based method with state-of-the-art methods.

4.1.5 Preliminaries

For basic terminology and notation we refer to Section 2.2.2. For nodes $u, v \in V$ let $\omega_i(u,v)$ denote the length of the shortest u-v path with respect to the arc weights $\mathbf{w}_i$, i.e., $\omega_i(u,v) = \min\{\mathbf{w}_i(\pi) : \pi \in \mathcal{P}_{uv}\}$; whereas $\sigma(u,v)$ denotes that with respect to the arc costs c, i.e., $\sigma(u,v) = \min\{c(\pi) : \pi \in \mathcal{P}_{uv}\}$. Note that $\omega_i(u,u) = 0$ and $\sigma(u,u) = 0$ for each node u and $i = 1,\dots,k$. If no u-v path exists then $\omega_i(u,v) = \infty$ for all $i = 1,\dots,k$, and $\sigma(u,v) = \infty$.

We say that a tuple (i.e., an ordered sequence) of arcs $\tau = (e_1, e_2 \dots, e_p)$ is *compatible with respect to* $\mathbf{w}_i$ if

$$\omega_i(s, \text{tail}(e_1)) + \mathbf{w}_i(e_1) + \omega_i(\text{head}(e_1), \text{tail}(e_2)) + \mathbf{w}_i(e_2) + \ldots + \mathbf{w}_i(e_p) + \omega_i(\text{head}(e_p), t) \leq \mathbf{L}_i$$

holds; otherwise τ is *incompatible with respect to* $\mathbf{w}_i$. Moreover, τ is *compatible*, if it is compatible with respect to $\mathbf{w}_i$ for all $i = 1,\dots,k$; otherwise τ is *incompatible*. Specially, if $\pi = (e_1, e_2 \dots, e_p)$ is a path, we say that π is a *feasible/infeasible subpath with respect to* $\mathbf{w}_i$, and π is a *feasible/infeasible*

subpath, respectively. Finally, if $\pi = (e_1, e_2 \ldots, e_p)$ is an s-t path, we say that π is a *feasible/infeasible s-t path with respect to* $\mathbf{w}_i$, and π is a *feasible/infeasible s-t path*, respectively.

4.2 Preprocessing, variable fixing, primal heuristics

4.2.1 Previous work

Several preprocessing methods are proposed for RCSPP in order to reduce the size of the underlying graph. Clearly, a node $v \in V$ cannot be in any feasible s-t path if

$$\omega_i(s, v) + \omega_i(v, t) > \mathbf{L}_i \tag{4.6}$$

holds for some $i \in \{1, \ldots, k\}$, or

$$\sigma(s, v) + \sigma(v, t) > U, \tag{4.7}$$

where U is an upper bound on the value of an optimal s-t path. Similarly, an arc $e = (u, v) \in A$ cannot be in any feasible s-t path if

$$\omega_i(s, u) + \mathbf{w}_i(e) + \omega_i(v, t) > \mathbf{L}_i \tag{4.8}$$

holds for some $i \in \{1, \ldots, k\}$, or

$$\sigma(s, u) + c(e) + \sigma(v, t) > U, \tag{4.9}$$

where U is an upper bound on the value of an optimal s-t path.

Aneja et al. (1983) repeatedly remove nodes with (4.6) and arcs with (4.8) until no other nodes and arcs can be deleted, or no s-t path remains in the reduced graph (which means that the problem is infeasible). Beasley and Christofides (1989) extend this approach by considering cost bounds to erase additional nodes and arcs from the underlying graph, similar to (4.7) and (4.9). In a branch-and-bound procedure in each enumeration tree node they calculate cost bounds through Lagrangean relaxation and eliminate nodes and arcs from the graph that could not appear in an optimal s-t path. Dumitrescu and Boland (2003) combine and simplify these two approaches. That is, they repeatedly remove nodes with (4.6) or (4.7), and arcs with (4.8) or (4.9), where the upper bound U may be calculated and updated during the procedure. We refer to this procedure as DB-*preprocessing*. Note that the final upper bound can be also used e.g., to improve a branch-and-bound procedure.

Garcia (2009) also extends the preprocessing scheme of Aneja et al. (1983) by applying their procedure on graph $D[v]$ for each node v. Recall that $D[v]$ is the subgraph of D containing all the s-t paths traversing through node v. So, if no s-t path remains in $D[v]$ after the reduction, then there exists no feasible s-t path in D traversing through node v, thus node v can be removed from D. We refer to this procedure as G-*preprocessing*. Note that this approach can be applied not only to a node v, but also to an arc e, that is, one can apply the preprocessing

scheme of Aneja et al. (1983) on graph $D[e]$. Since applying this procedure for all arcs can be expensive, Garcia (2009) suggests to use it as a variable fixing method. That is, once a fractional solution $\bar{x}$ to the LP-relaxation is available, one can preprocess $D[e]$ for each arc e with $\bar{x}_e > 0$, and fix x_e to zero, if no s-t path remains in $D[e]$ after the preprocessing. We refer to this approach as G-*varfix*.

4.2.2 New results

Primal heuristics

The basic idea of our primal heuristics is to find a feasible s-t path in the support graph $D_{\bar{x}}$ of the solution $\bar{x} \in [0,1]^A$ of the node-LP of the corresponding enumeration tree node (that is, $D_{\bar{x}} = (V, A_{\bar{x}})$ with $A_{\bar{x}} = \{e \in A : \bar{x}_e > 0\}$), if any, and update the best upper bound on an optimal solution to improve the branch-and-bound procedure. The basis of this idea is the observation that $\bar{x}$ is a convex combination of s-t paths (note we can omit cycles with 0 cost, if any). Briefly stated, we perform a depth-first search from s on $D_{\bar{x}}$, and once we reach a processed node v we can examine one or more s-t paths through v. For details we refer to (Horváth and Kis, 2016a).

Variable fixing procedure

The basic idea of our variable fixing procedure is to apply the DB-preprocessing in enumeration tree nodes (prior to solving the corresponding node LP) in order to fix arcs (i.e., the corresponding variables) to 0 if they cannot appear in any optimal solution of the subtree rooted at the corresponding node. Recall that DB-preprocessing is originally used to remove nodes and arcs from the underlying graph which cannot be in any feasible s-t path with respect to a single resource, or in any s-t path with cost at most U, where U is an upper bound on an optimal s-t path. If no s-t path remains in the reduced graph, then the problem is proven to be infeasible. Moreover, when a feasible s-t path is found during the procedure, the upper bound will be updated, and the final bound can be used to improve e.g., a branch-and-bound procedure.

In an enumeration tree node some variables may be already fixed to 0 or 1 (e.g., due to branching decisions), that is, in the subtree rooted at that tree node we seek a minimum cost, feasible s-t path which passes through every arc e such that x_e is fixed to 1 (*mandatory arcs*), and avoids each arc e such that x_e is fixed to 0 (*forbidden arcs*). We call such an s-t path *proper*. Thus we obtain a directed graph from the original (possibly preprocessed) graph by removing nodes and arcs that cannot appear in any proper path (shortly, after the forbidden arcs are erased from the graph we remove all nodes and arcs which cannot appear in any s-t path which contains all the mandatory arcs) and then we apply DB-preprocessing on it. We fix each variable corresponding to an erased arc to 0 since a proper path cannot passes through it. If there is no s-t path in the reduced graph, we prune the enumeration tree node. Note that

this variably fixing procedure is also a primal heuristics since each feasible s-t path found by DB-preprocessing is a feasible solution for problem (4.1)–(4.5). For details we refer to (Horváth and Kis, 2016a).

4.3 Valid inequalities for the RCSPP-polytope

4.3.1 Previous work

s-t cut precedence inequalities

Consider an s-t cut $(X, \bar{X})$ and an arc $e \in A$ with $\text{head}(e) \in X$. Clearly, each s-t path through e contains at least one arc f from $\delta^{out}(X)$, moreover, for a feasible s-t path (e, f) is compatible. Using this observation, for $i = 1, \dots, k$ one can define the sets

$$\phi_i^{out}(X, e) := \left\{ f \in \delta^{out}(X) : (e, f) \text{ is compatible with respect to } \mathbf{w}_i \right\}$$

and

$$F_i^{out}(X, e) := \left\{ f \in \phi_i^{out}(X, e) : \text{tail}(f) \neq \text{tail}(e) \text{ and } \text{head}(f) \neq \text{tail}(e) \right\}.$$

Then the *s-t cut precedence inequality with respect to arc e, cut $(X, \bar{X})$, and resource $\mathbf{w}_i$* is

$$x_e \leq x\left(F_i^{out}(X, e)\right), \tag{4.10}$$

and the *s-t cut precedence inequality with respect to arc e and cut $(X, \bar{X})$* is

$$x_e \leq x\left(\bigcap_{i=1}^{k} F_i^{out}(X, e)\right). \tag{4.11}$$

Analogously, for an s-t cut $(X, \bar{X})$, an arc $e \in A$ with $\text{tail}(e) \in \bar{X}$, and $i = 1, \dots, k$ one can define the sets

$$\phi_i^{in}(X, e) := \left\{ f \in \delta^{out}(X) : (f, e) \text{ is compatible with respect to } \mathbf{w}_i \right\},$$

and

$$F_i^{in}(X, e) := \left\{ f \in \phi_i^{in}(X, e) : \text{tail}(f) \neq \text{head}(e) \text{ and } \text{head}(f) \neq \text{head}(e) \right\}.$$

Then the *reverse s-t cut precedence inequality with respect to arc e, cut $(X, \bar{X})$, and resource $\mathbf{w}_i$* is

$$x_e \leq x\left(F_i^{in}(X, e)\right), \tag{4.12}$$

and the *reverse s-t cut precedence inequality with respect to arc e and cut $(X, \bar{X})$* is

$$x_e \leq x\left(\bigcap_{i=1}^{k} F_i^{in}(X, e)\right). \tag{4.13}$$

Garcia (2009) shows that inequalities (4.10), (4.11), (4.12), and (4.13) are valid for the RCSPP-polytope and can be separated by computing a minimum cut in a graph derived from D.

Subpath precedence inequalities

Let $\pi = (e_1, \ldots, e_{p-1})$ be a path in D, and let $v_i = \text{tail}(e_i)$ for all $i = 1, \ldots, p-1$, and $v_p = \text{head}(e_{p-1})$. Assume that π is an infeasible subpath with respect to $\mathbf{w}_i$, that is, $\omega_i(s, v_1) + \sum_{\ell=1}^{p-1} \mathbf{w}_i(e_\ell) + \omega_i(v_p, t) > \mathbf{L}_i$. If e_1 is an arc of a feasible s-t path $\pi^\star$, then $\pi^\star$ cannot contain all the arcs of π, and therefore, it must leave path π on some arc $f = (v_q, u)$ with $2 \leq q \leq v_{p-1}$ such that the path $(e_1, \ldots, e_{q-1}, f)$ is a feasible subpath with respect to $\mathbf{w}_i$. Using this observation, for $q = 2, \ldots, p-1$ one can define the sets

$$\phi_i^{out}(\pi, q) := \left\{ f \in \delta^{out}(v_q) : \omega_i(s, v_1) + \sum_{\ell=1}^{q-1} \mathbf{w}_i(e_\ell) + \mathbf{w}_i(f) + \omega_i(\text{head}(f), t) \leq \mathbf{L}_i \right\},$$

$$F_i^{out}(\pi, q) := \left\{ f \in \phi_i^{out}(\pi, q) : \text{head}(f) \neq v_1, \ldots, v_{q-1} \right\},$$

and

$$F_i^{out}(\pi) := \bigcup_{q=2}^{p-1} F_i^{out}(\pi, q).$$

Then the *subpath precedence inequality with respect to infeasible subpath π and resource $\mathbf{w}_i$* is

$$x_{e_1} \leq x\left(F_i^{out}(\pi)\right), \tag{4.14}$$

and the *subpath precedence inequality with respect to infeasible subpath π* is

$$x_{e_1} \leq x\left(\bigcap_{i=1}^{k} F_i^{out}(\pi)\right). \tag{4.15}$$

Analogously, for $q = 2 \ldots, p-1$ one can define the sets

$$\phi_i^{in}(\pi, q) := \left\{ f \in \delta^{in}(v_q) : \omega_i(s, \text{tail}(f)) + \mathbf{w}_i(f) + \sum_{\ell=q}^{p-1} \mathbf{w}_i(e_\ell) + \omega_i(v_p, t) \leq \mathbf{L}_i \right\},$$

$$F_i^{in}(\pi, q) := \left\{ f \in \phi_i^{in}(\pi, q) : \text{tail}(f) \neq v_{q+1}, \ldots, v_p \right\},$$

and

$$F_i^{in}(\pi) := \bigcup_{q=2}^{p-1} F_i^{in}(\pi, q).$$

Then the *reverse subpath precedence inequality with respect to infeasible subpath π and resource $\mathbf{w}_i$* is

$$x_{e_p} \leq x\left(F_i^{in}(\pi)\right), \tag{4.16}$$

and the *reverse subpath precedence inequality with respect to infeasible subpath π* is

$$x_{e_p} \leq x\left(\bigcap_{i=1}^{k} F_i^{in}(\pi)\right). \tag{4.17}$$

Garcia (2009) shows that inequalities (4.14), (4.15), (4.16), and (4.17) are valid for the RCSPP-polytope and provides a heuristic separation procedure for them, however, provides neither a polynomial time separation procedure, nor a proof that the separation problem is NP-hard.

4.3.2 New results

In this section we generalize the valid inequalities for the RCSPP-polytope described before.

Cut based inequalities

To define s-t cut precedence inequalities we considered an arc and an appropriate cut in the graph. Now, we will generalize these inequalities by considering a pair of arcs instead of a single one. Fix a pair of arcs $e, f \in A$ such that $\text{tail}(e) \neq \text{tail}(f)$, $\text{head}(e) \neq \text{head}(f)$, and $\text{head}(f) \neq \text{tail}(e)$. For an s-$\text{tail}(e)$ cut $(X, \bar{X})$ and $i = 1, \ldots, k$ we define the sets

$$\phi_i^1(X, e, f) := \{g \in \delta^{out}(X) : (g, e, f) \text{ is compatible with respect to } \mathbf{w}_i\}$$

and

$$F_i^1(X, e, f) := \left\{g \in \phi_i^1(X, e, f) : \text{tail}(g), \text{head}(g) \notin \{\text{head}(e), \text{tail}(f), \text{head}(f), t\}\right\},$$

and the inequalities

$$x_e + x_f - 1 \leq x\left(F_i^1(X, e, f)\right) \tag{4.18}$$

and

$$x_e + x_f - 1 \leq x\left(\bigcap_{i=1}^{k} F_i^1(X, e, f)\right). \tag{4.19}$$

Proposition 4.2. *If D contains no path from* $\text{head}(f)$ *to* $\text{tail}(e)$*, then inequality* (4.19) *(and thus inequality* (4.18)*) is valid for the* RCSPP*-polytope.*

Proof. Consider any feasible s-t path π, and let $x^\pi \in \{0,1\}^A$ be its characteristic vector, i.e., $x_g^\pi = 1$ if π contains arc g, otherwise $x_g^\pi = 0$. If π does not contain any of the edges $\{e, f\}$ then the left-hand side of (4.19) is at most 0, while the right-hand side is at least 0 by the non-negativity of x^π, thus the claim follows. So, suppose π passes through both of e and f, i.e., $x_e^\pi = x_f^\pi = 1$. We claim that the right-hand side of (4.19) is at least 1. It suffices to verify that $x_g^\pi = 1$ for some arc $g \in F_i^1(X, e, f)$. Since D contains no directed path from $\text{head}(f)$ to $\text{tail}(e)$, π passes through e first, and then through f, thus (g, e, f) is compatible for all the arcs g on the subpath π' from s to $\text{tail}(e)$, and clearly, π' contains no nodes from $\{\text{head}(e), \text{tail}(f), \text{head}(f), t\}$. Since $(X, \bar{X})$ is an s-$\text{tail}(e)$ cut, at least one edge g of π' must belong to $\delta^{out}(X)$, and the claim is verified. □

Now we give an example showing that a member of this new class of inequalities (4.18) may be violated, while all s-t cut precedence inequalities (4.10) and (4.12) are satisfied.

Example 4.1. *Consider the* RCSPP *instance in Fig. 4.1. There is a single resource function with limit equals to 8, and the weights are indicated on the arcs. The only infeasible s-t path $\psi = (s, v_2, v_4, t)$ is not cut off by any s-t cut precedence inequalities, but the inequality* (4.18) *for arcs (v_2, v_4) and (v_4, t), and cut $(X, \bar{X})$ with $X = \{s\}$, i.e., $x_{(v_2,v_4)} + x_{(v_4,t)} - 1 \leq x_{(s,v_1)}$ is violated by the incidence vector of ψ.*

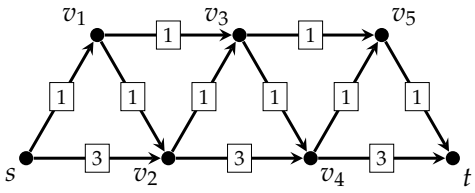


Figure 4.1: [Example 4.1] RCSPP instance with a single resource

Analogously, for a head(e)-tail(f) cut $(X, \bar{X})$, and $i = 1, \ldots, k$ we define the sets

$$\phi_i^2(X, e, f) := \{g \in \delta^{out}(X) : (e, g, f) \text{ is compatible with respect to } \mathbf{w}_i\}$$

and

$$F_i^2(X, e, f) := \left\{g \in \phi_i^2(X, e, f) : \text{tail}(g), \text{head}(g) \notin \{s, \text{tail}(e), \text{head}(f), t\}\right\},$$

and the inequality

$$x_e + x_f - 1 \leq x\left(\bigcap_{i=1}^{k} F_i^2(X, e, f)\right). \tag{4.20}$$

Finally, for a head(f)-t cut $(X, \bar{X})$, and $i = 1, \ldots, k$ we define the sets

$$\phi_i^3(X, e, f) := \{g \in \delta^{out}(X) : (e, f, g) \text{ is compatible with respect to } \mathbf{w}_i\}$$

and

$$F_i^3(X, e, f) := \left\{g \in \phi_i^3(X, e, f) : \text{tail}(g), \text{head}(g) \notin \{s, \text{tail}(e), \text{head}(e), \text{tail}(f)\}\right\},$$

and the inequality

$$x_e + x_f - 1 \leq x\left(\bigcap_{i=1}^{k} F_i^3(X, e, f)\right). \tag{4.21}$$

Similarly to the proof of Proposition 4.2, one can prove the following proposition.

Proposition 4.3. *If D contains no path from* head(f) *to* tail(e)*, then inequalities* (4.20) *and* (4.21) *are valid for the* RCSPP*-polytope.*

Separation of the inequalities. Let $\bar{x}$ be a fractional solution of the LP-relaxation of (4.1)–(4.5) (possibly augmented by some valid inequalities, and in which some variables may be fixed to 0 or 1 due to preprocessing, branching, or variable fixing). In order to find an inequality (4.19) violated by $\bar{x}$, we will search for each pair of arcs a maximally violated inequality over all cuts, if any, by calculating a minimum cut in an appropriate graph. So, fix a pair of arcs (e, f) such that there is no directed path from head(f) to tail(e) in D. First, we define a capacity function $\kappa : A \to \mathbb{R}_{\geq 0}$ as follows. Let $\kappa(g) := \bar{x}_g$ for each arc g such that (g, e, f) is compatible, and tail(g), head$(g) \notin \{\text{head}(e), \text{tail}(f), \text{head}(f), t\}$; otherwise let $\kappa(g) := 0$. Then, we determine a minimum capacity s-tail(e) cut $(X, \bar{X})$ with respect to κ, and if $\bar{x}_e + \bar{x}_f - 1 > \kappa(X)$, then

a violated inequality is found. It is easy to see that this procedure is of polynomial time for a fixed pair of arcs, and since the number of such pairs is $O(|A|^2)$, one can separate the class of inequalities (4.19) in polynomial time. Similarly, one can separate the classes of inequalities (4.20) and (4.21).

Infeasible subpath based inequalities

Let $\pi = (e_1, \dots, e_{p-1})$ be an infeasible subpath in D with $p \geq 4$, and let $v_q = \text{tail}(e_q)$ for $q = 1, \dots, p-1$ and $v_p = \text{head}(e_{p-1})$. Clearly, any feasible s-t path visiting e_1 and e_{p-1} in this order must leave π on some arc $f = (v_q, u)$ with $2 \leq q \leq p-2$, otherwise it would contain all the arcs of an infeasible subpath. Moreover, for such and arc, tuple $(e_1, \dots, e_{q-1}, f, e_{p-1})$ must be compatible. Therefore, for all $q = 2, \dots, p-2$, and $i = 1, \dots, k$ we define the sets

$$\gamma_i^{out}(\pi, q) := \left\{ f \in \delta^{out}(v_q) \setminus \{e_q\} : (e_1, \dots, e_{q-1}, f, e_{p-1}) \text{ is compatible with respect to } \mathbf{w}_i \right\}$$

$$G_i^{out}(\pi, q) := \left\{ f \in \gamma_i^{out}(\pi, q) : \text{head}(f) \neq v_1, \dots, v_{q-1} \text{ and } \text{head}(f) \neq v_p \right\},$$

and

$$G_i^{out}(\pi) := \bigcup_{q=2}^{p-2} G_i^{out}(\pi, q),$$

and the inequalities

$$x_{e_1} + x_{e_{p-1}} - 1 \leq x\left(G_i^{out}(\pi)\right) \tag{4.22}$$

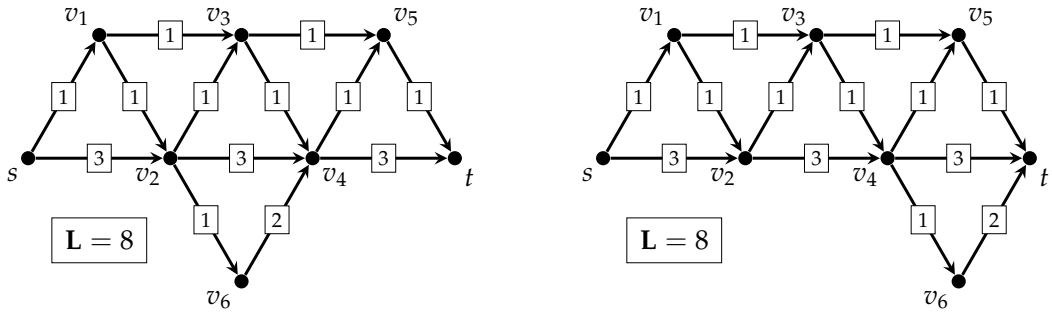
and

$$x_{e_1} + x_{e_{p-1}} - 1 \leq x\left(\bigcap_{i=1}^{k} G_i^{out}(\pi)\right). \tag{4.23}$$

Proposition 4.4. *If D contains no directed path from v_p to v_1, then inequality (4.23) (and thus inequality (4.22)) is valid for the* RCSPP*-polytope.*

Proof. Consider any feasible s-t path ψ, and let $x^\psi \in \{0,1\}^A$ be its characteristic vector. If ψ does not contain any of the edges $\{e_1, e_{p-1}\}$ then the left-hand side of (4.22) is at most 0, while the right-hand side is at least 0 by the non-negativity of x^ψ, thus the claim follows. So, suppose ψ passes through both of e_1 and e_{p-1}, i.e., $x^\psi_{e_1} = x^\psi_{e_{p-1}} = 1$. We claim that the right-hand side of (4.18) is at least 1. It suffices to verify that $x^\psi_f = 1$ for some arc $f \in \bigcap_{i=1}^k G_i^{out}(\pi)$. Since D contains no directed path from v_p to v_1, ψ passes through e_1 first, and then through e_{p-1}, and thus ψ cannot contain all the arcs $e_2, \dots, e_{p-2}$ since $\pi = (e_1, \dots, e_{p-1})$ is an infeasible subpath. Hence, ψ must contain an arc emanating from one of the nodes $v_2, \dots, v_{p-2}$, so let $f = (v_q, u)$ be the first emanating arc for some $2 \leq q \leq p-2$, i.e., $f \neq e_q$. Since ψ is a path, $u \neq v_1, \dots, v_{q-1}$ and $u \neq v_p$, and since ψ is feasible, thus $f \in G_i^{out}(\pi, q) \subseteq G_i^{out}(\pi)$ for all $i = 1, \dots, k$, and the statement of the proposition is proved. □

The following two examples show that neither the subpath precedence inequalities, nor the infeasible subpath based inequalities dominate the other.



(a) Instance of Example 4.2 (b) Instance of Example 4.3

Figure 4.2: [Examples 4.2 and 4.3] RCSPP instances with a single resource

Example 4.2. *Consider the* RCSPP *instance depicted in Fig. 4.2a. There is single resource with limit equals to 8, and there are two infeasible subpaths:* $\pi_1 = (s, v_2, v_4, t)$ *and* $\pi_2 = (s, v_2, v_6, v_4, t)$*. The subpath precedence inequalities (4.14) for* π_1 *and* π_2*, and the infeasible subpath based inequalities (4.22) for* $\{\pi_1, \pi_2\}$ *are: (i)* $x_{(s,v_2)} \leq x_{(v_2,v_3)} + x_{(v_2,v_6)} + x_{(v_4,v_5)}$*, (ii)* $x_{(s,v_2)} \leq x_{(v_2,v_3)} + x_{(v_2,v_4)} + x_{(v_4,v_5)}$*, and (iii)* $x_{(s,v_2)} + x_{(v_4,t)} \leq x_{(v_2,v_3)} + 1$*, respectively. Inequality (iii) excludes both* π_1 *and* π_2*, however, (i) excludes only* π_1*, and (ii) excludes only* π_2*.*

Example 4.3. *Consider the* RCSPP *instance depicted in Fig. 4.2b. There is single resource with limit equals to 8, and there are three infeasible subpaths:* $\pi_1 = (s, v_2, v_4, t)$*,* $\pi_2 = (s, v_2, v_4, v_6)$ *and* $\pi_3 = (s, v_2, v_4, v_6, t)$*. The subpath precedence inequalities (4.14) for* $\{\pi_1, \pi_2, \pi_3\}$*, and the infeasible subpath based inequalities (4.22) for* π_1*,* π_2*, and* π_3 *are: (i)* $x_{(s,v_2)} \leq x_{(v_2,v_3)} + x_{(v_4,v_5)}$*, (ii)* $x_{(s,v_2)} + x_{(v_4,t)} \leq x_{(v_2,v_3)} + 1$*, (iii)* $x_{(s,v_2)} + x_{(v_4,v_6)} \leq x_{(v_2,v_3)} + 1$*, and (iv)* $x_{(s,v_2)} + x_{(v_6,t)} \leq x_{(v_2,v_3)} + 1$*, respectively. Inequality (i) excludes all of* π_1*,* π_2*, and* π_3*, however, (ii), (iii), and (iv) only excludes* π_1*,* π_2 *and* π_3*, respectively.*

Similarly to the previous case, for all $q = 3, \ldots, p-1$, and $i = 1, \ldots, k$ we define the sets

$$\gamma_i^{in}(\pi, q) := \left\{ f \in \delta^{in}(v_q) \setminus \{e_{q-1}\} : (e_1, f, e_q, \ldots, e_{p-1}) \text{ is compatible with respect to } \mathbf{w}_i \right\},$$

$$G_i^{in}(\pi, q) := \left\{ f \in \gamma_i^{in}(\pi, q) : \text{tail}(f) \neq v_{q+1}, \ldots, v_p \text{ and } \text{tail}(f) \neq v_1 \right\},$$

and

$$G_i^{in}(\pi) := \bigcup_{q=3}^{p-1} G_i^{in}(\pi, q),$$

and the inequality

$$x_{e_1} + x_{e_{p-1}} - 1 \leq x\left(\bigcap_{i=1}^{k} G_i^{in}(\pi) \right). \tag{4.24}$$

Similarly to Proposition 4.4 one can prove the following proposition.

Proposition 4.5. *If* D *contains no directed path from* v_p *to* v_1*, then inequality (4.24) is valid for the* RCSPP*-polytope.*

Separation of inequalities. We show that separating the class of subpath precedence inequalities (4.14) and the infeasible subpath based inequalities (4.23) are NP-hard.

Problem 4.1 (SPP-SEP)**.** *Given an instance of* RCSPP *and a feasible solution $\bar{x}$ for the* LP*-relaxation of* (4.1)–(4.5) *(possibly augmented by some valid inequalities for* RCSPP*, and in which some variables may be fixed to 0 or 1 due to preprocessing, branching, or variable fixing). Is there an infeasible subpath $\pi = (e_1, \dots, e_{p-1})$ with $p \geq 4$ such that $\bar{x}_{e_1} > \bar{x}(F_i^{out}(\pi))$ for some $i \in \{1, \dots, k\}$?*

Theorem 4.1. SPP-SEP *is* NP*-hard.*

Proof. Clearly, it is sufficient to consider the single-resource case. We reduce the Binary Knapsack Problem (KP) to SPP-SEP. Recall that an instance of (the decision version of) KP is given by a set of n items, each with a profit $p_i \in \mathbb{Z}_{\geq 0}$ and a weight $a_i \in \mathbb{Z}_{\geq 0}$, a capacity value $B \in \mathbb{Z}_{>0}$, and a desired profit $P \in \mathbb{Z}$; and the question is whether a subset J of items exists such that $p(J) > P$ and $a(J) < B$ (cf. (Garey and Johnson, 1979, Problem MP9)).

Given an instance of KP. Without loss of generality we may assume that $1 \leq a_1 \leq \ldots \leq a_n \leq B$, and $1 \leq p_i \leq P$ for all $i = 1, \dots, n$. Let

$$\bar{a}_i := \frac{a_i}{a_n + B} \text{ for all } i = 1, \dots, n, \ \bar{a}_0 := 0, \ \text{ and } \ \bar{B} := \frac{B}{a_n + B}.$$

Since $\bar{B} < 1$, by multiplying values p_i and P by a suitable integer, we may assume that

$$\bar{B} \leq \frac{P + p_{sum} + 1}{P + p_{sum} + 2} \tag{4.25}$$

holds, where $p_{sum} := \sum_{i=1}^{n} p_i$.

We create an acyclic directed graph $\bar{D}$ with node set $V(\bar{D}) = \{v_0, \dots, v_{n+2}\}$. Every pair of nodes (v_i, v_{i+1}), $i = 1, \dots, n+1$ is connected by two arcs: $e^+_{i,i+1}$ and $e^0_{i,i+1}$. Furthermore, for all $i = 1, \dots, n$ there is an arc from v_0 to v_i: $e^0_{0,i}$. Finally, there is an arc $e^+_{0,1}$ from v_0 to v_1, and an arc $e^+_{1,n+2}$ from v_1 to v_{n+2}. Let $s = v_0$ and $t = v_{n+2}$, we define arc-weights $\mathbf{w}$, and a feasible solution $\bar{x}$ for the LP-relaxation of (4.1)–(4.5) as follows:

$$\mathbf{w}(e) := \begin{cases} p_i & \text{if } e = e^+_{i,i+1} \ (i = 1, \dots, n), \\ p_{sum} + 1 & \text{if } e = e^+_{0,1} \text{ or } e = e^+_{n+1,n+2}, \\ P + p_{sum} + 2 & \text{if } e = e^+_{1,n+2}, \\ 0 & \text{otherwise} \end{cases}$$

$$\bar{x}_e := \begin{cases} \bar{a}_i & \text{if } e = e^0_{i,i+1} \ (i = 1, \dots, n), \\ \bar{a}_n & \text{if } e = e^0_{n+1,n+2}, \\ \bar{a}_i - \bar{a}_{i-1} & \text{if } e = e^0_{0,i} \ (i = 1, \dots, n), \\ \bar{B} & \text{if } e = e^+_{0,1} \text{ or } e = e^+_{1,n+2}, \\ 0 & \text{otherwise} \end{cases}$$

and finally let $\mathbf{L} := P + 2p_{sum} + 2$. In Fig. 4.3 we depict the sketch of the constructed graph where the weights and solution values are indicated on the arcs. Clearly, $\bar{x}$ is a feasible solution

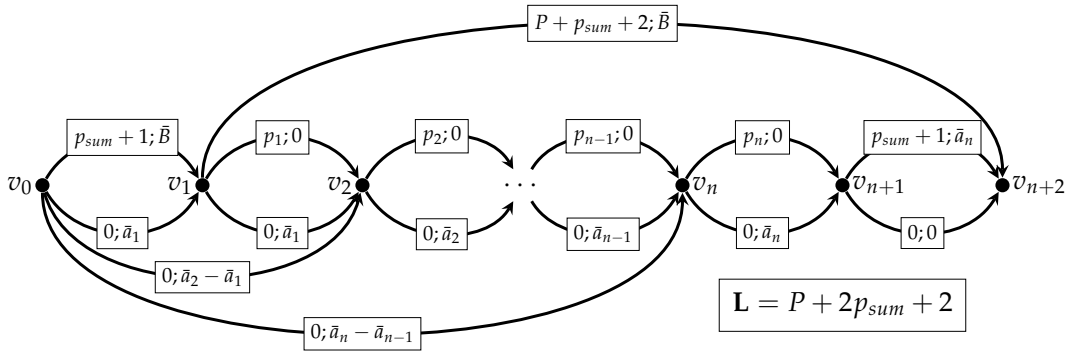


Figure 4.3: [Proof of Theorem 4.1] Reducing KP to SPP-SEP

for the LP-relaxation of (4.1)–(4.5) since it is an s-t flow of value $\bar{a}_n + \bar{B} = 1$, and the resource limit is satisfied due to (4.25), that is, $\sum_{e \in E} \mathbf{w}(e)\bar{x}_e = (p_{sum} + 1)(\bar{a}_n + \bar{B}) + (P + p_{sum} + 2)\bar{B} \leq P + 2p_{sum} + 2$. We claim there exists a solution for KP if and only if there exists a solution for SPP-SEP in $\bar{D}$.

First, suppose that SPP-SEP admits a solution, and let $\pi = (e_1, \ldots, e_{p-1})$ be an infeasible subpath with $p \geq 4$ such that $\bar{x}(F^{out}(\pi)) < \bar{x}_{e_1}$. Note that π does not contain arc $e^+_{1,n+2}$ since it cannot appear in an infeasible subpath with minimum length 3. Clearly, $\omega(s, v_i) = \omega(v_i, t) = 0$ for all $i = 1, \ldots, n+2$, and $\sum_{i=1}^{n} \max\{\mathbf{w}(e^+_{i,i+1}), \mathbf{w}(e^0_{i,i+1})\} = p_{sum}$, hence an infeasible subpath with minimum length 3 contains both of the arcs $e^+_{0,1}$ and $e^+_{n+1,n+2}$, and thus π is an s-t path. Now we determine $F^{out}(\pi)$. First, notice that $e^+_{1,n+2} \notin F^{out}(\pi)$, since $\mathbf{w}(e^+_{0,1}) + \mathbf{w}(e^+_{1,n+2}) = P + p_{sum} + 3 > \mathbf{L}$. Second, notice that $\mathbf{w}(\pi_{s,v_i}) \leq 2p_{sum} + 1$ holds for the subpath π_{s,v_i} of π from s to v_i for all $i = 1, \ldots, n+1$, $\mathbf{w}(e^0_{i,i+1}) = 0$, $\mathbf{w}(e^+_{i,i+1}) = p_i$, and $\omega(v_{i+1}, t) = 0$, thus if π contains an arc $e^+_{i,i+1}$ for some $i \in \{1, \ldots, n+1\}$, then $F^{out}(\pi)$ comprises arc $e^0_{i,i+1}$ and if π contains an arc $e^0_{i,i+1}$ for some $i \in \{1, \ldots, n\}$, then $F^{out}(\pi)$ comprises arc $e^+_{i,i+1}$. Therefore, let $I := \{i \in \{1, \ldots, n\} : e^+_{i,i+1} \in \pi\}$ and we have

$$F^{out}(\pi) = \left(\bigcup_{i \in I} \{e^0_{i,i+1}\} \right) \cup \left(\bigcup_{i \notin I} \{e^+_{i,i+1}\} \right) \cup \{e^0_{n+1,n+2}\},$$

and $\bar{x}(F^{out}(\pi)) = \bar{a}(I)$. Thus, if π is a solution for SPP-SEP, i.e., $\bar{x}(F^{out}(\pi)) < \bar{x}(e^+_{0,1})$, that is, $\bar{a}(I) < \bar{B}$, then $a(I) < B$ and thus I is a solution for KP, since $p(I) = \mathbf{w}(\pi) - \mathbf{w}(e^+_{0,1}) - \mathbf{w}(e^+_{n+1,n+2}) > P$.

Conversely, let I be a solution for KP. We define path π with arc set

$$\left\{ e^+_{0,1}, e^+_{n+1,n+2} \right\} \cup \left(\bigcup_{i \in I} \{e^+_{i,i+1}\} \right) \cup \left(\bigcup_{i \notin I} \{e^0_{i,i+1}\} \right).$$

Similarly to the previous case, it is easy to verify that $F^{out}(\pi) = \left(\bigcup_{i \in I} \{e^0_{i,i+1}\} \right) \cup \left(\bigcup_{i \notin I} \{e^+_{i,i+1}\} \right) \cup \{e^0_{n+1,n+2}\}$, and inequality (4.14) is violated by $\bar{x}$. □

Problem 4.2 (ISPB-SEP). *Given an instance of* RCSPP, *a feasible solution* $\bar{x}$ *for the* LP*-relaxation of* (4.1)–(4.5) *(possibly augmented by some valid inequalities for* RCSPP, *and in which some variables may be fixed to 0 or 1 due to preprocessing, branching, or variable fixing), and two arcs* $e, f \in A$. *Is there an infeasible subpath* $\pi = (e_1, \ldots, e_{p-1})$ *with* $p \geq 5$ *and* $e_1 = e$, $e_{p-1} = f$ *such that* $\bar{x}_{e_1} + \bar{x}_{e_{p-1}} - 1 > \bar{x}(G^{in}(\pi))$ *for some* $i \in \{1, \ldots, k\}$*?*

Theorem 4.2. ISPB-SEP *is* NP*-hard.*

Proof. The construction is almost the same as that in the proof of Theorem 4.1. To be suitable for the present claim, extend the graph $\bar{D}$ with a new node v_{n+2} and a new arc $e^0_{n+2,n+3}$ from v_{n+2} to v_{n+3}. The weight of this the new arc is 0, and let $\bar{x}_{e^0_{n+2,n+3}} = 1$. Accordingly, the new destination node is $t = v_{n+3}$, and the source node remains $s = v_0$. Let $e = e^+_{0,1}$ and $f = e^0_{n+2,n+3}$.

One may verify that KP problem admits a solution I if and only if there is an infeasible subpath π with length at least 4, with $e^+_{0,1}$ as the first edge, $e^0_{n+2,n+3}$ as the last edge, and $\bar{x}_{e^+_{0,1}} + \bar{x}_{e^0_{n+2,n+3}} - 1 > \bar{x}(G^{in}(\pi))$. □

4.4 Computational results: Evaluation of cutting planes and heuristics

In this section we summarize our computational experiments, where the main goals were

- to show that some of the new cutting planes can improve the performance of a branch-and-bound procedure for solving RCSPP,
- to assess the effectiveness of the new preprocessing and heuristic algorithms, and
- to find the best combination of the various techniques for solving hard instances.

4.4.1 Test environment and implementation

All the computational experiments were performed on a workstation with 4GB RAM, and XEON X5650 CPU of 2.67 GHz, and under Linux operating system using a single thread only. Our procedure is implemented in C++ programming language using FICO Xpress Solver (Xpress, version 28.01.04) as a branch-and-cut framework, and Library for Efficient Modeling and Optimization in Networks (Lemon, version 1.3.1) to handle graphs and to perform graph algorithms.

4.4.2 Preliminary experiments

We performed experiments on several widely-used instance sets from the literature (see Table 4.1), where we applied DB-preprocessing on the instances to reduce the underlying graph, then we solved them with Xpress applying its default settings. We summarize these results in Table 4.2, where for each instance set we indicate the number of instances (#*Instances*) in

Table 4.1: Properties of instance sets

Instance set		#Resources	#Nodes		#Arcs	
			Min	Max	Min	Max
I1	Beasley and Christofides (1989)	1	100	500	955	4 978
I2	Beasley and Christofides (1989)	10	100	500	990	4 868
D1	Dumitrescu and Boland (2003)	1	10 002	135 002	29 900	404 850
D2	Dumitrescu and Boland (2003)	1	625	40 000	2400	159 200
S	Santos et al. (2007)	1	10 000	40 000	15 000	800 000

Table 4.2: Average results on instance sets

Set	#Instances	Preprocessing		Xpress	
		Time	#Solved	#Nodes	Time
I1	12	0.01	12	-	-
I2	12	0.03	10	1.0	0.45
D1	8	0.38	0	1.0	179.28
D2	56	0.12	1	3.0	22.96
S	880	0.12	878	2.0	160.71

the set, the average preprocessing time in seconds (*Preprocessing/Time*) over the instance set, the number of instances solved optimally in the preprocessing phase (*#Solved*), the average execution time (*Branch-and-cut/Time*) in seconds, and the average number of investigated enumeration tree nodes (*#Nodes*) of the branch-and-cut procedure over the instance set.

One can notice that most of these instances are solved optimally in the preprocessing phase, and the rest of the instances are solved with Xpress investigating a small number of enumeration tree nodes. Thus, these instances cannot be used to challenge a branch-and-cut procedure, and therefore, we decided to generate our own instances using the similar generation method as in (Garcia, 2009) that we describe next.

4.4.3 Instances

To construct a directed graph we used a method similar to that of (Beasley and Christofides, 1989). Let n be the number of desired nodes, and denote $V = \{1, 2, \ldots, n\}$ the set of nodes with $s = 1$ and $t = n$. For all $i = 1, \ldots, n-1$ and for all $j = i+1, \ldots, \min\{n, i + \lfloor n/4 \rfloor\}$ we randomly include arc (i, j) with a probability such that the expected value of the number of arcs is $10n$. Since for all arcs (i, j), $j - i \leq n/4$, every s-t path consisted of at least 4 arcs. Clearly, the generated graphs are directed, acyclic, and do not contain parallel arcs.

In each of our instances all arc weights and all arc costs are integer. The weights were

Table 4.3: Summary of instance classes (each instance contains 10 resources)

Resource limit type		Class	#Nodes	Class	#Nodes	Class	#Nodes
Reducing	$p = 20$	G1	500	G5	1000	G9	1500
Uniform	$W = 20$	G2	500	G6	1000	G10	1500
Uniform	$W = 30$	G3	500	G7	1000	G11	1500
Uniform	$W = 40$	G4	500	G8	1000	G12	1500

uniformly and independently generated from $[0, 5]$, and arc costs were uniformly and independently generated from $[-5, 0]$. To create resource limits we used two different methods. The first one is similar to that in (Beasley and Christofides, 1989), that is, we searched a minimal cost s-t path and computed its resource consumptions. The resource limits were derived from these values, reduced by a given percentage p. In the second method we chose a fixed uniform limit W for all resources, like in (Garcia, 2009).

We generated 20 graphs for each $n \in \{500, 1000, 1500\}$. For each graph we generated a cost function and 10 resource functions, and then we derived four instances by the four ways of setting the resource limits. That is, we used the reducing method with $p = 20\%$, and the other 3 instances had uniform resource limits with $W = 20$, 30, and 40, respectively. Since every s-t path consists of at least 4 arcs, and the maximum arc weight is 5, each RCSPP instance with uniform resource limits has a feasible solution.

In summary, we created $240 = 20 \times 3 \times 4$ RCSPP instances which can be grouped into 12 classes according to their sizes, and the method used to generate their resource limits as we summarized in Table 4.3. For detailed information about the instances we refer to (Horváth and Kis, 2016b, Tables A1-A3).

In all of the following experiments, before building the ILP model, we applied the DB- and the G-preprocessing procedure to reduce the size of the underlying graph (see Section 4.2.1).

4.4.4 Experiments with heuristics and variable fixing procedures

In this section we present the results of the experiments with primal heuristics and variable fixing procedures described in Section 4.2.2. Our purpose was to investigate how these methods can improve a simple branch-and-bound procedure, thus in these experiments we turned off every Xpress presolving and heuristic methods, and we forbade Xpress to add any built-in cutting plane to the problem.

In these experiments we compared five scenarios corresponding to the settings summarized in Table A.1 of Appendix A. Method *BnB (Arcfix)* refers to the use of the variable fixing method of Garcia (2009) at the root node. In case of method *BnB (Varfix)* we used our variable fixing method in every enumeration tree node before solving the LP-relaxation, while in the case of method *BnB (Heuristics)* we used our primal heuristics in every enumeration tree node

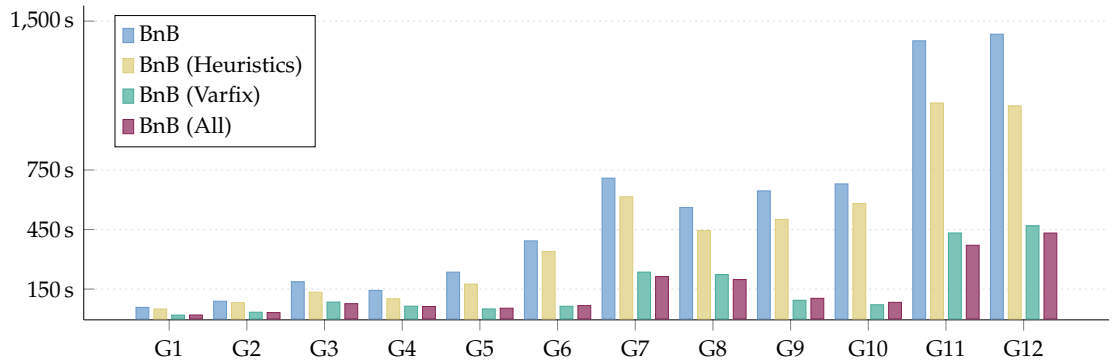


Figure 4.4: Results of experiments with primal heuristics and variable fixing procedures

after an optimal solution for the LP-relaxation had been found. Method *BnB (All)* refers to the use of all these components. Remind that in each case DB- and G-preprocessing were applied on the input graph.

The summary of these experiments can be found in Table A.2 of Appendix A, however, in Fig. 4.4 we depict the average execution times in seconds over the instance sets for the branch-and-bound method and the best performing methods.

The most successful method among the aboves is definitely our variable fixing technique. Method BnB (Varfix) can reduce the total time by $54,8 - 89,6\%$ with respect to the plain branch-and-bound method BnB. Moreover, in most of the cases we obtained the best results by method BnB (All), that is, when we combined all of the variable fixing and heuristic search techniques.

4.4.5 Experiments with cutting planes based on *s*-*t* cuts

In this section we summarize the experiments with inequalities described in Section 4.3.1 and Section 4.3.2. Our purpose was to compare the performance of the *s*-*t* cut precedence inequalities and that of our generalized inequalities. Again, we used the plain setting of Xpress, that is, we turned off every Xpress presolving and heuristic methods, and we forbade Xpress to add any built-in cutting plane to the problem. Moreover, in these experiments we gave the optimal solution value to the solver and set it as a cutoff value.

In these experiments we compared three scenarios corresponding to the settings summarized in Table A.1 of Appendix A. Method *BnC (STCP)* refers to the use of the *s*-*t* cut precedence inequalities (4.11) and (4.13), while our cut based inequalities (4.19), (4.20), and (4.21) were used in method *BnC (CB)*. In the case of method *BnC (STCP+CB)* we used all the inequalities mentioned above. Remind that in each case DB- and G-preprocessing were applied on the input graph. Cuts were generated in each enumeration tree node with depth at most 8 in one round, except the root node where we separate inequalities in 20 rounds.

The summary of the experiments can be found in Table A.3 of Appendix A, however, in Fig. 4.5 we depict the average execution times in seconds over the instance sets of the methods.

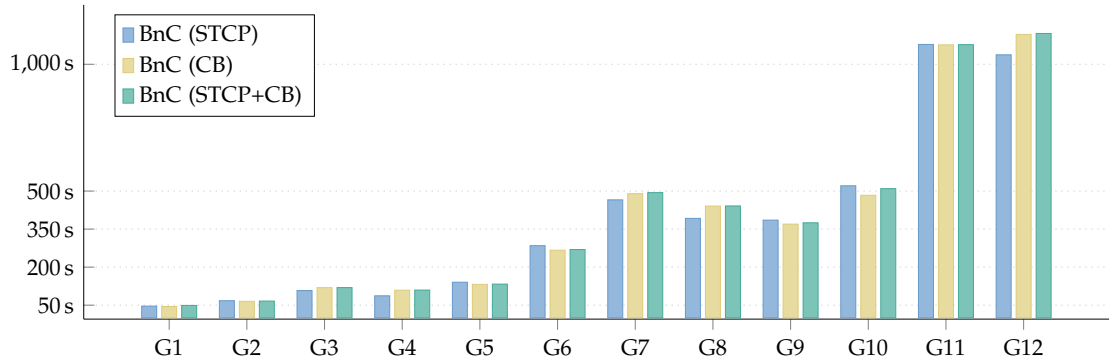


Figure 4.5: Results of experiments with cutting planes based on cuts

The results are diversified, however, we observe that the efficiency of the methods depends on the types of the instances rather than their sizes. That is, for all problem sizes, for resource limit types Reducing(20) and Uniform(20) (i.e., classes G1, G2, G5, G6, G9, G10) we obtained the best results by method BnC (CB), furthermore, for resource limit types Uniform(30) and Uniform(40) (i.e., classes G3, G4, G7, G8, G11, G12) method BnC (STCP) gave the best results in almost all cases. One of the reasons for this is that RCSPP instances with resource limit types Uniform(30) and Uniform(40) contain a few incompatible arc pairs, thus very few inequalities can be generated, however, the separation of the cut based inequalities are more expensive in total than the separation of the *s-t* cut precedence inequalities.

4.4.6 Experiments with cutting planes based on infeasible subpaths

In this section we summarize the experiments with cutting planes based on infeasible subpaths described in Section 4.3.1 and Section 4.3.2. Our purpose was to compare the performance of the subpath precedence inequalities and that of our generalized inequalities. Again, we used the plain setting of Xpress, that is, we turned off every Xpress presolving and heuristic methods, and we forbade Xpress to add any cutting plane of its own to the problem. Moreover, in these experiments we gave the optimal solution value to the solver and set it as a cutoff value.

In these experiments we compared three scenarios corresponding to the settings summarized in Table A.1 of Appendix A. Method *BnC (SPP)* refers to the use of the strengthened subpath precedence inequalities (4.15) and (4.17), while our infeasible subpath based inequalities (4.23) and (4.24) were used in the case of method *BnC (ISPB)*. In the case of method *BnC (SPP+ISPB)* we used all the inequalities mentioned above. Remind that in each case DB- and G-preprocessing were applied on the input graph. Cuts were generated in each enumeration tree node with depth at most 8 in one round, except the root node where we separate inequalities in 20 rounds.

The summary of these experiments can be found in Table A.4 of Appendix A, however, in Fig. 4.6 we depict the average execution times in seconds over the instance sets of the methods.

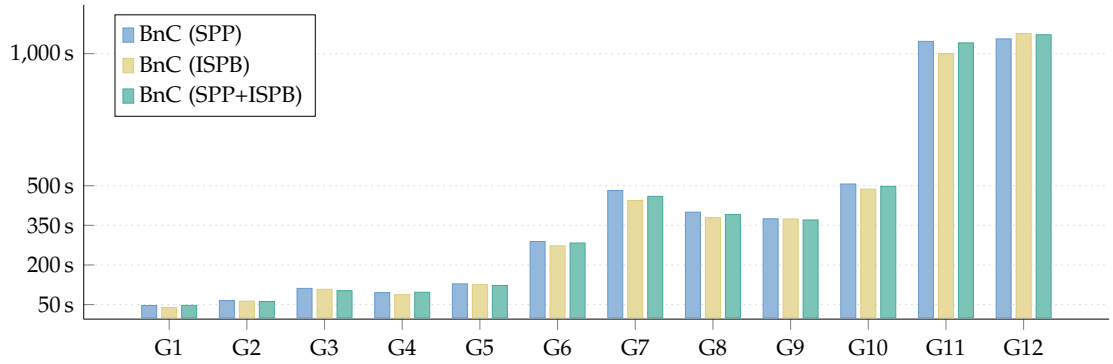


Figure 4.6: Results of experiments with cutting planes based on infeasible subpaths

We can observe that in almost all cases BnC (ISPB) or BnC (SPP+ISPB) proved to be the most effective method, except on the instances in class G12, where BnC (SPP) was the winner both in computation time and number of enumeration tree nodes explored. The reason for this is that the infeasible subpath based inequalities could be generated in a much greater number than the subpath precedence inequalities.

4.4.7 Experiments with combination of various components

In the experiments presented below we combined the various components to find the best way of using them together for solving hard instances. We report only on the most successful combinations.

Method *BnC (XC)* refers to the default Xpress settings, however, we turned off every presolving methods. In these experiments we compared four scenarios corresponding to the settings summarized in Table A.1 of Appendix A. In the case of methods *BnC (GC)* and *BnC (NC)* we used the plain setting of Xpress, that is, we turned off every Xpress presolving and heuristic methods, and we forbade Xpress to add any cutting plane of its own to the problem. In the former case we separated the *s-t* cut precedence and the subpath precedence inequalities, and the latter case we separated the *s-t* cut precedence, the cut based, and the infeasible subpath based inequalities. Method *BnC (XC+NC)* is the extension of method *BnC (XC)*, where we separated the same class of inequalities as in method *BnC (NC)*. Remind that in each case DB- and G-preprocessing were applied on the input graph.

The summary of these experiments can be found in Table A.5 of Appendix A, however, in Fig. 4.4 we depict the average execution times in seconds over the instance sets for the branch-and-bound method and the best performing methods.

We can observe that for all problem sizes, and all types of resource limits, we obtained the best results (in term of solving time) either by the method BnC (GC) or BnC (NC), and Xpress cuts and heuristics does not improve the performance of the branch-and-cut procedure.

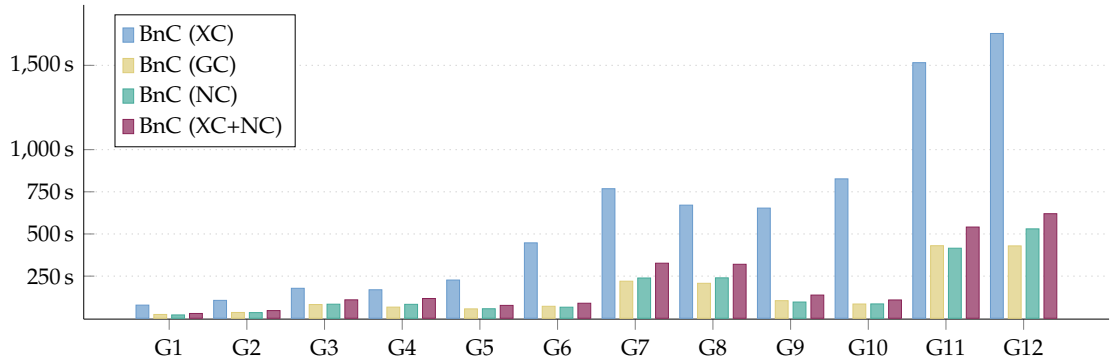


Figure 4.7: Results of experiments with combination of various components

4.4.8 Conclusions

The above tests suggest that our primal heuristics and mainly our variable fixing method can significantly reduce the execution time of a branch-and-bound procedure (Section 4.4.4). We can also see that both of our cutting planes and the cutting planes from literature can reduce the computation times and the number of enumeration tree nodes of a plain branch-and-bound procedure (Section 4.4.5 and Section 4.4.6). However, if we compare the results of the combined experiments with the results of the heuristics and variable fixing experiments, we can conclude that adding cutting planes on top of heuristics and variable fixing methods does not improve, and in most cases degrades the overall performance (cf. Section 4.4.4 and Section 4.4.7).

4.5 Computational results: Comparison with state-of-the-art methods

In this section we present our computational experiments, where the main goal was to compare our branch-and-cut approach with other approaches from the literature, namely the Reference Point Method of Pugliese and Guerriero (2013a), and the Pulse Algorithm of Lozano and Medaglia (2013).

4.5.1 Test environment and implementation

All the computational experiments were performed on a notebook with Intel Core i7-4710MQ, 2.5 GHz CPU, under Windows 7. Our procedure is implemented in C++ programming language using FICO Xpress Solver (Xpress, version 28.01.04) as a branch-and-cut framework, and Library for Efficient Modeling and Optimization in Networks (Lemon, version 1.3.1) to handle graphs and to perform graph algorithms.

The solution method of Pugliese and Guerriero (2013a) has been implemented in Java programming language and tested by using a PC with Intel Core i7-620M, 2.67 GHz CPU,

under Windows 7, while the solution approach of Lozano and Medaglia (2013) has been implemented in Java programming language and tested by using a PC with Intel Core 2 Duo P8600, 2.4 GHz CPU, under Windows XP. Since these environments differ from each other and from ours, we do not intend to directly compare the running times of the different procedures, but we want to investigate how they behave on different sets of instances.

4.5.2 Instances

For these experiments we used well-known instance sets from the literature, see Table 4.1. Instance sets I1 and I2 were given by Beasley and Christofides (1989), and also used by Dumitrescu and Boland (2003). Instance sets D1 and D2 were developed by Dumitrescu and Boland (2003), and also used by Pugliese and Guerriero (2013a). Instance sets S were given by Santos et al. (2007), and also used by Pugliese and Guerriero (2013a) and Lozano and Medaglia (2013). Originally, instance set S contains 900 instances, however, we have got only 880 of them. All of the instances have a single resource and can be classified into 18 classes according to their sizes (S1-S18) or into 5 groups according to their resource types (Group 1-Group 5). For details we refer to (Santos et al., 2007).

4.5.3 Experiments

In these experiments we tested only one version of our LP-based methods based on the previous results on the various components. That is, we applied DB-preprocessing on the instances to reduce the underlying graph, then we solved them with a branch-and-cut procedure where we used the default settings of Xpress in combination with our variable fixing procedure.

Experiments on instance sets I1 and I2

As we already mentioned in Section 4.4.2, 10 out of 12 instances of set I2, and all of the 12 instances of set I1 can be solved optimally only using DB-preprocessing; and the remaining two instances are easy to solve with a branch-and-cut procedure (see Table 4.2). For detailed results we refer to (Horváth and Kis, 2016b, Appendix B).

Experiments on instance sets D1 and D2

We present results in Tables 4.4 and 4.5 where we indicate execution times in seconds of the Reference Point Method (*RPM*) of Pugliese and Guerriero (2013a) (*LC* and *ISSA* refer to the two parts of the method, these are, the Label Correcting method and the Interactive Search Strategy Algorithm) and our branch-and-cut method (LPB) (where *PP* refers to the preprocessing procedure and *BNC* refers to the branch-and-cut procedure). For detailed results we refer to (Horváth and Kis, 2016b, Appendix D).

We recall that the solution approaches are tested on different platforms, thus we do not recommend a direct comparison of running times. However, we can observe that our LP-

Table 4.4: Computation times (in seconds) on instance set D1

Instance	RPM [1]		LPB [2]		Instance	RPM		LPB	
	LC	ISSA	PP	BNC		LC	ISSA	PP	BNC
D1-L/1	0.02	2.50	0.02	0.48	D1-M/1	3.60	11.31	0.01	2.04
D1-L/2	3.68	2324.59	0.23	64.34	D1-M/2	96.92	564.33	0.13	47.66
D1-L/3	3.62	1330.74	0.42	65.40	D1-M/3	377.97	1180.05	0.25	100.77
D1-L/4	21.89	21 926.07	1.31	719.84	D1-M/4	1 849.89	15 635.17	0.70	403.16

[1] method of Pugliese and Guerriero (2013a); tested with Intel Core i7-620M, 2.67 GHz CPU, under Windows 7
[2] our method; tested with Intel Core i7-4710MQ, 2.5 GHz CPU, under Windows 7

Table 4.5: Computation times (in seconds) on instance set D2

Instance set	RPM [1]		LPB [2]		Instance set	RPM		LPB	
	LC	ISSA	PP	BNC		LC	ISSA	PP	BNC
D2-L-1	0.00	0.00	0.00	0.01	D2-M-1	0.01	0.04	0.00	0.04
D2-L-2	0.02	0.04	0.01	0.24	D2-M-2	0.18	0.27	0.01	0.33
D2-L-3	0.04	0.17	0.02	0.73	D2-M-3	0.62	4.65	0.02	2.99
D2-L-4	0.33	1.36	0.07	3.10	D2-M-4	9.30	16.27	0.09	11.82
D2-L-5	0.72	2.96	0.15	8.98	D2-M-5	36.60	89.46	0.14	19.42
D2-L-6	1.01	10.16	0.20	32.07	D2-M-6	63.26	475.64	0.19	42.95
D2-L-7	2.09	52.60	0.36	59.65	D2-M-7	243.91	487.75	0.28	104.79

[1] method of Pugliese and Guerriero (2013a); tested with Intel Core i7-620M, 2.67 GHz CPU, under Windows 7
[2] our method; tested with Intel Core i7-4710MQ, 2.5 GHz CPU, under Windows 7

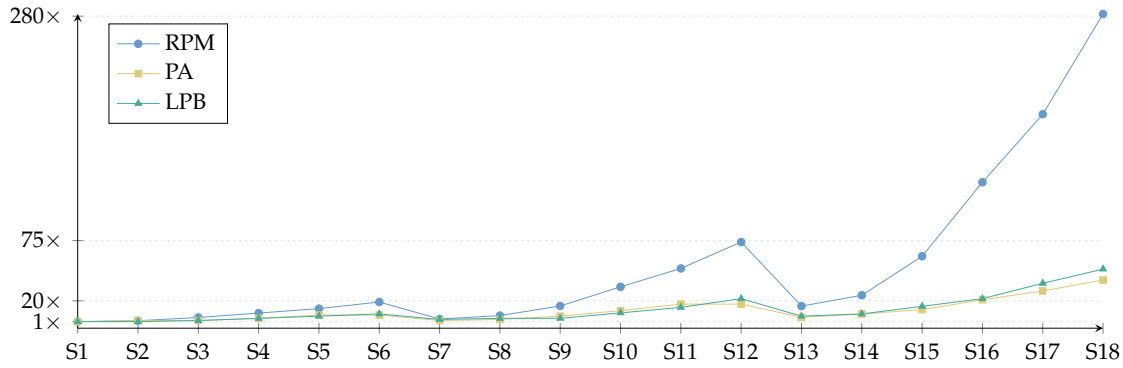


Figure 4.8: Computation times for Group 1 scaled to set S1

based method behaves in a more stable way than the Reference Point Method, since for the latter the running time grows more rapidly with the size of the instances.

Experiments on instance set S

We summarize results in Table A.6 of Appendix A. For detailed results we refer to (Horváth and Kis, 2016b, Appendix S). In Fig. 4.8 we indicate the average computation times of the Reference Point Method (RPM), the Pulse Algorithm (PA), and our branch-and-cut procedure (LPB) over the instance sets of Group 1 scaled to set S1. We have the following conclusions:

- We already showed in Section 4.4.2 that all but two instances (that is, 878 out of 880) are solved optimally in a split of second by just applying preprocessing (see also Table 4.2) which is a very surprisingly result. Note that Pugliese and Guerriero (2013a) also confirmed the efficiency of DB-preprocessing on these instances, but our computation times are orders of magnitude better than theirs. A possible reason for their high computation times is that they implemented the methods in Java (while we used C++), and on the other hand we used very efficient graph structures and shortest path algorithms provided by Lemon.
- Recall that the solution approaches have been tested on different platforms, thus we do not recommend to compare the corresponding execution times with each other, however, we notice that the computation times of PA and LPB are of the same magnitude, and RPM is significantly slower than them (see Table A.6). We remark that for LPB we have two salient results, namely for Group 2 in set S6 and set S18. These two sets contain the two instances not solved in the preprocessing phase.
- PA and LPB behave more stable way than RPM, that is, execution times increase less by the increase of the input size (see Fig. 4.8). For example, between sets S12 (20 000 nodes, 400 000 arcs) and S18 (40 000 nodes, 800 000 arcs) of Group 1 the computation time of PA and LPB increased by $120 - 130\%$, however, in the case of RPM the same value is 280%.

4.5.4 Conclusions

The above experiments suggest that our LP-based methods are competitive with other solution approaches to solve RCSPP. As we summarized in Table 4.2, the vast majority of the instances from sets I1, I2, and S can be solved optimally using only the well-known DB-preprocessing procedure. On harder instances, i.e., set D1 and D2, preprocessing techniques (both prior to forming the ILP, and in the course of branch-and-bound) play a key role in reducing the computation times.

Chapter 5

Position-based scheduling of chains on a single machine

In this chapter we consider a scheduling problem, where a set of unit-time jobs has to be sequenced on a single machine without any idle times between the jobs. Preemption of processing is not allowed. The processing cost of a job is determined by the position in the sequence, i.e., for each job and each position, there is an associated weight, and one has to determine a sequence of jobs which minimizes the total weight incurred by the positions of the jobs. In addition, the ordering of the jobs must satisfy the given chain-precedence constraints. We show that this problem is NP-hard even in a special case, where each chain consists of two jobs. Further on, we study the polyhedron associated with the problem, and present a class of valid inequalities along with a polynomial-time separation procedure, and show that some of these inequalities are facet-defining in the special case. Finally, we present our computational results that confirm that separating these inequalities can significantly speed up a linear programming based branch-and-bound procedure to solve the problem with chains of two jobs.

5.1 Introduction

For basic terminology and notation we refer to Section 2.3.

5.1.1 Problem definition

Let $\mathcal{J} = \{J_1, \ldots, J_n\}$ be the set of unit-time jobs. For a given schedule S and job J_j let $\sigma_j^S \in \{1, \ldots, n\}$ indicate the position of the job in the sequence (that is, $\sigma_j^S = k$ if exactly $k-1$ jobs are scheduled before J_j). For each job J_j and position k there is a weight $w_{jk} \in \mathbb{Q}$, and thus the weight of job J_j for a given schedule S is w_{j,σ_j^S}. The goal of the problem is to determine a schedule S that minimizes the total weight $\sum_{j=1}^{n} w_{j,\sigma_j^S}$. Using the classification of deterministic sequencing and scheduling problems introduced by Graham et al. (1979), we denote the prob-

lem as $1 \mid p_j = 1 \mid \sum w_{j,\sigma_j}$. In the case of precedence constraints, chain precedence constraints, and chain precedence constraints with chains consisting exactly two jobs (*two-chains*) we use notation $1 \mid prec, p_j = 1 \mid \sum w_{j,\sigma_j}$, $1 \mid chains, p_j = 1 \mid \sum w_{j,\sigma_j}$, and $1 \mid 2\text{-}chains, p_j = 1 \mid \sum w_{j,\sigma_j}$, respectively. Note that problem $1 \mid p_j = 1 \mid \sum w_{j,\sigma_j}$ is equivalent to the well-known assignment problem (Motzkin, 1956), thus problem $1 \mid prec, p_j = 1 \mid \sum w_{j,\sigma_j}$ can be considered as a generalized assignment problem, where the set of positions is ordered, and the assignment must satisfy the given precedence constraints.

5.1.2 Related work

Lenstra and Rinnooy Kan (1980) and Leung and Young (1990) present complexity results for scheduling unit-time jobs on a single machine with chain-precedence constraints, i.e., problems of the form $1 \mid chains, p_j = 1 \mid \gamma$. Clearly, problems with $\gamma = C_{\max}$ and $\gamma = \sum C_j$ are trivial (since each feasible schedule is optimal), and polynomially solvable for $\gamma = \sum w_j C_j$ (see e.g., (Lawler, 1978)). Lenstra and Rinnooy Kan (1980) and Leung and Young (1990) show that problems with $\gamma = \sum U_j$ and $\gamma = \sum T_j$ are strongly NP-hard, respectively. Our results in this chapter imply that the problem with $\gamma = \sum w_{j,\sigma_j}$ is NP-hard in the strong sense even if each chain in the precedence relation has length 2. We summarize these results in Table 5.1. Although we do not consider multiple-machine scheduling problems in this chapter, for the sake of completeness we also refer to some results about scheduling unit-time jobs on parallel machines under precedence constraints, i.e., problems of the form $P \mid prec, p_j = 1 \mid \gamma$. Ullman (1975) shows that problem $P \mid prec, p_j = 1 \mid C_{\max}$ is strongly NP-hard, however, problems $P \mid chains, p_j = 1 \mid C_{\max}$ and $P2 \mid prec, p_j = 1 \mid C_{\max}$ are polynomially solvable (see e.g., (Hu, 1961) and (Coffman and Graham, 1972), respectively). Hoogeveen et al. (2001) show that problem $P \mid prec, p_j = 1 \mid \sum C_j$ is APX-hard, however, problems $P \mid chains, p_j = 1 \mid \sum C_j$ and $P2 \mid prec, p_j = 1 \mid \sum C_j$ are polynomially solvable (see e.g., (Hu, 1961) and (Coffman and Graham, 1972), respectively). Finally, Timkovsky (2003) shows that problem $P2 \mid chains, p_j = 1 \mid \sum w_j C_j$ is strongly NP-hard.

The traditional precedence constraints can be considered as AND-precedence constraints, that is, a job can only be started after all of its (immediate) predecessors are completed. In contrast, in the case of OR-precedence constraints, a job can be started as soon as one of its immediate predecessors is completed. Note that in this case the precedence graph can be cyclic, however, one can decide in linear time whether the problem has a feasible solution (see e.g., (Möhring et al., 2004)). According to this, problem $1 \mid or\text{-}prec, p_j = 1 \mid \gamma$ is trivial for $\gamma = C_{\max}$ and $\gamma = \sum C_j$, where *or-prec* refers to the presence of OR-precedence constraints. Among other results, Johannes (2005) shows that problem $1 \mid or\text{-}prec, p_j = 1 \mid \sum w_j C_j$ is strongly NP-hard. Note that the chain-precedence constraints are both AND- and OR-precedence constraints, since in this case each job has at most one immediate predecessor, thus problems of the form $1 \mid chains, p_j = 1 \mid \gamma$ considered in this chapter are special cases of problem $1 \mid or\text{-}prec, p_j = 1 \mid \gamma$. We also summarize these results in Table 5.1.

Table 5.1: Problem $1 \mid \beta, p_j = 1 \mid \gamma$: scheduling unit-time jobs on a single machine under precedence constraints

	$\beta = chains$	$\beta = prec$	$\beta = or\text{-}prec$
$\gamma = C_{\max}$	in P (trivial)[1]		
$\gamma = \sum C_j$	in P (trivial)[1]		
$\gamma = \sum w_j C_j$	in P (Lawler, 1978)	strongly NP-hard (Lenstra and Rinnooy Kan, 1978)	strongly NP-hard (Johannes, 2005)
$\gamma = \sum T_j$	strongly NP-hard (Leung and Young, 1990)		
$\gamma = \sum U_j$	strongly NP-hard (Lenstra and Rinnooy Kan, 1980)		
$\gamma = \sum w_{j,\sigma_j}$	strongly NP-hard (in this chapter)		

[1] each feasible schedule is optimal

Wan and Qi (2010) introduce new scheduling models where time slot costs have to be taken into consideration. In their models the planning horizon is divided into $K \geq \sum_{j=1}^{n} p_j$ time slots with unit length, where the kth time slot has cost π_k, and the time slot cost of a job J_j with starting time t is $\sum_{k \in s_j} \pi_k$, where $s_j = \{t+1, \ldots, t+p_j\}$. The objective of their models is a combination of the total time slot cost with a traditional scheduling criterion, that is, they consider problems of the form $1 \mid slotcost \mid \gamma + \sum_j \sum_{k \in s_j} \pi_k$. Wan and Qi (2010) show that in the case of non-decreasing time slot costs (that is, $\pi_1 \leq \ldots \leq \pi_K$) the problem can be reduced to one without slot costs. Under the assumption of arbitrarily varied time slot costs they prove that the problems with $\gamma = \sum C_j$, $\gamma = L_{\max}$, $\gamma = T_{\max}$, $\gamma = \sum U_j$ and $\gamma = \sum T_j$ are strongly NP-hard. They also show that in the case of non-increasing time slot costs some of these problems can be solved in polynomial or pseudo-polynomial time. Zhao et al. (2016) prove that in the case of non-increasing time slot costs, problem $1 \mid slotcost \mid \sum(C_j + \sum_{k \in s_j} \pi_k)$ is NP-hard in the strong sense. Kulkarni and Munagala (2012) introduce a model similar to that of (Wan and Qi, 2010), however, they deal with online algorithms to minimize the total time slot costs plus the total weighted completion time. Note that the problem investigated in this chapter can be considered as a generalization of a special case of the model of Wan and Qi (2010). That is, in the case of unit-time jobs (with $K = \sum_{j=1}^{n} p_j = n$) problem $1 \mid slotcost, p_j = 1 \mid \sum_j \sum_{k \in s_j} \pi_k$ is similar to that of $1 \mid p_j = 1 \mid \sum w_{j,\sigma_j}$, however, in the latter problem the time slot costs depend on the jobs.

5.1.3 Problem formulation

Recall that $\mathcal{J} = \{J_1, \ldots, J_n\}$ is the set of unit-time jobs, and let $\mathcal{P} = \{1, \ldots, n\}$ be the set of positions. Recall that we write $J_{i_1} \prec\prec J_{i_2}$ ($J_{i_1} \prec J_{i_2}$) if $J_{i_1} \neq J_{i_2}$ and J_{i_1} is a predecessor (immediate predecessor) of J_{i_2}. Let x_{ij} be the binary variable indicating whether job J_i is

assigned to position j. We formulate problem $1 \,|\, prec, p_j = 1 \,|\, \sum w_{j,\sigma_j}$ as

$$\text{minimize} \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_{ij} \tag{5.1}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \text{for all } i = 1,\dots,n \tag{5.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \text{for all } j = 1,\dots,n \tag{5.3}$$

$$\sum_{j=1}^{k+1} x_{i_2,j} \le \sum_{j=1}^{k} x_{i_1,j} \qquad \text{for all } J_{i_1} \prec J_{i_2}, \text{ and } k = 1,\dots,n-1 \tag{5.4}$$

$$x_{ij} \in \{0,1\} \qquad \text{for all } i = 1,\dots,n, \text{ and } j = 1,\dots,n, \tag{5.5}$$

where constraints (5.2) and (5.3) model the job-position assignment constraints. Inequality (5.4) ensures that the precedence constraints are satisfied. That is, for each pair of jobs J_{i_1} and J_{i_2} such that $J_{i_1} \prec J_{i_2}$, there are $n-1$ linear constraints ensuring that job J_{i_2} cannot be assigned to the same or to an earlier position than job J_{i_1}. Let $S_n^{prec} := \{x \in \{0,1\}^{n\times n} : x \text{ satisfies (5.2)} - \text{(5.4)}\}$ be the set of feasible solutions, and $P_n^{prec} := \text{conv}(S_n^{prec})$ be the polytope of feasible solutions of (5.2)–(5.5). By construction, we have the following proposition.

Proposition 5.1. *S_n^{prec} is the set of incidence vectors corresponding to feasible job-position assignments.*

For later use we provide some valid equations for P_n^{prec}. Let $\mathcal{J}_i^+ := \{J_j \in \mathcal{J} : J_i \prec\prec J_j\}$ ($\mathcal{J}_i^- := \{J_j \in \mathcal{J} : J_j \prec\prec J_i\}$) be the set of successors (predecessors) of job J_i. Clearly, for each point $x \in S_n^{prec}$ we have

$$x_{ij} = 0 \qquad \text{for all } i = 1,\dots,n, \text{ and } j = 1,\dots,|\mathcal{J}_i^-| \tag{5.6}$$

and

$$x_{ij} = 0 \qquad \text{for all } i = 1,\dots,n, \text{ and } j = n - |\mathcal{J}_i^+| + 1,\dots,n. \tag{5.7}$$

Since P_n^{prec} is the convex hull of the points S_n^{prec}, these equations are valid for P_n^{prec}.

5.1.4 Our contribution

We show that problem $1 \,|\, prec, p_j = 1 \,|\, \sum w_{j,\sigma_j}$ is strongly NP-hard even in the case of two-chains.

We study the polytopes associated with the problem in case of chains and two-chains as well. In case of chain precedence constraints we present a class valid inequalities for the corresponding polytope along with a polynomial-time separation procedure. In case of two-chains we prove that a subclass of these inequalities are facet-defining.

Finally, we present our computational results that confirm that separating these inequalities can significantly speed up a linear programming based branch-and-bound procedure to solve the problem with chains of two jobs.

5.2 Problem $1 \,|\, chains, p_j = 1 \,|\, \gamma$

In this section we present a class of valid inequalities for the case of chain-precedence constraints along with a polynomial time separation procedure. We derive these inequalities by using the so-called parity inequalities, which constitute the non-trivial facets of the parity polytope (see Section 5.2.1).

For chain-precedence constraints, let S_n^{chain} and P_n^{chain} denote the set of feasible solutions and the polytope of feasible solutions, of (5.1)–(5.5), respectively. Let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be the set of chains (i.e., chain-precedence constraints), where $C_i = (J_{i_1}, \ldots, J_{i_\ell})$ with $J_{i_1} \prec \ldots \prec J_{i_\ell}$ for all $i = 1, \ldots, m$. The length of a chain C, i.e., the number of its jobs, is denoted by $|C|$. For a given integer k we denote the index set $\{1, \ldots, k\}$ by $[k]$.

5.2.1 Parity polytope, parity inequalities

Let P_d^{even} (P_d^{odd}) be the convex hull of those d-dimensional 0-1 vectors in which the number of 1's is even (odd). The characterization of these polytopes is attributed to Jeroslow (1975), however, for a direct proof of this result we refer to (Lancia and Serafini, 2018).

Theorem 5.1 (Lancia and Serafini (2018))**.**

$$P_d^{even} = \left\{ z \in [0,1]^d : \sum_{i \in S} z_i - \sum_{i \notin S} z_i \leq |S| - 1 \text{ for all odd-subset } S \subset [d] \right\},$$

$$P_d^{odd} = \left\{ z \in [0,1]^d : \sum_{i \in S} z_i - \sum_{i \notin S} z_i \leq |S| - 1 \text{ for all even-subset } S \subset [d] \right\}.$$

We say that a subset $S \subseteq [d]$ is an *odd-subset* (*even-subset*) if its cardinality $|S|$ is odd (even), and we call the inequalities of Theorem 5.1 *parity inequalities*.

Separation of parity inequalities

Since we have not been able to find any paper that provides a separation procedure for parity inequalities, we provide our own procedure. First, we reformulate parity inequalities as

$$1 \leq \sum_{i \in S} (1 - z_i) + \sum_{i \notin S} z_i \qquad \text{for each odd-subset } S \subseteq [d] \tag{5.8}$$

and

$$1 \leq \sum_{i \in S} (1 - z_i) + \sum_{i \notin S} z_i \qquad \text{for each even-subset } S \subseteq [d]. \tag{5.9}$$

Note that in the sake of convenience we allow S to be the complete set $[d]$, with this the corresponding inequality is still valid but redundant.

Theorem 5.2. *Inequalities* (5.8) *and* (5.9) *can be separated in polynomial time, that is, for a given vector* $\bar{z} \in [0,1]^d$ *the following problems can be solved in polynomial time:*

$$\text{maximize} \left\{ 1 - \left(\sum_{i \in S} (1 - \bar{z}_i) + \sum_{i \notin S} \bar{z}_i \right) : S \subseteq [d] \text{ is an odd-subset} \right\}, \tag{5.10}$$

$$\text{maximize} \left\{ 1 - \left(\sum_{i \in S} (1 - \bar{z}_i) + \sum_{i \notin S} \bar{z}_i \right) : S \subseteq [d] \text{ is an even-subset} \right\}. \tag{5.11}$$

Clearly, if the maximum value is less than or equal to zero then all of the inequalities are satisfied, otherwise, the corresponding subset gives one of the most violated inequalities.

Lemma 5.1. *Let* $1 \geq v_1 \geq v_2 \geq \ldots \geq v_d \geq 0$, *and let* $f(S) := \sum_{i \in S}(1 - v_i) + \sum_{i \notin S} v_i$ *for all* $S \subseteq [d]$. *Consider the following problems:*

$$\text{minimize} \{ f(S) : S \subseteq [d] \text{ is an odd-subset} \}, \tag{5.12}$$

$$\text{minimize} \{ f(S) : S \subseteq [d] \text{ is an even-subset} \}. \tag{5.13}$$

a) Let $S_0 := \emptyset$ *and* $S_i := [i]$ *for all* $i = 1, \ldots, d$. *There is an optimal solution* S_{OPT} *for problem* (5.12) *(problem* (5.13)*) such that* $S_{OPT} = S_i$ *for some* $i = 0, \ldots, d$.

b) Let $t := 0$ *if* $1 - v_i > v_i$ *holds for all* $i = 1, \ldots, d$, *and let* $t := \max\{i : 1 - v_i \leq v_i\}$ *otherwise. One of the sets* S_{t-1}, S_t *and* S_{t+1} *is an optimal solution for problem* (5.12) *(problem* (5.13)*).*

Proof. To prove statement a), consider an optimal solution S_{OPT} for problem (5.12) which maximizes the parameter $p := \max\{i : S_i \subseteq S_{OPT}\}$, i.e., for any optimal solution $S^\star$ we have $\max\{i : S_i \subseteq S^\star\} \leq p$. Clearly, $p + 1 \notin S_{OPT}$. Suppose for the sake of a contradiction that there is an index $q > p + 1$ such that $q \in S_{OPT}$. Let $S' := (S_{OPT} \cup \{p + 1\}) \setminus \{q\}$. Now, we have $f(S_{OPT}) \leq f(S') = f(S_{OPT}) + (1 - v_{p+1}) - v_{p+1} - (1 - v_q) + v_q = f(S_{OPT}) + 2(v_q - v_{p+1}) \leq f(S_{OPT})$, thus S' is also an optimal solution for problem (5.12), however $p < \max\{i : S_i \subseteq S'\}$ which contradicts our assumption for S_{OPT}, and the statement is proved.

According to statement a) problems (5.12) and (5.13) can be restricted to subsets of the form S_i, $i \in \{0, \ldots, d\}$. For each $i < t$, $1 - v_{i+1} \leq v_{i+1}$, thus $f(S_{i+1}) = f(S_i) + (1 - v_{i+1}) - v_{i+1} \geq f(S_i)$. For each $i > t$, $1 - v_i > v_i$, thus $f(S_i) = f(S_{i-1}) + (1 - v_i) - v_i < f(S_{i+1})$. Therefore, we have

$$f(S_1) \geq \ldots \geq f(S_{t-1}) \geq f(S_t) \text{ and } f(S_t) < f(S_{t+1}) < \ldots < f(S_n),$$

thus if S_t has odd (even) cardinality, then it is an optimal solution for problem (5.12) (problem (5.13)), otherwise, $\arg\min\{f(S_{t-1}), f(S_{t+1})\}$ is an optimal solution for problem (5.12) (problem (5.13)). □

Proof of Theorem 5.2. For a given vector $\bar{z} \in [0,1]^d$ let $v_i := \bar{z}_i$ for all $i = 1, \ldots, d$, and let $f(S) := \sum_{i \in S}(1 - v_i) + \sum_{i \notin S} v_i$ for all $S \subseteq [d]$. Without loss of generality (e.g., by sorting and reindexing the values), we can assume that $v_1 \geq v_2 \geq \ldots \geq v_d$. By this, separation problem (5.10) (problem (5.11)) is equivalent to problem (5.12) (problem (5.13)) which can be solved in polynomial time according to Lemma 5.1. □

5.2.2 Valid inequalities for P_n^{chain}

We introduce variables z_{ij} ($i = 1,\dots,m$, $j = 1,\dots,n$) indicating whether the number of jobs from chain C_i that are assigned to one of the positions from $\{1,\dots,j\}$ is odd ($z_{ij} = 1$) or even ($z_{ij} = 0$).

Claim 5.1. *Let $x \in S_n^{chain}$. For each chain $C_i = (J_{i_1},\dots,J_{i_\ell})$ and each position $j \in \{1,\dots,n\}$ we have*

$$z_{ij} = \sum_{k=1}^{\ell} (-1)^{k-1} \sum_{p=1}^{j} x_{i_k,p}.$$

Proof. For an $x \in S_n^{chain}$ the value $\delta_k := \sum_{p=1}^{j} x_{i_k,p}$ ($k = 1,\dots,\ell$) equals to 1 if and only if job J_{i_k} is assigned to one of the positions $\{1,\dots,j\}$, otherwise it is 0. Clearly, for jobs $J_{i_1} \prec \dots \prec J_{i_\ell}$ we have $1 \geq \delta_1 \geq \dots \geq \delta_\ell \geq 0$, thus summing these values with alternating factors $(-1)^{k-1}$ ($k = 1,\dots,\ell$), the sum (i.e., z_{ij}) is 1 if the number of δ-values that are equal to 1 is odd, otherwise it is 0. □

Claim 5.2. *For an even (odd) position $j \in \{1,\dots,n\}$ the number of 1's in vector $(z_{1j},\dots,z_{mj})$ is even (odd).*

Proof. If j is even (odd), then the number of chains C_i such that the cardinality of the set $\{k \in \{1,\dots,j\} : z_{ik} = 1\}$ is odd (i.e., C_i has an odd number of jobs assigned to the positions $1,\dots,j$) must be even (odd). □

According to Claim 5.2, the corresponding parity inequalities are valid for the polytope of feasible solutions of the formulation extended by the z-variables. However, due to Claim 5.1, one can transform these inequalities to the original x-variables, thus we have the following theorem.

Theorem 5.3. *The following inequalities are valid for P_n^{chain}:*

$$\sum_{i\in S}\left(\sum_{k=1}^{|C_i|}(-1)^{k-1}\sum_{p=1}^{j} x_{i_k,p}\right) - \sum_{i\notin S}\left(\sum_{k=1}^{|C_i|}(-1)^{k-1}\sum_{p=1}^{j} x_{i_k,p}\right) \leq |S| - 1$$

for each even position j and odd-subset $S \subseteq [m]$, (5.14)

and

$$\sum_{i\in S}\left(\sum_{k=1}^{|C_i|}(-1)^{k-1}\sum_{p=1}^{j} x_{i_k,p}\right) - \sum_{i\notin S}\left(\sum_{k=1}^{|C_i|}(-1)^{k-1}\sum_{p=1}^{j} x_{i_k,p}\right) \leq |S| - 1$$

for each odd position j and even-subset $S \subseteq [m]$. (5.15)

The separation procedure of the class of inequalities (5.14) (inequalities (5.15)) is similar to the separation procedure of the class of inequalities (5.8) (inequalities (5.9)), that is, for a given vector $\bar{x} \in [0,1]^{n\times n}$, fix an even (odd) position j, and let $\bar{z}_i := \sum_{k=1}^{\ell}(-1)^{k-1}\sum_{p=1}^{j}\bar{x}_{i_k,p}$ for each chain $C_i = (J_{i_1},\dots,J_{i_\ell})$, $i = 1,\dots,m$. By this, one can use the separation procedure described in Section 5.2.1.

5.3 Problem $1 \,|\, 2\text{-}chains, p_j = 1 \,|\, \gamma$

In this section we investigate problem $1 \,|\, 2\text{-}chains, p_j = 1 \,|\, \gamma$. Recall that in this problem we have an even number of jobs ($2n$), and relation $\prec$ partitions the set of jobs into n disjoint pairs, i.e., each jobs has exactly one predecessor or one successor, but not both. In Section 5.3.1 we reformulate the ILP of Section 5.1.3 to make our notation easier and reflect that each chain consists of two jobs. The problem $1 \,|\, 2\text{-}chains, p_j = 1 \,|\, \sum w_{j,\sigma_j}$ is shown to be strongly NP-hard in Section 5.3.2. In Section 5.3.3 we analyze the polyhedron spanned by the feasible solutions of our integer programming formulation, namely, we determine its dimension, and then in Section 5.3.4 we show that some of the inequalities from Section 5.2 are facet-defining.

5.3.1 Problem formulation

In order to simplify our notation, in this section let $\mathcal{J} = \{J_1, \dots, J_{2n}\}$ be the set of unit-time jobs, and $\mathcal{C} = \{C_1, \dots, C_n\}$ be the set of 2-chains, where $C_i = (J_{2i-1}, J_{2i})$, that is, $J_{2i-1} \prec J_{2i}$ for each $i = 1, \dots, n$. We say that job J_{2i-1} (J_{2i}) is the first (second) job of chain C_i. In addition, let $\mathcal{P} = \{1, \dots, 2n\}$ be the set of positions.

Let s_{ij} (e_{ij}) indicate whether the first (second) job of chain $C_i \in \mathcal{C}$ is assigned to position $j \in \mathcal{P}$. Note that we just renamed the variables of the formulation (5.2)–(5.7), that is, $s_{ij} := x_{2i-1,j}$ and $e_{ij} := x_{2i,j}$, thus we get the following equivalent formulation:

$$\sum_{j=1}^{2n} s_{ij} = 1 \qquad \text{for all } i = 1, \dots, n \tag{5.16}$$

$$\sum_{j=1}^{2n} e_{ij} = 1 \qquad \text{for all } i = 1, \dots, n \tag{5.17}$$

$$s_{i,2n} = 0 \qquad \text{for all } i = 1, \dots, n \tag{5.18}$$

$$e_{i,1} = 0 \qquad \text{for all } i = 1, \dots, n \tag{5.19}$$

$$\sum_{i=1}^{n} s_{i,1} = 1 \tag{5.20}$$

$$\sum_{i=1}^{n} \left(s_{ij} + e_{ij}\right) = 1 \qquad \text{for all } j = 2, \dots, 2n-1 \tag{5.21}$$

$$\sum_{i=1}^{n} e_{i,2n} = 1 \tag{5.22}$$

$$\sum_{j=1}^{k+1} e_{ij} \le \sum_{j=1}^{k} s_{ij} \qquad \text{for all } i = 1, \dots, n, \text{ and } k = 1, \dots, 2n-2 \tag{5.23}$$

$$s_{ij},\ e_{ij} \in \{0, 1\} \qquad \text{for all } i = 1, \dots, n, \text{ and } j = 1, \dots, 2n. \tag{5.24}$$

Constraints (5.16)–(5.17) and (5.20)–(5.22) are the job-position assignment constraints (cf. (5.2) and (5.3)). Inequality (5.23) ensures that each first-job precedes the corresponding second-job (cf. (5.4)). Finally, constraints (5.18)–(5.19) forbid to assign a first-job to the last, or a second-job

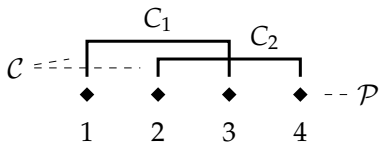


Figure 5.1: Representation of point $P = (s, e) \in S_4^{2\text{-}chains}$ with $s_{1,1} = e_{1,3} = s_{2,2} = e_{2,4} = 1$

to the first position (cf. (5.6)–(5.7)). Similarly to the general case in Section 5.1.3, we introduce the set of feasible solutions $S_{2n}^{2\text{-}chains} := \{(s, e) \in \{0,1\}^{n\times 2n} \times \{0,1\}^{n\times 2n} : (5.16) - (5.23) \text{ holds}\}$, and the polytope of feasible solutions $P_{2n}^{2\text{-}chains} := \text{conv}(S_{2n}^{2\text{-}chains})$ of (5.16)–(5.24).

Preliminaries

For a given point $P = (s, e) \in S_{2n}^{2\text{-}chains}$, let $s(P, i) = j$ ($e(P, i) = j$) if $s_{ij} = 1$ ($e_{ij} = 1$). For a given $i = 1, \ldots, n$ let $\sigma_i(P)$ be a 2-dimensional vector such that $\sigma_i(P) = (s(P, i), e(P, i))$, and $\sigma(P)$ be a $2n$-dimensional vector such that $\sigma(P) = (\sigma_1(P), \ldots, \sigma_n(P))$. For example, for the point P indicated in Fig. 5.1 we have $P = (1,0,0,0,\ 0,1,0,0,\ 0,0,1,0,\ 0,0,0,1)$, $\sigma_1(P) = (1,3)$, $\sigma_2(P) = (2,4)$, and $\sigma(P) = (1,3,2,4)$.

5.3.2 Complexity of problem $1 \,|\, 2\text{-}chains, p_j = 1 \,|\, \sum w_{j,\sigma_j}$

In Theorem 5.4 we will show that problem

$$1 \,|\, 2\text{-}chains, p_j = 1 \,|\, \sum w_{j,\sigma_j} \qquad (\star)$$

is NP-hard in the strong sense.

Sketch of proof of Theorem 5.4. We will transform the Independent Set Problem (ISP) to problem ($\star$). Recall that an instance of ISP is given by an undirected graph $G = (V, E)$ with node set $V = \{v_1, \ldots, v_n\}$, and a maximum size subset of nodes $I \subseteq V$ is sought such that $|\{u, v\} \cap I| \leq 1$ holds for each edge $\{u, v\} \in E$. The basic idea of the transformation can be seen in Fig. 5.2, where we depict the construction for the 2-length path (without the dummy chains). Briefly stated, we will create a chain t_i for each node v_i and two chains f_{ij} and g_{ij} for each edge $\{v_i, v_j\}$ of the ISP instance, and some additional dummy chains. To each of these chains we will designate two potential start and two potential end positions. First, by determining appropriate weights we ensure that in each solution with non-positive total weight, each of these chains either starts and ends at its first start and end position, respectively, or at its second start and end position. In Fig. 5.2 we depict the two potential states of these chains. Second, by designating these positions properly, it is guaranteed that each solution with a non-positive total weight represents an independent set in the ISP instance and vice versa. Namely, a node is in the independent set if and only if the corresponding chain starts and ends its second start and end position, respectively. Note that the role of the edge-chains

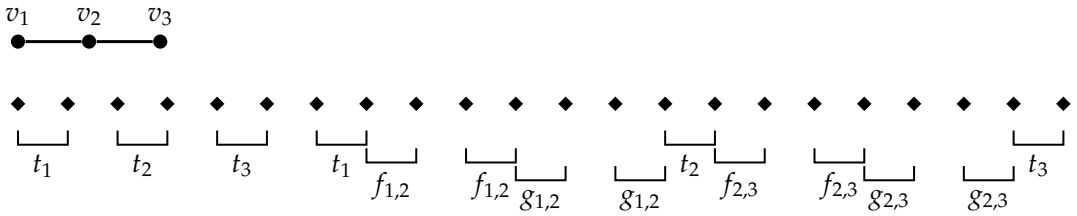


Figure 5.2: [Sketch of proof of Theorem 5.4] Construction for the 2-length path where we depict the possible states of chains (dummy chains are not depicted)

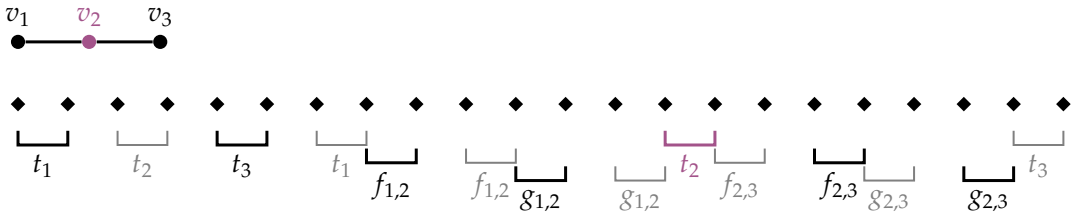


Figure 5.3: [Sketch of proof of Theorem 5.4] Solution representing independent set $\{v_2\}$

is to ensure that for adjacent vertices one of the corresponding node-chains must start and end at its first start and end position, respectively, i.e., at most one of these nodes can be in the independent set. For example, in Fig. 5.3 we depict the solution that represents the independent set $\{v_2\}$ (without the dummy chains). Note that since chain t_2 starts/ends at its second start/end position, i.e., v_2 is in the independent set, thus chains $g_{1,2}$, $f_{1,2}$ and therefore t_1 must start/end at its first start/end position, i.e., v_1 cannot be in the independent set. Similarly, t_3 cannot start/end at its second start/end position, that is, v_3 cannot be in the independent set. In Fig. 5.4 we depict the solution that represents the independent set $\{v_1, v_3\}$ (without the dummy chains).

Theorem 5.4. *Problem* $1 \mid 2\text{-chains}, p_j = 1 \mid \sum w_{j,\sigma_j}$ *is strongly* NP*-hard.*

Proof. We transform the Independent Set Problem (ISP) problem to problem ($\star$). Let $G = (V, E)$ be an instance for ISP with node set $V = \{v_1, \ldots, v_n\}$ and edge set E, and let $\vec{E} := \{(v_i, v_j) : \{v_i, v_j\} \in E,\ i < j\}$ be the set of directed edges, i.e., we replace undirected edge $\{v_i, v_j\}$ by directed edge (v_i, v_j) for $i < j$. For a node v_i let $\text{succ}(i) := \{v_j : (v_i, v_j) \in \vec{E}\}$

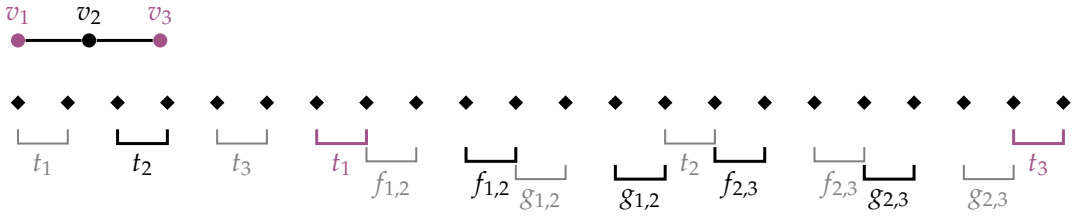


Figure 5.4: [Sketch of proof of Theorem 5.4] Solution representing independent set $\{v_1, v_3\}$

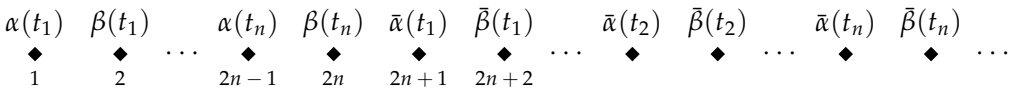


Figure 5.5: [Proof of Theorem 5.4] Designated positions for node-chains

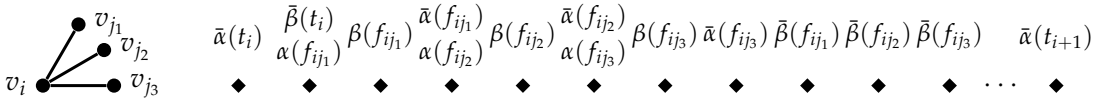


Figure 5.6: [Proof of Theorem 5.4] Designated positions for edge-chains (part 1)

($\text{pred}(i) := \{v_j : (v_j, v_i) \in \vec{E}\}$) denote its immediate successors (predecessors). Based on the ISP instance we will construct an instance for problem ($\star$) with $2|V| + 3|E|$ chains (that is, we will create 1 chain for each node, 2 chains for each edge, and $|V| + |E|$ additional dummy chains), and $4|V| + 6|E|$ positions.

For each $v_i \in V$ we create a node-chain t_i, and for each edge $(v_i, v_j) \in \vec{E}$ we create edge-chains f_{ij} and g_{ij}. Let $\mathcal{T}_V = \{t_i : v_i \in V\}$ and $\mathcal{T}_{\vec{E}} = \{f_{ij}, g_{ij} : (v_i, v_j) \in \vec{E}\}$. To each node-chain $t_i \in \mathcal{T}_V$ we designate four distinct positions: $\alpha(t_i) < \beta(t_i) < \bar{\alpha}(t_i) < \bar{\beta}(t_i)$ such that

(i) $2i - 1 = \alpha(t_i) = \beta(t_i) - 1$ for all $i = 1, \ldots, n$,
(ii) $2n + 1 = \bar{\alpha}(t_1) = \bar{\beta}(t_1) - 1$, and
(iii) $\bar{\beta}(t_i) < \bar{\alpha}(t_{i+1}) = \bar{\beta}(t_{i+1}) - 1$ for all $i = 1, \ldots, n-1$,

see Fig. 5.5. To each edge-chain $f_{ij} \in \mathcal{T}_{\vec{E}}$ we designate four distinct positions: $\alpha(f_{ij}) < \beta(f_{ij}) < \bar{\alpha}(f_{ij}) < \bar{\beta}(f_{ij})$. Consider a node $v_i \in V$ and its immediate successors $\text{succ}(i) = \{v_{j_1}, \ldots, v_{j_{|\text{succ}(i)|}}\}$. Let

(iv) $\alpha(f_{i,j_1}) = \bar{\beta}(t_i)$,
(v) $\alpha(f_{i,j_\ell}) = \beta(f_{i,j_\ell}) - 1 = \bar{\alpha}(f_{i,j_\ell}) - 2$ for all $\ell = 1, \ldots, |\text{succ}(i)|$,
(vi) $\bar{\alpha}(f_{i,j_\ell}) = \alpha(f_{i,j_{\ell+1}})$ for all $\ell = 1, \ldots, |\text{succ}(i)| - 1$,
(vii) $\bar{\alpha}(f_{i,j_{|\text{succ}(i)|}}) = \bar{\beta}(f_{i,j_1}) - 1 = \bar{\beta}(f_{i,j_2}) - 2 = \ldots = \bar{\beta}(f_{i,j_{|\text{succ}(i)|}}) - |\text{succ}(i)|$,
(viii) $\bar{\beta}(f_{i,j_{|\text{succ}(i)|}}) < \bar{\alpha}(t_{i+1})$,

see Fig. 5.6. Finally, to each edge-chain $g_{ij} \in \mathcal{T}_{\vec{E}}$ we designate four distinct positions: $\alpha(g_{ij}) < \beta(g_{ij}) < \bar{\alpha}(g_{ij}) < \bar{\beta}(g_{ij})$. Consider a node $v_j \in V$ and its immediate predecessors $\text{pred}(j) = \{v_{i_1}, \ldots, v_{i_{|\text{pred}(j)|}}\}$. Let

(ix) $\bar{\beta}(g_{i_1,j}) = \bar{\alpha}(t_j)$,
(x) $\beta(g_{i_\ell,j}) = \bar{\alpha}(g_{i_\ell,j}) - 1 = \bar{\beta}(g_{i_\ell,j}) - 2$ for all $\ell = 1, \ldots, |\text{pred}(j)|$,
(xi) $\beta(g_{i_\ell,j}) = \bar{\beta}(g_{i_{\ell+1},j})$ for all $\ell = 1, \ldots, |\text{pred}(j)| - 1$,
(xii) $\alpha(g_{i_\ell,j}) = \bar{\beta}(f_{i_\ell,j})$ for all $\ell = 1, \ldots, |\text{pred}(j)|$,
(xiii) $\bar{\beta}(t_{j-1}) < \beta(g_{i_1,|\text{pred}(j)|})$,

see Fig. 5.7.

For each $v_i \in V$ we have created 1 chain and designated 4 positions, and for each $(v_i, v_j) \in \vec{E}$ we have created 2 chains and designated 8 positions, however, positions $\alpha(f_{ij})$, $\bar{\beta}(g_{ij})$ and $\bar{\beta}(f_{ij})$ coincide with other positions (see (iv), (vi), (ix), (xi), and (xii)), hence we have $|V| + 2|E|$

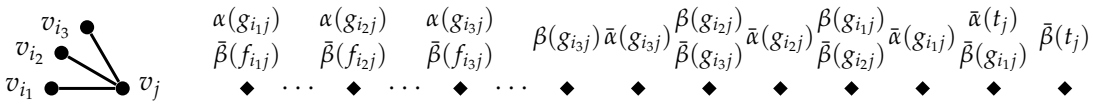


Figure 5.7: [Proof of Theorem 5.4] Designated positions for edge-chains (part 2)

chains, and $4|V| + 5|E|$ distinct positions. Thus, we also create $|V| + |E|$ dummy chains and $|E|$ dummy positions, therefore we have $2|V| + 3|E|$ chains and $2 \times (2|V| + 3|E|)$ positions, that is, we have a valid instance for problem $1 \,|\, 2\text{-}chains, p_j = 1 \,|\, \sum w_{j,\sigma_j}$.

Let $M > n$. For each $t_i \in \mathcal{T}_V$ let

$$w^s(t_i, j) := \begin{cases} M & \text{if } j = \alpha(t_i), \\ 0 & \text{if } j = \bar\alpha(t_i), \\ 2M & \text{otherwise,} \end{cases} \quad \text{and} \quad w^e(t_i, j) := \begin{cases} -M & \text{if } j = \beta(t_i), \\ -1 & \text{if } j = \bar\beta(t_i), \\ 2M & \text{otherwise.} \end{cases}$$

For each $t_{ij} \in \mathcal{T}_{\vec E}$ (t_{ij} is either f_{ij} or g_{ij}) let

$$w^s(t_{ij}, k) := \begin{cases} M & \text{if } k = \alpha(t_{ij}), \\ 0 & \text{if } k = \bar\alpha(t_{ij}), \\ 2M & \text{otherwise,} \end{cases} \quad \text{and} \quad w^e(t_{ij}, k) := \begin{cases} -M & \text{if } k = \beta(t_{ij}), \\ 0 & \text{if } k = \bar\beta(t_{ij}), \\ 2M & \text{otherwise.} \end{cases}$$

Finally, let $w^s(t, j) := w^e(t, j) := 0$, for each dummy chain t and for all $j = 1, \ldots, (4|V| + 6|E|)$.

Remarks 5.1. *By construction, in any feasible solution for the constructed problem, for each $t \in \mathcal{T}_V$ we have*

$$\sum_j w^s(t, j) + \sum_j w^e(t, j) = \begin{cases} 0 & \text{if } s_{t,\alpha(t)} = e_{t,\beta(t)} = 1, \\ -1 & \text{if } s_{t,\bar\alpha(t)} = e_{t,\bar\beta(t)} = 1, \\ \geq M & \text{otherwise,} \end{cases}$$

and for each $t \in \mathcal{T}_{\vec E}$ we have

$$\sum_j w^s(t, j) + \sum_j w^e(t, j) = \begin{cases} 0 & \text{if } s_{t,\alpha(t)} = e_{t,\beta(t)} = 1 \text{ or } s_{t,\bar\alpha(t)} = e_{t,\bar\beta(t)} = 1, \\ \geq M & \text{otherwise.} \end{cases}$$

Remark that $M > n = |\mathcal{T}_V|$, thus a solution for the created problem has non-positive total weight if and only if each chain $t \in \mathcal{T}_V \cup \mathcal{T}_{\vec E}$ starts/ends either its first start/end or its second start/end position.

Proposition 5.2. *Let $I \subseteq V$ an independent set in $G = (V, E)$. Then the corresponding scheduling problem instance admits a feasible solution of total weight $-|I|$.*

Proof. If $v_i \notin I$, then let $s_{t_i,\alpha(t_i)} := e_{t_i,\beta(t_i)} := 1$, for each $(v_i, v_j) \in \vec E$ let $s_{f_{ij},\alpha(f_{ij})} := e_{f_{ij},\beta(f_{ij})} := 1$, and for each $(v_k, v_i) \in \vec E$ let $s_{g_{ki},\bar\alpha(g_{ki})} := e_{g_{ki},\bar\beta(g_{ki})} := 1$ (see Fig. 5.8). Otherwise, if $v_i \in I$, then let $s_{t_i,\bar\alpha(t_i)} := e_{t_i,\bar\beta(t_i)} := 1$, for each $(v_i, v_j) \in \vec E$ let $s_{f_{ij},\bar\alpha(f_{ij})} := e_{f_{ij},\bar\beta(f_{ij})} := 1$, and for each $(v_k, v_i) \in \vec E$ let $s_{g_{ki},\alpha(g_{ki})} := e_{g_{ki},\beta(g_{ki})} := 1$ (see Fig. 5.9). The variables for dummy chains can be arbitrarily fixed. First, we claim that this assignment yields a feasible solution. We need

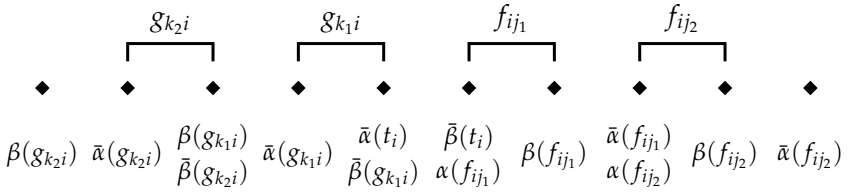


Figure 5.8: [Proof of Theorem 5.4] Assignments for node $v_i \in V \setminus I$

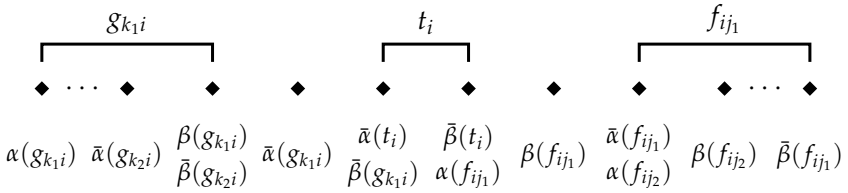


Figure 5.9: [Proof of Theorem 5.4] Assignments for node $v_i \in I$

to show, that each position that designated to multiple jobs is assigned to a single job. It is easy to check that it is true for positions $\alpha(f_{ij})$ and $\bar{\beta}(g_{ij})$. We also know, that $\bar{\beta}(f_{ij}) = \alpha(g_{ij})$ for all edge $(v_i, v_j) \in \vec{E}$ (see (xii)), however, we assigned position $\bar{\beta}(f_{ij})$ to job f_{ij} and position $\alpha(g_{ij})$ to job g_{ij} if and only if $v_i \in I$ and $v_j \in I$, respectively, however it is impossible, since I is independent. Second, it is clear that the weight of the solution is equal to $-|I|$. □

Proposition 5.3. *For an independent set problem in graph $G = (V, E)$, suppose the corresponding scheduling problem admits a feasible solution of value $W < 0$. Then there is an independent set I in G with $|I| = -W$.*

Proof. Since W is non-positive, according to Remark 5.1, for each $t \in \mathcal{T}_V \cup \mathcal{T}_{\vec{E}}$ we have either $s_{t,\alpha(t)} = e_{t,\beta(t)} = 1$ or $s_{t,\bar{\alpha}(t)} = e_{t,\bar{\beta}(t)} = 1$. We claim that the node set $I = \{v_i \in V : s_{t_i,\bar{\alpha}(t_i)} = e_{t_i,\bar{\beta}(t_i)} = 1\}$ is independent.

Suppose for a contradiction that there is an edge $(v_i, v_j) \in \vec{E}$ such that $v_i, v_j \in I$. Let $succ(i) = \{v_{j_1}, \ldots, v_{j_{|succ(i)|}}\}$ be the set of the immediate successors of node v_i. Since $e_{t_i,\bar{\beta}(t_i)} = 1$ and by construction $\bar{\beta}(t_i) = \alpha(f_{ij_1})$, thus $s_{f_{ij_1},\alpha(f_{ij_1})} = 0$ and therefore $s_{f_{ij_1},\bar{\alpha}(f_{ij_1})} = e_{f_{ij_1},\bar{\beta}(f_{ij_1})} = 1$. Again, by construction $\bar{\alpha}(f_{ij_1}) = \alpha(f_{ij_2})$, thus $s_{f_{ij_2},\alpha(f_{ij_2})} = 0$ and therefore $s_{f_{ij_2},\bar{\alpha}(f_{ij_2})} = e_{f_{ij_2},\bar{\beta}(f_{ij_2})} = 1$. Similarly, we can show that $s_{f_{ij_\ell},\bar{\alpha}(f_{ij_\ell})} = e_{f_{ij_\ell},\bar{\beta}(f_{i,j_\ell})} = 1$ holds for all $\ell = 1, \ldots, |succ(i)|$, moreover, since $j = j_\ell$ for some $\ell \in \{1, \ldots, |succ(i)|\}$ we have $e_{f_{ij},\bar{\beta}(f_{ij})} = 1$.

Let $pred(j) = \{v_{i_1}, \ldots, v_{i_{|pred(j)|}}\}$ be the set of the immediate predecessors of node v_j. Similarly, we can show that $s_{g_{i_\ell j},\alpha(g_{i_\ell j})} = e_{g_{i_\ell j},\beta(g_{i_\ell j})} = 1$ holds for all $\ell = 1, \ldots, |pred(j)|$, and since $i = i_\ell$ for some $\ell \in \{1, \ldots, |pred(j)|\}$ we have $s_{g_{ij},\alpha(g_{ij})} = 1$.

To sum up, we have $e_{f_{ij},\bar{\beta}(f_{ij})} = s_{g_{ij},\alpha(g_{ij})} = 1$ which yields a contradiction, since by construction $\bar{\beta}(f_{ij}) = \alpha(g_{ij})$. □

Finally, it is easy to see that our transformation is a pseudo-polynomial transformation,

thus the problem is NP-hard in the strong sense. □

Corollary 5.1. *Problem* $1 \,|\, chains, p_j = 1 \,|\, \sum w_{j,\sigma_j}$ *is strongly* NP*-hard even in the case of chains of length at most 2.*

Corollary 5.2. *Problem* $1 \,|\, prec, p_j = 1 \,|\, \sum w_{j,\sigma_j}$ *is strongly* NP*-hard.*

Corollary 5.3. *Problem* $1 \,|\, or\text{-}prec, p_j = 1 \,|\, \sum w_{j,\sigma_j}$ *is strongly* NP*-hard.*

5.3.3 Dimension of $P_{2n}^{2\text{-}chains}$

In this section we investigate the dimension of the polytope $P_{2n}^{2\text{-}chains}$.

Theorem 5.5.

$$\dim(P_{2n}^{2\text{-}chains}) = \begin{cases} 0 & \text{if } n = 1, \\ 4 & \text{if } n = 2, \\ 4n^2 - 6n + 1 & \text{if } n \geq 3. \end{cases}$$

Sketch of the proof of Theorem 5.5 ($n \geq 3$). In the case of $n \geq 3$ we will use Proposition 2.1 to prove the theorem. That is, we will provide an equation system for $P_{2n}^{2\text{-}chains}$ (see Theorem 5.6) with rank $6n - 1$ (see Proposition 5.6), which gives that the dimension of $P_{2n}^{2\text{-}chains} \subseteq \mathbb{R}^{4n^2}$ is $4n^2 - (6n - 1)$. The detailed proof of Theorem 5.5 can be found at the end of Section 5.3.3.

Theorem 5.6. *Let $n \geq 3$. The equation set $\mathcal{E} := \{(5.16) - (5.22)\}$ is an equation system for $P_{2n}^{2\text{-}chains}$.*

Proof. Clearly, the equations of $\mathcal{E}$ hold for every point of $P_{2n}^{2\text{-}chains}$, since they are defining equations for this polyhedron. In order to show that $\mathcal{E}$ is an equation system for $P_{2n}^{2\text{-}chains}$, we show that any other equation which holds for all points of $P_{2n}^{2\text{-}chains}$ is a linear combination of the equations of $\mathcal{E}$. Assume that

$$\sum_{i=1}^{n} \sum_{j=1}^{2n} \alpha_{ij} s_{ij} + \sum_{i=1}^{n} \sum_{j=1}^{2n} \beta_{ij} e_{ij} = \gamma \tag{5.25}$$

holds for all $(s, e) \in P_{2n}^{2\text{-}chains}$. To show that equation (5.25) is a linear combination of equations (5.16)–(5.22) we explicitly create a linear combination (5.26), and in Proposition 5.4 and Proposition 5.5 we prove that (5.25) and (5.26) are the same. In those proposition we use Lemma 5.2, however, for its proofs we refer to the appendix.

Lemma 5.2. *Equation (5.25) satisfies the following properties:*

(i) $\alpha_{p,j''} - \alpha_{p,j'} = \beta_{q,j''} - \beta_{q,j'}$ *for all* $p, q = 1, \ldots, n$ *and* $1 < j' < j'' < 2n$,
(ii) $\alpha_{p,j''} - \alpha_{p,j'} = \alpha_{q,j''} - \alpha_{q,j'}$ *for all* $p, q = 1, \ldots, n$ *and* $1 \leq j' < j'' < 2n$,
(iii) $\beta_{p,j''} - \beta_{p,j'} = \beta_{q,j''} - \beta_{q,j'}$ *for all* $p, q = 1, \ldots, n$ *and* $1 < j' < j'' \leq 2n$.

Note that in the case of (i) p may be equal to q.

Consider the linear combination of equations (5.16)–(5.22) with coefficients $\lambda_i^{5.16}$, $\lambda_i^{5.17}$, $\lambda_i^{5.18}$, $\lambda_i^{5.19}$, $\lambda^{5.20}$, $\lambda_j^{5.21}$, $\lambda^{5.22}$ ($i = 1, \ldots, n$, $j = 2, \ldots, 2n - 1$), respectively, where

- $\lambda_i^{5.16} = \alpha_{i,1} - \alpha_{1,1}$ for all $i = 1, \ldots, n$,
- $\lambda_i^{5.17} = \beta_{i,2n} - \beta_{1,2n} + \beta_{1,2} - \alpha_{1,2}$ for all $i = 1, \ldots, n$,
- $\lambda_i^{5.18} = \alpha_{i,2n} - \alpha_{i,1} + \alpha_{1,1}$ for all $i = 1, \ldots, n$,
- $\lambda_i^{5.19} = \beta_{i,1} - \beta_{i,2n} + \beta_{1,2n} - \beta_{1,2} + \alpha_{1,2}$ for all $i = 1, \ldots, n$,
- $\lambda^{5.20} = \alpha_{1,1}$,
- $\lambda_j^{5.21} = \alpha_{1,j}$ for all $j = 2, \ldots, 2n-1$,
- $\lambda^{5.22} = \beta_{1,2n} - \beta_{1,2} + \alpha_{1,2}$.

Let

$$\sum_{i=1}^{n}\sum_{j=1}^{2n} \hat{\alpha}_{ij} s_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{2n} \hat{\beta}_{ij} e_{ij} = \hat{\gamma} \tag{5.26}$$

be the equation obtained. Note that the left-hand side can be written as

$$\sum_{i=1}^{n} \Big((\lambda_i^{5.16} + \lambda^{5.20}) s_{i,1} + (\lambda_i^{5.16} + \lambda_i^{5.18}) s_{i,2n} + (\lambda_i^{5.17} + \lambda_i^{5.19}) e_{i,1} + (\lambda_i^{5.17} + \lambda^{5.22}) e_{i,2n} \Big) + \\ + \sum_{i=1}^{n}\sum_{j=2}^{2n-1} \Big((\lambda_i^{5.16} + \lambda_j^{5.21}) s_{ij} + (\lambda_i^{5.17} + \lambda_j^{5.21}) e_{ij} \Big).$$

Proposition 5.4. *Equation* (5.26) *satisfies the following:*
(I) $\hat{\alpha}_{ij} = \alpha_{ij}$ *for all* $i = 1, \ldots, n$ *and* $j = 1, \ldots, 2n$.

Proof. Let $i = 1, \ldots, n$ be fixed. For $j = 1$ we have

$$\hat{\alpha}_{i,1} = \lambda_i^{5.16} + \lambda^{5.20} = (\alpha_{i,1} - \alpha_{1,1}) + \alpha_{1,1} = \alpha_{i,1},$$

and for $j = 2n$ we have

$$\hat{\alpha}_{i,2n} = \lambda_i^{5.16} + \lambda_i^{5.18} = (\alpha_{i,1} - \alpha_{1,1}) + (\alpha_{i,2n} - \alpha_{i,1} + \alpha_{1,1}) = \alpha_{i,2n}.$$

For a given $j = 2, \ldots, 2n-1$ we have

$$\hat{\alpha}_{ij} = \lambda_i^{5.16} + \lambda_j^{5.21} = (\alpha_{i,1} - \alpha_{1,1}) + \alpha_{1,j} \overset{(ii)}{=} \alpha_{ij},$$

where for the last equation we use statement (ii) of Lemma 5.2 with $p = 1$, $q = i$, $j' = 1$ and $j'' = j$. □

Proposition 5.5. *For linear combination* (5.26) *the following statement holds:*
(II) $\hat{\beta}_{ij} = \beta_{ij}$ *for all* $i = 1, \ldots, n$ *and* $j = 1, \ldots, 2n$.

Proof. Let $i = 1, \ldots, n$ be fixed. For $j = 1$ we have

$$\hat{\beta}_{i,1} = \lambda_i^{5.17} + \lambda_i^{5.19} = (\beta_{i,2n} - \beta_{1,2n} + \beta_{1,2} - \alpha_{1,2}) + (\beta_{i,1} - \beta_{i,2n} + \beta_{1,2n} - \beta_{1,2} + \alpha_{1,2}) = \beta_{i,1},$$

and for $j = 2n$ we have

$$\hat{\beta}_{1,2n} = \lambda_i^{5.17} + \lambda^{5.22} = (\beta_{i,2n} - \beta_{1,2n} + \beta_{1,2} - \alpha_{1,2}) + (\beta_{1,2n} - \beta_{1,2} + \alpha_{1,2}) = \beta_{i,2n}.$$

For a given $j = 2, \dots, 2n-1$ we have

$$\hat{\beta}_{ij} = \lambda_i^{5.17} + \lambda_j^{5.21} = (\beta_{i,2n} - \beta_{1,2n} + \beta_{1,2} - \alpha_{1,2}) + \alpha_{1,j} \overset{(iii)}{=} \beta_{i,2} - \alpha_{1,2} + \alpha_{1,j} \overset{(i)}{=} \beta_{ij}.$$

since $\beta_{i,2n} - \beta_{1,2n} + \beta_{1,2} = \beta_{i,2}$ according to statement (iii) of Lemma 5.2, and $\beta_{i,2} - \alpha_{1,2} + \alpha_{1,j} = \beta_{ij}$ due to statement (i) of Lemma 5.2. □

Corollary 5.4. *Equation* (5.26) *is equivalent to* (5.25).

Proof. According to Propositions 5.4 and 5.5, the left-hand-sides of (5.26) and (5.25) are the same. Since both of them are satisfied by all the points in $P_{2n}^{2\text{-chains}}$, the right-hand-sides also coincide. □

□

Proposition 5.6. *Let* $n \geq 3$. *The rank of the equation system* $\mathcal{E} = \{(5.16) - (5.22)\}$ *is* $6n - 1$.

Proof. Consider a linear combination of equations (5.16)–(5.22) with coefficients $\lambda_i^{5.16}$, $\lambda_i^{5.17}$, $\lambda_i^{5.18}$, $\lambda_i^{5.19}$, $\lambda^{5.20}$, $\lambda_j^{5.21}$, $\lambda^{5.22}$ ($i = 1, \dots, n$, $j = 2, \dots, 2n-1$), respectively. This linear combination can be written as

$$\sum_{i=1}^{n} \Big((\lambda_i^{5.16} + \lambda^{5.20}) s_{i,1} + (\lambda_i^{5.16} + \lambda_i^{5.18}) s_{i,2n} + (\lambda_i^{5.17} + \lambda_i^{5.19}) e_{i,1} + (\lambda_i^{5.17} + \lambda^{5.22}) e_{i,2n} \Big) +$$

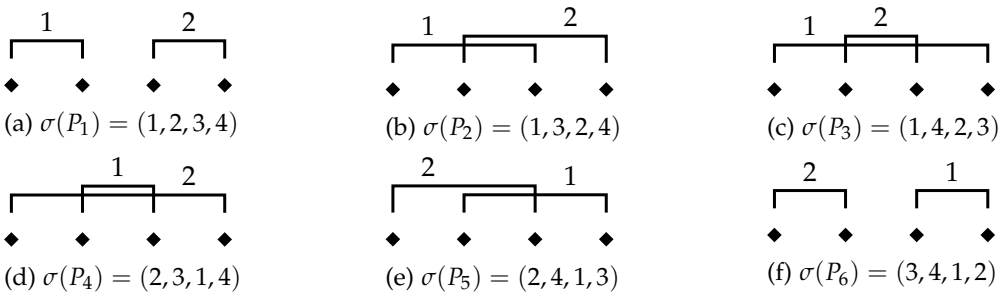
$$+ \sum_{i=1}^{n} \sum_{j=2}^{2n-1} \Big((\lambda_i^{5.16} + \lambda_j^{5.21}) s_{ij} + (\lambda_i^{5.17} + \lambda_j^{5.21}) e_{ij} \Big) = \lambda^{5.20} + \lambda^{5.22} + \sum_{i=1}^{n} \Big(\lambda_i^{5.16} + \lambda_i^{5.17} \Big) + \sum_{j=1}^{2n} \lambda_j^{5.21}.$$

The expression above reduces to the zero-equation ($0 \cdot s + 0 \cdot e = 0$) if and only if $\lambda_i^{5.18} = -\lambda_i^{5.16} = \lambda^{5.20}$, $\lambda_i^{5.16} = -\lambda_j^{5.21} = \lambda_i^{5.17}$ and $\lambda_i^{5.19} = -\lambda_i^{5.17} = \lambda^{5.22}$ hold for all $i = 1, \dots, n$, $j = 2, \dots, 2n-1$ and the right-hand side is zero. On the one hand, it is clear that we can easily choose non-zero coefficients that yield the zero-equation, thus the equations are linearly dependent. On the other hand, if we omit a single equation from (5.16)–(5.22), that is, we fix a single coefficient from $\lambda_i^{5.16}, \dots, \lambda^{5.22}$ to zero, then all the remaining coefficients will be zero, that is, that remaining equations are linearly independent. Hence, the equation system $\{(5.16) - (5.22)\}$ containing $6n$ equations has rank $6n - 1$. □

Proof of Theorem 5.5. In the case of $n = 1$, $S_2^{2\text{-chains}}$ consists of a single point P with $\sigma(P) = (1, 2)$, thus $\dim(P_2^{2\text{-chains}}) = 0$.

In the case of $n = 2$ in order to prove that $\dim(P_4^{2\text{-chains}}) = 4$ we show that the maximum number of affinely independent points in $S_4^{2\text{-chains}}$ is 5. We have $S_4^{2\text{-chains}} = \{P_1, \dots, P_6\}$, where $\sigma(P_1) = (1, 2, 3, 4)$, $\sigma(P_2) = (1, 3, 2, 4)$, $\sigma(P_3) = (1, 4, 2, 3)$, $\sigma(P_4) = (2, 3, 1, 4)$, $\sigma(P_5) = (2, 4, 1, 3)$, $\sigma(P_6) = (3, 4, 1, 2)$, see Fig. 5.10. The linear combination of these points with coefficients $\lambda_1, \dots, \lambda_6$, respectively, is

$$\begin{aligned}(\lambda_1 + \lambda_2 + \lambda_3) s_{1,1} + (\lambda_4 + \lambda_5) s_{1,2} + \lambda_6 s_{1,3} + (\lambda_4 + \lambda_5 + \lambda_6) s_{2,1} + (\lambda_2 + \lambda_3) s_{2,2} + \\ + \lambda_1 s_{2,3} + \lambda_1 e_{1,2} + (\lambda_2 + \lambda_4) e_{1,3} + (\lambda_3 + \lambda_5 + \lambda_6) e_{1,4} + \lambda_6 e_{2,2} + \\ + (\lambda_3 + \lambda_5) e_{2,3} + (\lambda_1 + \lambda_2 + \lambda_4) e_{2,4}.\end{aligned}$$



(a) $\sigma(P_1) = (1,2,3,4)$ (b) $\sigma(P_2) = (1,3,2,4)$ (c) $\sigma(P_3) = (1,4,2,3)$

(d) $\sigma(P_4) = (2,3,1,4)$ (e) $\sigma(P_5) = (2,4,1,3)$ (f) $\sigma(P_6) = (3,4,1,2)$

Figure 5.10: [Proof of Theorem 5.5] The six points in $S_4^{2\text{-chains}}$

Clearly, we get the zero-vector if and only if $\lambda_1 = 0$, $\lambda_6 = 0$ and $\lambda_2 = -\lambda_3 = \lambda_5 = -\lambda_4$. On the one hand, we can easily choose non-zero $\lambda_2, \ldots, \lambda_5$ coefficients to get the zero-vector such that $\lambda_1 + \ldots + \lambda_6 = 0$ also holds, thus points $P_1, \ldots, P_6$ are affinely dependent. On the other hand, if we omit for example P_2, i.e., we fix $\lambda_2 = 0$, we could get the zero-vector if and only if $\lambda_1 = \ldots = \lambda_6 = 0$, that is, points P_1, P_3, P_4, P_5, P_6 are linearly and hence affinely independent. Therefore $\dim(P_2^{2\text{-chains}}) = 4$.

Finally, assume that $n \geq 3$. According to Theorem 5.6, the equation set $\mathcal{E} = \{(5.16) - (5.22)\}$ is an equation system for $P_{2n}^{2\text{-chains}}$, and according to Proposition 5.6, the rank of this system is $6n - 1$. Since we have $4n^2$ variables, thus the dimension of $P_{2n}^{2\text{-chains}}$ is $4n^2 - (6n - 1)$. □

5.3.4 Parity inequalities

In the case of general chain-precedence constraints we showed that parity inequalities (5.14) and (5.15) are valid for P_n^{chain} (see Section 5.2), thus they are also valid in the case of 2-chains. Using the replacement of the variables (remark that $s_{ij} = x_{2i-1,j}$ and $e_{ij} = x_{2i,j}$) the following parity inequalities are valid for $P_{2n}^{2\text{-chains}}$:

$$\sum_{i \in S} \sum_{j=1}^{2k} (s_{ij} - e_{ij}) - \sum_{i \notin S} \sum_{j=1}^{2k} (s_{ij} - e_{ij}) \leq |S| - 1,$$

for all odd-subset $S \subseteq [n]$, and $k < n$, (5.27)

and

$$\sum_{i \in S} \sum_{j=1}^{2k-1} (s_{ij} - e_{ij}) - \sum_{i \notin S} \sum_{j=1}^{2k-1} (s_{ij} - e_{ij}) \leq |S| - 1,$$

for all even-subset $S \subseteq [n]$, and $k \leq n$. (5.28)

In this section we show that some of the inequalities (5.27) are facet-defining for $P_{2n}^{2\text{-chains}}$. Similarly, one can show that a subset of inequalities (5.28) are also facet-inducing.

Let $3 \leq t < n$ be a fixed odd number, $1 \leq k < n$ such that $t < 2k$ and $t < 2(n-k)$ hold, and $S \subseteq [n]$ with cardinality $|S| = t$. To simplify our notation, without loss of generality, we assume that $S = \{1, \ldots, t\}$. The corresponding parity inequality is:

$$\sum_{i=1}^{t} \sum_{j=1}^{2k} (s_{ij} - e_{ij}) \leq t - 1 + \sum_{i=t+1}^{n} \sum_{j=1}^{2k} (s_{ij} - e_{ij}). \tag{5.29}$$

Theorem 5.7. *Let $3 \leq t < n$ be a fixed odd number, $1 \leq k < n$ such that $t < 2k$ and $t < 2(n-k)$ hold, and $S = \{1, \ldots, t\}$. Inequalities (5.29) are facet-defining for $P_{2n}^{2\text{-chains}}$.*

Remarks 5.2. *Consider a point from $S_{2n}^{2\text{-chains}}$. We say that a chain $C_i = (J_{2i-1}, J_{2i})$ is active in interval $[2k, 2k+1]$ if $\sum_{j=1}^{2k}(s_{ij} - e_{ij}) = 1$ holds (that is, its first job J_{2i-1} is assigned before position $2k+1$, and its second job J_{2i} is assigned after position $2k$). A point from $S_{2n}^{2\text{-chains}}$ satisfies (5.29) with equality if and only if*

- *exactly $t-1$ chains from $\{1, \ldots, t\}$ and no chain from $\{t+1, \ldots, n\}$ are active in interval $[2k, 2k+1]$; or*
- *exactly t chains from $\{1, \ldots, t\}$ and exactly 1 chain from $\{t+1, \ldots, n\}$ are active in interval $[2k, 2k+1]$.*

Sketch of the proof of Theorem 5.7. Let us define the set of points

$$S_{2n}^{parity} := \{(s, e) \in S_{2n}^{2\text{-chains}} : (s, e) \text{ satisfies (5.29) with equality}\},$$

and the polyhedron of their convex hull $P_{2n}^{parity} := \text{conv}(S_{2n}^{parity})$. Note that P_{2n}^{parity} is a proper face of $P_{2n}^{2\text{-chains}}$. To prove that inequalities (5.29) are facet-defining for $P_{2n}^{2\text{-chains}}$, we will show that P_{2n}^{parity} is a facet of $P_{2n}^{2\text{-chains}}$, i.e., $\dim(P_{2n}^{parity}) = \dim(P_{2n}^{2\text{-chains}}) - 1$. To do this, we apply a similar procedure as in Section 5.3.3, that is, we will prove that the set $\mathcal{E}' := \mathcal{E} \cup \{(5.30)\} = \{(5.16) - (5.22), (5.30)\}$ of equations contains a minimal equation system for P_{2n}^{parity} with rank $\dim(P_{2n}^{2\text{-chains}}) - 1$, where we have

$$\sum_{i=1}^{t} \sum_{j=1}^{2k} (s_{ij} - e_{ij}) + \sum_{i=t+1}^{n} \sum_{j=1}^{2k} (e_{ij} - s_{ij}) = t - 1. \tag{5.30}$$

The detailed proof can be found in the end of Section 5.3.4.

Theorem 5.8. *The equation set $\mathcal{E}' = \{(5.16) - (5.22), (5.30)\}$ is an equation system for P_{2n}^{parity}.*

Proof. Clearly, the equations of $\mathcal{E}'$ hold for every point of P_{2n}^{parity} since they are defining equations for S_{2n}^{parity}. Assume that

$$\sum_{i=1}^{n} \sum_{j=1}^{2n} \alpha_{ij} s_{ij} + \sum_{i=1}^{n} \sum_{j=1}^{2n} \beta_{ij} e_{ij} = \gamma \tag{5.31}$$

holds for all $(s, e) \in P_{2n}^{parity}$. In order to show that equation (5.31) is a linear combination of equations (5.16)–(5.22) and (5.30) we explicitly create a linear combination (5.32), and in Propositions 5.7 to 5.10 we prove that (5.31) and (5.32) are the same. In those proposition we use Lemmas 5.3 and 5.4, however for their proofs we refer to the appendix.

Lemma 5.3. *For equation (5.31) the following statements hold:*

(i) $\alpha_{p,j''} - \alpha_{p,j'} = \alpha_{q,j''} - \alpha_{q,j'}$ *for all* $p,q = 1,\dots,t$ *and* $1 \le j' < j'' \le 2k$,
(ii) $\alpha_{p,j''} - \alpha_{p,j'} = \alpha_{q,j''} - \alpha_{q,j'}$ *for all* $p,q = 1,\dots,t$ *and* $1 \le j' \le 2k < j'' \le 2n-1$,
(iii) $\beta_{p,j''} - \beta_{p,j'} = \beta_{q,j''} - \beta_{q,j'}$ *for all* $p,q = 1,\dots,t$ *and* $2k < j' < j'' \le 2n$,
(iv) $\beta_{p,j''} - \beta_{p,j'} = \beta_{q,j''} - \beta_{q,j'}$ *for all* $p,q = 1,\dots,t$ *and* $2 \le j' \le 2k < j'' \le 2n$,
(v) $\alpha_{p,j''} - \alpha_{p,j'} = \beta_{q,j''} - \beta_{q,j'}$ *for all* $p,q = 1,\dots,t$ *and* $1 < j' < j'' \le 2k$,
(vi) $\alpha_{p,j''} - \alpha_{p,j'} = \beta_{q,j''} - \beta_{q,j'}$ *for all* $p,q = 1,\dots,t$ *and* $2k < j' < j'' < 2n$.

Note that in the case of (v) and (vi) p *may be equal to* q.

Lemma 5.4. *For equation (5.31) the following statements hold:*

(vii) $\alpha_{p,j''} - \alpha_{p,j'} = \alpha_{\bar{q},j''} - \alpha_{\bar{q},j'}$ *for all* $p = 1,\dots,t$, $\bar{q} = t+1,\dots,n$ *and* $1 \le j' < j'' \le 2k$,
(viii) $\beta_{p,j''} - \beta_{p,j'} = \beta_{\bar{q},j''} - \beta_{\bar{q},j'}$ *for all* $p = 1,\dots,t$, $\bar{q} = t+1,\dots,n$ *and* $2k < j' < j'' \le 2n$,
(ix) $\alpha_{p,j''} - \alpha_{p,j'} = \beta_{\bar{q},j''} - \beta_{\bar{q},j'}$ *for all* $p = 1,\dots,t$, $\bar{q} = t+1,\dots,n$ *and* $1 < j' < j'' \le 2k$,
(x) $\alpha_{p,j''} - \alpha_{p,j'} = \beta_{\bar{q},j''} - \beta_{\bar{q},j'}$ *for all* $p = 1,\dots,t$, $\bar{q} = t+1,\dots,n$ *and* $1 < j' \le 2k < j'' < 2n$,
(xi) $\beta_{p,j''} - \beta_{p,j'} = \alpha_{\bar{q},j''} - \alpha_{\bar{q},j'}$ *for all* $p = 1,\dots,t$, $\bar{q} = t+1,\dots,n$ *and* $1 < j' \le 2k < j'' < 2n$,
(xii) $\beta_{p,j''} - \beta_{p,j'} = \alpha_{\bar{q},j''} - \alpha_{\bar{q},j'}$ *for all* $p = 1,\dots,t$, $\bar{q} = t+1,\dots,n$ *and* $2k < j' < j'' < 2n$.

Consider the linear combination of equations (5.16)–(5.22) and (5.30) with coefficients $\lambda_i^{5.16}$, $\lambda_i^{5.17}$, $\lambda_i^{5.18}$, $\lambda_i^{5.19}$, $\lambda^{5.20}$, $\lambda_j^{5.21}$, $\lambda^{5.22}$ and $\lambda^{5.30}$, $(i = 1,\dots,n,\ j = 2,\dots,2n-1)$ respectively, where

- $\lambda^{5.30} = \lambda$, where $\lambda := (\alpha_{1,2} - \alpha_{1,2k+1} - \beta_{1,2} + \beta_{1,2k+1})/2$,
- $\lambda_i^{5.16} = \begin{cases} \alpha_{i,1} - \alpha_{1,1} & \text{if } i = 1,\dots,t, \\ \alpha_{i,1} - \alpha_{1,1} + 2\lambda & \text{if } i = t+1,\dots,n, \end{cases}$
- $\lambda_i^{5.17} = \mu_i$, where $\mu_i := \begin{cases} \beta_{1,2} - \alpha_{1,2} + 2\lambda & \text{if } i = 1, \\ \beta_{i,2n} - \beta_{1,2n} + \mu_1 & \text{if } i = 2,\dots,n, \end{cases}$
- $\lambda_i^{5.18} = \begin{cases} \alpha_{1,2n} & \text{if } i = 1, \\ \alpha_{i,2n} - \alpha_{i,1} + \alpha_{1,1} & \text{if } i = 2,\dots,t, \\ \alpha_{i,2n} - \alpha_{i,1} + \alpha_{1,1} - 2\lambda & \text{if } i = t+1,\dots,n, \end{cases}$
- $\lambda_i^{5.19} = \begin{cases} \beta_{i,1} + \lambda - \mu_i & \text{if } i = 1,\dots,t, \\ \beta_{i,1} - \lambda - \mu_i & \text{if } i = t+1,\dots,n, \end{cases}$
- $\lambda^{5.20} = \alpha_{1,1} - \lambda$,
- $\lambda_j^{5.21} = \begin{cases} \alpha_{1,j} - \lambda & \text{if } j = 2,\dots,2k, \\ \alpha_{1,j} & \text{if } j = 2k+1,\dots,2n-1, \end{cases}$
- $\lambda^{5.22} = \beta_{1,2n} - \mu_1$.

Let

$$\sum_{i=1}^{n}\sum_{j=1}^{2n} \hat{\alpha}_{ij} s_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{2n} \hat{\beta}_{ij} e_{ij} = \hat{\gamma} \tag{5.32}$$

be the resulting equation. Note that the left-hand side can be written as

$$\begin{aligned}
&\sum_{i=1}^{t}\Big((\lambda_i^{5.16}+\lambda^{5.20}+\lambda^{5.30})s_{i,1}+(\lambda_i^{5.17}+\lambda_i^{5.19}-\lambda^{5.30})e_{i,1}\Big)\\
&+\sum_{i=t+1}^{n}\Big((\lambda_i^{5.16}+\lambda^{5.20}-\lambda^{5.30})s_{i,1}+(\lambda_i^{5.17}+\lambda_i^{5.19}+\lambda^{5.30})e_{i,1}\Big)\\
&+\sum_{i=1}^{n}\Big((\lambda_i^{5.16}+\lambda_i^{5.18})s_{i,2n}+(\lambda_i^{5.17}+\lambda^{5.22})e_{i,2n}\Big)\\
&+\sum_{i=1}^{t}\sum_{j=2}^{2k}\Big((\lambda_i^{5.16}+\lambda_j^{5.21}+\lambda^{5.30})s_{ij}+(\lambda_i^{5.17}+\lambda_j^{5.21}-\lambda^{5.30})e_{ij}\Big)\\
&+\sum_{i=t+1}^{n}\sum_{j=2}^{2k}\Big((\lambda_i^{5.16}+\lambda_j^{5.21}-\lambda^{5.30})s_{ij}+(\lambda_i^{5.17}+\lambda_j^{5.21}+\lambda^{5.30})e_{ij}\Big)\\
&+\sum_{i=1}^{n}\sum_{j=2k+1}^{2n-1}\Big((\lambda_i^{5.16}+\lambda_j^{5.21})s_{ij}+(\lambda_i^{5.17}+\lambda_j^{5.21})e_{ij}\Big).
\end{aligned}$$

Proposition 5.7. *For linear combination* (5.32) *the following statement holds:*
(I) $\hat{\alpha}_{ij} = \alpha_{ij}$ *for all* $i = 1, \dots, t$ *and* $j = 1, \dots, 2n$.

Proof. By construction, the statement clearly holds for $i = 1$. Let $i = 2, \dots, t$ be fixed. For $j = 1$ we have

$$\hat{\alpha}_{i,1} = \lambda_i^{5.16} + \lambda^{5.20} + \lambda^{5.30} = (\alpha_{i,1} - \alpha_{1,1}) + (\alpha_{1,1} - \lambda) + \lambda = \alpha_{i,1},$$

and for $j = 2n$ we have

$$\hat{\alpha}_{i,2n} = \lambda_i^{5.16} + \lambda_i^{5.18} = (\alpha_{i,1} - \alpha_{1,1}) + (\alpha_{i,2n} + \alpha_{1,1} - \alpha_{i,1}) = \alpha_{i,2n}.$$

For a given $j = 2, \dots, 2k$ we have

$$\hat{\alpha}_{ij} = \lambda_i^{5.16} + \lambda_j^{5.21} + \lambda^{5.30} = (\alpha_{i,1} - \alpha_{1,1}) + (\alpha_{1,j} - \lambda) + \lambda = \alpha_{1,j} - \alpha_{1,1} + \alpha_{i,1} \overset{(i)}{=} \alpha_{ij},$$

where for the last equation we use statement (i) of Lemma 5.3 with $p = 1$, $q = i$, $j' = 1$, and $j'' = j$. Finally, for a given $j = 2k+1, \dots, 2n-1$ we have

$$\hat{\alpha}_{ij} = \lambda_i^{5.16} + \lambda_j^{5.21} = \alpha_{1,j} - \alpha_{1,1} + \alpha_{i,1} \overset{(ii)}{=} \alpha_{ij},$$

where for the last equation we use statement (ii) of Lemma 5.3 with $p = 1$, $q = i$, $j' = 1$ and $j'' = j$. □

Proposition 5.8. *For linear combination* (5.32) *the following statement holds:*
(II) $\hat{\beta}_{ij} = \beta_{ij}$ *for all* $i = 1, \dots, t$ *and* $j = 1, \dots, 2n$.

Proof. First, assume that $i = 1$. For $j = 1$ we have

$$\hat{\beta}_{1,1} = \lambda_1^{5.17} + \lambda_1^{5.19} - \lambda^{5.30} = \mu_1 + (\beta_{1,1} + \lambda - \mu_1) - \lambda = \beta_{1,1},$$

and for $j = 2n$ we have

$$\hat{\beta}_{1,2n} = \lambda_1^{5.17} + \lambda^{5.22} = \mu_1 + (\beta_{1,2n} - \mu_1) = \beta_{1,2n}.$$

For a given $j = 2, \dots, 2k$ we have

$$\hat{\beta}_{1,j} = \lambda_1^{5.17} + \lambda_j^{5.21} - \lambda^{5.30} = (\beta_{1,2} - \alpha_{1,2} + 2\lambda) + (\alpha_{1,j} - \lambda) - \lambda = \beta_{1,2} + \alpha_{1,j} - \alpha_{1,2} \overset{(v)}{=} \beta_{1,j},$$

where the last equation clearly holds for $j = 2$, and for $2 < j$ we can use statement (v) of Lemma 5.3 with $p = q = 1$, $j' = 2$ and $j'' = j$. For a given $j = 2k+1, \dots, 2n-1$ we have

$$\hat{\beta}_{1,j} = \lambda_1^{5.17} + \lambda_j^{5.21} = \beta_{1,2} + \alpha_{1,j} - \alpha_{1,2} + 2\lambda = \alpha_{1,j} - \alpha_{1,2k+1} + \beta_{1,2k+1} \overset{(vi)}{=} \beta_{ij},$$

according to statement (vi) of Lemma 5.3 with $p = q = 1$, $j' = 2k+1$ and $j'' = j$. Now, let $i = 2, \dots, t$. For $j = 1$ we have

$$\hat{\beta}_{i,1} = \lambda_i^{5.17} + \lambda_i^{5.19} - \lambda^{5.30} = \mu_i + (\beta_{i,1} + \lambda - \mu_i) - \lambda = \beta_{i,1},$$

and for $j = 2n$ we have

$$\hat{\beta}_{1,2n} = \lambda_i^{5.17} + \lambda^{5.22} = (\beta_{i,2n} - \beta_{1,2n} + \mu_1) + (\beta_{1,2n} - \mu_1) = \beta_{i,2n}.$$

For a given $j = 2, \dots, 2k$ we have

$$\begin{aligned}\hat{\beta}_{ij} = \lambda_i^{5.17} + \lambda_j^{5.21} - \lambda^{5.30} &= (\beta_{i,2n} - \beta_{1,2n} + \beta_{1,2} - \alpha_{1,2} + 2\lambda) + (\alpha_{1,j} - \lambda) - \lambda \\ &= \beta_{i,2n} - \beta_{1,2n} + \beta_{1,2} - \alpha_{1,2} + \alpha_{1,j} \overset{(v)}{=} \beta_{i,2n} - \beta_{1,2n} + \beta_{1,j} \overset{(iv)}{=} \beta_{ij},\end{aligned}$$

since $\beta_{1,2} - \alpha_{1,2} + \alpha_{1,j} = \beta_{1,j}$ according to statement (v) of Lemma 5.3 with $p = q = 1$, $j' = 2$ and $j'' = j$, and $\beta_{i,2n} - \beta_{1,2n} + \beta_{1,j} = \beta_{ij}$ due to statement (iv) of Lemma 5.3 with $p = 1$, $q = i$, $j' = j$ and $j'' = 2n$. Finally, for a given $j = 2k+1, \dots, 2n-1$ we have

$$\hat{\beta}_{ij} = \lambda_i^{5.17} + \lambda_j^{5.21} = \beta_{i,2n} - \beta_{1,2n} + \alpha_{1,j} - \alpha_{1,2k+1} + \beta_{1,2k+1} \overset{(vi)}{=} \beta_{i,2n} - \beta_{1,2n} + \beta_{1,j} \overset{(iv)}{=} \beta_{ij},$$

since $\alpha_{1,j} - \alpha_{1,2k+1} + \beta_{1,2k+1} = \beta_{1,j}$ according to statement (vi) of Lemma 5.3 with $p = q = 1$, $j' = 2k+1$ and $j'' = j$, and $\beta_{i,2n} - \beta_{1,2n} + \beta_{1,j} = \beta_{ij}$ due to statement (iv) of Lemma 5.3 with $p = 1$, $q = i$, $j' = j$ and $j'' = 2n$. □

Proposition 5.9. *For linear combination* (5.32) *the following statement holds:*
(III) $\hat{\alpha}_{ij} = \alpha_{ij}$ *for all* $i = t+1, \dots, n$ *and* $j = 1, \dots, 2n$.

Proof. Let $i = t+1, \dots, n$ be fixed. For $j = 1$ we have

$$\hat{\alpha}_{i,1} = \lambda_i^{5.16} + \lambda^{5.20} - \lambda^{5.30} = (\alpha_{i,1} - \alpha_{1,1} + 2\lambda) + (\alpha_{1,1} - \lambda) - \lambda = \alpha_{i,1},$$

and for $j = 2n$ we have

$$\hat{\alpha}_{i,2n} = \lambda_i^{5.16} + \lambda_i^{5.18} = (\alpha_{i,1} - \alpha_{1,1} + 2\lambda) + (\alpha_{i,2n} - \alpha_{i,1} + \alpha_{1,1} - 2\lambda) = \alpha_{i,2n}.$$

For a given $j = 2, \ldots, 2k$ we have

$$\hat{\alpha}_{ij} = \lambda_i^{5.16} + \lambda_j^{5.21} - \lambda^{5.30} = (\alpha_{i,1} - \alpha_{1,1} + 2\lambda) + (\alpha_{1,j} - \lambda) - \lambda = \alpha_{1,j} - \alpha_{1,1} + \alpha_{i,1} \overset{(vii)}{=} \alpha_{ij},$$

where for the last equation we use statement (vii) of Lemma 5.4 with $p = 1$, $\bar{q} = i$, $j' = 1$ and $j'' = j$. Finally, for a given $j = 2k+1, \ldots, 2n-1$ we have

$$\begin{aligned}\hat{\alpha}_{ij} = \lambda_i^{5.16} + \lambda_j^{5.21} &= (\alpha_{i,1} - \alpha_{1,1} + \alpha_{1,2} - \beta_{1,2} + \beta_{1,2k+1} - \alpha_{1,2k+1}) + \alpha_{1,j} \\ &\overset{(vi)}{=} \alpha_{i,1} - \alpha_{1,1} + \alpha_{1,2} - \beta_{1,2} + \beta_{1,j} \overset{(vii)}{=} \alpha_{i,2} - \beta_{1,2} + \beta_{1,j} \overset{(xi)}{=} \alpha_{ij},\end{aligned}$$

since $\beta_{1,2k+1} - \alpha_{1,2k+1} + \alpha_{1,j} = \beta_{1,j}$ according to statement (vi) of Lemma 5.3 with $p = q = 1$, $j' = 2k+1$ and $j'' = j$, and $\alpha_{i,1} - \alpha_{1,1} + \alpha_{1,2} = \alpha_{i,2}$ according to statement (vii) of Lemma 5.4 with $p = 1$, $\bar{q} = i$, $j' = 1$ and $j'' = 2$, and $\alpha_{i,2} - \beta_{1,2} + \beta_{1,j} = \alpha_{ij}$ due to statement (xi) of Lemma 5.4 with $p = 1$, $\bar{q} = i$, $j' = 2$ and $j'' = j$. □

Proposition 5.10. *For linear combination* (5.32) *the following statement holds:*
(IV) $\hat{\beta}_{ij} = \beta_{ij}$ *for all* $i = t+1, \ldots, n$ *and* $j = 1, \ldots, 2n$.

Proof. Let $i = t+1, \ldots, n$ be fixed. For $j = 1$ we have

$$\hat{\beta}_{i,1} = \lambda_i^{5.17} + \lambda_i^{5.19} + \lambda^{5.30} = \mu_i + (\beta_{i,1} - \lambda - \mu_i) + \lambda = \beta_{i,1},$$

and for $j = 2n$ we have

$$\hat{\beta}_{1,2n} = \lambda_i^{5.17} + \lambda^{5.22} = (\beta_{i,2n} - \beta_{1,2n} + \mu_1) + (\beta_{1,2n} - \mu_1) = \beta_{i,2n}.$$

For a given $j = 2, \ldots, 2k$ we have

$$\begin{aligned}\hat{\beta}_{ij} = \lambda_i^{5.17} + \lambda_j^{5.21} + \lambda^{5.30} &= \beta_{i,2n} + \alpha_{1,j} - \beta_{1,2n} - \alpha_{1,2k+1} + \beta_{1,2k+1} \\ &\overset{(viii)}{=} \alpha_{1,j} - \alpha_{1,2k+1} + \beta_{i,2k+1} \overset{(x)}{=} \beta_{ij},\end{aligned}$$

since $\beta_{i,2n} - \beta_{1,2n} + \beta_{1,2k+1} = \beta_{i,2k+1}$ according to statement (viii) of Lemma 5.4 with $p = 1$, $\bar{q} = i$, $j' = 2k+1$ and $j'' = 2n$, and $\alpha_{1,j} - \alpha_{1,2k+1} + \beta_{i,2k+1} = \beta_{ij}$ due to statement (x) of Lemma 5.4 with $p = 1$, $\bar{q} = i$, $j' = j$ and $j'' = 2k+1$. Finally, for a given $j = 2k+1, \ldots, 2n-1$ we have

$$\hat{\beta}_{ij} = \lambda_i^{5.17} + \lambda_j^{5.21} = \beta_{i,2n} - \beta_{1,2n} + \alpha_{1,j} - \alpha_{1,2k+1} + \beta_{1,2k+1} \overset{(vi)}{=} \beta_{i,2n} - \beta_{1,2n} + \beta_{1,j} \overset{(viii)}{=} \beta_{ij},$$

since $\alpha_{1,j} - \alpha_{1,2k+1} + \beta_{1,2k+1} = \beta_{1,j}$ according to statement (vi) of Lemma 5.4 with $p = 1$, $\bar{q} = i$, $j' = 2k+1$ and $j'' = j$, and $\beta_{i,2n} - \beta_{1,2n} + \beta_{1,j} = \beta_{ij}$ due to statement (viii) of Lemma 5.4 with $p = 1$, $\bar{q} = i$, $j' = j$ and $j'' = 2n$. □

Corollary 5.5. *Linear combination* (5.32) *yields equation* (5.31).

Proof. According to Propositions 5.7 to 5.10, the left-hand sides of (5.31) and (5.32) are the same. Since both of them are satisfied for the points from S_{2n}^{parity}, the right-hand sides also coincide with each other. □

□

Proof of Theorem 5.7. First, by definition, $rank(\mathcal{E}') \leq rank(\mathcal{E}) + 1$, thus (according to Theorems 5.6 and 5.8) $\dim(P_{2n}^{parity}) \geq \dim(P_{2n}^{2\text{-}chains}) - 1$. Second, $\dim(P_{2n}^{parity}) \leq \dim(P_{2n}^{2\text{-}chains}) - 1$ since P_{2n}^{parity} is a proper face of $P_{2n}^{2\text{-}chains}$, thus $\dim(P_{2n}^{parity}) = \dim(P_{2n}^{2\text{-}chains}) - 1$ and P_{2n}^{parity} is a facet of $P_{2n}^{2\text{-}chains}$. □

5.4 Computational results

In this section we present the results of our computational experiments, where the main goal was to examine the effectiveness of our parity inequalities. Since we proved that some of these inequalities are facet-defining if each chain has length two, our experiments focused on problems $1\,|\,2\text{-}chains, p_j = 1\,|\,\sum w_{j,\sigma_j}$ and $1\,|\,chain\text{-}length \in \{1,2\}, p_j = 1\,|\,\sum w_{j,\sigma_j}$, where in the latter case each chain has length at most two.

5.4.1 Test environment and implementation

All the computational experiments were performed on a workstation with 8GB RAM and Intel(R) Xeon(R) CPU E5-2630 v4 of 2.20 GHz, and under Linux operating system using a single thread only. Our solution method is implemented in C++ programming language using CPLEX Optimizer (CPLEX, version 12.6.3.0) as a branch-and-cut framework.

5.4.2 Instances

For problem $1\,|\,2\text{-}chains, p_j = 1\,|\,\sum w_{j,\sigma_j}$ we generated two families of instances, Family 1 and Family 2, that differ in the method of generating the cost functions, and for problem $1\,|\,chain\text{-}length \in \{1,2\}, p_j = 1\,|\,\sum w_{j,\sigma_j}$ one family of instances, Family 3, is generated. Both families consist of 30 instances, which can be further divided into problems with $n \in \{50, 100, 150\}$ jobs, i.e., 10 instances for each n. In order to generate challenging instances, for each first-job we assigned higher weight for the early positions than for the late ones, however, for each second-job we assigned lower weight for the early positions than for the late ones. Formally, in the case of Family 1, we partitioned the set of positions into 9 sets such that $\mathcal{P}_k = \{\lceil (k-1) \cdot 2n/9 \rceil + 1, \ldots, \lceil k \cdot 2n/9 \rceil\}$ for each $k = 1, \ldots, 9$, then for job J_i and position j we chose w_{ij} uniformly at random such that

- $w_{ij} \in \{10(10-k), \ldots, 10(11-k)-1\}$ if J_i is a first-job, and $j \in \mathcal{P}_k$,
- $w_{ij} \in \{10k, \ldots, 10(k+1)-1\}$ if J_i is a second-job, and $j \in \mathcal{P}_k$.

Table 5.2: Solver settings of the different methods (symbols '★' and '·' indicate if the corresponding component is used by the method or not)

	CPLEX			Parity cuts		
Method	Presolve	Heuristics	Cuts	Root	Non-root	Time limit
BnB	·	·	·	·	·	600 s
BnC (Default)	★	★	★	·	·	600 s
BnC (Parity)–1	·	·	·	·	★	600 s
BnC (Parity)–2	·	·	·	★	★	600 s

In the case of Family 2, we partitioned the set of positions into 17 subsets such that $\mathcal{P}_k = \{\lceil (k-1)\cdot 2n/17\rceil + 1, \ldots, \lceil k\cdot 2n/17\rceil\}$ for each $k = 1, \ldots, 17$, then for job J_i and position j we chose w_{ij} uniformly at random such that

- $w_{ij} \in \{10k, \ldots, 10(k+1)-1\}$ if J_i is a first-job, $k \leq 9$, and $j \in \mathcal{P}_k$,
- $w_{ij} \in \{10(18-k), \ldots, 10(19-k)-1\}$ if J_i is a first-job, $9 < k$, and $j \in \mathcal{P}_k$,
- $w_{ij} \in \{10(10-k), \ldots, 10(11-k)-1\}$ if J_i is a second-job, $k \leq 9$, and $j \in \mathcal{P}_k$,
- $w_{ij} \in \{10(k-9), \ldots, 10(k-8)-1\}$ if J_i is a second-job, $9 < k$, and $j \in \mathcal{P}_k$.

In the case of Family 3, we considered an n-length path (in terms of number of its nodes) as the precedence graph, and we randomly removed arcs from that path such that the remaining sub-paths (i.e, chains) have length at most two. We used the same weight-generation method similarly to Family 1.

5.4.3 Experiments

In these experiments we compared three solution approaches, more precisely, four scenarios corresponding to the settings summarized in Table 5.2. Method *BnB* is a pure branch-and-bound procedure, where we turned off all the presolves, heuristics and forbid CPLEX to generate built-in cuts. Method *BnC (Default)* refers to the default CPLEX settings (i.e., CPLEX performs presolves and heuristics, and generates built-in cuts). Methods *BnC (Parity)–1* and *BnC (Parity)–2* use the same solver settings as method BnB, but we also separate parity inequalities, i.e., both of these methods separate parity inequalities in enumeration tree node of depth at least 1, but method BnC (Parity)–1 does not generate any cuts in the root, while method BnC (Parity)–2 does. In each case we had a runtime limit of 600 seconds, i.e., the search was stopped upon reaching the time limit.

We summarize our results in Tables 5.3 to 5.5, where we indicate the number of jobs (n), the settings of the solver (*Method*), the lower bound after the root node is solved (LB^r), the final lower and upper bounds (LB^f and, UB^f), the final gap (Gap^f) calculated as $100 \times (UB^f - LB^f)/LB^f$, the number of investigated enumeration tree nodes (#*Nodes*), the number of generated parity inequalities (#*Cuts*), and the execution time (*Time*) in seconds.

Table 5.3: Computational results for Family 1 (averages over 10 instances)

n	Method	LB^r	LB^f	UB^f	Gap^f	#Nodes	#Cuts	Time
50	BnB	2521.1	2525.4	2525.4	0.0	3196.1	0.0	17.2
	BnC (Default)	2522.3	2525.4	2525.4	0.0	839.9	0.0	7.7
	BnC (Parity)–1	2521.1	2525.4	2525.4	0.0	4.3	19.7	1.1
	BnC (Parity)–2	2525.0	2525.4	2525.4	0.0	2.3	20.2	0.6
100	BnB	5004.8	5006.9	5021.9	0.3	14 017.2	0.0	416.9
	BnC (Default)	5005.1	5007.5	5011.0	0.1	10 670.4	0.0	397.2
	BnC (Parity)–1	5004.8	5008.5	5008.5	0.0	140.0	35.7	25.0
	BnC (Parity)–2	5006.9	5008.5	5008.5	0.0	127.6	26.4	23.3
150	BnB	7500.0	7500.0	7513.7	0.2	3740.6	0.0	346.6
	BnC (Default)	7500.0	7500.0	7500.1	0.0	1257.2	0.0	227.5
	BnC (Parity)–1	7500.0	7500.0	7500.0	0.0	12.8	35.4	42.8
	BnC (Parity)–2	7500.0	7500.0	7500.0	0.0	82.8	26.7	82.0

Experiments on problem $1 \,|\, 2\text{-}chains, p_j = 1 \,|\, \sum w_{j,\sigma_j}$

In Tables 5.3 and 5.4 we summarize our results on Family 1 and Family 2, while for the detailed results we refer to (Horváth and Kis, 2019b, Tables 6–11). Our observations are the followings.

- Methods BnC (Parity)–1 and BnC (Parity)–2 significantly outperformed the other ones in all aspects. First, only these methods were able to solve all instances optimally (one can see that the average gap is always 0.0). Second, for each instance, method BnC (Parity)–1 needed shorter execution time than methods BnB and BnC (Default). Note that on average, method BnC (Parity)–2 was also significantly faster than methods BnB and BnC (Default) (often faster than method BnC (Parity)–1 as well), however, for some instances one of the other two methods outperformed it. Finally, both of the methods BnC (Parity)–1 and BnC (Parity)–2 significantly reduced the number of the explored enumeration tree nodes as well.
- Separating parity inequalities at the root node (method BnC (Parity)–2) yielded the best (i.e., highest) lower bounds at the root node, however, on large instances with 150 jobs the separation procedure at the root node took a lot of time, which resulted in longer execution times than method BnC (Parity)–1. For example, in the case of Family 1 and $n = 150$, where the LP-relaxation of the problem (see column LB^r of the pure branch-and-bound method BnB) is basically strong, separating these inequalities at the root node could not help a lot, and method BnC (Parity)–1 outperformed method BnC (Parity)–2.

Table 5.4: Computational results for Family 2 (averages over 10 instances)

n	Method	LB^r	LB^f	UB^f	Gap^f	#Nodes	#Cuts	Time
50	BnB	1800.4	1816.5	1816.5	0.0	925.1	0.0	3.3
	BnC (Default)	1815.9	1816.5	1816.5	0.0	14.7	0.0	0.6
	BnC (Parity)–1	1800.4	1816.5	1816.5	0.0	3.5	2.7	0.3
	BnC (Parity)–2	1816.3	1816.5	1816.5	0.0	1.7	2.6	0.2
100	BnB	3596.2	3598.6	3663.4	1.8	63 707.8	0.0	600.0
	BnC (Default)	3602.3	3616.5	3644.5	0.8	14 762.8	0.0	600.0
	BnC (Parity)–1	3596.2	3642.1	3642.1	0.0	4.3	27.1	5.4
	BnC (Parity)–2	3642.1	3642.1	3642.1	0.0	4.5	16.9	5.2
150	BnB	5340.7	5340.9	5399.8	1.1	9099.5	0.0	600.0
	BnC (Default)	5344.3	5352.8	5360.5	0.1	4944.9	0.0	574.0
	BnC (Parity)–1	5340.7	5360.4	5360.4	0.0	5.0	6.9	15.4
	BnC (Parity)–2	5360.4	5360.4	5360.4	0.0	6.8	10.0	18.1

Experiments on problem $1 \mid chains, chain\text{-}length \in \{1,2\}, p_j = 1 \mid \sum w_{j,\sigma_j}$

In Table 5.5 we summarize our results on Family 3, and for detailed results we refer to (Horváth and Kis, 2019b, Tables 12–14). Similarly to the previous experiments, methods BnC (Parity)–1 and BnC (Parity)–2 outperformed the other ones. For smaller instances with 50 and 100 jobs, method BnC (Parity)–2 slightly outperformed method BnC (Parity)–1 in terms of enumeration tree nodes and total running time, but on large instances with 150 jobs, the method BnC (Parity)–1 proved better.

5.4.4 Conclusions and final remarks

The computational results presented in the previous section show that separating parity inequalities (5.14) and (5.15) can significantly improve an LP-based branch-and-bound procedure if the length of each chain is at most two. Although these inequalities are also valid in the case of chain-precedence constraints with arbitrary chain-lengths, according to our preliminary computational experiments, separating these inequalities could not improve a branch-and-bound procedure in that case.

Table 5.5: Computational results for Family 3 (averages over 10 instances)

n	Method	LB^r	LB^f	UB^f	Gap^f	#Nodes	#Cuts	Time
50	BnB	2039.6	2056.1	2056.1	0.0	28 892.6	0.0	47.0
	BnC (Default)	2049.8	2056.1	2056.1	0.0	851.1	0.0	2.0
	BnC (Parity)–1	2039.6	2056.1	2056.1	0.0	4.2	7.8	0.3
	BnC (Parity)–2	2055.6	2056.1	2056.1	0.0	2.9	5.3	0.3
100	BnB	4053.6	4056.1	4087.7	0.8	60 305.6	0.0	600.0
	BnC (Default)	4056.8	4070.4	4078.7	0.2	26 954.3	0.0	590.3
	BnC (Parity)–1	4053.6	4076.7	4076.7	0.0	56.0	16.0	8.6
	BnC (Parity)–2	4075.7	4076.7	4076.7	0.0	44.4	8.5	6.9
150	BnB	6062.4	6062.8	6109.5	0.8	16 331.7	0.0	600.0
	BnC (Default)	6063.7	6068.3	6084.3	0.3	9923.6	0.0	600.0
	BnC (Parity)–1	6062.4	6081.8	6081.8	0.0	32.9	16.4	23.3
	BnC (Parity)–2	6081.4	6081.8	6081.8	0.0	196.9	10.6	38.3

Chapter 6

Multiple-depot vehicle and crew scheduling problem

The vehicle scheduling and the crew scheduling problems are two main planning problems that arise in the operational phase of the planning process of public transport companies, and have several real-world applications. Briefly stated, the aim of these problems is to find an assignment of minimum cost of a given set of trips to vehicles, and to create a minimal cost set of crew duties that cover tasks resulted from vehicle schedules. In the traditional sequential approach, the vehicle scheduling problem is solved first and then the crew scheduling problem next, but scheduling vehicles independently of the crew is seriously criticized, because in the mass transit case crew costs mostly dominate vehicle operating costs. The integrated vehicle and crew scheduling problem aims to schedule vehicles and the crew simultaneously, rather than sequentially. In this chapter we present a novel mathematical formulation for the integrated problem, and we propose a branch-and-price solution procedure. The motivation for our contribution was the fact that to our best knowledge, the only paper proposing an exact method for the integrated problem in the case of multiple depots is that of Mesquita et al. (2009), however, in that paper a variant of the problem is studied, but in that variant some of the common assumptions other authors make on feasible crew schedules are neglected. Thus, our branch-and-price procedure is the first exact solution approach for the general case with multiple depots.

6.1 Introduction

6.1.1 Problem definition

A *trip* is a project for vehicles to carry passengers between two given stations, and we assume that each trip is timetabled, that is, it has fixed departure and arrival time. A fleet of vehicles may consist of different vehicle types, and some trips may not be operated by all vehicle types. Thus, although a depot basically is a storage facility, where vehicles can be parked when not

in use, we treat a *depot* as a facility with homogeneous fleet of vehicles (that is, if such a facility consists of several vehicle types, we partition its inhomogeneous fleet into homogeneous ones). The Vehicle Scheduling Problem (VSP) can be stated as follows: we are given a set of trips, a fleet of vehicles divided into depots and the goal is to find an assignment of trips to vehicles such that each trip is assigned exactly once; each vehicle performs a feasible sequence of trips; each sequence starts and ends at the same depot; and assets and operational costs are minimized. Based on the number of depots, we have the Single-Depot Vehicle Scheduling Problem (SDVSP), or the Multiple-Depot Vehicle Scheduling Problem (MDVSP).

A *vehicle itinerary* describes the route of a vehicle, i.e., the movements made by the vehicle, e.g., performing a trip, waiting at a station or in a depot, pulling out from/pulling in a depot, performing a deadhead (that is, traveling between stations without passengers). Each vehicle itinerary starts with a *pull-out* and ends with a *pull-in*, but vehicles can return to the depot at any time. A *vehicle block* is the part of the vehicle itinerary between a pair of consecutive pull-out and pull-in (both included). In Fig. 6.1 we depict a vehicle itinerary consisting of two vehicle blocks. Some vehicle movements require driver attendance (e.g., performing a trip/deadhead or pulling out from/pulling in a depot), while typically no driver is required to be present if the vehicle is waiting in a depot. Drivers can board/leave the vehicle only at *relief points*, these are the depots and certain designated stations. Moreover, each trip has at most two relief points: one at the beginning and one at the end of the trip, i.e., drivers cannot board/leave the vehicle while it is performing a trip. According to these restrictions, each vehicle itinerary defines tasks that have to be assigned to drivers. More precisely, a *task* is a sequence of driver requiring vehicle movements between two consecutive relief points, i.e., tasks are the most elementary portion of work that can be assigned to a driver. For example, in Fig. 6.1 we present a situation, where a driver is required to be present if a vehicle is outside of the depot, and the only relief point other than the depot is station C. Thereby, vehicle block I and vehicle block II consist of 3 and 1 tasks, respectively. A *piece of work* is a sequence of tasks without any break (i.e., each task in a piece of work begins at the time point when the previous one ends), and a *(driver) duty* is either a single piece of work or a sequence of pieces of work separated by breaks. The first three tasks in Fig. 6.1 could define six pieces of work (these are (task I), (task II), (task III), (task I, task II), (task II, task III) and (task I, task II, task III)), while task IV can be contained by only one piece of work. In this figure we depict only three pieces of work. Again, these three pieces of work could define four driver duties (these are (piece of work I), (piece II), (piece III), and (piece II, piece III)), however we depict only one.

The Crew Scheduling Problem (CSP) can be stated as follows: find a set of duties for a given set of tasks such that each task is covered by a duty that can be performed by a single driver; each duty satisfies a wide variety of federal laws, safety regulations, and (collective) in-house agreements; and labor costs are minimized.

Finally, the Integrated Vehicle and Crew Scheduling Problem (VCSP) can be stated

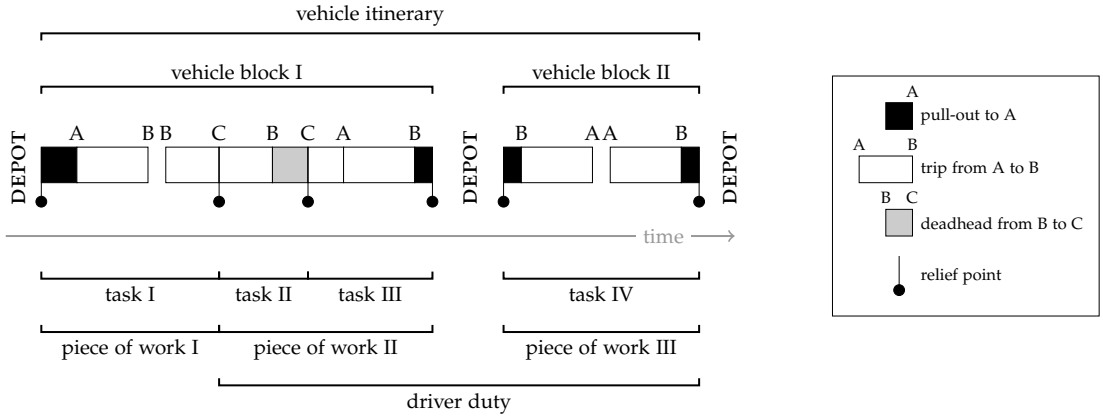


Figure 6.1: Vehicle itinerary with driver activities (based on (Steinzen, 2007, Figure 1.4))

as follows: for a set of trips find a minimum cost set of vehicle itineraries and driver duties such that both the vehicle and the crew schedules are feasible and compatible with each other (that is, the driver schedule is feasible according to tasks determined by the vehicle schedule). Again, based on the number of depots we have the Single-Depot Vehicle and Crew Scheduling Problem (SDVCSP), or the Multiple-Depot Vehicle and Crew Scheduling Problem (MDVCSP).

Assumptions

In the followings we introduce our assumptions about MDVCSP.

Rule 1. *Each vehicle is assigned to a depot where its daily schedule starts end ends. Each depot is unlimited in capacity, that is, it can store an unlimited number of vehicles.*

Rule 2. *A vehicle returns to its depot if the idle time between two consecutive trips is long enough to perform a round trip to the depot.*

Rule 3. *Each driver is assigned to a depot and may only conduct tasks on vehicles from this particular depot. However, a duty does not necessarily start and end in this depot.*

Rule 4. *A piece of work is only restricted by its duration. It may have a minimum and maximum duration.*

Rule 5 (continuous attendance)**.** *A driver is required to be present if a vehicle is outside of a depot, while no driver is needed when the vehicle is parked in the depot.*

Rule 6 (restricted changeover)**.** *Drivers may only change their vehicle during a break, i.e., between two pieces of work.*

Rules 1 to 6 are customary assumptions in the literature (Huisman, 2004; Huisman et al., 2005; Steinzen, 2007; Steinzen et al., 2010).

Rule 2 was originally proposed for vehicle scheduling problems to reduce the number of constraints by introducing the concept of short arcs, and long arcs, see e.g., Freling et al. (1995). Basically, in their network model arcs representing vehicle movements with appropriate long idle time were replaced with so-called long arcs representing round trips to the depot, and for such arcs they did not require the continuous attendance. This idea was applied for integrated problems as well (e.g., Freling et al. (2003); Huisman et al. (2005); Steinzen et al. (2010)), however, it is worth mentioning that omitting such long waiting and deadheads may change the set of potential tasks (see Rule 5), hence the set of feasible duties can be changed. Rule 2 can create another problem when time-space network approaches are used for VCSP. Steinzen (2007) and Steinzen et al. (2010) suggest to eliminate appropriate (long) arcs from network to ensure Rule 2, but it is not sufficient by itself as we will show in Section 6.1.4. That is why we will handle Rule 2 as a lazy rule, i.e., we will eliminate long arcs from the network model of the problem, but we will not make further efforts to satisfy Rule 2.

To ensure Rule 6 we need to redefine the concept of a piece of work, that is, in the rest of this chapter a piece of work is a sequence of tasks without any break that is performed by the same vehicle. Remark that pieces of work in Fig. 6.1 correspond to the new concept.

Rule 7. *A duty consists of one or two pieces of work. Each duty starts with a sign-on and ends with a sign-off by the driver. Feasibility of a duty can depend only on earliest/latest (sign-on) start/(sign-off) end time; minimum/maximum piece length; minimum/maximum break length; minimum/maximum working time; minimum/maximum spread time.*

In our terminology *working time* is the time that driver spends on the vehicle (i.e., the total duration of the pieces of work consisted by the duty), and *spread time* is the total duration of the sign-on, the sign-off, the pieces of work and the breaks.

Rule 8. *Vehicle cost is a combination of a fixed asset cost for using the vehicle and a variable operation cost. Asset cost depends only on depot. Operation cost is a linear function of travel and idle time outside the depot.*

Rule 9. *Duty cost is a combination of a fixed driver cost for using a driver and a variable working cost. Driver cost depends only on depot. Working cost is a linear function of working time.*

In fact, fixed costs in Rules 8 and 9 are not restrictions as we assumed that each depot consists of a homogeneous fleet of vehicles and crew is a group of anonymous drivers.

6.1.2 Literature review

Sequential vehicle and crew scheduling

MDVSP is shown to be NP-hard by Bertossi et al. (1987), which is in strong contrast with the polynomial solvability of SDVSP, see e.g., (Freling et al., 2001). An overview of different vehicle scheduling models can be found in (Bunte and Kliewer, 2009), and for heuristic solution approaches for MDVSP we refer to (Pepin et al., 2006).

Both of VSP and CSP can be interpreted as an assignment problem, however, CSP is more complicated than VSP because of the wide variety of working rules (e.g., minimum/maximum working time for drivers, minimum/maximums spread time for duties, etc.). Fischetti et al. (1987, 1989) show that CSP is NP-hard if either spread time or working time constraints are present.

Partial integration

Until the late nineties the complete integration of vehicle scheduling and crew scheduling was computationally intractable, thus most of the early approaches are based on a heuristic integration.

Ball et al. (1983) propose the first partially integrated approach for the single-depot case. They schedule crew first including vehicle scheduling considerations and construct a feasible vehicle schedule afterward. Similar heuristics for the single-depot case are proposed by Tosini and Vercellis (1988), Falkner and Ryan (1992), and Patrikalakis and Xerocostas (1992).

Other approaches schedule vehicles first but include crew scheduling considerations and subsequently generate feasible crew schedules, see e.g., (Scott, 1985) and (Darby-Dowman et al., 1988).

Gintner et al. (2008) apply another partial integration approach for the multiple-depot case. They perform vehicle scheduling first and crew scheduling afterward, but they use a time-space network approach for vehicle scheduling that allow to change the corresponding optimal vehicle schedule without loss of optimality in the crew scheduling phase.

Complete integration – Single-depot case

In Table 6.1 we collect the core of modeling and solution approaches of completely integrated models, details are explained below.

Freling et al. (1995) propose the first fully integrated approach for the single-depot case. Their integer programming model uses a so-called connection-based network and consists of three components: a quasi-assignment formulation for vehicle scheduling, a set partitioning formulation for crew scheduling, and additional linking constraints that ensure the compatibility of vehicle and crew schedules. Their solution approach uses column generation in combination with Lagrangian relaxation. That is, linking constraints are relaxed in a Lagrangian way and the crew scheduling part is relaxed to a set covering formulation that yields two independent Lagrangian subproblems: a single-depot vehicle scheduling problem and a selection problem. They solve the Lagrangian dual problem with a subgradient algorithm, and suggest a two-phase pricing method to generate new columns (i.e., duties) for the crew scheduling part. They apply several heuristics to obtain feasible integer solutions for the original problem. This modeling and solution approach provides the basis for many other publications, e.g., Freling et al. (2003); Huisman (2004); Huisman et al. (2005); Steinzen (2007); Steinzen et al. (2010).

Friberg and Haase (1999) propose the first exact algorithm for the single-depot case. Their mathematical programming formulation is a combination of set partitioning formulations for the vehicle scheduling problem and for the crew scheduling problem, respectively. They develop a branch-and-cut-and-price algorithm, i.e., the LP-relaxation in each node of the enumeration tree is solved by column generation, moreover, polyhedral cuts are added to strengthen the relaxation. Columns for the vehicle scheduling subproblem are generated by solving shortest path problems on acyclic graphs, however, the pricing problem for the crew scheduling subproblem is modeled as a resource constrained shortest path problem which is solved by a dynamic programming algorithm.

Haase et al. (2001) propose another exact solution approach for the single-depot case. In their view each driver duty must start and end in the depot. Their crew-based mathematical model is a multicommodity flow formulation that relies on a so-called driver network structure. Side constraints are used to guarantee that an optimal compatible vehicle schedule could be derived. That formulation uses a set of path flow variables for drivers and only one additional variable to count vehicles. They propose a branch-and-price algorithm, where cutting planes are added to the master problem to reinforce linear relaxations throughout the enumeration tree. Each pricing problem is transformed into a shortest path problem with resource constraints and solved by a dynamic programming algorithm.

Laurent and Hao (2008) consider a situation where all vehicles are parked in the same depot, however, the vehicles may belong to different categories. Thus, their case is more general than a single-depot case, but more special than the general multiple-depot case which we consider in this chapter. They also use simplified crew constraints in contrast to Rule 7, e.g., they have restrictions only for the spread and working times. Their formulation relies on a constraint satisfaction and optimization model, and they apply a heuristic greedy randomized adaptive search procedure to solve the problem.

Complete integration – Multiple-depot case

In Table 6.1 we collect the core of modeling and solution approaches of completely integrated models, details are explained below.

Gaffi and Nonato (1999) introduce the integrated problem for the multiple-depot case. However, their approach is developed for the extra-urban mass transit setting, where drivers are virtually tied to their vehicles. Hence, for example, they assume that a driver is assigned to the same vehicle during the whole duty, and all pieces of work start and end in the depot. Their heuristic procedure is based on column generation in combination with Lagrangian relaxation.

Huisman (2004) and Huisman et al. (2005) propose the first general approaches for the multiple-depot case. Huisman (2004) explicitly introduces Rules 1 and 3 to 6, and Rule 2 is applied in his mathematical formulation to reduce the number of constraints. That formulation complies also with Rules 7 to 9. Huisman (2004) and Huisman et al. (2005) extend

Table 6.1: Modeling and solution approaches for the complete integration

Reference	Modeling approach			Solution approach	
	Network[a]	Vehicle scheduling part	Crew scheduling part	Type[b]	Core
Single-depot case					
Freling et al. (1995)	CB	quasi-assignment	set partitioning	H	LR-CG[c]
Friberg and Haase (1999)	CB	set partitioning	set partitioning	E	BCP[d]
Haase et al. (2001)	DB	side constraints	multicommodity flow	E	BCP
Freling et al. (2003)	CB	quasi-assignment	set partitioning	H	LR-CG
Laurent and Hao (2008)		(constraint programming approach)		H	GRASP[e]
Multiple-depot case					
Gaffi and Nonato (1999)	CB	quasi-assignment	set partitioning	H	LR-CG
Huisman et al. (2005)	CB	multicommodity flow	set partitioning	H	LR-CG
Borndörfer et al. (2008)	CB	multicommodity flow	set partitioning	H	LR-CG
Mesquita and Paias (2008)	CB	multicommodity flow	set partitioning/covering	H	PB[f]
Mesquita et al. (2009)	CB	multicommodity flow	set partitioning/covering	H/E	BP[g]
Steinzen et al. (2010)	TS	multicommodity flow	set partitioning	H	LR-CG

[a] CB: connection-based; DB: driver-based; TS: time-space
[b] H: heuristic approach; E: exact method
[c] Lagrangian relaxation based column generation
[d] branch-and-cut-and-price
[e] greedy randomized adaptive search procedure
[f] (LP-relaxation based) price-and-branch
[g] branch-and-price

the modeling and solution approaches of Freling et al. (2003) and Haase et al. (2001) for the multiple-depot case. That is, they use a multicommodity flow formulation for the vehicle scheduling part which is based on connection-based networks, and additional constraints are used to link duty and flow variables. In the first phase of their solution approach they calculate a lower bound on the optimum using a column generation algorithm where the master problem is solved with Lagrangian relaxation by a subgradient algorithm. For generating duties they apply a two-step procedure similar to that of Freling et al. (1995), that is, they generate pieces of work with shortest path algorithms, while duties are generated by a simple enumerating procedure. Feasible solutions are obtained in the second phase. Huisman (2004) and Huisman et al. (2005) propose an alternative formulation obtained from the previous one containing only variables related to crew duties. However, additional constraints are added to count the number of vehicles and to consider fixed vehicle costs. They apply a solution approach similar to the one for the previous formulation.

In (Huisman, 2004; Huisman et al., 2005) the authors propose their randomly generated instances which are widely used in the literature (Borndörfer et al., 2008; Mesquita and Paias, 2008; Mesquita et al., 2009; Steinzen, 2007; Steinzen et al., 2010) and in this chapter as well.

Borndörfer et al. (2008) use a modeling approach similar to that of (Freling et al., 1995). Their solution approach also relies on a Lagrangian relaxation based column generation procedure, but they use inexact proximal bundle method to solve Lagrangian dual problems. The bundle method is embedded in a backtracking procedure to produce an integer solution in

the second phase.

Mesquita and Paias (2008) propose a modeling approach similar to that of Huisman (2004). However, there are some fundamental differences between the problem definition of Mesquita and Paias (2008) and of Huisman (2004). For example, in (Mesquita and Paias, 2008) the authors consider each end location of a trip as a potential relief point. Moreover, they allow drivers to change vehicles whenever there is a relief point, and to use vehicles from any depot, that is, their model do not comply with Rules 3 and 6. They use a multicommodity flow formulation for the vehicle scheduling part, and set partitioning/covering formulations for the crew scheduling part. They apply a price-and-branch algorithm, that is, they solve the LP-relaxation of the problem with a column generation approach, and if the resulted optimal solution is fractional they apply a branch-and-bound procedure to obtain feasible integer solution to the problem. The pricing problems are modeled as resource constrained shortest path problems and are solved by a dynamic programming algorithm.

Mesquita et al. (2009) propose exact and non-exact branch-and-price procedures for the same problem definition and formulation as in (Mesquita and Paias, 2008).

Steinzen (2007) and Steinzen et al. (2010) use a similar modeling approach for the multiple-depot case as in (Huisman, 2004), however, their mathematical formulation is based on time-space networks. Their Lagrangian relaxation based column generation approach is also similar to that of Huisman (2004), but in their case pricing problems are modeled by resource constrained shortest path problems on time-space networks which are solved by a dynamic programming algorithm. Finally, they devise a heuristic branch-and-price procedure which alternates between vehicle and crew scheduling to obtain feasible solutions.

6.1.3 Our contribution

We present a novel problem formulation for MDVCSP, where we combine the advantages of the existing modeling approaches. While most of the known MILP formulations model the vehicle and crew schedules separately, and join the two parts by linking constraints, we model crew schedules along with some extra variables and constraints that ensure that from any integer feasible solution a valid vehicle schedule can be deduced as well. Our modeling approach is quite general, the set of columns represents the valid crew schedules, and a subset of it is generated in the course of the solution procedure guided by the rules to be observed by valid driver schedules.

We developed an exact branch-and-price procedure including (i) an effective pricing procedure based on that of Freling et al. (1995) using several acceleration strategies, (ii) some branching strategies, and (iii) a simple primal heuristics. To our best knowledge, the only paper proposing an exact method for the multiple-depot integrated vehicle and crew scheduling problem is that of Mesquita et al. (2009), where a variant of the problem is studied in which some of the common assumptions we and other authors make on feasible crew schedules are neglected. Their MILP formulation, unlike ours, models vehicle and crew schedules

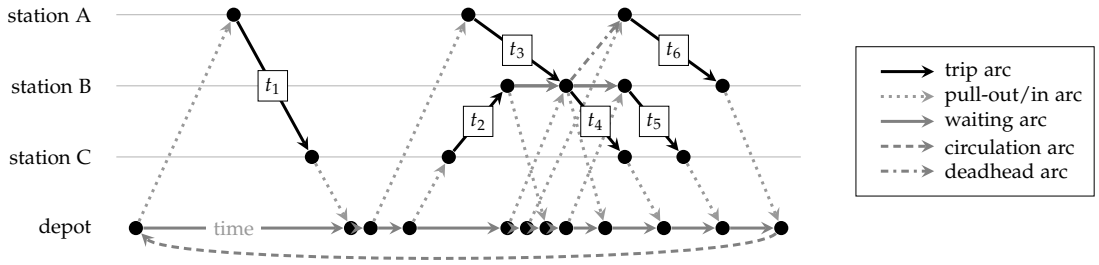


Figure 6.2: Example for a time-space network

separately and contains additional linking constraints to join the two parts.

We also present our computational results compared with other well-known solution approaches. As we discussed above, several problem definitions have been proposed for the (integrated) vehicle and crew scheduling problem. Because of the differences between these assumptions, fair comparisons cannot be established between all approaches. That is, a feasible solution for a given approach may not be feasible for another one, and vice-versa. As we mentioned in Section 6.1.1, our assumptions comply with those of Huisman et al. (2005); Steinzen et al. (2010), however, they differ from the assumptions of Mesquita and Paias (2008); Mesquita et al. (2009).

6.1.4 Problem formulation

Now we discuss our mathematical formulation for MDVCSP, and we shortly present the well-known time-space network structure the formulation bases on.

Time-space network structure

In a time-space network each node represents a (time, space) pair (where space is either a station or the depot), and arcs represent vehicle movements. In the following we present how we build a time-space network for a given depot. For a detailed description about building time-space networks we refer to (Kliewer et al., 2006).

For each trip that can be operated from the depot we add four nodes to the network representing the (departing time, departing station), (arriving time, arriving station), (pull-out time, depot) and (pull-in time, depot) pairs, respectively. Additionally, we add a *trip arc* to the network from the *departing node* to the *arriving node*, and a *pull-out arc* (*pull-in arc*) from the *pull-out node* (arriving node) to the departing node (*pull-in node*). Of course, if a node or a pull-in/out arc already exists we do not duplicate them (e.g., arriving node of trip t_3 and departing node of trip t_4 are the same in Fig. 6.2).

To represent waiting at a station or in the depot we create for each space its *timeline*, that is, we collect all nodes that represent this space and sort them in increasing order according to their represented time, then we add a *waiting arc* between consecutive nodes. Let s and t be the first and last node of the timeline of the depot, respectively. We add an extra *circulation*

arc from t to s. Note that at a station it is sufficient to start that connecting process with the first node that represents arriving event, since there is no reason for a vehicle to wait at a station until a trip ends there. Moreover, according to Rule 2 we do not connect consecutive nodes together if the duration of that waiting arc would not be shorter than the duration of a round trip. As you can see in Fig. 6.2 we do not connect the arriving node of trip t_1 with the departing node of trip t_2 at station C, since there is enough time for a vehicle to perform a round trip. It is worth mentioning that both of the two waiting arcs are necessary at station B — as they ensure the connections between trips t_2 and t_4, and trips t_3 and t_5, respectively — however, a vehicle operating trips t_2 and t_5 can use these arcs to wait in station B instead of performing a round trip as required by Rule 2. That is why we mentioned that omitting long arcs is not sufficient to satisfy Rule 2, and that is why we do not strive to satisfy Rule 2 in the rest of our solution approach.

To represent deadhead movements between stations we add *deadhead arcs* connecting the arriving node of a trip with the departing node of an another trip. One of the most important properties of time-space networks is that we should not represent all of the deadhead movements explicitly. For example, in Fig. 6.2 trips t_3 and t_6 are compatible (i.e., can be performed by the same vehicle), thus we connect their corresponding arriving/departing nodes with a deadhead arc. However, trips t_2 and t_6 are also compatible, but is not necessary to add any deadhead arc between them, since these can be operated by the same vehicle by using the first waiting arc and the deadhead arc. Of course, we omit a deadhead arc if it is longer than the corresponding round trip.

Note that each path from s to t corresponds to a vehicle itinerary (and vice versa), and a piece of work can be represented as a path between two relief points using nondepot-arcs only.

Problem formulation of Steinzen et al. (2010)

Let $\mathcal{D} = \{d_1, d_2, \ldots, d_{|\mathcal{D}|}\}$ be the set of depots, and $\mathcal{T}$ be the set of trips. Let $D^d = (V^d, A^d)$ be the time-space network for depot $d \in \mathcal{D}$, and let $\tilde{A}^d \subset A^d$ be the set of nondepot-arcs (i.e., all arcs but the arcs of the timeline of the depot and the circulation arc). It is worth mentioning that $\tilde{A}^d$ is the set of arcs that require both of vehicle and driver activities. Remember that a path between two nodes that correspond to relief points and using nondepot-arcs only represents a piece of work. Let K^d be the set of feasible duties that can be operated from depot $d \in \mathcal{D}$ and $K^d(i,j) \subseteq K^d$ the set of duties covering arc $(i,j) \in \tilde{A}^d$. For depot $d \in \mathcal{D}$ we denote by $A^d(t) \subseteq A^d$ the set of arcs corresponding to trip $t \in \mathcal{T}$. Note that $A^d(t)$ is empty if trip t cannot be operated from depot d, otherwise it contains a single arc.

Steinzen et al. (2010) use two types of variables. First, they associate a flow variable y_{ij}^d with each arc $(i,j) \in A^d$ indicating whether that arc is used and assigned to depot $d \in \mathcal{D}$. The binary duty variables x_k^d ($k \in K^d$) indicate whether duty k is selected for depot $d \in \mathcal{D}$.

On the one hand, Steinzen et al. (2010) assign a vehicle cost c_{ij}^d to each arc $(i,j) \in A^d$. That

is, c^d_{ij} is the asset cost for using a vehicle if (i,j) is the circulation arc of D^d; c^d_{ij} is the operation cost of the represented vehicle movement if $(i,j) \in \tilde{A}^d$; otherwise c^d_{ij} is equal to zero. On the other hand, one could associate a working cost g^d_{ij} with each arc $(i,j) \in \tilde{A}^d$. With this, the duty cost f^d_k of duty $k \in K^d$ is the sum of the fixed driver cost, and the working cost of its pieces of work. The formulation of Steinzen et al. (2010) is the following:

$$\text{minimize} \sum_{d\in\mathcal{D}} \sum_{(i,j)\in A^d} c^d_{ij} y^d_{ij} + \sum_{d\in\mathcal{D}} \sum_{k\in K^d} f^d_k x^d_k \tag{6.1}$$

$$\sum_{d\in\mathcal{D}} \sum_{(i,j)\in A^d(t)} y^d_{ij} = 1 \qquad \text{for all } t \in \mathcal{T} \tag{6.2}$$

$$\sum_{j:(j,i)\in A^d} y^d_{ji} - \sum_{j:(i,j)\in A^d} y^d_{ij} = 0 \qquad \text{for all } d \in \mathcal{D}, \text{ and } i \in V^d \tag{6.3}$$

$$\sum_{k\in K^d(i,j)} x^d_k - y^d_{ij} = 0 \qquad \text{for all } d \in \mathcal{D}, \text{ and } (i,j) \in \tilde{A}^d \tag{6.4}$$

$$0 \le y^d_{ij} \le u^d_{ij},\ y^d_{ij} \in \mathbb{N} \qquad \text{for all } d \in \mathcal{D}, \text{ and } (i,j) \in A^d \tag{6.5}$$

$$x^k_d \in \{0,1\} \qquad \text{for all } d \in \mathcal{D}, \text{ and } k \in K^d. \tag{6.6}$$

The objective (6.1) minimizes the sum of vehicle and crew costs. Constraint set (6.2) ensures that the set of trips are partitioned among the depots and each trip is covered by a single vehicle. Constraints (6.3) are the flow conservation constraints corresponding to the multi-commodity flow formulation for the vehicle scheduling problem. Constraint set (6.4) links the vehicle and crew schedules, that is, each nondepot-arc should be covered by the same number of vehicles and duties. Constraints (6.5) ensure that the maximum capacity of flow variables is satisfied. Steinzen et al. (2010) set u^d_{ij} to 1 on trip arcs $(i,j) \in \tilde{A}^d$, however, these constraints are redundant according to (6.2). They also set u^d_{ij} to 1 on pull-in/out arcs $(i,j) \in \tilde{A}^d$, which are technical constraints (note that they use unique pull-in/out arcs for each trip). For all other arcs they use maximum capacity u^d equal to the number of vehicles available in depot $d \in \mathcal{D}$.

Our problem formulation

Our mathematical programming formulation is obtained from that of Steinzen et al. (2010) described above by dropping the redundant and technical capacity constraints from (6.5), and eliminating most of the flow variables by substituting them using constraints (6.4).

However, our formulation can also be interpreted directly from the problem definition. We use the same notations as before. Further on, let $\bar{A}^d = A^d \setminus \tilde{A}^d$ be the set of depot-arcs (i.e., the arcs of the timeline of the depot and the circulation arc), and $\bar{V}^d \subset V^d$ be the set of depot-nodes of D^d (i.e., nodes of the timeline of the depot). For depot $d \in \mathcal{D}$ we denote by $K^d(t) \subseteq K^d$ the set of duties covering trip $t \in \mathcal{T}$, furthermore, we denote by $K^d_-(i) \subseteq K^d$ ($K^d_+(i) \subseteq K^d$) the set of duties that contain a piece of work starting (ending) in node $i \in V^d$.

We also use two types of variables. First, we associate a *flow variable* y^d_{ij} with each depot-arc $(i,j) \in \bar{A}^d$ indicating the number of vehicles that cross arc (i,j). To ensure continuous

attendance (Rule 5), and restricted changeover (Rule 6), the second type of our variables combines drivers and vehicles outside of a depot. Remember that a path between two nodes that correspond to relief points and using nondepot-arcs only represents a piece of work. From a different angle, such a path can be considered as a part of some vehicle block, that is why we can handle a piece of work as a driver-vehicle pair. That is, binary *duty variable* x_k^d indicates whether duty $k \in K^d$ is selected for depot $d \in \mathcal{D}$, if so, it means that a driver is assigned to duty k and for each piece of work of the duty a vehicle is assigned.

We also assign vehicle costs c_{ij}^d to each arc $(i,j) \in A^d$, and a working cost g_{ij}^d to each arc $(i,j) \in \tilde{A}^d$. With this, the driver cost (vehicle cost) of a piece of work is the cost of the corresponding path according to arc costs g_{ij}^d (c_{ij}^d), and the combined duty cost $\tilde{f}_k^d$ of duty $k \in K^d$ is the sum of the fixed driver cost, the vehicle cost of its pieces of work, and the working cost of its pieces of work. Now, we formulate MDVCSP as:

$$\text{minimize} \sum_{d\in\mathcal{D}} \sum_{(i,j)\in\bar{A}^d} c_{ij}^d y_{ij}^d + \sum_{d\in\mathcal{D}} \sum_{k\in K^d} \tilde{f}_k^d x_k^d \tag{6.7}$$

$$\sum_{d\in\mathcal{D}} \sum_{k\in K^d(t)} x_k^d = 1 \qquad \text{for all } t \in \mathcal{T} \tag{6.8}$$

$$\sum_{k\in K_+^d(i)} x_k^d - \sum_{k\in K_-^d(i)} x_k^d = 0 \qquad \text{for all } d \in \mathcal{D}, \text{ and } i \in V^d \setminus \bar{V}^d \tag{6.9}$$

$$\sum_{(i,j)\in\bar{A}^d} y_{ij}^d + \sum_{k\in K_+^d(i)} x_k^d - \sum_{(j,i)\in\bar{A}^d} y_{ji}^d - \sum_{k\in K_-^d(i)} x_k^d = 0 \qquad \text{for all } d \in \mathcal{D}, \text{ and } i \in \bar{V}^d \tag{6.10}$$

$$0 \le y_{ij}^d,\ y_{ij}^d \in \mathbb{Z} \qquad \text{for all } d \in \mathcal{D}, \text{ and } (i,j) \in \bar{A}^d \tag{6.11}$$

$$x_d^k \in \{0,1\} \qquad \text{for all } d \in \mathcal{D}, \text{ and } k \in K^d. \tag{6.12}$$

The objective (6.7) minimizes the sum of vehicle and crew costs, as the fixed asset costs for the vehicles are built in the first term of (6.7), and all the other costs are contained in the second term of (6.7). Constraint (6.8) ensures that each trip is covered by exactly one duty. Constraints (6.9)–(6.10) connect flow variables with the vehicle part of duty variables. That is, (6.9) specifies for a nondepot-node i that the number of pieces of work ending in node i (i.e., the number of vehicles arriving at node i) must be equal to the number of pieces of work starting in node i (i.e., the number of vehicles departing from node i). Constraint (6.10) is analogous for depot-nodes, but it takes into consideration that vehicles can wait in the depots. Note that flow variables are implicit integer, that is, they are always integer if duty variables are integer.

It is worth mentioning that in our formulation a duty variable (i.e., the corresponding column) contains only relevant information about the duty, namely, the start/end nodes of the piece(s) of work of the duty and the trips covered by the duty, if any. Notice that deadhead routes (e.g., routes between two consecutive trips) are not considered by the constraints. Moreover, the rules concerning the feasibility of duties do not appear explicitly in this formulation, only in the set K^d.

Note that limits on the number of vehicles in depots can be imposed by adding the constraints $y^d_{ts} \leq u^d$ to the model, where (t, s) is the circulation arc of the corresponding depot.

By construction, we have the following result.

Proposition 6.1. *Each optimal solution of the formulation* (6.7)–(6.12) *corresponds to an optimal solution for* MDVCSP*, and each optimal solution for* MDVCSP *is represented as an optimal solution for the formulation* (6.7)–(6.12)*.*

6.2 Solution approach

In this section we present our solution method for MDVCSP which is a branch-and-price procedure to solve *master problem* (6.7)–(6.12). That is, we compute a MILP containing just a few columns of the master problem (called *restricted master problem*) and perform a branch-and-bound procedure such that in each node of the enumeration tree we may add new columns (i.e., duties) to the LP-relaxation of the current restricted master problem.

More precisely, we create an initial restricted master problem (described in Section 6.2.1). We solve each node-LP optimally, that is, for each node we generate new duties until no one with a negative reduced cost is left as we describe in Section 6.2.2. At the root node we apply a two-stage approach. In the first stage we generate duties that contain one or two pieces of work starting and ending in the depot, and at the end of this stage we perform a primal solution search (described in Section 6.2.4). The reason for this is that with such a column set the constraints (6.9)–(6.10) are easy to satisfy, hence we expect that the search procedure can quickly find a good primal solution. In the second stage we generate duties without any limitations for their start and end locations, and we may also perform a primal solution search at the end of the stage. We describe our branching rules in Section 6.2.3. Our primary branching strategy is to assign trips to depots, and we use the SPP-based branching strategy as a secondary rule (if the primary rule failed to branch), and as a last resort, one may rely on the default branching strategy of the MILP solver.

6.2.1 Initial restricted master problem

The initial restricted master problem contains all of the flow variables and a set of initial duty variables that we create by obtaining a feasible solution for MDVCSP by using a sequential procedure. That is, we first formulate MDVSP as a minimum cost multicommodity flow problem on the time-space networks using the given vehicle costs as in (Kliewer et al., 2006), and solve the MILP model with a standard software. Then, independently for each depot we create a set-partitioning formulation for CSP (e.g., (Freling et al., 2003)) to assign drivers to the obtained vehicle schedules. We solve the LP-relaxations of these problems with a column generation approach similar to the one we discuss in Section 6.2.2, then we solve the resulting restricted master problems with branch-and-bound, and use the solutions as initial column set for MDVCSP.

Note, that if we failed to obtain feasible integer solution for any of CSPs, we could use fictive columns for the initial restricted master problem penalized by a high cost, or we could start branch-and-price with an initial restricted master problem containing no duty variables (see Farkas pricing in Section 6.2.2).

6.2.2 Pricing variables

Once the corresponding restricted master problem is solved we attempt to price out new variables (i.e., new duties) by using the dual information of the solution. Let λ_t ($t \in \mathcal{T}$) and μ_i^d ($i \in V^d$, $d \in \mathcal{D}$) be the dual variables associated to constraints (6.8) and (6.9)–(6.10), respectively.

To generate feasible duties we use a two-phase procedure similar to the one proposed by Freling et al. (1995), that is, in the first phase we generate a set of feasible pieces by using a so-called piece generation network, and in the second phase we derive feasible duties. Since we generate pieces of work and duties independently for each depot, in the rest of this section we fix a depot $d \in \mathcal{D}$.

Generation of pieces of work

For each depot we derive a piece generation network from the corresponding time-space network consisting of all original arcs but depot-arcs, that is, the piece generation network for depot d is $\hat{D}^d = (V^d, A^d \setminus \bar{A}^d)$. We recall that each path in $\hat{D}^d$ between two nodes that correspond to relief points represents a piece of work. For a piece of work p let $A[p]$ and $\mathcal{T}[p]$ be the set of arcs and the set of trips covered by p, respectively, and let $s[p]$ and $e[p]$ be the start and the end node of p, respectively. The combined cost h of a piece of work p is the sum of vehicle and driver costs for all arcs covered by the piece of work, formally

$$h(p) := \sum_{ij \in A[p]} c_{ij}^d + g_{ij}^d.$$

The reduced cost $\hat{h}$ of a piece of work p (and the reduced cost of the corresponding path) is

$$\hat{h}(p) := h(p) - \mu_{s[p]}^d + \mu_{e[p]}^d - \sum_{t \in \mathcal{T}[p]} \lambda_t^d.$$

For the sake of efficiency, we do not generate all of the pieces of work, but obtain a set of feasible pieces by considering only the minimum reduced cost path between any two nodes in $\hat{D}^d$. To do this, we predetermine a processing order of nodes of $\hat{D}^d$ (which is a topological order in case the network is acyclic). By that, for any given node we can determine the shortest path arborescence in $O(|A|)$ time, thus we can determine the minimum reduced cost path for each pair of nodes in $O(|A||V|)$ total time.

At the root node of the enumeration tree it is clear that considering only the minimum reduced cost paths is sufficient in the sense that we will find at least one piece of work with

negative reduced cost, if any. However, when branching decisions are to be considered this strategy may fail to find appropriate pieces. For example, assume that piece of work p corresponding to the minimum reduced cost path between nodes u and v is infeasible according to some of the branching decisions, but there is an another u–v path with negative reduced cost that admits a piece of work p' which is feasible according to all of the branching decisions. It is clear that we will fail to find the feasible piece of work p', since it is overshadowed by the infeasible piece of work p. That is why we should take branching decisions into consideration during piece or/and duty generation. We postpone the details until Section 6.2.3.

Generation of duties

Duties consisting of one piece of work can be easily generated by iterating over the previously obtained piece of work set. To generate combined duties (i.e., duties consisting of two pieces of work) we apply a straightforward pairing procedure using proper data structures and several acceleration techniques in order to avoid enumerating inherently infeasible pairs. For details we refer to (Horváth and Kis, 2019a).

Farkas pricing

After branching is performed the restricted master problem of a new node may be infeasible due to fixings, but it does not mean that the master problem of the node is infeasible, so the node cannot be pruned.

Again, one could resolve this issue by adding fictive columns to the LP penalized by a high cost, but instead, in such cases we perform a so-called Farkas pricing. That is, if the restricted master problem is infeasible we can obtain dual Farkas multipliers $\bar{\lambda}_t$ and $\bar{\mu}_v^d$ associated with constraints (6.8) and (6.9)–(6.10), respectively, to prove infeasibility according to the Farkas-Lemma. To make restricted master problem feasible we have to find a new column that violates this proof. It can be shown that this pricing problem is similar to the pricing problem for reduced cost pricing, but now we use a zero objective function and the dual Farkas multipliers instead of the original objective function and the dual solution. Thus, we can use the pricing method discussed in Section 6.2.2 with a minor modification to make the restricted master problem feasible.

6.2.3 Branching strategies

Now, we present our strategies to perform branch in an enumeration tree node where the optimal solution for the final restricted master problem is fractional. Remark, that flow variables y_{ij}^d are implicit integer, hence it is sufficient to take only duty variables x_k^d into consideration.

Assign trips to depots

Our first branching strategies can be used in the multiple-depot case when there exists a trip that belongs to several depots in the current LP-relaxation. Formally, consider a fractional solution $(\bar{x}, \bar{y})$ to the relaxation of the corresponding restricted master problem, and let $C_{\bar{x}}(t,d)$ denote the commitment of trip t to depot d, that is

$$C_{\bar{x}}(t,d) := \sum_{k \in K^d(t)} \bar{x}_k^d.$$

If $0 < C_{\bar{x}}(t,d) < 1$ holds for a trip t and a depot d, i.e., trip t is committed to multiple depots according to solution $\bar{x}$, we choose a trip $\bar{t}$ and a depot $\bar{d}$ such that $(\bar{t},\bar{d}) = \arg\min_{(t,d)} |C_{\bar{x}}(t,d) - 0.5|$. We have two possibilities to perform branch on pair $(\bar{t},\bar{d})$:

1. *Partitioning:* We create exactly two branches. We require to cover trip $\bar{t}$ by a duty from depot $\bar{d}$ on the one branch, and to cover by a duty from a depot that differs from $\bar{d}$ on the other branch. Formally,

$$\sum_{k \in K^{\bar{d}}(\bar{t})} x_k^{\bar{d}} = 1 \qquad \text{(binding branch)} \tag{6.13}$$

$$\sum_{k \in K^{\bar{d}}(\bar{t})} x_k^{\bar{d}} = 0 \qquad \text{(banning branch).} \tag{6.14}$$

2. *Splitting:* Assume that trip $\bar{t}$ can be performed from depots $d_{i_1}, d_{i_2}, \ldots, d_{i_q}$. We create q branches, and force to cover trip $\bar{t}$ by a duty from depot d_{i_j} on the jth branch $(1 \le j \le q)$. Formally, for the jth branch we have

$$\sum_{k \in K^{d_{i_j}}(\bar{t})} x_k^{d_{i_j}} = 1. \tag{6.15}$$

Note that these two branching strategies are the same if we have exactly two depots.

As we remarked above, these branching strategies are not complete in the sense that they cannot be used if each trip t is committed for a single depot, i.e., $C_{\bar{x}}(t,d) = 1$ holds for some depot d. However, handling these branching rules is quite easy without adding any inequalities of (6.13)–(6.14) or (6.15) to the problem. That is, on the one hand we can easily fix the appropriate existing variables to zero according to the corresponding branch. On the other hand, if a trip is forbidden to cover by a duty from the depot for which we want to price out new duties, we just erase the corresponding trip-arc from the piece generation network of the depot, and the pricing procedure described in Section 6.2.2 can be used without any modification.

SPP-*based branching*

Our branching strategy based on the branching scheme proposed by Ryan and Foster (1981) (see Section 2.6.5), that is, we branch on duty variables utilizing the set partitioning structure

of constraints (6.8). Consider a fractional solution $(\bar{x}, \bar{y})$ to the relaxation of the corresponding restricted master problem, and for trips t, u and depot d let

$$B_{\bar{x}}(t,u;d) := \sum_{k \in K^d(t,u)} \bar{x}_k^d,$$

where $K^d(t,u) \subset K^d$ is the set of duties covering both trips t and u. We select a pair of trips $(\bar{t}, \bar{u})$ and a depot $\bar{d}$ to branch on such that $(\bar{t}, \bar{u}; \bar{d}) = \arg\min_{(t,u;d)} |B_{\bar{x}}(t,u;d) - 0.5|$. The branching scheme requires to cover trips $\bar{t}$ and $\bar{u}$ by the same duty from depot d on one branch and not to cover by the same duty from depot d on the other. Formally,

$$\sum_{k \in K^{\bar{d}}(\bar{t},\bar{u})} x_k^d = 1 \qquad \text{(same branch)} \tag{6.16}$$

$$\sum_{k \in K^{\bar{d}}(\bar{t},\bar{u})} x_k^d = 0 \qquad \text{(diff branch).} \tag{6.17}$$

Note that this branching strategy can be used for both of the single-depot and the multiple-depot case if there exists trips t, u and a depot d such that $0 < B_{\bar{x}}(t,u;d) < 1$.

Again, we do not intend to add any of the inequalities (6.16)–(6.17) to the restricted master problem, however, handling this branching rule in the pricing procedure is a bit cumbersome as we explain in the following. Assume that in a node we would like to generate new feasible duties for a given depot, but a branching decision requires not to cover trips t and u by the same duty. In addition, assume that a combined duty consisting of pieces of work p_t and p_u has a negative reduced cost, where pieces of work p_t and p_u contain trips t and u, respectively. This duty is infeasible according to the branching decision, and it may shadow a feasible duty with negative reduced cost. Thus we have to ensure that pieces of work (i) containing trip t, (ii) not containing trip t, (iii) containing trip u, (iv) not containing trip u are also generated. These terms are going to be more complicated in nodes with higher depth. In order to resolve this difficulties we apply a two-step procedure. That is, in the first step we generate duties as we described before until no more duties with negative reduced cost are left. If in the last pricing round we do not refuse any duties according to branching decisions, we can stop (i.e., the node is solved optimally), since no overshadowed duties with negative reduced costs are left. Otherwise, in the second step we choose a duty which was refused in the last pricing round and generate all duties that may be overshadowed by this duty. More specifically, assume that the refused duty consists of pieces of work p_1 and p_2 where p_i refers to an u_i-v_i path for $i = 1, 2$, respectively. We construct a piece of work set S by generating all pieces of work that correspond to an u_i-v_i path ($i = 1, 2$). To generate duties in the second step we use the piece of work set corresponding to the shortest paths along with the piece of work set S. We repeat this procedure until no duties with negative reduced cost are left or refused.

Default 0-1 branching

As we mentioned before, when all of our strategies failed to branch, as a last resort we rely on the default branching strategy of the MILP solver. That is, a fractional duty variable x_k^d is

chosen, and it is forced to 0 on the first branch and 1 on the second branch. In the former case we need to ensure that this forbidden duty will be not regenerated during the pricing procedure. Again, for details we refer to (Horváth and Kis, 2019a).

6.2.4 Primal heuristics

Any time during the solution method we can perform an obvious primal solution search approach, that is we solve problem (6.7)–(6.12) with the current column set. However, such a problem can be hard to solve, so it is not worth to apply this method frequently.

6.3 Computational results

In this section we present our computational experiments, where the main goals were

- to evaluate our integrated method, and
- to compare our method with other solution approaches from the literature.

6.3.1 Test environment and implementation

All the computational experiments were performed on a workstation with 4GB RAM, and XEON X5650 CPU of 2.67 GHz, and under Linux operating system using a single thread only. Our solution method is implemented in C++ programming language using SCIP Optimization Suite (SCIP, version 3.1.1) as a branch-and-price framework. We also used FICO Xpress Solver (Xpress, version 28.01.09) to solve certain phases; and Library for Efficient Modeling and Optimization in Networks (Lemon, version 1.3.1) to handle graphs and to perform network algorithms.

6.3.2 Instances and problem parameters

We tried to comply with Steinzen et al. (2010) as much as possible, that is, we used the same instance set, the same duty parameters and the same costs as in (Steinzen et al., 2010).

We used the randomly generated problem instances of Huisman et al. (2005) available in (Huisman, 2003). These instances are classified into two classes according to travel speed (i.e., length of the trips), that is, class A consists of shorter trips than class B, hence vehicle blocks and duties cover more trips, thus instances in class A can be considered more difficult. In class A for each $n = 80, 100, 160, 200, 320$ there are 10 instances (one trip-file and one deadhead-file) containing n trips and requiring 4 depots and 4 or 5 stations.

In accordance with Huisman (2004) we used five types of duties with the properties described in Table 6.2. A *tripper duty* consists of one piece of work with length between 30 minutes and 5 hours, while the combined duties (*early*, *day*, *late*, *split*) contains exactly two pieces of work separated by a break. For duties starting (ending) in a depot we assessed a sign-on (sign-off) time of 10 (of 5) minutes, and for duties starting (ending) at a station we assessed a

Table 6.2: Properties of duty types

	Tripper		Early		Day		Late		Split	
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
start time					8:00		13:15			
end time				16:30		18:14				19:30
piece length	0:30	5:00	0:30	5:00	0:30	5:00	0:30	5:00	0:30	5:00
break length	-	-	0:45		0:45		0:45		1:30	
spread time				9:45		9:45		9:45		12:00
working time				9:00		9:00		9:00		9:00

sign-on (sign-off) time of 15 minutes plus the deadhead time between the start (end) station and the depot. Start and end times in Table 6.2 correspond to the sign-on start and sign-off end time of the duty, respectively.

We assigned a fixed cost of 1000 for each vehicle and a cost of 1 for each minute a vehicle is outside of the depot. We assigned a fixed cost of 1000 for each duty and a cost of 0.1 for each minute a driver is working.

6.3.3 Experiments

In these experiments we solved problems with gap limit set to 0.5%, and time limit set to $20 \times |\mathcal{T}|$ seconds, i.e., the solution process could be stopped due to three reasons: (i) the best solution was proven to be optimal, (ii) the gap limit was reached (i.e., the relative gap between the lower bound and the current best solution was at most 0.5%), (iii) the time limit was reached (i.e., the execution time exceeded $20 \times |\mathcal{T}|$ seconds).

As we mentioned in Section 6.2, at the root node we used a two-stage approach for generating duties. At the end of the first stage we applied our primal heuristics, that is, we called Xpress with time limit set to 60 seconds to solve the current restricted master problem. At the end of the second stage we applied this heuristics only if the number of variables did not exceed 30 000.

Experiments on branching strategies

In these experiments we aimed to compare the partitioning and splitting branching strategies described in Section 6.2.3. In order to make a more extensive experiment we matched all of the trip-files with all of the deadhead-files for these tests, i.e., we used $10 \times 10 = 100$ problem instances. In Table 6.3 we present our results where we indicate the summarized solution status (*Status*) (these are, the number of instances that solved optimally (*O*), the number of instances where gap limit was reached (*G*), the number of instances where solving process was stopped due to time limit (*T*)); the best lower (*Lower*) and upper bound (*Upper*) and the

Table 6.3: Summary of experiments on branching strategies

Instances	Strategy	Status	Bound			Best solution			Time
		O/G/T	Lower	Upper	Gap	v	d	v+d	
80A	Partitioning	9/50/41	34 772.2	35 410.8	1.8%	9.5	18.5	28.0	755.6
	Splitting	8/49/43	34 769.4	35 481.4	2.0%	9.5	18.6	28.0	777.5
100A	Partitioning	6/49/45	41 624.0	42 464.2	2.0%	11.4	22.1	33.5	1136.6
	Splitting	5/43/52	41 621.7	42 533.2	2.2%	11.4	22.1	33.6	1205.4

corresponding gap (*Gap*) which is calculated as $100 \times (\text{Upper} - \text{Lower})/\text{Lower}$; the number of vehicles (v) and the number of drivers (d) in the best solution; and the execution time in seconds (*Time*).

Both for 80-trip and 100-trip instances, the partitioning based branching strategy gave the best results in terms of execution time, and the quality of solutions as well. Moreover, more instances were solved within the time limit with that rule. According to these results, in the following experiments we used the partitioning rule as the primary branching strategy.

Evaluation of the integrated method

In these experiments we evaluated or integrated method, and we present the results on 80-trip and 100-trip instances in Tables 6.4 and 6.5. In these tables we indicate the solution status (*Status*) (optimal: the instance is solved optimally; gap/time limit: the solving process is stopped due to the gap/time limit was reached); the lower bound at the root node (*Root*), and at the end of the procedure (*Global*); the value of the best solution (*Upper bound*); the corresponding gap (*Gap*) which is calculated as $100 \times (\text{Upper bound} - \text{Global})/\text{Global}$; and the execution time in seconds (*Time*).

We can see that 2 out of 10 80-trip instances are solved optimally, and 3 other instances are solved with gap less than 0.5%, moreover, the average gap of the 80-trips instances is 2.5%. For the 100-trip instances we also solved 2 out of 10 instances optimally, and 4 more instances are solved with gap limit, while the average gap is 2.2%.

We also remark that most of the computation time was spent at the root node for finding the optimal LP solution which sometimes required the generation of thousands of columns. In the other enumeration tree nodes, finding the optimum solution took much less effort in general.

Comparison of methods

In these experiments we considered four scenarios in order to compare of our sequential and integrated methods, and the integrated method of Steinzen et al. (2010). Method *Seq.* refers to the sequential approach we used to obtain in the initial restricted master problem, while

Table 6.4: Summary of the evaluation of the integrated method on instance set 80A

Instance	Status	Lower bound		Upper bound	Gap	Time
		Root	Global			
1	gap limit	31 619.6	31 619.6	31 702.3	0.3%	88.6
2	time limit	27 497.8	27 498.3	29 079.7	5.8%	1602.5
3	optimal	32 750.7	32 750.7	32 750.7	0.0%	87.1
4	time limit	34 162.4	34 169.8	34 922.2	2.2%	1600.6
5	gap limit	32 175.4	32 175.4	32 188.6	0.0%	112.0
6	time limit	31 393.9	31 407.5	32 879.4	4.7%	1602.6
7	gap limit	36 133.7	36 133.7	36 266.6	0.4%	115.9
8	time limit	43 017.6	43 040.9	44 419.3	3.2%	1601.1
9	optimal	34 638.4	34 638.4	34 643.9	0.0%	734.0
10	time limit	42 583.6	42 619.4	45 716.3	7.3%	1601.3
average		34 597.3	34 605.4	35 456.9	2.5%	914.6

Table 6.5: Summary of the evaluation of the integrated method on problem set 100A

Instance	Status	Lower bound		Upper bound	Gap	Time
		Root	Global			
1	optimal	49 183.8	49 183.8	49 183.8	0.0%	390.8
2	time limit	41 311.8	41 326.8	43 552.4	5.4%	2002.3
3	time limit	35 896.6	35 910.3	38 519.5	7.3%	2000.8
4	gap limit	40 217.2	40 217.2	40 255.5	0.1%	175.1
5	optimal	45 424.8	45 424.8	45 424.8	0.0%	344.5
6	gap limit	35 543.3	35 543.3	35 543.8	0.0%	230.0
7	time limit	36 242.3	36 257.3	37 231.3	2.7%	2003.2
8	gap limit	45 403.5	45 403.5	45 453.4	0.1%	237.1
9	time limit	50 566.0	50 572.6	53 708.4	6.2%	2002.7
10	gap limit	33 912.2	33 912.2	34 001.5	0.3%	683.7
average		41 370.2	41 375.2	42 287.4	2.2%	1007.0

Table 6.6: Comparing sequential and integrated methods

Instance set	Method	Best solution				Time
		v	d	v+d	Cost	
80A	Seq. [1]	9.2	24.3	33.5	40 588.0	1.2
	Int. (first) [1]	9.6	18.6	28.2	35 668.5	4.1
	Int. (timelimit) [1]	9.5	18.5	28.0	35 456.9	914.6
	Int. (Steinzen et al.) [2]	9.2	19.1	28.2		235.0
100A	Seq.	11.0	28.2	39.2	47 792.7	1.6
	Int. (first)	11.4	22.0	33.4	42 428.5	31.8
	Int. (timelimit)	11.4	21.9	33.3	42 287.4	1007.0
	Int. (Steinzen et al.)	11.0	22.7	33.7		369.0

[1] our methods; tested on a workstation with 4GB RAM, and XEON X5650 CPU of 2.67 GHz, and under Linux operating system.

[2] integrated method of Steinzen et al. (2010); tested on a Dell OptiPlex GX620 personal computer with an Intel Pentium IV 3.4 GHz processor and 2 GB of main memory under Windows XP.

the next two methods refer to our integrated approach. In the case of method *Int. (first)* we interrupted the solution procedure right after we found a feasible solution to the problem. In the case of method *Int. (timelimit)* we interrupted our procedure only when the time limit was reached (or we found a good enough solution). Method *Int. (Steinzen et al.)* refers to the integrated approach of Steinzen et al. (2010) which was tested on a Dell OptiPlex GX620 personal computer with an Intel Pentium IV 3.4 GHz processor under Windows XP.

In Table 6.6 we summarize our comparison of sequential and integrated methods where we indicate the number of vehicles (v), the number of drivers (d), the cost of the best solution (*Cost*); and the execution time in seconds (*Time*). For the detailed results we refer to (Horváth and Kis, 2019a, Tables 7–8).

Our remarks and observations are the following.

- Note that in the case of method Int. (Steinzen et al.) we do not indicate the solution costs since these are not provided in (Steinzen et al., 2010). We contacted the authors, however, they could not provide these detailed results.
- On the one hand, our experiments re-proved that one can obtain better solutions using the integrated approach instead of the sequential method. On the other hand, observe that we could improve on the first integer solution if we run the procedure until a time limit or a gap limit is reached, however, the average improvement over the first integer feasible solution is 1.1% in the 80-trip case, and 0.3% in the 100-trip case.
- One can see that our integrated method found solutions with fewer vehicles plus drivers than Steinzen et al. (2010). Both for 80-trip and 100-trip instances, our method found the

first integer solution quickly in 4.1 and 31.8 seconds, respectively, and on average it was at least as good as the final solution of Steinzen et al. (2010).

6.3.4 Conclusions and final remarks

Our computational results show that with limited computational resources (computation time and single CPU thread), near optimal schedules can be found for problems with 80–100 trips and 4 depots.

Note that Steinzen et al. (2010) present computational results for instances with $n = 160, 200, 320, 400, 640$, as well, however, solving instances with 160 trips took already about 1600 seconds on average, while 640-trip instances required about 16 hours. We also made experiments on the 160-trip instances, however, we were not able to solve any of these instances neither optimally nor with gap limit, in fact, the column generation procedure at the root node required more than 3 hours on average. Our best solutions yielded 11.5% gap on average, and the average number of vehicles and drivers used in these solutions ($v + d = 50.5$) is worse than that of (Steinzen et al., 2010) ($v + d = 46.6$).

In order to increase the problem size, one possible direction is to exploit multiple CPU cores/threads, but for that, one needs a parallel branch-and-price solver. Currently, the parallel branch-and-price implementation of SCIP is at the conceptual stage. Another option would be to get lower bounds faster, for which further acceleration strategies are needed.

Appendix A

Computational results of Chapter 4

A.1 Setting of methods investigated in Section 4.4

In Table A.1 we summarize the settings of methods investigated in Section 4.4. With symbols '★' and '·' we indicate if the corresponding component is used by the method or not, respectively.

A.2 Computational results of Section 4.4.4

In Table A.2 we summarize the results of experiments with primal heuristics and variable fixing procedures described in Section 4.4.4. We indicate the average number of the explored enumeration tree nodes (#*Nodes*) and the average running time (*Time*) of the entire branch-and-bound procedure in seconds. The detailed computations are provided in (Horváth and Kis, 2016b, Table A4).

A.3 Computational results of Section 4.4.5

In Table A.3 we summarize the results of experiments with cutting planes based on cuts described in Section 4.4.5. We indicate the average number of cutting planes added to the problem (#*Cutting planes*), the average number of the explored enumeration tree nodes (#*Nodes*), and the average running time (*Time*) of the entire branch-and-cut procedure in seconds. The detailed computations are provided in (Horváth and Kis, 2016b, Table A5).

A.4 Computational results of Section 4.4.6

In Table A.4 we summarize the results of experiments with cutting planes based on infeasible subpaths described in Section 4.4.6. We indicate the average number of cutting planes added to the problem (#*Cutting planes*), the average number of the explored enumeration tree

Table A.1: Settings of methods for experiments

Method	Xpress			Preprocessing		Variable fixing		Heuristics[4]	Cutting planes								
	P[1]	C[1]	H[1]	DB[2]	G[3]	G-varfix[3]	DB-based[4]		*s-t* cut prec[5]		cut based[6]			subpath prec[5]		inf subpath based[6]	
									(4.11)	(4.13)	(4.19)	(4.20)	(4.21)	(4.15)	(4.17)	(4.23)	(4.24)
BnB	·	·	·	★	★	·	·	·	·	·	·	·	·	·	·	·	·
BnB (Arcfix)	·	·	·	★	★	★	·	·	·	·	·	·	·	·	·	·	·
BnB (Varfix)	·	·	·	★	★	·	★	·	·	·	·	·	·	·	·	·	·
BnB (Heuristics)	·	·	·	★	★	·	·	★	·	·	·	·	·	·	·	·	·
BnB (All)	·	·	·	★	★	★	★	★	·	·	·	·	·	·	·	·	·
BnC (STCP)	·	·	·	★	★	·	·	·	★	★	·	·	·	·	·	·	·
BnC (CB)	·	·	·	★	★	·	·	·	·	·	★	★	★	·	·	·	·
BnC (STCP+CB)	·	·	·	★	★	·	·	·	★	★	★	★	★	·	·	·	·
BnC (SPP)	·	·	·	★	★	·	·	·	·	·	·	·	·	★	★	·	·
BnC (ISPB)	·	·	·	★	★	·	·	·	·	·	·	·	·	·	·	★	★
BnC (SPP+ISPB)	·	·	·	★	★	·	·	·	·	·	·	·	·	★	★	★	★
BnC (XC)	·	★	★	★	★	·	·	·	·	·	·	·	·	·	·	·	·
BnC (GC)	·	·	·	★	★	·	·	·	★	★	·	·	·	★	★	·	·
BnC (NC)	·	·	·	★	★	·	·	·	★	★	★	★	★	·	·	★	★
BnC (XC+NC)	·	★	★	★	★	·	·	·	★	★	★	★	★	·	·	★	★

[1] Xpress presolve (P), built-in cuts (C), and heuristics (H)
[2] preprocessing procedure of Dumitrescu and Boland (2003)
[3] preprocessing and variable fixing procedure of Garcia (2009)
[4] our primal heuristics and variable fixing procedure
[5] the *s-t* cut precedence and subpath precedence inequalities of Garcia (2009)
[6] our cut based and infeasible subpath based inequalities

Table A.2: Summary of experiments with primal heuristics and variable fixing procedures

Method	Class	#Nodes	Time	Class	#Nodes	Time	Class	#Nodes	Time
BnB	G1	1360.1	57.7	G5	1499.2	235.3	G9	2165.9	644.7
BnB (Arcfix)		1153.0	52.0		1328.2	211.9		1884.0	554.4
BnB (Varfix)		881.7	19.0		1000.3	54.0		1101.4	93.5
BnB (Heuristics)		1191.3	49.6		1223.9	175.2		1841.0	501.1
BnB (All)		775.3	19.4		877.8	53.6		1090.3	103.5
BnB	G2	1883.6	88.9	G6	2621.3	392.9	G10	2308.4	679.9
BnB (Arcfix)		1982.6	93.7		2464.7	375.0		1932.0	623.4
BnB (Varfix)		1351.3	33.5		1149.5	63.5		782.5	71.0
BnB (Heuristics)		1769.3	81.5		2303.7	339.2		2092.1	580.8
BnB (All)		1265.9	31.8		1086.1	66.8		660.6	83.2
BnB	G3	4275.3	187.1	G7	4877.5	709.0	G11	5533.7	1400.7
BnB (Arcfix)		4275.3	187.7		4877.5	706.8		5533.7	1407.4
BnB (Varfix)		3775.9	84.5		4281.7	235.4		4864.3	432.6
BnB (Heuristics)		3261.0	134.2		4410.4	615.0		4637.2	1087.5
BnB (All)		3412.2	76.6		3722.3	213.0		3990.3	370.9
BnB	G4	3699.0	143.4	G8	4222.5	560.8	G12	6044.6	1434.0
BnB (Arcfix)		3699.0	141.9		4222.5	556.6		6044.6	1429.1
BnB (Varfix)		3294.8	63.9		4620.6	223.2		5636.2	469.2
BnB (Heuristics)		2742.5	101.0		3644.5	444.9		5150.7	1073.3
BnB (All)		3102.2	62.4		3982.5	197.9		5067.4	432.3

Table A.3: Summary of experiments with cutting planes based on cuts

Method	Class	#Cutting planes		#Nodes	Time	Class	#Cutting planes		#Nodes	Time	Class	#Cutting planes		#Nodes	Time
		STCP	CB				STCP	CB				STCP	CB		
BnC (STCP)	G1	121.4	0.0	1193.8	46.3	G5	163.2	0.0	1068.4	140.5	G9	192.6	0.0	1541.0	385.4
BnC (CB)		0.0	515.1	1080.2	44.7		0.0	868.0	993.1	132.0		0.0	1082.1	1381.7	369.6
BnC (STCP+CB)		128.0	425.4	1134.4	49.0		162.7	889.8	1000.4	133.1		163.3	746.9	1445.3	374.9
BnC (STCP)	G2	65.2	0.0	1548.4	67.6	G6	172.4	0.0	2104.8	284.6	G10	318.0	0.0	2007.5	520.8
BnC (CB)		0.0	693.7	1429.4	65.2		0.0	1003.0	1958.8	266.8		0.0	747.4	1824.0	483.3
BnC (STCP+CB)		57.6	648.4	1484.5	66.2		186.9	808.1	2011.5	269.5		290.1	697.5	1930.4	509.5
BnC (STCP)	G3	0.3	0.0	2812.4	107.7	G7	5.2	0.0	3513.1	465.3	G11	14.7	0.0	3972.9	1078.2
BnC (CB)		0.0	171.3	2809.1	119.0		0.0	717.3	3539.1	489.7		0.0	1159.9	3810.8	1076.5
BnC (STCP+CB)		0.4	170.7	2809.4	119.3		5.1	742.5	3553.8	493.8		16.1	1200.2	3886.8	1077.3
BnC (STCP)	G4	0.0	0.0	2403.2	86.9	G8	0.1	0.0	3299.8	392.3	G12	0.3	0.0	4326.4	1037.3
BnC (CB)		0.0	3.8	2405.2	109.1		0.0	30.6	3333.1	440.8		0.0	95.3	4350.6	1117.8
BnC (STCP+CB)		0.0	3.8	2405.2	109.6		0.1	30.1	3330.5	441.3		0.4	95.3	4349.0	1121.5

nodes (#*Nodes*), and the average running time (*Time*) of the entire branch-and-cut procedure in seconds. The detailed computations are provided in (Horváth and Kis, 2016b, Table A6).

A.5 Computational results of Section 4.4.7

In Table A.5 we summarize the results of experiments with cutting planes based on infeasible subpaths described in Section 4.4.7. We indicate the average number of the explored enumeration tree nodes (#*Nodes*) and the average running time (*Time*) of the entire branch-and-cut procedure in seconds. The detailed computations are provided in (Horváth and Kis, 2016b, Table A7).

A.6 Computational results of Section 4.5.3

In Table A.6 we summarize the results of experiments on instance sets of Santos et al. (2007). We indicate the average running time over the instance sets in seconds of the Reference Point Method (RPM), the Pulse Algorithm (PA), and our branch-and-cut algorithm (LPB). The detailed computations are provided in (Horváth and Kis, 2016b, Appendix S).

Table A.4: Summary of experiments with cutting planes based on infeasible subpaths

Method	Class	#Cutting planes		#Nodes	Time	Class	#Cutting planes		#Nodes	Time	Class	#Cutting planes		#Nodes	Time
		SPP	ISPB				SPP	ISPB				SPP	ISPB		
BnC (SPP)	G1	591.9	0.0	1249.2	46.7	G5	759.8	0.0	1040.2	128.4	G9	711.7	0.0	1458.2	374.8
BnC (ISPB)		0.0	1031.8	1094.2	39.2		0.0	1545.0	1001.3	125.1		0.0	1839.8	1420.2	374.1
BnC (SPP+ISPB)		561.4	819.6	1261.4	47.1		715.0	1113.6	1024.5	122.3		702.4	1197.1	1474.9	370.4
BnC (SPP)	G2	697.1	0.0	1537.2	65.2	G6	725.8	0.0	2261.6	288.7	G10	759.6	0.0	1977.7	506.4
BnC (ISPB)		0.0	1140.3	1480.3	62.9		0.0	1134.2	2019.2	272.5		0.0	977.8	1815.8	487.0
BnC (SPP+ISPB)		712.5	979.1	1539.9	61.5		697.0	729.9	2188.0	283.0		705.6	658.2	1940.8	497.5
BnC (SPP)	G3	970.7	0.0	2845.1	111.3	G7	1032.2	0.0	3568.1	481.8	G11	1084.5	0.0	3907.9	1046.4
BnC (ISPB)		0.0	3212.3	2917.2	107.8		0.0	3968.5	3320.7	444.0		0.0	4436.8	3673.2	1000.0
BnC (SPP+ISPB)		908.6	2704.1	2809.3	102.7		983.3	2808.7	3483.2	459.7		1104.5	3024.8	3848.7	1040.7
BnC (SPP)	G4	951.6	0.0	2650.2	95.3	G8	1146.3	0.0	3248.2	399.9	G12	1379.1	0.0	4392.6	1055.8
BnC (ISPB)		0.0	3828.5	2428.4	87.7		0.0	5138.4	3123.7	378.9		0.0	5964.9	4403.2	1076.3
BnC (SPP+ISPB)		809.5	3607.9	2694.3	96.7		1058.7	4513.1	3237.6	390.9		1263.6	5715.6	4439.7	1071.6

Table A.5: Summary of combined experiments

Method	Class	#Nodes	Time	Class	#Nodes	Time	Class	#Nodes	Time
BnC (XC)	G1	1744.3	78.0	G5	1450.4	226.7	G9	2215.9	653.4
BnC (GC)		990.1	22.6		904.3	55.7		1142.3	104.3
BnC (NC)		816.0	19.8		886.2	56.5		991.3	96.3
BnC (XC+NC)		1194.9	28.5		1040.9	76.4		1337.5	137.6
BnC (XC)	G2	2241.1	106.2	G6	2720.8	447.2	G10	2664.7	826.7
BnC (GC)		1370.6	34.4		1262.0	71.1		994.5	84.6
BnC (NC)		1283.7	33.6		1087.8	65.7		921.8	85.1
BnC (XC+NC)		1617.7	45.6		1489.8	89.2		1112.9	108.6
BnC (XC)	G3	4578.3	177.8	G7	4961.6	768.3	G11	5508.8	1515.8
BnC (GC)		3587.6	81.5		3575.7	219.9		4290.4	430.3
BnC (NC)		3362.5	83.5		3623.9	238.8		4156.6	415.5
BnC (XC+NC)		4317.0	109.3		4345.0	326.6		4502.6	541.0
BnC (XC)	G4	4341.8	169.0	G8	5021.0	670.9	G12	5777.3	1689.0
BnC (GC)		3133.3	66.2		3859.2	207.4		4830.0	429.2
BnC (NC)		3124.3	82.7		3771.6	239.7		5372.3	530.2
BnC (XC+NC)		4172.3	117.3		4697.7	320.2		5280.8	620.1

Table A.6: Average computation times in seconds on instance set S

Set	Group 1			Group 2			Group 3			Group 4			Group 5		
	RPM [1]	PA [2]	LPB [3]	RPM	PA	LPB	RPM	PA	LPB	RPM	PA	LPB	RPM	PA	LPB
S1	0.88	0.01	0.01	0.81	0.01	0.01	0.81	0.01	0.01	0.80	0.01	0.01	0.84	0.01	0.02
S2	1.61	0.02	0.01	1.56	0.02	0.02	1.59	0.02	0.02	1.67	0.02	0.02	1.85	0.02	0.02
S3	4.20	0.02	0.02	3.57	0.02	0.02	3.76	0.02	0.02	4.51	0.02	0.03	5.31	0.02	0.03
S4	7.73	0.04	0.04	6.96	0.05	0.05	8.81	0.04	0.05	11.18	0.05	0.05	13.06	0.04	0.03
S5	11.38	0.07	0.06	13.47	0.06	0.09	17.49	0.07	0.09	21.10	0.07	0.06	24.48	0.06	0.04
S6	16.61	0.07	0.08	20.05	0.14	0.77	25.26	0.16	0.12	30.04	0.10	0.10	33.82	0.08	0.07
S7	3.00	0.02	0.03	3.00	0.02	0.03	3.00	0.02	0.03	2.97	0.02	0.03	2.97	0.02	0.03
S8	5.73	0.03	0.04	5.66	0.03	0.04	5.69	0.03	0.04	5.92	0.03	0.04	6.33	0.03	0.03
S9	13.45	0.06	0.04	12.39	0.06	0.04	13.83	0.06	0.04	16.92	0.06	0.04	20.22	0.06	0.04
S10	28.84	0.11	0.09	28.68	0.09	0.11	33.79	0.09	0.10	42.92	0.09	0.09	50.11	0.12	0.08
S11	43.66	0.17	0.14	49.47	0.14	0.19	65.49	0.15	0.18	78.18	0.15	0.15	89.40	0.19	0.14
S12	64.85	0.17	0.22	77.41	0.22	0.34	96.36	0.28	0.24	111.99	0.23	0.20	127.56	0.18	0.16
S13	13.39	0.05	0.06	12.09	0.05	0.05	12.50	0.05	0.05	12.81	0.05	0.07	12.99	0.05	0.06
S14	22.05	0.08	0.08	22.06	0.08	0.08	23.02	0.08	0.08	25.26	0.08	0.11	27.46	0.08	0.11
S15	53.39	0.12	0.15	49.39	0.12	0.13	57.26	0.12	0.13	74.26	0.12	0.18	88.16	0.12	0.18
S16	112.92	0.21	0.22	108.43	0.22	0.23	142.34	0.26	0.25	180.23	0.22	0.20	215.63	0.21	0.21
S17	167.53	0.29	0.36	201.34	0.34	0.49	270.36	0.31	0.38	331.30	0.31	0.39	435.46	0.31	0.36
S18	248.18	0.39	0.49	321.77	0.47	32.20	420.77	0.52	0.56	504.70	0.55	0.37	567.61	0.43	0.33

[1] Reference Point Method of Pugliese and Guerriero (2013a); tested with Intel Core i7-620M, 2.67 GHz CPU, under Windows 7

[2] Pulse Algorithm of Lozano and Medaglia (2013); tested with Intel Core 2 Duo P8600, 2.4 GHz CPU, under Windows XP

[3] our branch-and-cut approach; tested with Intel Core i7-4710MQ, 2.5 GHz CPU, under Windows 7

Appendix B

Proofs postponed from Chapter 5

B.1 Proof of Lemma 5.2

Proof of statement (i). Let $p, q = 1, \dots, n$ be distinct elements, $1 \le j_1 < j_2 < j_3 < j_4 \le 2n$ and consider points $P_1 = (s^1, e^1), P_2 = (s^2, e^2) \in S_{2n}^{2\text{-chains}}$ such that $\sigma_p(P_1) = (j_1, j_2)$, $\sigma_q(P_1) = (j_3, j_4)$ and $\sigma_q(P_2) = (j_1, j_3)$, $\sigma_q(P_2) = (j_2, j_4)$ and $\sigma_r(P_1) = \sigma_r(P_2)$ for all $r \notin \{p, q\}$, i.e.,

$$s^1_{p,j_1} = e^1_{p,j_2} = s^1_{q,j_3} = e^1_{q,j_4} = 1 \quad \text{and} \quad s^2_{p,j_1} = e^2_{p,j_3} = s^2_{q,j_2} = e^2_{q,j_4} = 1,$$

and $s^1_{r,j} = s^2_{r,j}$, $e^1_{r,j} = e^2_{r,j}$ for all $r \notin \{p, q\}$ and $j = 1, \dots, 2n$. Since P_1 and P_2 satisfy (5.25), we have

$$\alpha_{p,j_1} + \beta_{p,j_2} + \alpha_{q,j_3} + \beta_{q,j_4} + \sum_{\substack{r=1 \\ r \neq p,q}}^{n} \sum_{j=1}^{2n} \left(\alpha_{r,j} s^1_{r,j} + \beta_{r,j} e^1_{r,j} \right) = \gamma,$$

and

$$\alpha_{p,j_1} + \beta_{p,j_3} + \alpha_{q,j_2} + \beta_{q,j_4} + \sum_{\substack{r=1 \\ r \neq p,q}}^{n} \sum_{j=1}^{2n} \left(\alpha_{r,j} s^2_{r,j} + \beta_{r,j} e^2_{r,j} \right) = \gamma,$$

thus, by subtracting the second equation from the first one, we have $\beta_{p,j_2} + \alpha_{q,j_3} = \alpha_{q,j_2} + \beta_{p,j_3}$ $(1 < j_2 < j_3 < 2n)$, that is, statement (i) holds for $p \neq q$.

Since $n \ge 3$, we can choose pairwise distinct elements $p, q, r = 1, \dots, n$, therefore we have

$$\alpha_{p,j''} - \alpha_{p,j'} = \beta_{q,j''} - \beta_{q,j'} = \alpha_{r,j''} - \alpha_{r,j'} = \beta_{p,j''} - \beta_{p,j'},$$

that is, statement (i) also holds for $p = q$. □

Proof of statement (ii). Let $p, q = 1, \dots, n$ be distinct elements, $1 \le j_1 < j_2 < j_3 < j_4 \le 2n$ and consider points $P_1, P_2 \in S_{2n}^{2\text{-chains}}$ such that $\sigma_p(P_1) = (j_1, j_3)$, $\sigma_q(P_1) = (j_2, j_4)$ and $\sigma_p(P_2) = (j_2, j_3)$, $\sigma_q(P_2) = (j_1, j_4)$ and $\sigma_r(P_1) = \sigma_r(P_2)$ for all $r \notin \{p, q\}$. Since P_1 and P_2 satisfy (5.25), we have $\alpha_{p,j_2} - \alpha_{p,j_1} = \alpha_{q,j_2} - \alpha_{q,j_1}$ $(1 \le j_1 < j_2 < 2n - 1)$, that is, statement (ii) holds for $j'' < 2n - 1$.

Now, consider points $P_3, P_4 \in S_{2n}^{2\text{-chains}}$ such that $\sigma_p(P_3) = (j_1, 2n-2)$, $\sigma_q(P_3) = (2n-1, 2n)$ and $\sigma_p(P_4) = (2n-1, 2n)$, $\sigma_q(P_4) = (j_1, 2n-2)$ and $\sigma_r(P_3) = \sigma_r(P_4)$ for all $r \notin \{p, q\}$. Since P_3 and P_4 satisfy (5.25), we have $\alpha_{p,j_1} + \beta_{p,2n-2} + \alpha_{q,2n-1} + \beta_{q,2n} = \alpha_{p,2n-1} + \beta_{p,2n} + \alpha_{q,j_1} + \beta_{q,2n-2}$. According to statement (i) (note that $1 < 2n-2$) we have $\beta_{p,2n} - \beta_{p,2n-2} = \beta_{q,2n} - \beta_{q,2n-2}$, therefore $\alpha_{p,j_1} + \alpha_{q,2n-1} = \alpha_{q,j_1} + \alpha_{p,2n-1}$, that is, statement (ii) also holds for $j'' = 2n-1$. □

Proof of statement (iii). Let $p, q = 1, \dots, n$ be distinct elements, $1 \le j_1 < j_2 < j_3 < j_4 \le 2n$ and consider points $P_1, P_2 \in S_{2n}^{2\text{-chains}}$ such that $\sigma_p(P_1) = (j_1, j_3)$, $\sigma_q(P_1) = (j_2, j_4)$ and $\sigma_p(P_2) = (j_1, j_4)$, $\sigma_q(P_2) = (j_2, j_3)$ and $\sigma_r(P_1) = \sigma_r(P_2)$ for all $r \notin \{p, q\}$. Since P_1 and P_2 satisfy (5.25), we have $\beta_{p,j_4} - \beta_{p,j_3} = \beta_{q,j_4} - \beta_{q,j_3}$ ($2 < j_3 < j_4 \le 2n$), that is, statement (iii) holds for $2 < j'$.

Now, consider points $P_3, P_4 \in S_{2n}^{2\text{-chains}}$ such that $\sigma_p(P_3) = (1, 2)$, $\sigma_q(P_3) = (3, j_4)$ and $\sigma_p(P_4) = (3, j_4)$, $\sigma_q(P_4) = (1, 2)$ and $\sigma_r(P_3) = \sigma_r(P_4)$ for all $r \notin \{p, q\}$. Since P_3 and P_4 satisfy (5.25), we have $\alpha_{p,1} + \beta_{p,2} + \alpha_{q,3} + \beta_{q,j_4} = \alpha_{p,3} + \beta_{p,j_4} + \alpha_{q,1} + \beta_{q,2}$. According to statement (i) (note that $3 < 2n$) we have $\alpha_{p,3} - \alpha_{p,1} = \alpha_{q,3} - \alpha_{q,1}$, therefore $\beta_{p,2} + \beta_{q,j_4} = \beta_{q,2} + \beta_{p,j_4}$, that is, statement (iii) also holds for $j' = 2$. □

B.2 Proof of Lemma 5.3

Proof of statement (i). Let $p, q = 1, \dots, t$ be distinct elements, $1 \le j_1 < j_2 \le 2k < j_3 < j_4 \le 2n$ and consider points $P_1, P_2 \in S_{2n}^{parity}$ such that $\sigma_p(P_1) = (j_1, j_3)$, $\sigma_q(P_1) = (j_2, j_4)$ and $\sigma_p(P_2) = (j_2, j_3)$, $\sigma_q(P_2) = (j_1, j_4)$ and $\sigma_r(P_1) = \sigma_r(P_2)$ for all $r \notin \{p, q\}$, i.e.,

$$s^1_{p,j_1} = e^1_{p,j_3} = s^1_{q,j_2} = e^1_{q,j_4} = 1 \quad \text{and} \quad s^2_{p,j_2} = e^2_{p,j_3} = s^2_{q,j_1} = e^2_{q,j_4} = 1,$$

and $s^1_{r,j} = s^2_{r,j}$, $e^1_{r,j} = e^2_{r,j}$ for all $r \notin \{p, q\}$ and $j = 1, \dots, 2n$. Note that such points exist according to Remark 5.2. Since P_1 and P_2 satisfy (5.31), we have

$$\alpha_{p,j_1} + \beta_{p,j_3} + \alpha_{q,j_2} + \beta_{q,j_4} + \sum_{\substack{r=1 \\ r \ne p,q}}^{n} \sum_{j=1}^{2n} \left(\alpha_{r,j} s^1_{r,j} + \beta_{r,j} e^1_{r,j} \right) = \gamma,$$

and

$$\alpha_{p,j_2} + \beta_{p,j_3} + \alpha_{q,j_1} + \beta_{q,j_4} + \sum_{\substack{r=1 \\ r \ne p,q}}^{n} \sum_{j=1}^{2n} \left(\alpha_{r,j} s^2_{r,j} + \beta_{r,j} e^2_{r,j} \right) = \gamma,$$

thus, by subtracting the first one from the second one, we have $\alpha_{p,j_1} + \alpha_{q,j_2} = \alpha_{p,j_2} + \alpha_{q,j_1}$. □

Proof of statement (iii). Let $p, q = 1, \dots, t$ be distinct elements, $1 \le j_1 < j_2 \le 2k < j_3 < j_4 \le 2n$ and consider points $P_1, P_2 \in S_{2n}^{parity}$ such that $\sigma_p(P_1) = (j_1, j_3)$, $\sigma_q(P_1) = (j_2, j_4)$ and $\sigma_p(P_2) = (j_1, j_4)$, $\sigma_q(P_2) = (j_2, j_3)$ and $\sigma_r(P_1) = \sigma_r(P_2)$ for all $r \notin \{p, q\}$. Since P_1 and P_2 satisfy (5.31) we have $\beta_{p,j_4} - \beta_{p,j_3} = \beta_{q,j_4} - \beta_{q,j_3}$. □

Proof of statement (ii). Let $p, q = 1, \dots, t$ be distinct elements and $1 \le j_1 \le 2k < j_2 < j_3 < j_4 \le 2n$. First, consider points $P_1, P_2 \in S_{2n}^{parity}$ such that $\sigma_p(P_1) = (j_1, j_3)$, $\sigma_q(P_1) = (j_2, j_4)$

and $\sigma_p(P_2) = (j_2, j_3)$, $\sigma_q(P_2) = (j_1, j_4)$ and $\sigma_r(P_1) = \sigma_r(P_2)$ for all $r \notin \{p, q\}$. Since P_1 and P_2 satisfy (5.31) we have $\alpha_{p,j_2} - \alpha_{p,j_1} = \alpha_{q,j_2} - \alpha_{q,j_1}$, that is, statement (ii) holds if $2k < j'' < 2n-1$.

Now, consider points $P_3, P_4 \in S_{2n}^{parity}$ such that $\sigma_p(P_3) = (j_1, 2k+1)$, $\sigma_q(P_3) = (2n-1, 2n)$ and $\sigma_p(P_4) = (2n-1, 2n)$, $\sigma_q(P_4) = (j_1, 2k+1)$ and $\sigma_r(P_3) = \sigma_q(P_3)$ for all $r \notin \{p, q\}$. Since P_3 and P_4 satisfy (5.31) we have $\alpha_{p,j_1} + \beta_{p,2k+1} + \alpha_{q,2n-1} + \beta_{q,2n} = \alpha_{q,j_1} + \beta_{q,2k+1} + \alpha_{p,2n-1} + \beta_{p,2n}$. According to statement (iii), $\beta_{p,2n} - \beta_{p,2k+1} = \beta_{q,2n} - \beta_{q,2k+1}$, thus $\alpha_{p,j_1} + \alpha_{q,2n-1} = \alpha_{q,j_1} + \alpha_{p,2n-1}$, that is, statement (ii) also holds for $j'' = 2n-1$. □

Proof of statement (vi). Let $p, q = 1, \ldots, t$ be distinct elements, $1 \le j_1 < j_2 < j_3 \le 2k < j_4 \le 2n$. First, consider points $P_1, P_2 \in S_{2n}^{parity}$ such that $\sigma_p(P_1) = (j_1, j_3)$, $\sigma_q(P_1) = (j_2, j_4)$ and $\sigma_q(P_2) = (j_1, j_4)$, $\sigma_q(P_2) = (j_2, j_3)$ and $\sigma_r(P_1) = \sigma_r(P_2)$ for all $r \notin \{p, q\}$. Since P_1 and P_2 satisfy (5.31) we have $\beta_{p,j_4} - \beta_{p,j_3} = \beta_{q,j_4} - \beta_{q,j_3}$, that is, statement (iv) holds if $2 < j' \le 2k$.

Now, consider points $P_3, P_4 \in S_{2n}^{parity}$ such that $\sigma_p(P_3) = (1, 2)$, $\sigma_q(P_3) = (2k, j_4)$ and $\sigma_p(P_4) = (2k, j_4)$, $\sigma_q(P_4) = (1, 2)$ and $\sigma_r(P_3) = \sigma_r(P_4)$ for all $r \notin \{p, q\}$. Since P_3 and P_4 satisfy (5.31) we have $\alpha_{p,1} + \beta_{p,2} + \alpha_{q,2k} + \beta_{q,j_4} = \alpha_{q,1} + \beta_{q,2} + \alpha_{p,2k} + \beta_{p,j_4}$. According to statement (i), $\alpha_{p,2k} - \alpha_{p,1} = \alpha_{q,2k} - \alpha_{q,1}$, thus $\beta_{p,2} + \beta_{q,j_4} = \beta_{q,2} + \beta_{p,j_4}$, that is, statement (iv) also holds for $j' = 2$. □

Proof of statement (v). Let $p, q = 1, \ldots, t$ be distinct elements, $1 \le j_1 < j_2 < j_3 \le 2k < j_4 \le 2n$ and consider points $P_1, P_2 \in S_{2n}^{parity}$ such that $\sigma_p(P_1) = (j_1, j_3)$, $\sigma_q(P_1) = (j_2, j_4)$ and $\sigma_p(P_2) = (j_1, j_2)$, $\sigma_q(P_2) = (j_3, j_4)$ and $\sigma_r(P_1) = \sigma_r(P_2)$ for all $r \notin \{p, q\}$. Since P_1 and P_2 satisfy (5.31) we have $\alpha_{p,j_3} - \alpha_{p,j_2} = \beta_{q,j_3} - \beta_{q,j_2}$.

Since $3 \le t$, we can choose pairwise distinct element $p, q, r = 1, \ldots, t$, therefore we have

$$\alpha_{p,j_3} - \alpha_{p,j_2} = \beta_{q,j_3} - \beta_{q,j_2} = \alpha_{r,j_3} - \alpha_{r,j_2} = \beta_{p,j_3} - \beta_{p,j_2},$$

that is, statement (v) also holds for $p = q$. □

Proof of statement (vi). Let $p, q = 1, \ldots, t$ be distinct elements, $1 \le j_1 \le 2k < j_2 < j_3 < j_4 \le 2n$ and consider points $P_1, P_2 \in S_{2n}^{parity}$ such that $\sigma_p(P_1) = (j_1, j_3)$, $\sigma_q(P_1) = (j_2, j_4)$ and $\sigma_p(P_2) = (j_1, j_2)$, $\sigma_q(P_2) = (j_3, j_4)$ and $\sigma_r(P_1) = \sigma_r(P_2)$ for all $r \notin \{p, q\}$. Since P_1 and P_2 satisfy (5.31) we have $\alpha_{p,j_3} - \alpha_{p,j_2} = \beta_{q,j_3} - \beta_{q,j_2}$.

Since $3 \le t$, we can choose pairwise distinct element $p, q, r = 1, \ldots, t$, therefore we have

$$\alpha_{p,j_3} - \alpha_{p,j_2} = \beta_{q,j_3} - \beta_{q,j_2} = \alpha_{r,j_3} - \alpha_{r,j_2} = \beta_{p,j_3} - \beta_{p,j_2},$$

that is, statement (vi) also holds for $p = q$. □

B.3 Proof of Lemma 5.4

Proof of statement (vii). Let $p = 1, \ldots, t$, $\bar{q} = t+1, \ldots, n$ and $1 \le j_1 < j_2 \le 2k < j_3 < j_4 \le 2n$. Consider points $P_1, P_2 \in S_{2n}^{parity}$ such that $\sigma_p(P_1) = (j_1, j_3)$, $\sigma_{\bar{q}}(P_1) = (j_2, j_4)$ and $\sigma_p(P_2) = (j_2, j_3)$,

$\sigma_{\bar{q}}(P_2) = (j_1, j_4)$ and $\sigma_r(P_1) = \sigma_r(P_2)$ for all $r \notin \{p, \bar{q}\}$. Since P_1 and P_2 satisfy (5.31) we have $\alpha_{p,j_2} - \alpha_{p,j_1} = \alpha_{\bar{q},j_2} - \alpha_{\bar{q},j_1}$. □

Proof of statement (viii). Let $p = 1, \dots, t$, $\bar{q} = t+1, \dots, n$ and $1 \le j_1 < j_2 \le 2k < j_3 < j_4 \le 2n$. Consider points $P_1, P_2 \in S_{2n}^{parity}$ such that $\sigma_p(P_1) = (j_1, j_3)$, $\sigma_{\bar{q}}(P_1) = (j_2, j_4)$ and $\sigma_p(P_2) = (j_1, j_4)$, $\sigma_{\bar{q}}(P_2) = (j_2, j_3)$ and $\sigma_r(P_1) = \sigma_r(P_2)$ for all $r \notin \{p, \bar{q}\}$. Since P_1 and P_2 satisfy (5.31) we have $\beta_{p,j_4} - \beta_{p,j_3} = \beta_{\bar{q},j_4} - \beta_{\bar{q},j_3}$. □

Proof of statement (ix). Let $p = 1, \dots, t$, $\bar{q} = t+1, \dots, n$ and $1 \le j_1 < j_2 < j_3 \le 2k < j_4 \le 2n$. Consider points $P_1, P_2 \in S_{2n}^{parity}$ such that $\sigma_p(P_1) = (j_3, j_4)$, $\sigma_{\bar{q}}(P_1) = (j_1, j_2)$ and $\sigma_p(P_2) = (j_2, j_4)$, $\sigma_{\bar{q}}(P_2) = (j_1, j_3)$ and $\sigma_r(P_1) = \sigma_r(P_2)$ for all $r \notin \{p, \bar{q}\}$. Since P_1 and P_2 satisfy (5.31) we have $\alpha_{p,j_3} - \alpha_{p,j_2} = \beta_{\bar{q},j_3} - \beta_{\bar{q},j_2}$. □

Proof of statement (x). Let $p = 1, \dots, t$, $\bar{q} = t+1, \dots, n$ and $1 \le j_1 < j_2 \le 2k < j_3 < j_4 \le 2n$. Consider points $P_1, P_2 \in S_{2n}^{parity}$ such that $\sigma_p(P_1) = (j_3, j_4)$, $\sigma_{\bar{q}}(P_1) = (j_1, j_2)$ and $\sigma_p(P_2) = (j_2, j_4)$, $\sigma_{\bar{q}}(P_2) = (j_1, j_3)$ and $\sigma_r(P_1) = \sigma_r(P_2)$ for all $r \notin \{p, \bar{q}\}$. Since P_1 and P_2 satisfy (5.31) we have $\alpha_{p,j_3} - \alpha_{p,j_2} = \beta_{\bar{q},j_3} - \beta_{\bar{q},j_2}$. □

Proof of statement (xi). Let $p = 1, \dots, t$, $\bar{q} = t+1, \dots, n$ and $1 \le j_1 < j_2 \le 2k < j_3 < j_4 \le 2n$. Consider points $P_1, P_2 \in S_{2n}^{parity}$ such that $\sigma_p(P_1) = (j_1, j_2)$, $\sigma_{\bar{q}}(P_1) = (j_3, j_4)$ and $\sigma_p(P_2) = (j_1, j_3)$, $\sigma_{\bar{q}}(P_2) = (j_2, j_4)$ and $\sigma_r(P_1) = \sigma_r(P_2)$ for all $r \notin \{p, \bar{q}\}$. Since P_1 and P_2 satisfy (5.31) we have $\beta_{p,j_3} - \beta_{p,j_2} = \alpha_{\bar{q},j_3} - \alpha_{\bar{q},j_2}$. □

Proof of statement (xii). Let $p = 1, \dots, t$, $\bar{q} = t+1, \dots, n$ and $1 \le j_1 \le 2k < j_2 < j_3 < j_4 \le 2n$. Consider points $P_1, P_2 \in S_{2n}^{parity}$ such that $\sigma_p(P_1) = (j_1, j_2)$, $\sigma_{\bar{q}}(P_1) = (j_3, j_4)$ and $\sigma_p(P_2) = (j_1, j_3)$, $\sigma_{\bar{q}}(P_2) = (j_2, j_4)$ and $\sigma_r(P_1) = \sigma_r(P_2)$ for all $r \notin \{p, \bar{q}\}$. Since P_1 and P_2 satisfy (5.31) we have $\beta_{p,j_3} - \beta_{p,j_2} = \alpha_{\bar{q},j_3} - \alpha_{\bar{q},j_2}$. □

Bibliography

Abedinnia, H., Glock, C. H., Grosse, E. H., and Schneider, M. Machine scheduling problems in production: A tertiary study. *Computers & Industrial Engineering*, 111:403–416, 2017.

Aneja, Y. P., Aggarwal, V., and Nair, K. P. Shortest chain subject to side constraints. *Networks*, 13(2):295–302, 1983.

Árgilán, V., Balogh, J., Békési, J., Dávid, B., Krész, M., and Tóth, A. A flexible system for optimizing public transportation. In *Proceedings of the 8th International Conference on Applied Informatics*, volume 2, pages 181–190, 2010.

Avella, P., Boccia, M., and Sforza, A. Resource constrained shortest path problems in path planning for fleet management. *Journal of Mathematical Modelling and Algorithms*, 3(1):1–17, 2004.

Baldoquin, M. G. and Rengifo-Campo, A. J. A model for solving vehicle scheduling problems: a case study. *Revista Facultad de Ingeniería Universidad de Antioquia*, (88):16–25, 2018.

Ball, M., Bodin, L., and Dial, R. A matching based heuristic for scheduling mass transit crews and vehicles. *Transportation Science*, 17(1):4–31, 1983.

Beasley, J. E. and Christofides, N. An algorithm for the resource constrained shortest path problem. *Networks*, 19(4):379–394, 1989.

Békési, J., Brodnik, A., Krész, M., and Pash, D. An integrated framework for bus logistics management: Case studies. *Logistik Management*, pages 389–411, 2009.

Bertossi, A. A., Carraresi, P., and Gallo, G. On some matching problems arising in vehicle scheduling models. *Networks*, 17(3):271–281, 1987.

Borndörfer, R., Löbel, A., and Weider, S. A bundle method for integrated multi-depot vehicle and duty scheduling in public transit. In *Computer-Aided Systems in Public Transport*, pages 3–24. Springer, 2008.

Bunte, S. and Kliewer, N. An overview on vehicle scheduling models. *Public Transport*, 1(4): 299–317, 2009.

Chabrier, A. Vehicle routing problem with elementary shortest path based column generation. *Computers & Operations Research*, 33(10):2972–2990, 2006.

Coffman, E. G. and Graham, R. L. Optimal scheduling for two-processor systems. *Acta informatica*, 1(3):200–213, 1972.

CPLEX Optimizer. IBM. url:https://www.ibm.com/analytics/cplex-optimizer.

Dahl, G. and Gouveia, L. On the directed hop-constrained shortest path problem. *Operations Research Letters*, 32(1):15–22, 2004.

Dahl, G. and Realfsen, B. The cardinality-constrained shortest path problem in 2-graphs. *Networks*, 36(1):1–8, 2000.

Darby-Dowman, K., Jachnik, J., Lewis, R., and Mitra, G. Integrated decision support systems for urban transport scheduling: Discussion of implementation and experience. In *Computer-Aided Transit Scheduling*, pages 226–239. Springer, 1988.

Derigs, U., Friederichs, S., and Schäfer, S. A new approach for air cargo network planning. *Transportation Science*, 43(3):370–380, 2009.

Desrochers, M. and Soumis, F. A column generation approach to the urban transit crew scheduling problem. *Transportation science*, 23(1):1–13, 1989.

Dias, T. G., de Sousa, J. P., and Cunha, J. Genetic algorithms for the bus driver scheduling problem: a case study. *Journal of the Operational Research Society*, 53(3):324–335, 2002.

Dinur, I. and Safra, S. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, 162(1):439–485, 2005.

Dror, M. Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, 42(5):977–978, 1994.

Dumitrescu, I. and Boland, N. Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks*, 42(3):135–153, 2003.

Elimam, A. and Kohler, D. Two engineering applications of a constrained shortest-path model. *European Journal of Operational Research*, 103(3):426–438, 1997.

Ergun, F., Sinha, R., and Zhang, L. An improved FPTAS for restricted shortest path. *Information Processing Letters*, 83(5):287–291, 2002.

Falkner, J. and Ryan, D. Express: Set partitioning for bus crew scheduling in christchurch. In *Computer-Aided Transit Scheduling*, pages 359–378. Springer, 1992.

FICO Xpress Solver. Fair Isaac Corporation. url:www.fico.com/en/products/fico-xpress-optimization.

Fischetti, M., Martello, S., and Toth, P. The fixed job schedule problem with spread-time constraints. *Operations Research*, 35(6):849–858, 1987.

Fischetti, M., Martello, S., and Toth, P. The fixed job schedule problem with working-time constraints. *Operations Research*, 37(3):395–403, 1989.

Freling, R., Boender, C. G. E., and Paixão, J. M. P. An integrated approach to vehicle and crew scheduling. Technical Report 9503/A, Econometric Institute, Erasmus University Rotterdam, Rotterdam, The Netherlands, 1995.

Freling, R., Wagelmans, A. P., and Paixão, J. M. P. Models and algorithms for single-depot vehicle scheduling. *Transportation Science*, 35(2):165–180, 2001.

Freling, R., Huisman, D., and Wagelmans, A. P. Models and algorithms for integration of vehicle and crew scheduling. *Journal of Scheduling*, 6(1):63–85, 2003.

Friberg, C. and Haase, K. An exact branch and cut algorithm for the vehicle and crew scheduling problem. In *Computer-Aided Transit Scheduling*, pages 63–80. Springer, 1999.

Gaffi, A. and Nonato, M. An integrated approach to ex-urban crew and vehicle scheduling. In *Computer-Aided Transit Scheduling*, pages 103–128. Springer, 1999.

Gamache, M., Soumis, F., Marquis, G., and Desrosiers, J. A column generation approach for large-scale aircrew rostering problems. *Operations research*, 47(2):247–263, 1999.

Garcia, R. *Resource constrained shortest paths and extensions*. PhD thesis, Georgia Institute of Technology, 2009.

Garey, M. R. and Johnson, D. S. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman, 1979. isbn:0-7167-1044-7.

Garroppo, R. G., Giordano, S., and Tavanti, L. A survey on multi-constrained optimal path computation: Exact and approximate algorithms. *Computer Networks*, 54(17):3081–3107, 2010.

Gintner, V., Kliewer, N., and Suhl, L. A crew scheduling approach for public transit enhanced with aspects from vehicle scheduling. In *Computer-Aided Systems in Public Transport*, pages 25–42. Springer, 2008.

Goel, A., Ramakrishnan, K. G., Kataria, D., and Logothetis, D. Efficient computation of delay-sensitive routes from one source to all destinations. In *Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 854–858. IEEE, 2001.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.

Grandoni, F., Ravi, R., Singh, M., and Zenklusen, R. New approaches to multi-objective optimization. *Mathematical Programming*, 146(1-2):525–554, 2014.

Gurobi Optimizer. Gurobi optimization. url:https://www.gurobi.com/.

Haase, K., Desaulniers, G., and Desrosiers, J. Simultaneous vehicle and crew scheduling in urban mass transit systems. *Transportation Science*, 35(3):286–303, 2001.

Hassin, R. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations research*, 17(1):36–42, 1992.

Hastad, J. Clique is hard to approximate within $n^{1-\varepsilon}$. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 627–636. IEEE, 1996.

Hoogeveen, H., Schuurman, P., and Woeginger, G. J. Non-approximability results for scheduling problems with minsum criteria. *INFORMS Journal on Computing*, 13(2):157–168, 2001.

Horváth, M. and Kis, T. Solving resource constrained shortest path problems with LP-based methods. *Computers & Operations Research*, 73:150–164, 2016a. doi:10.1016/j.cor.2016.04.013.

Horváth, M. and Kis, T. Supplementary material to the paper "Solving resource constrained shortest path problems with LP-based methods", 2016b. url:dx.doi.org/10.1016/j.cor.2016.04.013.

Horváth, M. and Kis, T. Computing strong lower and upper bounds for the integrated multiple-depot vehicle and crew scheduling problem with branch-and-price. *Central European Journal of Operations Research*, 27(1):39–67, 2019a. doi:10.1007/s10100-017-0489-4.

Horváth, M. and Kis, T. Polyhedral results for position-based scheduling of chains on a single machine. *Annals of Operations Research*, pages 1–40, 2019b. doi:10.1007/s10479-019-03180-8.

Hu, T. C. Parallel sequencing and assembly line problems. *Operations research*, 9(6):841–848, 1961.

Huisman, D. Random data instances for multiple-depot vehicle and crew scheduling, 2003. url:http://people.few.eur.nl/huisman/instances.htm.

Huisman, D. *Integrated and Dynamic Vehicle and Crew Scheduling*. PhD thesis, Erasmus School of Economics (ESE), 2004.

Huisman, D., Freling, R., and Wagelmans, A. P. A robust solution approach to the dynamic vehicle scheduling problem. *Transportation Science*, 38(4):447–458, 2004.

Huisman, D., Freling, R., and Wagelmans, A. P. Multiple-depot integrated vehicle and crew scheduling. *Transportation Science*, 39(4):491–502, 2005.

Jepsen, M., Petersen, B., and Spoorendonk, S. A branch-and-cut algorithm for the elementary shortest path problem with a capacity constraint. Technical Report 08/01, Department of Computer Science, University of Copenhagen, 2008.

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

Johannes, B. On the complexity of scheduling unit-time jobs with OR-precedence constraints. *Operations research letters*, 33(6):587–596, 2005.

Kliewer, N., Mellouli, T., and Suhl, L. A time–space network based exact optimization model for multi-depot bus scheduling. *European Journal of Operational Research*, 175(3):1616–1627, 2006.

Kulkarni, J. and Munagala, K. Algorithms for cost-aware scheduling. In *WAOA 2012: Approximation and Online Algorithms*, pages 201–214. Springer, 2012.

Lancia, G. and Serafini, P. *Compact Extended Linear Programming Models*. Springer, 2018. isbn:978-3-319-63975-8.

Laporte, G., Pascoal, M., et al. The pipeline and valve location problem. *European Journal of Industrial Engineering*, 6(3):301–321, 2012.

Laurent, B. and Hao, J.-K. Simultaneous vehicle and crew scheduling for extra urban transports. In *New Frontiers in Applied Artificial Intelligence*, pages 466–475. Springer, 2008.

Lawler, E. L. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. In *Annals of Discrete Mathematics*, volume 2, pages 75–90. Elsevier, 1978.

Lenstra, J. K. and Rinnooy Kan, A. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, 1978.

Lenstra, J. K. and Rinnooy Kan, A. Complexity results for scheduling chains on a single machine. *European Journal of Operational Research*, 4(4):270–275, 1980.

Leung, J. Y.-T. and Young, G. H. Minimizing total tardiness on a single machine with precedence constraints. *ORSA Journal on Computing*, 2(4):346–352, 1990.

Library for Efficient Modeling and Optimization in Networks. Egerváry Research Group on Combinatorial Optimization. url:https://lemon.cs.elte.hu/trac/lemon.

Lorenz, D. H. and Raz, D. A simple efficient approximation scheme for the restricted shortest path problem. *Operations Research Letters*, 28(5):213–219, 2001.

Lozano, L. and Medaglia, A. L. On an exact method for the constrained shortest path problem. *Computers & Operations Research*, 40(1):378–384, 2013.

Mesquita, M. and Paias, A. Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem. *Computers & Operations Research*, 35(5):1562–1575, 2008.

Mesquita, M., Paias, A., and Respício, A. Branching approaches for integrated vehicle and crew scheduling. *Public Transport*, 1(1):21–37, 2009.

Möhring, R. H., Skutella, M., and Stork, F. Scheduling with AND/OR precedence constraints. *SIAM Journal on Computing*, 33(2):393–415, 2004.

Motzkin, T. The assignment problem. In Curtiss, J. H., editor, *Proceedings of Symposia in Applied Mathematics*, volume 6, pages 109–125, 1956.

Papadimitriou, C. H. and Yannakakis, M. On the approximability of trade-offs and optimal access of web sources. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 86–92. IEEE, 2000.

Patrikalakis, I. and Xerocostas, D. A new decomposition scheme of the urban public transport scheduling problem. In *Computer-Aided Transit Scheduling*, pages 407–425. Springer, 1992.

Pepin, A., Desaulniers, G., Hertz, A., and Huisman, D. Comparison of heuristic approaches for the multiple depot vehicle scheduling problem. Technical Report EI 2006-34, Econometric Institute, Erasmus University Rotterdam, Nov. 2006. url:http://hdl.handle.net/1765/8069.

Pugliese, L. D. P. and Guerriero, F. A reference point approach for the resource constrained shortest path problems. *Transportation Science*, 47(2):247–265, 2013a.

Pugliese, L. D. P. and Guerriero, F. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, 62(3):183–200, 2013b.

Ryan, D. M. and Foster, B. A. An integer programming approach to scheduling. In *Computer Scheduling of Public Transportation: Urban Passenger Vehicle and Crew Scheduling*, pages 269–280. North-Holland, Amsterdam, 1981.

Santos, L., Coutinho-Rodrigues, J., and Current, J. R. An improved solution algorithm for the constrained shortest path problem. *Transportation Research Part B: Methodological*, 41(7): 756–771, 2007.

SCIP Optimization Suite. Zuse Institute Berlin. url:https://scip.zib.de.

Scott, D. A large scale linear programming approach to the public transport scheduling and costing problem. In *Computer Scheduling of Public Transport 2*, pages 473–491. Elsevier, 1985.

Song, M. and Sahni, S. Approximation algorithms for multiconstrained quality-of-service routing. *IEEE Transactions on computers*, 55(5):603–617, 2006.

Steinzen, I. *Topics in integrated vehicle and crew scheduling in public transit*. PhD thesis, University of Paderborn, 2007.

Steinzen, I., Gintner, V., Suhl, L., and Kliewer, N. A time-space network approach for the integrated vehicle-and crew-scheduling problem with multiple depots. *Transportation Science*, 44(3):367–382, 2010.

Timkovsky, V. G. Identical parallel machines vs. unit-time shops and preemptions vs. chains in scheduling complexity. *European Journal of Operational Research*, 149(2):355–376, 2003.

Tosini, E. and Vercellis, C. An interactive system for extra-urban vehicle and crew scheduling problems. In *Computer-Aided Transit Scheduling*, pages 41–53. Springer, 1988.

Ullman, J. D. NP-complete scheduling problems. *Journal of Computer and System sciences*, 10 (3):384–393, 1975.

Wan, G. and Qi, X. Scheduling with variable time slot costs. *Naval Research Logistics*, 57(2): 159–171, 2010.

Wilhelm, W. E., Damodaran, P., and Li, J. Prescribing the content and timing of product upgrades. *IIE Transactions*, 35(7):647–663, 2003.

Wilhelm, W. E., Choudhry, N. D., and Damodaran, P. A model to optimize placement operations on dual-head placement machines. *Discrete Optimization*, 4(2):232–256, 2007.

Zabarankin, M., Uryasev, S., and Pardalos, P. Optimal risk path algorithms. In *Cooperative control and optimization*, pages 273–298. Springer, 2002.

Zhao, Y., Qi, X., and Li, M. On scheduling with non-increasing time slot cost to minimize total weighted completion time. *Journal of Scheduling*, 19(6):759–767, 2016.

Summary

In this thesis we investigate three combinatorial optimization problems with great practical relevance.

Resource constrained shortest path problem

First, we investigate multi-criteria approximation algorithms for the resource constrained shortest path problem (RCSPP).

- We show that there is no polynomial time multi-criteria approximation scheme for a set of k-budgeted combinatorial optimization problems (including RCSPP), if the number of weight functions is not a constant (i.e., part of the input), unless P = NP.
- We show that there is a fully polynomial time $(1; 1+\varepsilon, \ldots, 1+\varepsilon)$-approximation scheme for RCSPP, if the number of weight functions is a constant (i.e., not part of the input).

Then, we create linear programming based branch-and-bound solution methods for RCSPP.

- We propose efficient primal heuristics and variable fixing procedures.
- We generalize two classes of valid inequalities for the polytope of feasible solutions. For the first class we provide a polynomial time exact separation procedure. For the second class we prove that separating those inequalities (both the original and generalized versions) is NP-hard, however, we provide a heuristic separation procedure.
- We make thorough computational experiments to evaluate the efficiency of these components, and to compare our solution methods with state-of-the-art methods.

Position-based scheduling on a single machine

- We show that problem $1 \mid 2\text{-}chains, p_j = 1 \mid \sum w_{j,\sigma_j}$ and thus problem $1 \mid prec, p_j = 1 \mid \sum w_{j,\sigma_j}$ is strongly NP-hard.
- We propose a class of valid inequalities for the polytope associated with the feasible solutions of problem $1 \mid chains, p_j = 1 \mid \sum w_{j,\sigma_j}$. We show that a subclass of these inequalities are facet-defining in the case of two-chains.
- We make thorough computational experiments to show that our cutting planes can improve a linear programming based branch-and-bound procedure.

Multiple-depot integrated vehicle and crew scheduling problem

- We propose a novel mixed-integer linear programming formulation for the multiple-depot integrated vehicle and crew scheduling problem (MDVCSP).
- We develop an exact branch-and-price procedure for the proposed formulation consisting of an efficient variable pricing procedure, several problem-tailored branching strategies, and a simple primal heuristics.
- We make computational experiments to evaluate our method, and to compare it with other solution approaches from the literature.

Összefoglaló

A tézisben három, széles gyakorlati alkalmazással rendelkező kombinatorikus optimalizálási feladattal foglalkozunk.

Erőforrás-korlátos legrövidebb út feladat

Az erőforrás-korlátos legrövidebb út feladatra (RCSPP) több-kritériumú approximációs algoritmusokat vizsgálunk.

- Megmutatjuk, hogy nem létezik polinomiális idejű több-kritériumú approximációs séma több k-korlátos optimalizálási feladatra (köztük az RCSPP-re) sem, ha a súlyfüggvények száma nem konstans (feltéve, hogy P $\neq$ NP).
- Megmutatjuk, hogy ha a súlyfüggvények száma konstans (azaz nem része az inputnak), akkor létezik teljesen polinomiális idejű $(1; 1+\varepsilon, \ldots, 1+\varepsilon)$-approximációs séma az RCSPP-re.

Lineáris programozás alapú korlátozás-és-szétválasztás eljárásokat készítünk az RCSPP-re.

- Hatékony primál heurisztikát és változó rögzítési eljárást készítünk.
- A megengedett megoldások politópjára a szakirodalomban ismert vágósíkok két osztályát általánosítjuk. Megmutatjuk, hogy a második osztálybeli (mind az eredeti, mind az általánosított) egyenlőtlenségek szeparálása NP-nehéz. Az első osztálybeli egyenlőtlenségekre egzakt, a második osztálybeliekre heurisztikus szeparációs eljárást adunk.
- Átfogó számítógépes kísérleteket végzünk, melyben az említett komponenseket kiértékeljük, és módszerünket különböző state-of-the-art eljárásokkal összehasonlítjuk.

Pozíció alapú egy-gépes ütemezési feladat

- Megmutatjuk, hogy a $1\,|\,2\text{-}chains, p_j = 1\,|\,\sum w_{j,\sigma_j}$ probléma, és ezért a $1\,|\,prec, p_j = 1\,|\,\sum w_{j,\sigma_j}$ probléma erősen NP-nehéz.
- Érvényes egyenlőtlenségek egy osztályát adjuk a $1\,|\,chains, p_j = 1\,|\,\sum w_{j,\sigma_j}$ probléma megengedett megoldásainak politópjára. Megmutatjuk, hogy ezen egyenlőtlenségek egy rész kettő-láncok esetében a megfelelő politóp lapjait definiálják.
- Számítógépes kísérleteket végzünk, hogy igazoljuk, vágósíkjaink használata növelni tudja egy lineáris programozás alapú korlátozás-és-szétválasztás eljárás hatékonyságát.

Több-depós integrált jármű és vezető ütemezési feladat

- Egy új vegyes programozási felírást adunk a több-depós integrált jármű és vezető ütemezési feladatra (MDVCSP).
- Egy egzakt szétválasztás-és-árazás eljárást készítünk a feladat megoldására, mely tartalmaz egy hatékony oszlopgenerálás eljárást, több probléma-specifikus szétválasztási stratégiát, és egy egyszerű primál heurisztikát.
- Számítógépes kísérleteinkben kiértékeljük saját módszerünket, melyet utána egy, a szakirodalomban fellelhető megközelítéssel hasonlítunk össze.

ADATLAP
a doktori értekezés nyilvánosságra hozatalához*

I. A doktori értekezés adatai

A szerző neve: Horváth Markó
MTMT-azonosító: 10054564
A doktori értekezés címe és alcíme: Integer programming approaches for solving routing and scheduling problems
DOI-azonosító: 10.15476/ELTE.2019.241
A doktori iskola neve: Matematika Doktori Iskola
A doktori iskolán belüli doktori program neve: Alkalmazott matematika
A témavezető neve és tudományos fokozata: Kis Tamás, MTA doktora
A témavezető munkahelye: SZTAKI

II. Nyilatkozatok

1. A doktori értekezés szerzőjeként

a) hozzájárulok, hogy a doktori fokozat megszerzését követően a doktori értekezésem és a tézisek nyilvánosságra kerüljenek az ELTE Digitális Intézményi Tudástárban. Felhatalmazom a Természettudományi kar Dékáni Hivatal Doktori, Habilitációs és Nemzetközi Ügyek Csoportjának ügyintézőjét, hogy az értekezést és a téziseket feltöltse az ELTE Digitális Intézményi Tudástárba, és ennek során kitöltse a feltöltéshez szükséges nyilatkozatokat.

b) kérem, hogy a mellékelt kérelemben részletezett szabadalmi, illetőleg oltalmi bejelentés közzétételéig a doktori értekezést ne bocsássák nyilvánosságra az Egyetemi Könyvtárban és az ELTE Digitális Intézményi Tudástárban;

c) kérem, hogy a nemzetbiztonsági okból minősített adatot tartalmazó doktori értekezést a minősítés (dátum)-ig tartó időtartama alatt ne bocsássák nyilvánosságra az Egyetemi Könyvtárban és az ELTE Digitális Intézményi Tudástárban;

d) kérem, hogy a m? kiadására vonatkozó mellékelt kiadó szerződésre tekintettel a doktori értekezést a könyv megjelenéséig ne bocsássák nyilvánosságra az Egyetemi Könyvtárban, és az ELTE Digitális Intézményi Tudástárban csak a könyv bibliográfiai adatait tegyék közzé. Ha a könyv a fokozatszerzést követőn egy évig nem jelenik meg, hozzájárulok, hogy a doktori értekezésem és a tézisek nyilvánosságra kerüljenek az Egyetemi Könyvtárban és az ELTE Digitális Intézményi Tudástárban.

2. A doktori értekezés szerzőjeként kijelentem, hogy

a) az ELTE Digitális Intézményi Tudástárba feltöltendő doktori értekezés és a tézisek saját eredeti, önálló szellemi munkám és legjobb tudomásom szerint nem sértem vele senki szerzői jogait;

b) a doktori értekezés és a tézisek nyomtatott változatai és az elektronikus adathordozón benyújtott tartalmak (szöveg és ábrák) mindenben megegyeznek.

3. A doktori értekezés szerzőjeként hozzájárulok a doktori értekezés és a tézisek szövegének plágiumkereső adatbázisba helyezéséhez és plágiumellenőrző vizsgálatok lefuttatásához.

Kelt: Budapest, 2019. október 31.

a doktori értekezés szerzőjének aláírása

*ELTE SZMSZ SZMR 12. sz. melléklet