Irving van Heuven van Staereling

SCHEDULING PROBLEMS IN THE SERVICE INDUSTRY

THEORY AND APPLICATIONS

Irving van Heuven van Staereling

Scheduling Problems in the Service Industry

Scheduling Problems in the Service Industry: Theory and Applications

Irving van Heuven van Staereling

VRIJE UNIVERSITEIT

Scheduling Problems in the Service Industry
Theory and Applications

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. J.J.G. Geurts,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de School of Business and Economics
op woensdag 20 september 2023 om 15.45 uur
in een bijeenkomst van de universiteit,
De Boelelaan 1105

door

Irving Ian van Heuven van Staereling

geboren te Amsterdam

promotoren:	prof.dr. G. Schäfer prof.dr. R.D. van der Mei
promotiecommissie:	prof.dr. L. Stougie prof.dr. D. Huisman prof.dr. E.W. Hans prof.dr. J.L. Hurink dr. J.M. van den Akker

Acknowledgements

The completion of this PhD has been a challenging, but engaging journey, with many highlights and setbacks along the way. During this period, I have received an enormous amount of support and help from many others, whom I would like to thank personally.

First and foremost, I would like to express my sincere gratitude to my daily supervisor and promotor Guido Schäfer for his constructive feedback, unwavering support and invaluable patience throughout all the years. Words cannot describe my appreciation for all the efforts he has done for me and the opportunities he has given me. Starting from my first internship as a student until the very end of my PhD, he always was ready to spend time to help with finding results and solutions, for which I cannot thank enough for. Moreover, I am very grateful to my second promotor, Rob van der Mei, for the invaluable guidance and encouragement he has given me, especially during the final stage of my PhD. I truly appreciate everything that my promotors have done for me, and I hope I can do something significant in return in the future.

In addition, I would like to thank my co-authors for their contributions to the research that form the basis of this thesis. I am very thankful to have worked with René Bekker, whom I met on the first day of my academic career and who has guided me very well throughout my bachelor, master and PhD. Many thanks also go to Bart de Keijzer, one of the smartest and most dedicated researchers I have ever met, for his help and guidance on writing the more theoretical part of this thesis. Learning and working with both of them has been a very pleasant experience and I hope to come across opportunities to work with them again.

Furthermore, I would like to thank the reading committee consisting of Marjan van den Akker, Erwin Hans, Dennis Huisman, Johann Hurink and Leen Stougie. I highly appreciate the honest feedback and advice that this panel of experts has provided. They are all great researchers and teachers from whom I learned a lot, during both my studies and PhD, so I feel honored to be able to exchange thoughts with them about my research during the defense of my thesis.

I consider myself very lucky to have been able to conduct my research at CWI and VU University, and I want give thanks to all the fantastic colleagues that I have been able to spend time with during my stay. In particular, I would like to thank Asparuh, Bart, Carla, Daniel, David, Dylan, Elenna, Ewan, Jan-Pieter, Joost, Krzysztof, Lex, Matteo, Monique, Neil, Pieter, Sander, Sandjai, Sophie, Susanne and Teresa. It was a pleasure and privilege to be surrounded by such dedicated and smart researchers, who all have been a source of inspiration to me in their own way.

Moreover, special thanks go towards the people from Rovecom, for making my PhD possible and providing numerous interesting topics for my research. Working with George, Gerben, Jeroen and Jurriën has been a great pleasure and closing the bridge between theory and practice has been an engaging experience.

Finally, I would like to express my profound gratitude to my friends and family, especially to my parents Marcel and Julianti, for always being there for me throughout my years of studying, researching and writing this thesis. Their support has been a continuous source of motivation to keep going and this accomplishment would not have been possible without them. Thank you!

Irving van Heuven van Staereling
Amsterdam, September 2023

Contents

Chapter 1

Introduction

Scheduling problems belong to the classical optimization problems in Operations Research due to their practical significance and theoretical elegance. This chapter gives an introduction to basic scheduling concepts and their applicability, and describes how the contributions in this thesis fit within the research field of scheduling.

After illustrating the background of scheduling in Section 1.1, the basics of formulating and solving scheduling problems are given in Section 1.2. Subsequently, the goals and research questions of this thesis will be elaborated upon in Section 1.3. Finally, an outline including an overview of the contributions of this thesis is given in Section 1.4.

1.1 Background

Scheduling is a type of decision making that is used on a regular basis by practically all organizations and individuals. Generally speaking, scheduling problems deal with the allocation of *resources* to *tasks*, usually over a given *period of time*, with the goal to optimize one or more *objectives*, subject to a set of *constraints*. Typical large-scale processes where scheduling is of vital importance are production, transportation, distribution, construction, information processing and communication.

Within scheduling applications, there is a large variety in tasks, resources and objectives. Common real-life examples of tasks include assembling product parts, driving trips, or completing stages of a construction project. Examples of the corresponding resources are the machines in a factory, tracks at a train station, or workers available for the construction site for the given time period. The objectives for an organization often include the maximization of the profit or the minimization of the total completion time. However, large organizations often deal with multiple objectives. This can be represented in a multi-objective function, or combined in one objective function by assigning weights to every objective, representing their relative importance. Other special objectives include the environment (pollution minimization), safety (avoiding trucks in crowded areas) and even may have a political nature (equalize the benefit from a service in different areas). Alternatively, objectives can also be transformed into a constraint (e.g., at least 95% of the ambulances must arrive at location within 15 minutes after an emergency is recorded).

1.2 Scheduling problems

Many scheduling problems can be formulated as a *combinatorial optimization* problem. These types of problems are concerned with finding an optimal solution within a large, but finite set of solutions. To illustrate this, three fundamental combinatorial optimization problems and their practical relevance are described informally below. A formal definition of these problems can be found in Appendix B.

Example 1.1 (Assignment Problem)**.** Suppose a group of workers need to perform a set of tasks. Any worker can be assigned to any job, but every job-worker combination may have a unique cost. The goal is to assign exactly one job to every worker such that the total cost is minimized. In practice, the costs could be determined by the time that is required for a job by a worker, which may depend on e.g., the skills of every individual worker.

Example 1.2 (Vehicle Routing Problem)**.** A transportation company has a fleet of vehicles that can be used to serve a given set of customers. The goal is to visit all customers while minimizing the total distance (or time) of the routes of the vehicles. The practical context is often that goods or packages from a central depot have to be delivered to customers.

Example 1.3 (Machine Scheduling Problem)**.** A fixed number of machines is available to process a finite set of jobs with varying processing times, but every machine can process only one job at the same time. The goal is to minimize the total length of the schedule, i.e., the time when all jobs are finished. Applications of this problem include the manufacturing of products in a factory, but can also be translated to a context where jobs and machines are represented by different entities (e.g., customers and workers, respectively).

These typical optimization problems are formulated in a basic version, but many variations and extensions exist. The problem in Example 1.1 could be extended with sizes for every job and capacities for each worker. Workers are then allowed to take more jobs as long as the total size does not exceed the worker's capacity. In Example 1.2, one could include time windows on the delivery moments that need to be fulfilled. Most notably, machine scheduling problems such as in Example 1.3 have been the subject of extensive research for many decades due to their application in many fields. These problems are also used to introduce new students and researchers into scheduling, because it is easier to use standard combinatorial techniques and to analyze the computational complexity. Two major overviews on the theory of (machine) scheduling can be found in [10] and [99].

These three examples are only a limited sample of the wide range of optimization problems that can be found in practice and in the literature. Typically, scheduling problems play an important role in manufacturing and service industries, as well as some information processing environments. In this thesis, the focus lies on scheduling problems in the *service industry*. Although such problems generally have a higher complexity (e.g., more constraints), the practical impact of understanding and solving problems can be higher as well (in terms of saved costs and time).

Formulating a scheduling problem. Although scheduling problems in *theory* should have a clear input, objective and output, this is not always the case in *practice*. Formulating an optimization problem from a realistic setting to a mathematical model often has room for different interpretations or perspectives. Even the *objective function*, which is an indication of the quality of a schedule, might not be definable in a straightforward way.

For example, the classical Vehicle Routing Problem (as described in Example 1.2) and many of its variants have been studied very extensively, but a real-world transport organization usually cannot apply an existing method directly into practice. It is unlikely that the exact optimization problem of a transportation company is contained in the literature. In practice, there might be significantly more considerations that may need to be taken into account, such as:

- balancing the work load of drivers within a daily duty, but also on a weekly, monthly and even yearly basis,
- time windows in which deliveries can or should be done,
- stochasticity of driving and delivery times (including chances of disruptions, traffic jams, etc.),
- availability and skills of drivers with different contracts, and
- availability, equipment and sizes of vehicles with different driving costs.

In this (non-exhaustive) list, most considerations provide room for different interpretations. For example, when trying to balance the work load for a driver, it is not easy to define when one set of duties in a week for a driver is more "balanced" than another. Furthermore, one might question whether time windows are a hard or soft constraint, i.e., a requirement or a preference. In case of a preference, an optimization model needs to know what delay is acceptable, and whether there exists a penalty if the delay exceeds a certain amount of time. And even if all such performance measures were quantifiable, it may be difficult to formulate all practical considerations into a (mathematical) model, let alone solve it.

For such reasons, models in the literature are usually limited to only a subset of such extensions motivated by practice, with their own interpretation. The obvious downside is that this makes those models less interesting to use for some organizations, simply because some (company-specific) considerations could be missing that potentially form a hard constraint. Also, personal preferences of employees might not be able to be incorporated easily. This does not mean that all mathematical models in the literature are inapplicable in practice. Existing algorithms could potentially solve simplified versions of the problem. But the mentioned downsides could make automatic optimization for some organizations insufficient and/or too troublesome. After all, automated scheduling requires to formalize all resources and constraints into input data for an optimization model that also needs to be created and maintained.

The efforts that come with automatic optimization may urge some organizations to schedule manually instead, which might lead to a less efficient schedule. On the

other hand, manual scheduling is for other organizations impractical, because their problem has too many variables and constraints to solve by hand, or because the constraints are too complex that planners cannot even find a feasible or acceptable schedule.

Solving a scheduling problem. A scheduling problem is an optimization problem which can be solved by an *algorithm*. Informally, an algorithm is any well-defined procedure, typically implementable by a computer. An algorithm takes a set of values as input, and produces a specific set of values as output, such as a schedule.

Clearly, an optimization problem can be solved in multiple ways. Ideally, an algorithm can be found for which can be formally proven that it guarantees to find an optimal solution to the problem *efficiently*. An algorithm is considered efficient if its resource consumption (also known as its computational cost) is within an acceptable level, which will be formalized in Chapter 2.

However, this thesis mainly considers so-called NP-hard problems. These are difficult problems for which it is unlikely that an efficient algorithm exists (see Section 2.3 for formal definition). Therefore, alternative techniques are required. The established techniques that will be used to approach scheduling problems within this thesis, arise from different areas in Operations Research. An overview of these approaches, including a brief non-technical description, is given in alphabetical order below.

- *Approximation algorithms* (see, e.g., [122]) are efficient algorithms that provide a *provable* guarantee on the quality of its solution compared to an optimal solution of the same problem. In other words, the solution value corresponding to the solution produced by the algorithm lies within a specified and proved factor of the optimal solution value. The design and analysis of approximation algorithms usually involves a mathematical proof certifying the quality of the returned solutions in the worst case.

- *Branch and price* (see, e.g., [25]) is a method to solve combinatorial optimization problems that can be formulated as an Integer Linear Program (ILP), and is a hybrid of branch-and-bound and column generation. A branch-and-bound algorithm implicitly enumerates all possible solutions, applying pruning rules to subsets of solutions if those subsets cannot produce a better solution than the best solution found so far by the algorithm.

- *Dynamic programming* (see, e.g., [21]) is a method that breaks the problem down into subproblems that can be solved efficiently. It applies when the subproblems cover each other, i.e., when a subproblem of the main problem is a subproblem of a different (larger) subproblem of the main problem. A dynamic programming algorithm solves each subproblem only once in a bottom-up fashion and saves its solution in a table. This prevents the need to recompute the answer every time it solves every subproblem. Intuitively this means that optimal solutions of subproblems will be used recursively to find optimal solutions of slightly bigger subproblems, until the overall problem is solved.

- *Graph algorithms* (see, e.g., [21]) are the class of algorithms that are based on searching a graph. A graph is a commonly-used mathematical structure used to model pairwise relations between objects. The objects are referred to as vertices (or nodes), while any pair of vertices may be connected by an edge. The requirement is that the scheduling problem can be represented as a graph. A graph-searching algorithm may discover much about the structure of a graph.

- *Greedy algorithms* (see, e.g., [88]) are simpler and more efficient algorithms compared to the other methods in this list. Throughout the execution of the algorithm, such methods tend to make the choice that looks best at the moment. In other words, greedy algorithms make locally optimal choices, hoping that these choices eventually lead to a globally optimal solution. However, this may result in worse results in the long run (i.e. local optima).

- *Heuristics* (see, e.g., [88]) are methods based on intuitive and/or rational rules that, in contrast to the earlier mentioned types of algorithms, do not satisfy formal or theoretical properties. This means that heuristics have no theoretical guarantee about its solution quality or its running time. Nevertheless, heuristics are often used in practice with great effectiveness as they usually require significantly less running time than algorithms that are guaranteed to find an optimal solution.

Note that there is no strict separation between the mentioned types of algorithms. For example, graph algorithms may include dynamic programming, while greedy algorithms are by many researchers considered as heuristics. Also note that this list is only a small subset of all possible optimization algorithms for scheduling problems. It only gives an impression of the algorithms that will be used in this thesis specifically. The choice for these algorithms is based on the high occurrence within the literature, the proven effectiveness of these methods for (scheduling) problems and/or the theoretical insights they provide.

These techniques will be explained more elaborately in the preliminaries in Chapter 2 with the aid of examples, before they will be applied in subsequent chapters.

1.3 Goals and research questions

This thesis focuses on both the theory and applications of scheduling and contributes new methodologies and insights in both aspects. With this aim, this thesis studies four difficult scheduling problems from different settings: job covering, crew scheduling, hospital planning and train timetabling. The selection of these problems is based on:

- the insights that the corresponding models provide,

- the variety of methodologies that are needed for their theoretical analyses, and

- their relevance and importance with respect to real-world applications.

These selected areas are by no means an exhaustive list of scheduling areas. However, there is often a large overlap with problems from other scheduling areas (e.g., machine scheduling, airline routing, sports timetabling), meaning that insights gained for these problems may be extended to other (scheduling) areas. With these scheduling settings used as a framework, the following goals and research questions are considered in this thesis.

Goals. The primary goal of this thesis is to provide new perspectives, models and algorithms to solve scheduling problems. For this reason, a different approach will be proposed for each considered scheduling problem. The proposed methods usually consist of an adaptation or extension of well-known approaches and should give additional insights on the possibilities of the existing methods. Note that the goal is not necessarily to design an algorithm which gives the best solution, has the shortest computation time or has the best theoretical guarantee. Instead, the focus lies on contributing new insights to approach scheduling problems.

To obtain a better understanding of the complexity of scheduling problems, one can identify the types of constraints or structures that can make a scheduling problem hard to solve (efficiently). This forms as a secondary goal. To achieve this goal, special cases of the scheduling problem (where one or more constraints are added or omitted) will be considered for which an efficient solution method can be proposed. Note that there is a direct relation between the two goals. If an efficient algorithm for a special case of the scheduling problem can be found (secondary goal), one can consider this algorithm as a subroutine by solving a subproblem, to provide a method to solve the general scheduling problem (primary goal). This will be done for all four scheduling problems.

Finally, it is important to note that this thesis is not built up in such a way that the four problems and corresponding chapters are directly connected to each other. The research in the four chapters have different motivations and can be read independently from each other and may have their own approaches and conclusions. If possible, comparisons will be made across chapters in the conclusions, but this is not the intention of this thesis.

Research questions. To achieve the mentioned goals, the following research questions are formulated that provide a basis for every of the four chapters:

- Why is the scheduling problem difficult to solve optimally?
- Which special cases of the scheduling problem can be solved efficiently?
- How can algorithms for special cases be used to solve the general scheduling problem?
- How applicable are these algorithms in practice with regard to their computation time?

The aim is to answer these questions per scheduling problem as satisfactorily as possible in the conclusions of every chapter. As mentioned, such insights may also help in areas other than the ones considered in this thesis.

1.4 Outline and contributions

This thesis is structured as follows.

- Chapter 2 describes important concepts from combinatorial optimization that introduce the reader to techniques that will be used in further chapters. These include the running time of algorithms, complexity classes, reductions among optimization problems, approximation algorithms, dynamic programming, fully polynomial time approximation schemes and column generation.

- Chapter 3 introduces the Budgeted Job Coverage Problem, where jobs with costs and profits need to be covered by specific resources (e.g., staff, machines) to maximize profit, subject to a budget constraint.

  The contributions consist of a proof of hardness and a variety of algorithms for special cases of the problem. A $\frac{1}{2}(1-\frac{1}{\sqrt{e}})$-approximation algorithm will be proposed which crucially exploits the case if every resource processes exactly two jobs. Also, a bi-level dynamic program and a fully polynomial time approximation scheme will be proposed in case the problem can be transformed to an acyclic graph. Following up on this analysis, some insights will be given in how to the decompose the problem into such acyclic versions, and the computation cost that comes with it.

  The work in this chapter is based on: Irving I. van Heuven van Staereling, Bart de Keijzer, and Guido Schäfer. The ground-set-cost budgeted maximum coverage problem. In *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58, pages 50:1–50:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016.

- Chapter 4 introduces the European Crew Scheduling Problem, where trips need to be assigned to a crew of bus drivers, subject to resting and driving constraints for the bus drivers that are imposed by the European Union.

  The contributions consist of a proof of hardness, complexity analyses and algorithms for the problem where a subset of the constraints is considered. It will be shown that length bounds on duty times make the problem hard. Most importantly, a branch-and-price algorithm will be proposed for the problem where the most important constraints will be taken into account. The used subproblem (or pricing problem) crucially exploits the time structure of the underlying graph, such that it can be solved in an efficient manner. Moreover, a heuristic will be used with experimental results based on realistic data as comparison.

  The work in this chapter is based on: Irving I. van Heuven van Staereling and Guido Schäfer. A branch-and-cut algorithm for driver scheduling under European regulations. *Manuscript*.

- Chapter 5 introduces the Room and Ward Planning Problem, where the planning of operating rooms and wards within a hospital need to be integrated,

while taking the uncertainty of procedure times, hospitalization times and urgent arrivals into account.

The contributions consist of a linearization method for a model which can take many important practical constraints into account. This model succeeds to incorporate the mentioned uncertainties, while minimizing the weighted sum of important performance measures for operating rooms and wards, of which the combination could not have been found in the literature before. To this aim, an Integer Linear Program (ILP) will be proposed that applies linearization of constraints by crucially exploiting the structure of the model.

The work in this chapter is based on: Irving I. van Heuven van Staereling, René Bekker, and Cornelis P. Allaart. Stochastic scheduling techniques for integrated optimization of catheterization laboratories. In *Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling*, pages 313–329, 2018.

- Chapter 6 considers the Periodic Event Scheduling Problem, where a set of events need to be scheduled within a periodic timetable subject to a set of constraints.

  The contributions consist of a dynamic program for the problem in case the underlying graph is acyclic (i.e., a tree). Additionally, every variable may be bounded to a given subset of values. This provides the basis for the main contribution, which is a heuristic that decomposes the instance into trees that can be solved efficiently using the dynamic program. To find a composition of trees that can be unified into a feasible, high-quality solution, an intuitive enumeration method using greedy mechanics will be proposed and applied on publicly available data instances.

  The work in this chapter is based on: Irving I. van Heuven van Staereling. Tree decomposition methods for the periodic event scheduling problem. In *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018)*, volume 65, pages 6:1–6:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018.

Chapters 3 to 6 all consider different scheduling problems, but are structured similarly. First, the practical background, motivation and relevance will be illustrated. Afterwards, the problem will be defined formally including different models and illustrations, followed by a discussion of the related work (including classical results) regarding the defined scheduling problem. This gives a better idea of the contributions of this thesis that will be made explicit afterwards. Also, a proof of hardness will be given in every chapter.

The chapters are sorted from most theoretical (Chapter 3) to most practical (Chapter 6) and all chapters can be read independently from each other, with possibly requiring preliminary knowledge discussed in Chapter 2. Chapters 4 to 6 also include computational experiments and results that are all based on online benchmarks, real-world datasets and/or extensive case studies.

Acronyms & notation. Throughout this thesis, a lot of acronyms and notation are used. Even though all notation will be defined before they are used, a summary of all used acronyms and notation is given in Appendix A as a reference for the reader. Also, a wide variety of combinatorial optimization problems is used in this thesis, for which an overview can be found in Appendix B.

Chapter 2

Preliminaries

This chapter will explain several basic concepts within combinatorial optimization that will be used throughout this thesis.

After a short introduction to combinatorial optimization problems in Section 2.1, the running time of algorithms is formalized in Section 2.2. In Section 2.3, complexity classes such as P and NP will be discussed after which the concept of reducability among problems is explained in Section 2.4. Subsequently, techniques to handle difficult optimization problems will be discussed, including approximation algorithms in Section 2.5, dynamic programming in Section 2.6, fully polynomial time approximation schemes in Section 2.7 and column generation in Section 2.8.

2.1 Optimization vs. decision problems

Many theoretical and practical optimization problems can be formulated as a combinatorial optimization problem. Informally, such problems ask to find the best solution among a typically very large (but finite) set of feasible solutions [105]. Formally, such problems are defined as follows:

Definition 2.1. *A **combinatorial optimization problem** $\Pi = (\mathcal{I}, F, m, g)$ is defined by:*

- *a set of instances $\mathcal{I}$,*
- *a finite set of feasible solutions $F(I)$, for any instance $I \in \mathcal{I}$,*
- *a measure or objective function $m(I, \sigma)$, for every feasible solution $\sigma \in F(I)$, and*
- *a goal function g, either* min *or* max.

*The goal of the problem Π is to find an **optimal solution** σ^* for every instance $I \in \mathcal{I}$, i.e., a feasible solution for which $m(I, \sigma^*) = g\{m(I, \sigma) \mid \sigma \in F(I)\}$.*

Note that every combinatorial optimization problem may have an infinite number of instances, but every instance has a finite number of feasible solutions, of which

at least one is optimal. In further sections, the optimal solution value $m(I, \sigma^*)$ is for simplicity also referred to as $OPT(I)$.

An example of a combinatorial optimization problem is the Knapsack Problem, which will be used throughout this chapter to illustrate more basic combinatorial optimization concepts. The Knapsack Problem is defined as follows:

KNAPSACK PROBLEM	
Given:	A set of n items with corresponding weights $w_i > 0$ and profits $p_i > 0$ for $i \in [n]$ and a knapsack capacity $B > 0$.
Goal:	Find a subset of items $\sigma \subseteq [n]$ that maximizes $p(\sigma) = \sum_{i \in \sigma} p_i$ such that $w(\sigma) = \sum_{i \in \sigma} w_i \leq B$.

Here, $[n]$ is used to denote the set $\{1, \ldots, n\}$. To fill in the formalities of a combinatorial optimization problem given in Definition 2.1 for the Knapsack Problem:

- $\mathcal{I} = \{(n, w, p, B) \mid n \in \mathbb{N}; w_i, p_i > 0 \text{ for } i = 1, \ldots, n; B > 0\}$,
- $F(I) = \{\sigma \subseteq [n] \mid w(\sigma) \leq B\}$,
- $m(I, \sigma) = p(\sigma)$, and
- $g = \max$.

Example 2.1. Consider an instance $I \in \mathcal{I}$ of the Knapsack Problem where $n = 5$, $w = (2, 3, 4, 5, 6)$, $p = (1, 4, 5, 7, 8)$ and $B = 7$. Then the unique optimal solution is the subset of items $\sigma = \{2, 3\} \subseteq [n]$, with weight $w_2 + w_3 = 7 \leq B$ and profit $p_2 + p_3 = 9$.

Moreover, every combinatorial optimization problem $\Pi = (\mathcal{I}, F, m, g)$ has a corresponding decision problem Π' which is given by an additional threshold $K \in \mathbb{R}$.

Definition 2.2. *The **decision variant** of an optimization problem $\Pi = (\mathcal{I}, F, m, g)$ can be defined as $\Pi' = (\mathcal{I}, F, m, g, K)$ and asks the question whether there exists a solution $\sigma \in F(I)$ for instance $I \in \mathcal{I}$ for which:*

- $m(I, \sigma) \leq K$ *if* $g = \min$, *or*
- $m(I, \sigma) \geq K$ *if* $g = \max$.

If such a solution exists, then I is called a Yes-instance. Otherwise, I is called a No-instance.

For example, the goal of the decision variant of the Knapsack Problem is simply to determine whether there exists a feasible solution $\sigma \subseteq [n]$ for which $\sum_{i \in \sigma} p_i \geq K$. The instance in Example 2.1 is a Yes-instance for $K = 8$, but a No-instance for $K = 10$. Also note that there can be decision problems that do not necessarily have an optimization counterpart (see, e.g., 3-Partition in Appendix B).

This means in general that solving a decision problem does not necessarily require to find an optimal solution, and is in that sense easier than its optimization problem. In other words, an optimization problem can be seen as a generalization of its decision problem.

2.2 Running time of algorithms

Because the set of feasible solutions of any combinatorial optimization problem Π is finite, it is in theory often possible to solve any instance from Π. This is done by completely enumerating its set of feasible solutions and selecting the one with the best objective value. However, the number of feasible solutions is usually so large that such an enumeration would be too time-consuming for practical instances.

Fortunately, many problems can be solved by an algorithm that is more efficient than complete enumeration. To quantify the efficiency of an algorithm, it has to be noted that every algorithm conducts a sequence of specified actions, that can be broken down into *elementary operations*. Examples of such elementary operations include:

- variable assignments,
- simple arithmetic operations (addition, subtraction, multiplication, division, comparison of numbers),
- conditional jumps (if, then, else, go to), or
- access to variables whose index is stored elsewhere.

Using these concepts, one could determine the exact maximum number of elementary operations required for an algorithm to output a solution for a combinatorial optimization problem, but this may lead to overly complicated expressions and calculations. Instead, the Big-O notation is used to describe the rate of growth of an expression more conveniently.

Definition 2.3. *Let $f, g : D \to \mathbb{R}_+$ be two functions. If there exist two constants $\alpha, \beta > 0$ such that $f(x) \leq \alpha \cdot g(x) + \beta$ for every $x \in D$, then $f = \mathcal{O}(g)$.*

The input for an algorithm (an instance) usually consists of a list of numbers. It is well-known that computers encode natural numbers as a string of bits, i.e., $\mathbb{N} = \{1, 2, 3, 4, 5, \dots\}$ is encoded as $\{1, 10, 11, 100, \dots\}$. Thus, an integer $a \in \mathbb{N}$ can be encoded in binary representation using $\mathcal{O}(\log(|a|)$ bits. For rational numbers, the numerator and denominator can be encoded separately. Irrational numbers can never be fully encoded in binary notation, and therefore are usually encoded to the nearest rational number subject to a certain precision. These concepts suffice to quantify the running time of an algorithm on an instance.

Definition 2.4. *The **input size** $\text{size}(I)$ of an instance I is the total number of bits needed for its binary representation.*

In the following definitions in this section is assumed that an Algorithm $\mathcal{A}$ takes an instance $I \in \mathcal{I}$ from a combinatorial optimization problem $\Pi = (\mathcal{I}, F, m, g)$ as input.

Definition 2.5. *Let $\mathcal{A}$ be an algorithm and let $f : \mathbb{N} \to \mathbb{R}$. If there exist constants $\alpha, \beta > 0$ such that $\mathcal{A}$ terminates within at most $\alpha \cdot f(n) + \beta$ elementary steps for any $I \in \mathcal{I}$, then the **running time** (or **time complexity**) of $\mathcal{A}$ is $\mathcal{O}(f(n))$.*

Definition 2.6. *Let $\mathcal{A}$ be an algorithm with rational input. If there exists an integer k such that $\mathcal{A}$ runs in $\mathcal{O}(n^k)$ time, where n is the input size, and all intermediate computations can be stored in $\mathcal{O}(n^k)$ bits, then $\mathcal{A}$ runs in* ***polynomial time****. Also, $\mathcal{A}$ is called* ***efficient****.*

Within the polynomial-time algorithms, a distinction is made between two types.

Definition 2.7. *Let $\mathcal{A}$ be an algorithm with arbitrary input. If there exists an integer k such that $\mathcal{A}$ runs in $\mathcal{O}(n^k)$ time for any input consisting of n numbers and $\mathcal{A}$ runs in polynomial time for rational input, then Algorithm $\mathcal{A}$ runs in* ***strongly polynomial*** *time. If $\mathcal{A}$ runs in polynomial, but not strongly polynomial time, then $\mathcal{A}$ runs in* ***weakly polynomial*** *time.*

In other words, the running time of a strongly polynomial time algorithm depends solely on the number of input values. The running time of a weakly polynomial time algorithm might additionally take the size of the input values into account. Thus, a weakly polynomial time algorithm runs slower as the value of a specific number increases, while a strongly polynomial time algorithm would not. A clarifying example will be given further in this chapter.

2.3 P and NP

The idea that some problems can be solved more efficiently than others has provided the basis for the introduction of complexity classes for problems. To do this completely and precisely, one should actually provide introduce formal models (such as Turing machines), but this lies beyond the scope of this thesis. A more informal explanation is given instead. For formal details, the interested reader is referred to Chapter 15 of [71].

Complexity classes are defined with respect to decision problems. Several classes will be discussed in this chapter, for which some examples will be given further.

Definition 2.8. *A decision problem Π is in complexity class P is there exists an algorithm that can determine for every instance I of Π in polynomial time whether I is a Yes-instance or a No-instance.*

Problems in class P are generally considered tractable, as opposed to problems that require exponential time algorithms. Of course, it is theoretically possible that an algorithm running in $\mathcal{O}\left(1.001^n\right)$ requires fewer elementary operations than an algorithm running in $\mathcal{O}(n^{1000})$, even for realistic values of n. However, almost no practical problems require time of the order of such a low-based exponential or a high-degree polynomial. Moreover, experience has shown that even if the current best algorithm for a problem has a running time that includes such a high-degree polynomial, improvements can and will likely be discovered.

For the next complexity class, the concept of *verifying* a solution is used, which informally means whether a given solution to a problem can be checked for correctness (i.e., whether the solution is feasible). More formally, the following definition is used for this purpose.

Definition 2.9. *Given a Yes-instance $I \in \mathcal{I}$ of decision problem $\Pi = (\mathcal{I}, F, m, g, K)$, S is a* ***certificate*** *if $S \in F(I)$ and:*

- $m(I, \sigma) \leq K$ *if* $g = \min$*, or*
- $m(I, \sigma) \geq K$ *if* $g = \max$.

Note that every Yes-instance must have at least one certificate. However, not every certificate can be verified in polynomial time.

Definition 2.10. *A decision problem Π is in complexity class NP if every Yes-instance I of Π has a certificate that can be verified in polynomial time.*

For example, the Knapsack Problem is in NP. A certificate may simply be a solution $\sigma \subseteq [n]$ for which one can determine whether $\sum_{i \in \sigma} w_i \leq B$ and $\sum_{i \in \sigma} p_i \geq K$ in linear time. However, it is not directly clear whether it is in P. The most obvious way to solve this problem would be to enumerate over all 2^n solutions and keep track of the solution with maximum profit for which the weight does not exceed the capacity, but this algorithm does not run in polynomial time.

Note that every problem in P is also in NP, i.e., $P \subseteq NP$. After all, the algorithm for the problem in P can produce a certificate for which the validity has been verified throughout the execution of the algorithm in polynomial time. One of the biggest mathematical open questions is whether $NP \subseteq P$, which would imply $P = NP$. After all, there exist many problems that are in NP, but for which no polynomial time algorithm is known. Since many of these problems have a high practical relevance, the consequences could be substantial if it turns out these problems can be solved more efficiently.

However, it is currently generally accepted that the classes are not equal. When experts in 2019 have been asked the relationship between P and NP, 99% mentioned that they believe $P \neq NP$ [42]. Yet, a formal proof has never been given.

2.4 Polynomial-time reductions

The most common method to show that a decision problem Π lies within a specific complexity class, is by showing that another decision problem Π', for which it is known that it already lies in that specific complexity class, is at least as difficult as Π. This procedure is called a *reduction*.

Informally, a reduction from Π to Π' shows that if there exists an algorithm that can solve Π, it can be used to find an algorithm to solve Π'. Such a reduction is basically done by showing that every instance of Π is from a certain perspective simply a special case of Π'. More formally:

Definition 2.11. *A* ***polynomial-time reduction*** *from a decision problem Π to a decision problem Π' is a function $\phi : \mathcal{I} \to \mathcal{I}'$ that maps any instance $I \in \mathcal{I}$ of Π to an instance $I' \in \mathcal{I}'$ of Π' such that:*

- *I is a Yes-instance of Π if and only if I' is a Yes-instance of Π', and*

- $\phi(I)$ *can be computed within polynomial time of* size(I).

If such a polynomial-time reduction exists, then it is also said that Π *reduces to* Π'*, denoted by* $\Pi \propto \Pi'$.

An example of such a reduction is given below using the Partition Problem.

PARTITION PROBLEM	
Given:	A set of integers $a_1, \ldots, a_m$.
Goal:	Determine whether there exists a subset $\sigma \subseteq [m]$ such that $\sum_{i \in \sigma} a_i = \sum_{i \notin \sigma} a_i$.

Lemma 2.1. *Partition Problem* $\propto$ *Knapsack Problem*

Proof. Given an instance of the Partition Problem, construct the following instance for the decision variant of the Knapsack Problem:

- $n = m$,
- $p_i = w_i = a_i$ for $i = 1, \ldots, n$ and
- $B = K = \frac{1}{2} \sum_{i=1}^{n} a_i$.

If the instance for the Partition Problem is a Yes-instance, then this implies by definition that $\sum_{i \in \sigma} a_i = \frac{1}{2} \sum_{i=1}^{n} a_i$. Selecting the corresponding items for the Knapsack Problem gives by construction also a profit and weight of $\frac{1}{2} \sum_{i=1}^{n} a_i$. Thus, the total obtained profit is at least K and the capacity constraint is fulfilled, meaning that the instance for the Knapsack Problem also is a Yes-instance.

If the instance for the Knapsack Problem is a Yes-instance, then it must be that $\sum_{i \in \sigma} p_i \geq K$ and $\sum_{i \in \sigma} w_i \leq B$. Now note that $\sum_{i \in \sigma} p_i$ cannot be strictly larger K. Otherwise, the budget constraint would be violated, because $p_i = w_i$ for all $i \in [n]$ by construction. Therefore, it must be that $\sum_{i \in \sigma} p_i = \sum_{i \in \sigma} w_i = \frac{1}{2} \sum_{i=1}^{n} a_i$, from which directly follows that the integers in A corresponding to the items selected in S sum to $\frac{1}{2} \sum_{i=1}^{n} a_i$, meaning that the instance for the Partition Problem also is a Yes-instance.

For completeness, the reduction runs in $\mathcal{O}(n)$ because only n integers need to be computed. □

The following proposition is the primary motivation for introducing this concept of reductions.

Proposition 2.1. *If* $\Pi \propto \Pi'$ *and there exists a polynomial time-algorithm for* Π'*, then there also exists a polynomial-time algorithm for* Π.

Proof. Let $\mathcal{A}'$ be the algorithm that solves Π' in polynomial time. Consider the following Algorithm $\mathcal{A}$ for Π: transform the instance $I \in \mathcal{I}$ of Π to an instance $I' = \phi(I) \in \mathcal{I}'$ of Π'. Subsequently, run Algorithm $\mathcal{A}'$ on I' and output that:

- I is a Yes-instance if $\mathcal{A}'$ outputs that I' is a Yes-instance, and

- I is a No-instance, otherwise.

Note that the transformation runs in polynomial time of size(I) by definition of 2.11 and Algorithm $\mathcal{A}'$ runs in polynomial time by assumption, meaning that Algorithm $\mathcal{A}'$ also runs in polynomial time. □

This concept allows to define a class of problems, which represent the hardest problems in NP.

Definition 2.12. *A decision problem $\Pi \in NP$ is NP-complete if there exists a polynomial-time reduction from every problem in NP to Π.*

This means that every NP-complete problem can be reduced to each other. As a consequence, if an algorithm can be found that solves any NP-complete problem in polynomial time, then every problem in NP can be solved in polynomial time, which would imply that P = NP.

In the example used in this section, suppose that the Partition Problem is NP-complete. By definition, every problem in NP reduces to the Partition Problem. Using the reduction in the proof of Lemma 2.1, the Partition Problem reduces to the Knapsack Problem, meaning that all problems in NP also reduce to the Knapsack Problem.

Clearly, this definition is only useful if there exists at least one NP-complete problem, such that the above procedure can be done to prove the existence of more NP-complete problems. The proof of the existence of the first NP-complete problem is done in [20] for the Satisfiability Problem, but this proof lies beyond the scope of this thesis.

2.5 Approximation algorithms

Approximation algorithms are efficient algorithms that find a solution value with a provable performance guarantee. More formally:

Definition 2.13. *An algorithm $\mathcal{A}$ is an α-approximation algorithm for optimization problem Π if it finds for every instance I of Π in polynomial time a feasible solution with value:*

- $\mathcal{A}(I) \geq \frac{1}{\alpha} \cdot OPT$, *if Π is a maximization problem, or*
- $\mathcal{A}(I) \leq \alpha \cdot OPT$, *if Π is a minimization problem,*

where $OPT(I) = m(I, \sigma^*)$ is the value of an optimal solution σ^* for I and $\alpha \geq 1$.

It is desirable that α is as low as possible, since this implies that the (worst-case) performance is as good as possible.

An approximation algorithm will be proposed for the Knapsack Problem. Without loss of generality, assume that all items are ordered on their profit-to-weight (or efficiency), i.e.:

$$\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq \cdots \leq \frac{p_n}{w_n}.$$

This can be done as a preprocessing step in $\mathcal{O}(n \log(n))$ time.

Now consider the following Algorithm 1 for the Knapsack Problem. Let $k \leq n$ be the smallest index such that $\sum_{i=1}^{k} w_i > B$. Pick the more profitable set of items among $\{1, \ldots, k-1\}$ and $\{k\}$. This algorithm is described in pseudocode below more formally.

Algorithm 1 Greedy algorithm for the Knapsack Problem

1: $S \leftarrow \emptyset$
2: $k \leftarrow 1$
3: **while** $w(S \cup \{k\}) \leq B$ **do**
4: $\quad S \leftarrow S \cup \{k\}$
5: $\quad k \leftarrow k + 1$
6: **end while**
7: $\sigma \leftarrow \arg\max \{p(S), p(\{k\})\}$
8: **return** σ

Lemma 2.2 ([66]). *Algorithm 1 is a 2-approximation for the Knapsack Problem.*

Proof. It has to be shown that Algorithm 1 outputs a feasible solution in polynomial time. Note that at step 7 of the algorithm, both S and $\{k\}$ are feasible solutions, since $w(S) \leq B$ by construction and $w_k \leq B$ by definition. Hence, σ is also feasible.

To see the performance guarantee, note the following upper bound on OPT. Consider the theoretical situation in which item k can be taken fractionally, referred to as the fractional Knapsack Problem. If there is capacity left in the knapsack, i.e., if $B - \sum_{i=1}^{k-1} w_i > 0$, a fraction of $\alpha = \frac{B - \sum_{i=1}^{k-1} w_i}{w_k}$ of item k can be taken and added to S. Note that $\alpha < 1$, because item k could have been added to S otherwise. More importantly, note that this optimal solution value in the fractional Knapsack Problem:

$$\left(\sum_{i=1}^{k-1} p_i \right) + \alpha \cdot p_k \geq OPT.$$

This inequality holds because any feasible solution to the Knapsack Problem with integer items is also feasible for the fractional Knapsack Problem. In other words, the fractional Knapsack Problem is a strictly less restrictive problem than the original integer Knapsack Problem, meaning that the optimal solution value of the fractional Knapsack Problem is at least the optimal solution value of the original integer problem. Algorithm 1 has a solution value equal to $\max \left\{ \sum_{i=1}^{k-1} p_i, p_k \right\}$, which trivially must be at least half of the average of $\sum_{i=1}^{k-1} p_i$ and p_k, i.e.:

$$\max \left\{ \sum_{i=1}^{k-1} p_i, p_k \right\} \geq \frac{1}{2} \sum_{i=1}^{k-1} p_i + \frac{1}{2} p_k.$$

Combining all (in)equalities yields:

$$\max \left\{ \sum_{i=1}^{k-1} p_i, p_k \right\} \geq \frac{1}{2} \left(\sum_{i=1}^{k-1} p_i + p_k \right) \geq \frac{1}{2} \left(\sum_{i=1}^{k-1} p_i + \alpha \cdot p_k \right) \geq \frac{1}{2} OPT$$

As for the running time: ordering the items on profit-to-weight ratio can be done in $\mathcal{O}(n \log(n))$. Subsequently, at most n iterations need to be done in which a constant number of elementary operations need to be done. After the iterations, computing $p(S)$ is done in $\mathcal{O}(n)$ time. To conclude, the ordering has the highest time complexity which determines the running time of $\mathcal{O}(n \log(n))$, from which one can conclude that Algorithm 1 runs in polynomial time. □

2.6 Dynamic programming

Dynamic programming is a method to solve decision and optimization problems by breaking it down into smaller subproblems in a recursive manner. Intuitively this means that the optimal solutions of subproblems will be used to find the optimal solutions of slightly bigger subproblems, up until the overall problem is solved.

In this section, this technique is used to solve the Knapsack Problem. For the ease of explanation, the assumption is made that all profits are integer. In the next section will be described how this assumption can be dropped. Let P be the sum of profits of all items, i.e., $P = \sum_{i \in [n]} p_i$. Clearly, P is an upper bound on the optimal solution value.

Theorem 2.1 ([66]). *The Knapsack Problem with integer profits can be solved in $\mathcal{O}(nP)$ time.*

Proof. Define the dynamic programming function:

$$f(j,p) = \min_{\sigma \subseteq [j]} \left\{ \sum_{i \in \sigma} w_i \;\middle|\; \sum_{i \in \sigma} w_i \leq B, \sum_{i \in \sigma} p_i = p \right\}$$

for $j = 0, \ldots, n$ and $p = 0, \ldots, P$. In other words, $f(j,p)$ is the minimum total weight of any subset using only the first j items (rather than all items) such that the profit is *exactly* p and the capacity B is not exceeded. When a specific profit p using the subset $[j]$ cannot be obtained exactly, $f(j,p) = \infty$.

Since the goal of the Knapsack Problem is to maximize the profit while not exceeding the capacity B, the optimal solution value of an instance I of the Knapsack Problem can also be defined by:

$$OPT = \max\{p \in \{0, \ldots, P\} : f(n,p) \leq B\}.$$

Thus if all values of $f(j,p)$ can be computed, the optimal solution value OPT can be obtained as well. Dynamic programming does this in a bottom-up way as follows.

Initialization. Some values of the dynamic programming function have to be determined first to build on, which is referred to as the initialization. An easy initialization is to consider the empty subset of items by considering $j = 0$, as clearly only an empty solution can be made using this subset. This naturally corresponds to a weight and profit of 0. This means that:

$$f(0,p) = \begin{cases} 0 & \text{for } p = 0 \\ \infty & \text{otherwise} \end{cases}$$

as no other profit and weight can be obtained than 0.

Recursion. With this basis, one can define the recursion step in the dynamic programming as follows:

$$f(j,p) = \min\{f(j-1,p), f(j-1,p-p_j) + w_j\}$$

for $j = 1,\ldots,n$ and $p = 0,\ldots,P$. This simply means that two cases are considered when trying to obtain a profit of p: one in which $j \notin \sigma$ and one in which $j \in \sigma$, respectively. The corresponding minimum possible weights are given in the above expression and are justified as follows:

- If j is not in the minimum-weight solution σ that achieves profit p, then the weight of σ equals the minimum-weight solution where only the first $j-1$ items are considered, i.e., $f(j-1,p)$.

Note that prior to determining all $f(j,p)$ for a specific j and all p, only all optimal values $f(j-1,p)$ for all p need to be determined. Also note that this requires the assumption that all profits, weights and the budget are non-negative.

Retrieving the optimal solution. As mentioned, the optimal solution value OPT can is equal to $OPT = \max\{p \in \{0,\ldots,P\} : f(n,p) \leq B\}$. Retrieving the corresponding solution σ can be done by simply backtracking whether every item was used to obtain the profit OPT. Thus determine whether item n is used to obtain $f(n,OPT)$, one needs to look back whether:

- $f(j,OPT) = f(j-1,OPT-p_j) + w_j$, or
- $f(j,OPT) = f(j-1,OPT)$.

In the former case, there indeed exists an optimal solution σ^* which contains item n. This procedure can be repeated for $j = n-1,\ldots,1$ and updating the profit to look for accordingly.

Running time. Determining a single value of $f(j,p)$ requires a comparison of two values that can be retrieved in constant time. Since such a value has to be determined for all n items and at most P possible values for the profit, the running time of the dynamic program is $\mathcal{O}(nP)$. Afterwards, retrieving the optimal solution can be done by n comparisons of two computations. Therefore, the running time of the entire procedure is $\mathcal{O}(nP)$. □

2.7 Fully polynomial time approximation schemes

This section proposes a fully polynomial time approximation scheme (FPTAS) for the Knapsack Problem.

Definition 2.14. *An algorithm $\mathcal{A}$ is a fully polynomial time approximation scheme for a maximization problem Π if it finds for every error parameter $\epsilon > 0$ a feasible solution with value $\mathcal{A}(I) \geq (1-\epsilon) \cdot OPT(I)$ for every instance I of Π and its running time is polynomial in the input size and $\frac{1}{\epsilon}$.*

In other words, an FPTAS makes it possible to determine the optimality guarantee manually, but a better optimality comes with a larger running time.

Theorem 2.2 ([66])**.** *There exists an FPTAS for the Knapsack Problem that computes an $(1-\epsilon)$-approximate solution in $\mathcal{O}(n^3/\epsilon)$ for any $\epsilon > 0$.*

Proof. The idea is to truncate all profits, i.e.:

$$p'_i = \left\lfloor \frac{p_i}{10^t} \right\rfloor,$$

for any $t > 0$. Since all profits are truncated, a solution can now be determined in $\mathcal{O}(n\frac{P}{10^t})$ instead of $\mathcal{O}(nP)$ using the dynamic program described in Section 2.6.

However, an optimal solution corresponding to the truncated problem, say σ', clearly could be suboptimal with respect to the original problem. Let σ^* be an optimal solution to the original problem. A direct relationship can be seen from the two solutions, in order to derive an FPTAS, namely:

$$\sum_{i \in \sigma'} p_i \geq \sum_{i \in \sigma'} 10^t p'_i \geq \sum_{i \in \sigma^*} 10^t p'_i$$

The first equation follows directly from the definition of p'_i. The second inequality follows from the fact that σ' is optimal in the problem with truncated profits, but not necessarily in the original problem. Furthermore, because of rounding down, it must be that $10^t p'_i \geq p_i - 10^t$. Hence:

$$\sum_{i \in \sigma'} p_i \geq \sum_{i \in \sigma^*} (p_i - 10^t) \geq \sum_{i \in \sigma^*} p_i - n \cdot 10^t = OPT - n \cdot 10^t$$

$$= OPT \left(1 - \frac{n \cdot 10^t}{OPT}\right) \geq OPT \left(1 - \frac{n^2 \cdot 10^t}{P}\right)$$

The first equation follows from combining the earlier mentioned equations, while the second inequality holds because σ^* cannot contain more than n items by definition. The last inequality holds because $OPT \geq P/n$; after all, P/n is the average profit of the items, meaning that there exists an item with at least this profit. Selecting only this item is a feasible solution, so this is a lower bound on OPT.

Now fix $t = \log_{10}\left(\frac{\epsilon P}{n^2}\right)$ for some given $\epsilon > 0$, such that $\frac{n^2 \cdot 10^t}{P} = \epsilon$. Then:

$$\sum_{i \in \sigma'} p_i \geq OPT(1-\epsilon)$$

Meaning that a $(1-\epsilon)$-approximate solution for the Knapsack Problem is obtained. Since the running time of the dynamic program using truncated profits is $\mathcal{O}\left(n\frac{P}{10^t}\right)$ and because $\frac{n^2 \cdot 10^t}{P} = \epsilon$, the running time of the new algorithm is $\mathcal{O}(n^3/\epsilon)$. □

2.8 Column generation

Column generation is a method proposed by [36] that is used to solve large linear programs that are too time-consuming to solve due to the large number of decision variables. The motivation behind column generation is that while the number of variables is large, only a few variables take a positive value in an optimal solution. Therefore, only a very small subset of these variables is initially considered first to be able to find a feasible solution as soon as possible. The problem using this subset is called the *restricted master problem* (RMP).

However, because many variables in the RMP are disregarded, it is likely that an optimal solution to the original problem is not included in this subset. In other words, variables that can potentially improve the objective function, are missing in this subset. Column generation aims to identify such variables to add them to the subset. This is done by formulating and solving a *subproblem* (also referred to as *pricing problem*). Once such a variable is found, the subset of variables of the master problem is expanded with this variable and solved again. This procedure of identifying potentially improving variables is repeated until no improvement can be found, i.e., an optimal solution is contained within the subset.

This process is illustrated using an example when solving the following combinatorial optimization problem.

Bin Packing Problem (BPP)	
Given:	A bin capacity $B > 0$, a set of n items with size $0 < a_i \leq B$ for $i \in [n]$ and a maximum number of bins m.
Goal:	Minimize the number of required bins K for which an assignment $f : [n] \to [K]$ exists such that $\sum_{i:f(i)=j} a_i \leq B$ for all $j \in [K]$.

Standard IP formulation Define the following two types of decision variables for the BPP:

$$x_{ij} = \begin{cases} 1 & \text{if item } i \text{ is packed in bin } j, \text{ and} \\ 0 & \text{otherwise,} \end{cases}$$

and

$$y_j = \begin{cases} 1 & \text{if bin } j \text{ is used, and} \\ 0 & \text{otherwise.} \end{cases}$$

A straightforward way to formulate an Integer Program (IP) is then as follows:

$$\min \sum_{j=1}^{m} y_j \tag{2.0}$$

$$\text{s.t.} \sum_{j=1}^{m} x_{ij} \geq 1 \qquad i \in [n] \tag{2.1}$$

$$\sum_{i=1}^{n} a_i x_{ij} \leq B \qquad j \in [n] \tag{2.2}$$

$$x_{ij} \leq y_j \qquad i \in [n], j \in [m] \tag{2.3}$$

$$x_{ij} \in \{0,1\} \qquad i \in [n], j \in [m] \tag{2.4}$$

$$y_j \in \{0,1\} \qquad j \in [m] \tag{2.5}$$

where:

- constraint (2.1) guarantees that every item is packed,
- constraint (2.2) ensures that the bin capacity of every bin is not exceeded,
- constraint (2.3) implies that items are only placed in bins that are used, and
- constraint (2.4) and (2.5) ensure integrality of the taken bins and items put in bins.

However, this turns out to be not a good IP formulation. To see this, it is important to know how Linear Programs (LP's) are used as a subroutine to solve such integer programs.

Branch and bound. Integer Programs are often solved using a branch-and-bound method. In this method, the LP-relaxation is obtained first by simply allowing fractional decision variables, i.e., constraint (2.4) and (2.5) are simply replaced by:

$$0 \leq x_{ij} \leq 1 \qquad i \in [n], j \in [m] \tag{2.6}$$

$$0 \leq y_j \leq 1 \qquad j \in [m] \tag{2.7}$$

It is since 1979 well-known that LP's can be solved in polynomial time using a so-called ellipsoid method [67] and later even more efficiently using interior point methods [62]. There currently exist many methods to deal with LP's efficiently that lie beyond the scope of this thesis, so for now is simply assumed that solving an LP can be done quickly.

Now note that since this is a minimization problem, the optimal objective value of the LP is at most the optimal objective value of the IP. After all, every optimal solution to the IP is also feasible for the LP-relaxation. The reverse clearly does not always hold, unless every optimal LP-solution turns out to be integral. If the LP-solution is not integral, a branch-and-bound algorithm can be applied.

During branch-and-bound, a fractional variable, say y_j, in a fractional solution from the LP-relaxation is considered. The original problem *branches* then into two cases, or rather subproblems: one in which $y_j = 0$ and one in which $y_j = 1$. The subproblem with the lowest optimal solution value is the optimal solution of the overall problem, because the union of the state space of both subproblems comprises the state space of the overall problem.

Solving a subproblem is now done as for the original problem, i.e., by considering the LP-relaxation first. No further branching is needed when either:

- the optimal solution to the LP-relaxation is integral, or
- the optimal solution value to the LP-relaxation is worse than the solution value of an earlier found integral solution; this means that the optimal integral solution cannot be found in this branch, meaning that there is no point branching further in this subproblem.

Otherwise, the procedure repeats: branching on a fractional variable, solving the LP-relaxation(s) of the subproblems and verifying whether the solution is integral or can never improve an already found solution. When all possible subproblems are branched out, the overall problem is solved.

This procedure can clearly also be time-consuming, because in theory it is possible that on all n variables needs to be branched, leading to $\mathcal{O}(2^n)$ subproblems. For the process to terminate more quickly, it is desirable to have at least two characteristics of the IP:

1. The optimal solution value of the LP-relaxation is close to the optimal solution of the IP. This implies that few modifications to a fractional solution is needed to find an integral one.
2. The set of feasible solutions in the two possible branches do not contain similar solutions.

However, the former does not hold because the trivial solution to the LP-relaxation is $x_{ij} = y_i = \frac{1}{n}$ for $i \in [n]$ and $j \in [m]$. This represents the solution where for every item, a fraction of $\frac{1}{n}$ is packed in every bin. Under integrality conditions, constraint (2.3) would ensure that $y_j = 1$ (i.e., the bin is used). Within this LP-relaxation however, $y_j = 1$ is not enforced, regardless of how many partial items are packed in the bin, as y_j is equal to the maximum fraction of any item that is packed in the bin by this constraint.

This solution has the corresponding solution value equal to 1, which can be arbitrary far from the optimal integral solution. The latter does not hold because there the same integral solution can be represented in multiple ways (the selected items in bin j and j' can be swapped, which basically is same solution value). Hence, an alternative formulation is more suitable when using column generation.

Master problem. An alternative way to formulate the IP for the BPP is by defining bin configurations where the exact composition of a bin is predefined. In other words, a bin configuration simply is a possible set of items. Let $C = \{c \subseteq [n] \mid$

$\sum_{i:i\in c} a_i \leq B\}$ be the set of possible bin configurations. Furthermore, introduce for $c \in [C]$ the decision variable:

$$z_c = \begin{cases} 1 & \text{if bin configuration } c \text{ is used,} \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, introduce an indicator parameter:

$$I_{ic} = \begin{cases} 1 & \text{if item } i \text{ is contained in bin configuration } c, \\ 0 & \text{otherwise.} \end{cases}$$

Then an alternative IP formulation is given by:

$$\min \sum_{c\in C} z_c \tag{2.8}$$

$$\text{s.t.} \sum_{c\in C} I_{ic} z_c \geq 1 \qquad i \in [n] \tag{2.9}$$

$$z_c \in \{0,1\} \qquad c \in C \tag{2.10}$$

This is the master problem and a different way of formulating the IP. However, note that $|C| = \mathcal{O}(2^n)$, which may become too large. But as mentioned at the beginning of this section, for an optimal solution, only a very small subset of C has a positive value for z_c. The difficulty for the master problem is to find a subset of configurations $C' \subseteq C$ with $|C'| \ll |C|$ efficiently, such that an optimal solution of the master problem is guaranteed to be in C'.

Restricted master problem. The LP-relaxation master problem using the subset C' instead of C is called the *restricted master problem* (RMP):

$$\min \sum_{c\in C'} z_c \tag{2.11}$$

$$\text{s.t.} \sum_{c\in C'} I_{ic} z_c \geq 1 \qquad i \in [n] \tag{2.12}$$

$$0 \leq z_c \leq 1 \qquad c \in C' \tag{2.13}$$

The way this RMP is approached is as follows. First, it needs to be solved using an initial set of configurations (or more generally, columns) C' in which a feasible solution with certainty can be found. A simple way to achieve this for this problem is to choose n bin configurations, where bin configuration i contains only item i for $i \in [n]$. In this way, the values of I_{ic} correspond to an identity matrix. Alternatively, this can also be done by a greedy algorithm for two reasons. A greedy solution has a value which should be closer to the optimal solution value than a solution where every bin contains only one item. Also, a greedy solution is more likely to contain a bin configuration which is also found in an optimal solution, which would imply that the restricted master problem may need to identify fewer variables to be added to C'. For the procedure though, any subset $C' \subseteq C$ that contains any feasible solution suffices.

Pricing problem. After solving the RMP, let π_i be the dual variable corresponding to the j-th constraint. The interpretation of π_i within this context is basically how much the objective function could decrease if item i is not required to be packed anymore in the current optimal solution. Now consider a bin configuration $c \notin C'$. To determine whether a bin configuration $c \notin C'$ potentially can reduce the objective value of the master problem, it is required to compute its *reduced costs*:

$$r_c = 1 - \sum_{j=1}^{m} I_{ic}\pi_i.$$

If bin configuration c is used, the objective value increases by 1, which explains the first term. The second term describes how many bins this configuration c can save with regard to the objective value. It is the sum of the mentioned $\pi_i's$ for every item in bin configuration c. Now if $r_c < 0$, c can potentially improve the objective value of the RMP.

However, it is too time consuming to determine this for every $c \in C$ for which $c \notin C'$ as $|C| = \mathcal{O}(2^n)$. Instead, the following *pricing problem* (or *subproblem*) can be formulated to determine the bin configuration with the lowest reduced costs:

$$\min 1 - \sum_{i=1}^{n} \pi_i x_i \tag{2.14}$$

$$\text{s.t.} \sum_{i=1}^{n} a_i x_i \geq C \qquad i \in [n] \tag{2.15}$$

$$x_i \in \{0, 1\} \qquad i \in [n] \tag{2.16}$$

where the decision variable $x_i = 1$ if item i should be used in the bin configuration, and $x_i = 0$ otherwise.

This may initially not seem like a straightforward problem due to the integrality constraint. However, the objective function in (2.14) is the same problem as maximizing $\sum_{i=1}^{n} \pi_i x_i$. And with this objective function, this problem turns out to be a Knapsack Problem with profits π_i, weights a_i and knapsack capacity C. For this, a dynamic program has been described in Section 2.6 that can be used to determine the bin configuration.

If the optimal solution value $\sum_{i=1}^{n} \pi_i x_i > 1$, then this means that the original objective function $r = 1 - \sum_{i=1}^{n} \pi_i x_i < 0$, meaning that the corresponding variable indeed has the potential to improve the objective function. However, if no such variable can be found, there exists no variable that can improve the restricted master problem. This meaning that the optimal solution to the RMP is the same as the optimal solution value of the master problem such that the problem is solved.

Summary. To bring this method into context of solving the master problem, the procedure can be summarized in following steps:

1. Formulate the combinatorial optimization problem as a suitable *master problem*.

2. Consider a subset of the variables, a basis, of the master problem that contains with certainty a feasible solution, to obtain the *restricted master problem*.

3. Solve the relaxation of the *restricted master problem* and obtain the dual values.

4. Construct a *pricing problem* using the dual values to find variables (or columns) that can reduce the objective function (in case of a minimization problem).

5. Solve the pricing problem and determine whether there exists a variable with negative reduced costs. If so, go back to step 2.

6. If the solution is integral, an optimal solution is found. If the solution is not integral, branch on a fractional variable and go to step 3.

Since this procedure combines branch-and-bound and column generation methods, this procedure is also referred to as *branch and price*.

Chapter 3

Job Covering

This chapter introduces the Budgeted Job Coverage Problem (BJCP), where the goal is to select a subset of a given family of sets, over a set of jobs with costs and weights, in order to maximize the total weight while not exceeding a budget. Because costs are assigned to the jobs (or elements), rather than the sets, the problem complicates significantly and analyses from related work become unsuitable.

After discussing the background of covering problems in Section 3.1, the problem under consideration will be defined in Section 3.2. Due to the problem's complexity, the theoretical analysis considers three special cases. Section 3.3 considers the case where sets have cardinality 2, which allows a constant approximation algorithm. Subsequently, Section 3.4 considers an acyclic version, for which a dynamic program and fully polynomial time approximation scheme will be presented. These results will be used in Section 3.5 that aims to generalize the acyclic variant using a concept called feedback vertex sets. This chapter concludes with Section 3.6 by deducing which characteristics of the problem complicates the improvement of a theoretical analysis and proposes directions for future research.

The results in this chapter are based on [118] and require intermediate knowledge on combinatorial optimization, with graph and complexity theory in particular (see Sections 2.1 to 2.7).

3.1 Background

Covering problems belong to the classical problems within Operations Research and computer science. The basic variant is known as the Set Cover Problem (SCP), where one is given a family of sets $F = \{S_1, \ldots, S_m\}$ over a domain of elements X. The goal is to find a subcollection $\sigma \subseteq F$ of minimum cardinality such that its union equals X. The SCP is one of Karp's 21 NP-complete problems [63] for which many algorithms and heuristics have been proposed. In particular, the study of the SCP led to the development of fundamental techniques for the entire field of approximation algorithms [122]. Such techniques will also be applied within this chapter to analyze the complexity of the variant of the SCP that is considered here.

In practice, set covering problems play an important role in scheduling, such as crew scheduling. Elements may correspond to jobs (e.g., flights, bus trips) or

entire shifts, while sets correspond to feasible (or legal) working schedules for crew members (or resources). A common objective is to minimize the number of used sets or total weight. Within a practical context, this may be equivalent to minimizing the number of required crew members, while the cost of a set represents the cost of e.g., a crew member performing that specific job or shift. Since the problem is NP-hard, the problem has been the subject of many heuristics due to its frequent natural occurrence. This chapter considers a less studied variant of the SCP and provides new theoretical insights to the complexity of set covering problems.

3.2 The Budgeted Job Covering Problem

3.2.1 Motivation

The problem considered in this chapter comes from a setting where a set of *jobs* (e.g., serving customers) are given. The performing of a job has its own associated cost, and a weight representing its reward. Furthermore, there is a collection of *resources* (e.g., staff, machines) that each can perform their own specific subset of the jobs. The goal of the problem is to select a subset of the resources to *maximize the reward*, while the total cost of the performed jobs does not exceed a given *budget*. However, a crucial characteristic of the problem is that whenever a resource is selected, it becomes *obligatory* to perform all the jobs that it can perform.

The incorporation of an obligation is in contrast to most well-studied variants of the Set Cover Problem. Usually, whenever a resource is selected (e.g., constructing a factory, hiring a worker), this *enables* (rather than obliges) an organization to execute a set of jobs. Also, costs are usually assigned to the resources (e.g., for constructing a factory, or establishing a server) rather than the jobs.

In practice, these obligations are mainly the result of the necessity to maintain a service level, or because the jobs are very undesirable or irresponsible to refuse. Some examples include the offering of:

- Area-specific needs such as public electricity or wireless internet. Once such a public network is established, everyone in the neighborhood can use it. The costs can be calculated from the usage in that area, while the rewards have to be estimated from the increase in welfare in that area.

- Public recreational parks or attractions that require maintenance. The maintenance costs (cleaning jobs, security) are in proportion to the amount of people that decide to visit it, while the rewards also have to be estimated from the increase in welfare.

Note that if multiple resources offer the same service in the same area, it is assumed that the people in this area will use only one service (it remains one single job). If none is offered, people will not use the service in general at all, at the expense of the welfare. The addition of obligations within a set cover problem is more typical for city council or governments (rather than commercial companies that can refuse customers) that want to increase the prosperity of their city or country, while offering this service comes at a significant, area-dependent cost.

3.2.2 Definition

Define $F = \{S_1, \ldots, S_m\}$ as a family of resource jobsets over a set of jobs $J = [n]$, i.e., $S_i \subseteq J$ for each $i \in [m]$ such that $\cup_{i=1}^{m} S_i = J$. Each job $j \in J$ has an associated cost $c_j \geq 0$ and weight $w_j > 0$ and there is a budget B. Finally, given a solution $\sigma \subseteq F$, define $J(\sigma)$ as the union of all sets in σ, i.e., $J(\sigma) = \cup_{S \in \sigma} S \subseteq X$, representing all jobs that have to be executed.

The goal of the Budgeted Job Coverage Problem (BJCP) is to select a collection of resource jobsets σ that maximizes the total weight of the jobs $w(\sigma) = \sum_{j \in J(\sigma)} w_j$, while its total cost $c(\sigma) = \sum_{j \in J(\sigma)} c_j$ does not exceed B. Hence, subset S_i can be interpreted as the set of jobs that have to be executed when resource i (e.g., a server) is selected by σ. From a practical point of view, any job $j \in S_i$ may also be executed by another resource i', as long as $S_{i'} \in \sigma$. The problem considered in this chapter can then be summarized as follows.

Budgeted Job Coverage Problem (BJCP)	
Given:	A family of subsets $F = \{S_1, \ldots, S_m\}$ over a set of jobs $J = [n]$ with cost $c_j > 0$ and weight $w_j > 0$ for each $j \in J$, and a budget B.
Goal:	Find a collection of subsets $\sigma \subseteq F$ that maximizes $\sum_{j \in J(\sigma)} w_j$ such that $\sum_{j \in J(\sigma)} c_j \leq B$.

Moreover, the following concepts are introduced.

Definition 3.1. *The **incidence graph** $G(F)$ of F is defined as the bipartite graph $G(F) = (F \cup J, E)$ with $E = \{\{S, j\} \mid j \in S\}$.*

Definition 3.2. *F is **acyclic** if its incidence graph $G(F)$ does not contain a cycle. Given a subset $F' \subseteq F$, the term $G[F']$ is used to refer to the **subgraph** of G induced by the sets in F', i.e., $G[F'] = (F' \cup J', E')$ with $J' = J(F')$. A subset F' is called a **subtree** of F if $G[F']$ is acyclic.*

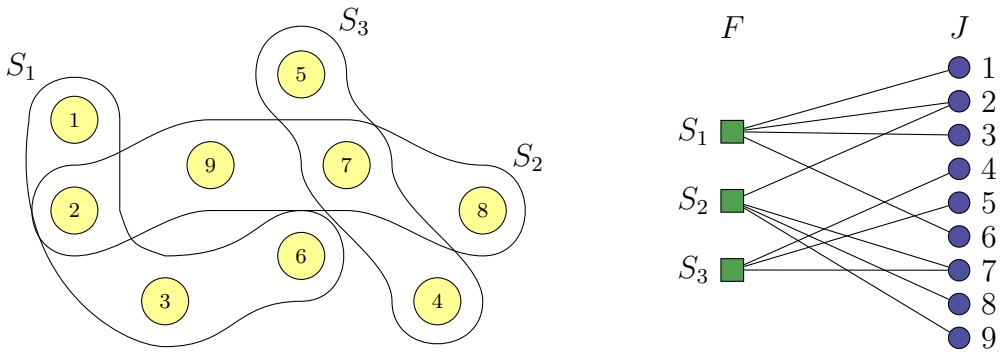

Figure 3.1: Example of a BJCP-instance (left) and its incidence graph (right)

See Figure 3.1 for a graphical example of an acyclic BJCP-instance and its incidence graph, where $F = \{S_1 = \{1, 2, 3, 6\}, S_2 = \{2, 7, 8, 9\}, S_3 = \{4, 5, 7\}\}$. Suppose all elements have unit weight and cost and $B = 6$. Then, the unique optimal solution is $\sigma = \{S_2, S_3\}$ with weight 6.

3.2.3 Related work

Much literature is available on the maximum coverage problem and its variants (see, e.g., [12, 19, 68]). The most relevant results are also discussed here.

Budgeted maximum coverage. Arguably the most closely related problem to the BJCP is the *Budgeted Maximum Coverage Problem* (BMCP), where the only, yet crucial, difference is that costs are assigned to the sets instead of the elements. This is a fundamental combinatorial optimization problem with many applications in resource allocation, job scheduling and facility location (see, e.g., [56] for examples). It has been shown that this problem is not polynomial-time approximable within a factor of $(1 - \frac{1}{e})$ unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log n)})$, even if all sets have unit cost [34].

A $(1-\frac{1}{e})$-approximation algorithm for the budgeted maximum coverage problem has been derived in [68], which is the best possible. These algorithms are based on a natural greedy approach in combination with a standard enumeration technique. Similar approaches were used to derive constant factor approximation algorithms for several other variants and generalizations of the maximum coverage problem.

Albeit seemingly minor, assigning the cost to elements instead of sets makes the problem much harder to tackle algorithmically. More specifically, the analysis of greedy approaches for the BMCP and other variants turn out to be inapplicable for the BJCP. The basic idea underlying these greedy approaches is to select in each iteration the most *cost-efficient* set, i.e., the set that maximizes the ratio of the profit of newly covered elements over the cost of selecting the set. A property that is crucially exploited in the analysis of these algorithms is that the cost for selecting a set is constant. This means that its cost-efficiency can only decrease throughout the course of the algorithm, as more of its elements get covered. However, this monotonicity property is not guaranteed for the BJCP because the cost for picking a set depends on the set of already covered jobs. In fact, it is not hard to see that the cost-efficiency of a set can change arbitrarily from one iteration to the next.

Other variants. Also closely related to the BJCP is the *budgeted bid optimization problem*, proposed in [35]. In this work, a $(1-\frac{1}{e})$-approximation algorithm is derived if the budget constraint is *soft*, i.e., has to be met in expectation only. However, this budget constraint is hard in the BJCP considered here.

Moreover, in [19], a generalized version of the BMCP is studied, but this does not include BJCP as a special case. Also note that the BJCP reduces to the knapsack problem if all sets have cardinality 1. This problem is known to be weakly NP-hard and admits an FPTAS (see, e.g., [66]).

Finally, the BJCP can be seen as a special case of a more general set of problems where a submodular profit function has to be maximized, subject to the constraint that a submodular cost function does not exceed a given budget. However, when there is oracle access to both submodular functions, it has been shown that this more general problem is not approximable within a factor of $\frac{\log(m)}{\sqrt{m}}$, where m is the number of elements in the ground set. This holds even for the special case that the objective function is the modular function that returns the cardinality of the set. This follows from Theorem 4.2 in [111]; see also [61].

3.2.4 Contributions

The contributions of this chapter are the following. First, in Section 3.2.5 will be shown that the BJCP cannot approximated within a factor of $1-\frac{1}{e}$ in Section 3.2.5, unless $\mathsf{NP} \subseteq DTIME(n^{O(\log\log n)})$. This proof is done by a reduction from the *maximum coverage problem*.

Subsequently in Section 3.3, a special case of the BJCP will be considered where each resource jobset has cardinality 2, for which a $\frac{1}{2}(1-\frac{1}{\sqrt{e}})$-approximation algorithm will be obtained. This algorithm crucially exploits the fact that every jobset has exactly 2 elements, and it will be argued that this approach is hard and tedious to generalize for larger fixed set cardinality than 2.

Moreover, in Section 3.4 will be shown that that in case the corresponding incidence graph is acyclic, the problem can be solved in $\mathcal{O}(mn^3W^2)$ time using a bi-level dynamic program. Using truncation techniques, this algorithm can be turned into an FPTAS with error parameter $\epsilon > 0$, which runs in $\mathcal{O}(mn^5/\epsilon^2)$.

These results suggest that cycles increase the computational complexity of the problem significantly. Following up on this analysis, some insights will be given in Section 3.5 on how to the decompose BJCP into such acyclic versions, and the computation cost that comes with it.

3.2.5 Complexity

Theorem 3.1. *The BJCP cannot be approximated within a factor of* $1-\frac{1}{e}$ *in polynomial time, unless* $\mathsf{NP} \subseteq DTIME(n^{O(\log\log n)})$.

Proof. The proof is done by a reduction from the *maximum coverage problem*, where one is given a family of sets $F = \{S_1, \ldots, S_m\}$ over a domain of elements $X = [n]$ and a parameter $k > 0$. The goal is to choose a subset $\sigma \subseteq F$ that maximizes the number of covered elements $|X(\sigma)|$ such that $|\sigma| \leq k$. This problem cannot be approximated within $1-\frac{1}{e}$, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log\log n)})$ [34].

Given an instance $I = (X, F, k)$ of the maximum coverage problem, construct an instance $I' = (J', F', c', w', B')$ of the BJCP as follows. Define:

- $J' := \{1, \ldots, n+m\}$, i.e., m additional elements (or jobs), each representing a set, are added,
- $F' := \{S_1 \cup \{n+1\}, \ldots, S_m \cup \{n+m\}\}$,
- $c'_j := \begin{cases} 0 & j = 1, \ldots, n, \\ 1 & j = n+1, \ldots, n+m, \end{cases}$
- $w'_j := \begin{cases} 1 & j = 1, \ldots, n, \\ 0 & j = n+1, \ldots, n+m, \text{ and} \end{cases}$
- $B' = k$.

It is easy to verify that for each subset $\sigma \subseteq F$, the set

$$\sigma' = \{S_j \cup \{n+j\} : S_j \in \sigma\} \subseteq F'$$

satisfies $\sum_{j \in J(\sigma')} c'_j \leq B'$ if and only if $|\sigma| \leq k$, because $B' = k$. Furthermore, $\sum_{j \in J(\sigma')} w'_j = |X(\sigma)|$ by construction of w'_j. Thus, I can be approximated within a factor of $1 - \frac{1}{e}$ if and only if I' can be approximated within a factor of $1 - \frac{1}{e}$, which proves Theorem 3.1. □

3.3 The BJCP with set cardinality 2

This section presents a $\frac{1}{2}(1 - \frac{1}{\sqrt{e}})$-approximation algorithm for the BJCP when each set has cardinality 2, i.e., $|S_i| = 2, \forall i \in [m]$. This is done by reducing the problem to the classical BMCP, for which several concepts need to be introduced first.

3.3.1 A cost-efficiency function for the BMCP

Recall that the BMCP is given by a family of subsets $F = \{S_1, \ldots, S_m\}$ of $X = [n]$ with associated weights $w_1, \ldots, w_n$, but the costs $c(S_1), \ldots, c(S_m)$ are now associated with the subsets. The goal is to find a solution $\sigma \subseteq F$ that maximizes the total weight $w(\sigma) = \sum_{j \in X(\sigma)} w_j$ while the total costs $c(\sigma) = \sum_{S \in \sigma} c(S)$ do not exceed the budget B.

A polynomial-time $(1 - \frac{1}{e})$-approximation algorithm for the BMCP is given in [68]. Also, various simpler algorithms with worse approximation factors are presented here. This section presents a variation of one of these algorithms that is required as a subroutine in the analysis for the BJCP.

Definition 3.3. *Let $I = (F, X, c, w, B)$ be an instance of the BMCP. For $\alpha \in [0, 1]$, an α-**approximate cost-efficiency oracle** for I is a function $f_I : 2^X \to F$ that maps a subset of the elements $Y \subseteq X$ to a subset $S \in F$ such that:*

$$\frac{\sum_{j \in S \setminus Y} w_j}{c(S)} \geq \alpha \cdot \frac{\sum_{j \in S' \setminus Y} w_j}{c(S')},$$

for any $S' \in F$.

Thus, a cost-efficiency oracle takes as input a subset of the elements $Y \subseteq X$, and selects a subset $S \in F$ with the approximately highest cost-efficiency (up to a factor α), excluding the profit that would be contributed by elements in Y. Only subsets of which the cost does not exceed the budget are considered.

Let $I = (F, X, c, w, B)$ be an instance of the BMCP, and let f_I be an α-approximate cost-efficiency oracle for this instance for some $\alpha \in (0, 1]$. The (greedy) algorithm that will be proposed now takes as input only f_I. Throughout the execution of the algorithm, σ represents a feasible solution, while $Y = X(\sigma)$ represents the set of elements covered by σ.

Using these representations, it is now possible to propose the following Algorithm $\mathcal{A}$ for the BMCP:

1. Initialize the solution $\sigma := \emptyset$ with $Y := \emptyset$.
2. Let $S := f_I(Y)$. If:

- $Y = X$ (i.e., there is no profitable subset left) or if $c(S)+\sum_{S'\in\sigma} c(S') > B$ (i.e., adding the new subset S to σ would exceed the budget), go to Step 3.
- $Y \neq X$, set $\sigma := \sigma \cup \{S\}$, $Y := X(\sigma)$ and repeat Step 2.

3. Output the solution with the highest total weight among the two solutions σ and $\{S\}$.

Theorem 3.2. *Algorithm $\mathcal{A}$ is a $\frac{1}{2}(1-\frac{1}{e^\alpha})$-approximation algorithm for BMCP that runs in $O(n \cdot t)$ time, where t is the amount of time it takes to evaluate f_I.*

The bound on the running time of Algorithm $\mathcal{A}$ follows because there are at most m iterations of Step 2. In each iteration, only one call to the oracle f_I is done. The approximation factor is obtained by generalizing the analysis given in [68] for a similar algorithm, by taking into account the additional factor α and the oracle-access assumption.

Define k as the iteration of Step 2 of Algorithm $\mathcal{A}$ running on I, and $S_k^{\mathcal{A}} \in F$ as the subset that f_I returns at the k-th iteration of Step 2. Furthermore, let $\sigma_0^{\mathcal{A}} = \emptyset$ and inductively define $\sigma_k^{\mathcal{A}} = \sigma_{k-1}^{\mathcal{A}} \cup \{S_k^{\mathcal{A}}\}$, i.e., $\sigma_k^{\mathcal{A}} = \{S_1^{\mathcal{A}}, \ldots, S_k^{\mathcal{A}}\}$.

Note that if ℓ is the total number of iterations of Step 2, then in Step 3 the solutions $\sigma_{\ell-1}^{\mathcal{A}}$ and $S_\ell^{\mathcal{A}}$ are considered. Moreover, note that $\sigma_\ell^{\mathcal{A}}$ is typically not a feasible solution, as its cost may exceed B. Finally, denote by σ^* an optimal solution to I.

Lemma 3.1. *For each iteration k of Step 2 of Algorithm $\mathcal{A}$, it holds that*

$$\sum_{j \in X(\sigma^*) \setminus X(\sigma_{k-1}^{\mathcal{A}})} w_j \leq \frac{B}{\alpha \cdot c(S_k^{\mathcal{A}})} \left(w(\sigma_k^{\mathcal{A}}) - w(\sigma_{k-1}^{\mathcal{A}}) \right).$$

Proof. Let $S_1^*, \ldots, S_{|\sigma^*|}^*$ be the subsets of the optimal solution σ^*. Let $S_{\max}^*$ be the subset of σ^* with the highest cost-efficiency when it would be added to the solution $\sigma_{k-1}^{\mathcal{A}}$. That is:

$$S_{\max}^* = \arg\max_{S \in \sigma^*} \frac{\sum_{j \in S \setminus X(\sigma_{k-1}^{\mathcal{A}})} w_j}{c(S)}.$$

Recall that $S_k^{\mathcal{A}}$ selected by the algorithm in iteration k has a cost-efficiency that is within a factor of α from that of $S_{\max}^*$.

Suppose now that all of σ^* would be added to $\sigma_{k-1}^{\mathcal{A}}$. The increase in weight, i.e., the total weight of all elements in σ^* that are not in $X\left(\sigma_{k-1}^{\mathcal{A}}\right)$, can then be bounded

by:

$$\begin{aligned}
\sum_{j \in X(\sigma^*) \setminus X(\sigma^{\mathcal{A}}_{k-1}} w_j &\leq \sum_{S \in \sigma^*} \sum_{j \in S \setminus X(\sigma^{\mathcal{A}}_{k-1})} w_j \\
&= \sum_{S \in \sigma^*} c(S) \sum_{j \in S \setminus X(\sigma^{\mathcal{A}}_{k-1})} \frac{w_j}{c(S)} \\
&\leq \sum_{S \in \sigma^*} c(S) \sum_{j \in S^*_{\max} \setminus X(\sigma^{\mathcal{A}}_{k-1})} \frac{w_j}{c(S^*_{\max})} \\
&\leq B \sum_{j \in S^*_{\max} \setminus X(\sigma^{\mathcal{A}}_{k-1}} \frac{w_j}{c(S^*_{\max})} \\
&\leq B \sum_{j \in S^{\mathcal{A}}_k \setminus X(\sigma^{\mathcal{A}}_{k-1})} \frac{w_j}{\alpha \cdot c(S^{\mathcal{A}}_k)} \\
&= \frac{B}{\alpha \cdot c(S^{\mathcal{A}}_k)} \left(w(\sigma^{\mathcal{A}}_k) - w(\sigma^{\mathcal{A}}_{k-1}) \right).
\end{aligned}$$

The first inequality holds because both terms contain the same elements, but the right term counts the weight every element possibly multiple times. The second inequality holds by definition of $S^*_{\max}$, while the third inequality holds because σ^* is feasible. Finally, the fourth inequality follows by using the definition of a 2-approximate cost-efficiency oracle (see Definition 3.3). □

Lemma 3.2. *For each iteration k of Step 2 of Algorithm $\mathcal{A}$, it holds that*

$$w(\sigma^{\mathcal{A}}_k) - w(\sigma^{\mathcal{A}}_{k-1}) \geq \frac{\alpha \cdot c(S^{\mathcal{A}}_k)}{B} \left(w(\sigma^*) - w(\sigma^{\mathcal{A}}_{k-1}) \right).$$

Proof. The derivation applies Lemma 3.1 as follows:

$$\begin{aligned}
\frac{B}{\alpha \cdot c(S^{\mathcal{A}}_k)} \left(w(\sigma^{\mathcal{A}}_k) - w(\sigma^{\mathcal{A}}_{k-1}) \right) &\geq \sum_{j \in X(\sigma^*) \setminus Y^{\mathcal{A}}_{k-1}} w_j \\
&\geq \sum_{j \in X(\sigma^*) \cap Y^{\mathcal{A}}_{k-1}} w_j + \sum_{j \in X(\sigma^*) \setminus Y^{\mathcal{A}}_{k-1}} w_j \\
&\quad - \sum_{j \in X(\sigma^*) \cap Y^{\mathcal{A}}_{k-1}} w_j - \sum_{j \in Y^{\mathcal{A}}_{k-1} \setminus X(\sigma^*)} w_j \\
&= w(\sigma^*) - w(\sigma^{\mathcal{A}}_{k-1}).
\end{aligned}$$

□

Lemma 3.3. *For each iteration k of Step 2 of algorithm $\mathcal{A}$, it holds that*

$$w(\sigma^{\mathcal{A}}_k) \geq w(\sigma^*) \left(1 - \prod_{i=1}^{k} \left(1 - \frac{\alpha \cdot c(S^{\mathcal{A}}_i)}{B} \right) \right).$$

Proof. The proof is done by induction on the number of iterations k. If $k = 1$, note that:

$$w(\sigma_k^{\mathcal{A}}) = w(\sigma_1^{\mathcal{A}}) - w(\sigma_0^{\mathcal{A}}) \geq \frac{\alpha \cdot c(S_1^{\mathcal{A}})}{B} \sum_{j \in X(\sigma^*)} w_j = w(\sigma^*) \left(1 - \left(1 - \frac{\alpha \cdot c(S_1^{\mathcal{A}})}{B}\right)\right),$$

where the inequality follows from Lemma 3.1.

Suppose now that the claim holds if $k \in [\ell]$ for some $\ell \geq 1$. It will be proven that the claim also holds for $k = \ell + 1$.

$$\begin{aligned}
w(\sigma_{\ell+1}^{\mathcal{A}}) &= w(\sigma_\ell^{\mathcal{A}}) + w(\sigma_{\ell+1}^{\mathcal{A}}) - w(\sigma_\ell^{\mathcal{A}}) \\
&\geq w(\sigma_\ell^{\mathcal{A}}) + \frac{\alpha \cdot c(S_{\ell+1}^{\mathcal{A}})}{B}(w(\sigma^*) - w(\sigma_\ell^{\mathcal{A}})) \\
&= \left(1 - \frac{\alpha \cdot c(S_{\ell+1}^{\mathcal{A}})}{B}\right) w(\sigma_\ell^{\mathcal{A}}) + \frac{\alpha \cdot c(S_{\ell+1}^{\mathcal{A}})}{B} w(\sigma^*) \\
&\geq w(\sigma^*) \left(1 - \frac{\alpha \cdot c(S_{\ell+1}^{\mathcal{A}})}{B}\right) \left(1 - \prod_{i=1}^{\ell} \left(1 - \frac{\alpha \cdot c(S_i^{\mathcal{A}})}{B}\right)\right) \\
&\quad + \frac{\alpha \cdot c(S_{\ell+1}^{\mathcal{A}})}{B} w(\sigma^*) \\
&= w(\sigma^*) \left(1 - \prod_{i=1}^{\ell+1} \left(1 - \frac{\alpha \cdot c(S_i^{\mathcal{A}})}{B}\right)\right),
\end{aligned}$$

where the first inequality follows from Lemma 3.2 and the second inequality follows from the induction hypothesis. □

The following is the final technical lemma that is required to prove the approximation bound.

Lemma 3.4. *Let $a \in \mathbb{R}_{\geq 0}$ be any nonnegative real number and $n \in \mathbb{N}_{>1}$ be a positive natural number. The function $g(x_1, \ldots, x_n) = 1 - \prod_{i=1}^{n} \left(1 - \frac{x_i}{B}\right)$ on the domain $D = \{x \in \mathbb{R}^n : \sum_{i=1}^{n} x_i = a, x_i \geq 0\}$ achieves its minimum at the point where $x_i = a/n$ for all $i \in [n]$.*

Proof. It will be shown that the function

$$g'(x) = 1 - g(x) = \prod_{i=1}^{n} \left(1 - \frac{x_i}{B}\right)$$

achieves its maximum when $x_i = \frac{a}{n}$ for all $i \in [n]$.

Suppose there is a solution $x \in D$ for which there are two indices i and j such that $x_i > \frac{a}{n}$ and $x_j < \frac{a}{n}$. Assume without loss of generality that $i = 1$ and $j = 2$. Let x' be the vector obtained from x' by subtracting an amount of $\epsilon = (x_1 - x_2)/2$ from x_1 and adding it to x_2. It will be shown that $g'(x') > g'(x)$.

Let $C = \prod_{i=3}^{n} \left(1 - \frac{x_i}{B}\right)$. Then indeed,

$$\begin{aligned}
g'(x') &= \left(1 - \frac{x_1 - \epsilon}{B}\right)\left(1 - \frac{x_2 + \epsilon}{B}\right)\prod_{i=3}^{n}\left(1 - \frac{x_i}{B}\right) \\
&= C \cdot \left(1 - \frac{x_1}{B} + \frac{\epsilon}{B}\right)\left(1 - \frac{x_2}{B} - \frac{\epsilon}{B}\right) \\
&= g'(x) + C \cdot \left(-\frac{\epsilon}{B} + \frac{x_1\epsilon}{B^2} + \frac{\epsilon}{B} - \frac{x_2\epsilon}{B^2} - \frac{\epsilon^2}{B^2}\right) \\
&= g'(x) + C \cdot \left(\frac{(x_1 - x_2)\epsilon}{B^2} - \frac{\epsilon^2}{B^2}\right) \\
&= g'(x) + C \cdot \left(\frac{(x_1 - x_2)^2}{2B^2} - \frac{(x_1 - x_2)^2}{4B^2}\right) \\
&= g'(x) + C \cdot \frac{(x_1 - x_2)^2}{4B^2} \\
&> g'(x).
\end{aligned}$$

This shows that a vector x does not maximize g' whenever not all elements of x are equal, and thus establishes the claim. □

These results suffice to prove the main result within this section.

Proof of Theorem 3.2. Let ℓ be the total number of iterations of Step 2 of Algorithm $\mathcal{A}$. If the solution returned by the algorithm covers all elements, then the solution is optimal.

Otherwise, the solution returned by the algorithm may not be optimal, in which case it holds that σ_ℓ violates the budget. Lemma 3.3 can be applied to iteration ℓ in order to derive

$$\begin{aligned}
w(\sigma_\ell^{\mathcal{A}}) &\geq w(\sigma^*)\left(1 - \prod_{k=1}^{\ell}\left(1 - \frac{\alpha \cdot c(S_k^{\mathcal{A}})}{B}\right)\right) \\
&\geq w(\sigma^*)\left(1 - \prod_{k=1}^{\ell}\left(1 - \frac{\alpha \cdot c(S_k^{\mathcal{A}})}{\sum_{S \in \sigma_\ell^{\mathcal{A}}} c(S)}\right)\right) \\
&\geq w(\sigma^*)\left(1 - \prod_{k=1}^{\ell}\left(1 - \frac{\alpha \sum_{S \in \sigma_\ell^{\mathcal{A}}} c(S)/\ell}{\sum_{S \in \sigma_\ell^{\mathcal{A}}} c(S)}\right)\right) \\
&= w(\sigma^*)\left(1 - \left(1 - \frac{\alpha}{\ell}\right)^{\ell}\right) \\
&\geq w(\sigma^*)\left(1 - \frac{1}{e^\alpha}\right),
\end{aligned}$$

where the second inequality follows from the fact that σ_ℓ violates the budget, and the third inequality follows from Lemma 3.4.

The algorithm outputs a set with a profit of $\max\{w(\sigma_{\ell-1}^{\mathcal{A}}), w(S_\ell^{\mathcal{A}})\}$ and from the above derivation it follows that:

$$\max\left\{w(\sigma_{\ell-1}^{\mathcal{A}}), w(S_\ell^{\mathcal{A}})\right\} \geq \frac{1}{2}(w(\sigma_{\ell-1}^{\mathcal{A}}) + w(S_\ell^{\mathcal{A}})) \geq \frac{1}{2}w(\sigma_\ell^{\mathcal{A}}) \geq \frac{1}{2}\left(1 - \frac{1}{e^\alpha}\right)w(\sigma^*),$$

which proves the claim. □

3.3.2 Approximation algorithm

As mentioned at the beginning of this section, a $\frac{1}{2}(1-\frac{1}{\sqrt{e}})$-approximation algorithm for the BJCP will be presented when every set has cardinality 2. This is done by reducing the problem to the BMCP. An instance I of the BJCP will be reduced to an instance $r(I)$ of the BMCP on the same set of elements, such that the optimal solution of $r(I)$ has the same profit as the optimal solution of I.

The instance $r(I)$ may have a superpolynomial number of sets. However, instead of generating the BMCP instance explicitly, only a $\frac{1}{2}$-approximate cost-efficiency oracle $f_{r(I)}$ for $r(I)$ will be constructed. Afterwards, Algorithm $\mathcal{A}$ on $f_{r(I)}$ will be used in order to obtain a $\frac{1}{2}(1-\frac{1}{\sqrt{e}})$-approximately optimal solution to $r(I)$ in polynomial time. Last, it will be shown how to transform in polynomial time a feasible solution for $r(I)$ into a feasible solution for I with equal profit. This explanation of this procedure starts with defining the reduction r.

Definition 3.4. *Let $I = (F, J, c, w, B)$ be an instance of the BJCP where all sets have cardinality 2. Define the BMCP instance $r(I) = (F', X, c', w, B)$, where*

- $F' = \bigcup_{j \in J} F'_j$ *with* $F'_j = \{S \cup \{j\} : |S| \geq 1, \forall i \in S : \{i, j\} \in F\}$, *and*
- $c'(S) = \sum_{j \in S} c_j$ *for every* $S \in F$.

The set of elements (or jobs), profit functions and budgets of I and $r(I)$ are equal.

That is, F'_j consists of the sets that all contain the element/job j, including a subset of the jobs in J that share a set with j in F. As an example, consider the following instance $I = (F, J, c, w, B)$ of the BJCP with set cardinality 2 in Figure 2. In this illustration, every pair of elements that share a set in F are connected through an edge.

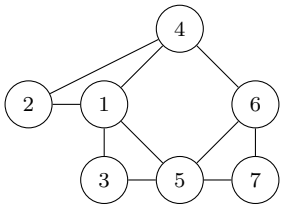

Figure 3.2: Example of the BJCP with set cardinality 2

Then, for element 1, the corresponding subfamily of sets F'_1 for the newly defined BMCP instance consists of the following subsets:

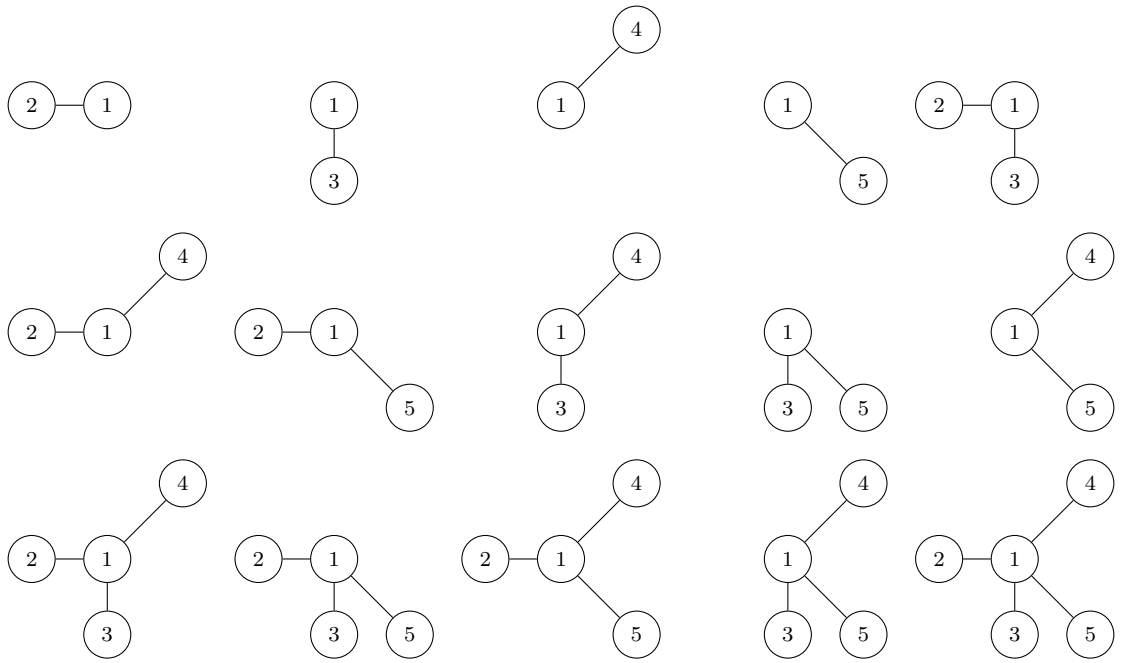

Figure 3.3: Illustration of F'_1 for the newly created BMCP instance.

In general, if j is contained in n_j sets (of cardinality 2) in F, $|F'_j| = 2^{n_j} - 1$ (all possible combinations, excluding the empty set). Even though the number of sets in F is not polynomial in the input size of the BJCP, recall that this instance will not be generated explicitly. A set in F'_j for the reduced instance will further be also referred to as a *star*, centered at j. Recall that c_j is the cost of an element in the original BJCP, while c' is a function that assigns a cost to every set in F'.

It will first be shown that every feasible solution σ' for $r(I)$ can be transformed into a feasible solution σ for I in polynomial time such that the profit is preserved. Consider the following function g_I that maps solutions of $r(I)$ to I:

Definition 3.5. *Let $I = (F, J, c, w, B)$ be an instance of BJCP and let σ' be a feasible solution for $r(I) = (F', X, c', w, B)$. The function g_I maps σ' to the following solution σ for I.*

$$g_I(\sigma') = \{\{j, j'\} \in F : j, j' \in X(\sigma')\}.$$

In other words, $g_I(\sigma')$ is the set of subsets of cardinality 2 of F that are contained in (a set of) σ'.

Lemma 3.5. *If σ' is a feasible solution for $r(I)$, then the transformed solution $\sigma = g_I(\sigma')$ is computable in time $O(mn|\sigma'|)$. Moreover, $g_I(\sigma')$ is feasible, i.e., the total cost of all elements covered by $g_I(\sigma')$ does not exceed B. Also, $w(\sigma') = w(g_I(\sigma'))$.*

Proof. For the first claim, observe that for each set $S' \in \sigma'$ and 2-set $S \in F$, it needs to be checked whether $S \subseteq S'$. This can be done in $O(n)$ time.

The second claim follows from the fact that $g_I(\sigma')$ covers the same elements as σ', and by definition

$$B \geq \sum_{S \in \sigma'} c'(S) = \sum_{j \in X(\sigma')} c_j \cdot |\{S' \in \sigma' : j \in S'\}| \geq \sum_{j \in X(\sigma')} c_j.$$

The third claim follows from the fact that $g_I(\sigma')$ covers the same set of elements as σ', and the weights of the elements are equal in both instances. □

Next will be shown that the optimal solution for I is at most the profit of the optimal solution for $r(I)$. Combined with the previous lemma, this entails that the optimal profits of I and $r(I)$ are equal.

Lemma 3.6. *Let w_{opt} be the maximum weight achievable in instance I. There exists a solution for $r(I)$ with weight w_{opt}.*

Proof. Let σ be a weight-maximizing feasible solution for I. Assume without loss of generality that no 2-set in σ covers two elements that are both already covered by other 2-sets in σ. Otherwise, such a set can be removed from σ without decreasing the profit. Under this assumption, σ is a set of stars.

A feasible solution σ' for $r(I)$ with equal profit can be constructed from σ as follows. Define σ' to be the collection of sets that correspond to the maximal stars of σ, i.e., for each maximal star of σ, we add to σ' the set consisting of the elements covered by the star.

Since no pair of sets (or stars) in σ' intersects, by definition of c' the total cost $\sum_{S \in \sigma'} c'(S)$ equals $\sum_{j \in X(\sigma)} c_j \leq B$, and therefore σ' is a feasible solution for $r(I)$. Moreover, σ and σ' cover the same set of elements, and therefore $w(\sigma)$ in I equals $w(\sigma')$ in $r(I)$. □

A final ingredient that is required is the $\frac{1}{2}$-approximate cost-efficiency oracle f for $r(I)$. Recall that Y represents the set of elements already covered during the execution of Algorithm $\mathcal{A}$.

Let Y be the input argument to f. The high level idea is that for each element j, a set of elements T_i of the star centered at j will be computed using a greedy procedure. The goal for each of these stars is to select for each such i the substar that is at least α times the highest possible cost-efficiency, such that the cost of the elements in the substar does not exceed the budget. The output will be the set in $\{T_j : j \in X\}$ with the highest cost-efficiency. The function f will be defined as Algorithm $\mathcal{B}$ as follows.

1. Let X' be subset of elements of X that have at least one attached element (i.e., are contained in the same set in F) that is not in Y. For each $i \in X'$ (note that i itself may be in Y):

   (a) Initialize $T_j := \{j\}$, and $d_j = c_j$. If $j \in Y$, set $p_j := 0$, and otherwise set $p_j := w_j$. Throughout this step, $\frac{p_j}{d_j}$ represents the cost-efficiency of the substar T_j.

   (b) Order non-increasingly the elements j' that are not in Y and are attached to j in F, according to ratio $\frac{w_{j'}}{c_{j'}}$. Denote this ordering by Ω_j.

(c) Let j' be the next element of Ω_i, starting with the first element. If $\frac{p_j+w_{j'}}{d_j+c_{j'}} \geq \frac{p_j}{d_j}$, then add j' to T_j, set $p_j := p_j + w_{j'}$, and set $d_j := d_j + c_{j'}$, and repeat this step if $d_j \leq B$. Otherwise, if $\frac{p_j+w_{j'}}{d_j+c_{j'}} < \frac{p_j}{d_j}$ or if $d_j > B$, proceed to the next step.

(d) If $d_j \leq B$, skip this step. This would mean the entire star centered at j is the most cost-efficient set, and does not exceed the budget.

Otherwise, let j' be the element last added in the previous step, being the element in T_j with lowest $\frac{w_{j'}}{c_{j'}}$. Consider two substars of T_j that are within the budget: $\{j, j'\}$ and $T_j \setminus \{j'\}$. Set T_j to be the substar with the highest cost-efficiency. Formally:

i. If $j \notin Y$: if $\frac{w_j+w_{j'}}{c_j+c_{j'}} \geq \frac{p_j-w_{j'}}{d_j-c_{j'}}$, then set $T_j = \{j, j'\}$, $p_j := w_j + w_{j'}$ and $d_j := c_j + c_{j'}$. Otherwise, set $T_j := T_j \setminus \{j'\}$, $p_j := p_j - w_{j'}$, and $d_j := d_j - c_{j'}$.

ii. If $j \in Y$: if $\frac{w_{j'}}{c_j+c_{j'}} \geq \frac{p_j-w_{j'}}{d_j-w_{j'}}$, then set $T_j := \{j, j'\}$, $p_j := w_{j'}$ and $d_j := c_j + c_{j'}$. Otherwise, set $T_j := T_j \setminus \{j'\}$, $p_j := p_j - w_{j'}$, and $d_j := d_j - c_{j'}$.

2. Output the set in $\{T_j : |T_j| \geq 2, j \in X'\}$ with the highest cost-efficiency, i.e., the ratio $\frac{n_j}{d_j}$.

Note that it is not needed to consider those $j \in X'$ for which $|T_j| = 1$. It can be easily verified that in this case, the optimal star centered at j is a single set $\{j, j'\}$. However, this set is also in $F'_{j'}$, and it is necessarily true that $|T'_j| \geq 2$.

Lemma 3.7. *The function f is a $\frac{1}{2}$-approximate cost-efficiency oracle for $r(I)$ and can be computed in time $O(n^2)$.*

Proof. It is easy to see that the set output by f is always a set in F', as it always outputs a set that corresponds to star of F. Moreover, the set $\{T_j : |T_j| \geq 2, j \in X'\}$ is never empty when $Y \neq X$ by definition of X'. This implies that f is a valid cost-efficiency oracle. It is also straightforward to see that f runs in time n^2. For each element, all its attached elements are considered, which each takes a constant amount of time.

It remains to prove the approximation factor. Consider the set $\{T_j : |T_j| \geq 2, j \in X'\}$. It suffices to show that for each $j \in X'$ for which $|T_j| \geq 2$, the ratio $\frac{n_j}{d_j}$ is at least $\frac{1}{2} \cdot \sum_{j \in S \setminus Y} \frac{w_j}{c(S)}$ for all $S \in F'_j$. In other words, the cost-efficiency $\frac{n_j}{d_j}$ of the set T_j is at least half the maximum cost-efficiency among all stars in F'_j, with respect to the input set Y.

Let $j \in X'$ such that $|T_j| \geq 2$. For the analysis, T_j will be compared to an optimal *fractional* substar centered at j, $T_j^{*\text{frac}}$. In this fractional solution, each of the elements j' attached to j and not in Y is picked with a certain fraction $t_{j'}^{*\text{frac}} \in [0, 1]$, and element j is selected with fraction $t_j^{*\text{frac}} = 1$. The cost-efficiency

of adding $T_j^{*\text{frac}}$ is defined as:

$$\frac{\sum_{j' \in T_j^{*\text{frac}} \setminus Y} t_{j'}^{*\text{frac}} w_{j'}}{\sum_{j' \in T_j^{*\text{frac}} \setminus Y} t_{j'}^{*\text{frac}} c_{j'}}.$$

It holds that the cost-efficiency of $T_j^{*\text{frac}}$ is at least the cost-efficiency of the substar centered at i that would maximize the integral variant.

Note that $T_j^{*\text{frac}}$ is obtained by selecting j, and afterwards greedily selecting elements attached to j according to non-increasing cost-efficiency, i.e., according to the order Ω_j as given in Algorithm $\mathcal{B}$. A considered element is selected with the highest possible fraction as long as the budget is not exceeded, and as long as adding the element increases the cost-efficiency of the solution. Hence, in $T_j^{*\text{frac}}$ all elements are selected with either fraction 0 or 1, except at most one element, which is selected with a fraction in $(0, 1)$.

To see why this is true, suppose for contradiction that $T_j^{*\text{frac}}$ is optimal but has a different structure.

- Suppose an element $j' \in T_j^{*\text{frac}}$ with $t_{j'}^{*\text{frac}} > 0$ such that the cost-efficiency of j' is less than the cost-efficiency of $T_j^{*\text{frac}}$. Then, setting $t_{j'}^{*\text{frac}}$ to 0 will increase the cost-efficiency of the solution, contradicting the optimality of $T_j^{*\text{frac}}$. Thus, every selected element has at least the cost-efficiency of $T_j^{*\text{frac}}$.

- Suppose there are two elements $j', j'' \in T_j^{*\text{frac}} \setminus j$ for which $t_{j'}^{*\text{frac}} < 1$ and $t_{j''}^{*\text{frac}} > 0$, and the cost-efficiency of j' exceeds that of j''. Then, decreasing $t_{j''}^{*\text{frac}}$ by an amount ϵ and increasing $t_{j'}^{*\text{frac}}$ by a maximal amount would increase the cost-efficiency (for a suitably small choice of ϵ), contradicting the optimality of $T_j^{*\text{frac}}$.

This shows that $T_j^{*\text{frac}}$ is obtained by the aforementioned greedy procedure.

Next, observe that if $T_j^{*\text{frac}}$ happens to be integral, then the set of integrally selected elements is precisely T_j, meaning that T_j is the set that maximizes cost-efficiency. In this case, the claim is proved.

Now consider the case that $T_j^{*\text{frac}}$ is not integral. Let j' be the element that is fractionally selected in $T_j^{*\text{frac}}$ and let S' be the integral elements in $T_j^{*\text{frac}}$ excluding j, i.e., $S' = \{j'' : j'' \neq j, t_{j''}^{*\text{frac}} = 1\}$. It follows from Definition 3.3 that T_j is either the set $\{j\} \cup S$ or the set $\{j, j'\}$.

Four (similar) subcases will be distinguished.

- Suppose $j \notin S$ and $w_{j'} \geq \sum_{j'' \in S'} w_{j''}$. Because $T_j^{*\text{frac}}$ is the optimal fractional substar centered at j, the cost-efficiency of $S' \cup \{j, j'\}$ exceeds the optimal fractional solution and thus also the cost-efficiency of the optimal integral

substar T_j^*. Therefore, the cost-efficiency of T_j:

$$\begin{aligned}\frac{w_j + w_{j'}}{c_j + c_{j'}} &\geq \frac{w_j + w_{j'}}{c_j + c_{j'} + \sum_{j'' \in S'} c_{j''}} \\ &\geq \frac{1}{2} \cdot \frac{w_j + w_{j'} + \sum_{j'' \in S'} w_{j''}}{c_j + c_{j'} + \sum_{j'' \in S'} c_{j''}} \\ &\geq \frac{1}{2} \cdot \frac{\sum_{j'' \in T_j^*} w_{j''}}{\sum_{j'' \in T_j^*} c_{j''}},\end{aligned}$$

as needed.

- Suppose $j \notin S$ and $w_{j'} < \sum_{j'' \in S'} w_{j''}$. Similarly, the cost-efficiency of T_j is:

$$\begin{aligned}\frac{w_j + \sum_{j'' \in S'} w_{j''}}{c_j + \sum_{j'' \in S'} c_{ji''}} &\geq \frac{w_j + \sum_{j'' \in S'} w_{j''}}{c_j + c_{j'} + \sum_{j'' \in S'} c_{j''}} \\ &\geq \frac{1}{2} \cdot \frac{w_j + w_{j'} + \sum_{j'' \in S'} w_{j''}}{c_j + c_{j'} + \sum_{j'' \in S'} c_{j''}} \\ &\geq \frac{1}{2} \cdot \frac{\sum_{j'' \in T_j^*} w_{j''}}{\sum_{j'' \in T_j^*} c_{j''}}.\end{aligned}$$

- The remaining two cases are analogous to the above two, where w_j is replaced with 0.

□

The lemmas provided in this section suffice to present a $\frac{1}{2}(1 - \frac{1}{\sqrt{e}})$-approximation algorithm for the BJCP with set cardinality 2. This algorithm is referred to as Algorithm $\mathcal{C}$ and is defined as follows. Let $I = (G = (V, E), c, p, B)$ be an input instance of the BJCP with set cardinality 2.

- Run Algorithm $\mathcal{A}$ using the $\frac{1}{2}$-approximate cost-efficiency oracle f described as Algorithm $\mathcal{B}$ on the reduced instance $r(I)$ given in Definition 3.4. This results in a solution σ^{BMCP} for instance $r(I)$.
- Compute and output $\sigma = g_I(\sigma')$ (see Definition 3.5).

The correctness, polynomial running time and approximation factor of $\frac{1}{2}(1 - \frac{1}{\sqrt{e}})$ of Algorithm $\mathcal{C}$ follow from Theorem 3.2, Lemma 3.5, Lemma 3.6 and Lemma 3.7.

3.4 The acyclic BJCP

In this section, a bi-level dynamic program will be proposed when the incidence graph is a forest (see Figure 3.1 for an example). This special case will be referred to as BJCP-Forest. Furthermore, it will be shown that this dynamic program can be turned into a fully polynomial time approximation scheme (FPTAS).

3.4.1 Dynamic program

Before proposing algorithms for the acyclic BJCP, define W as the maximum weight of an element, i.e., $W = \max_{j \in J} w_j$.

Theorem 3.3. *The BJCP-Forest can be solved optimally in time* $\mathcal{O}(mn^3W^2)$.

Before proving Theorem 3.3, several concepts will be introduced first. Suppose the given incidence graph is a forest and consists of z trees $T_1, \ldots, T_z$. In order to facilitate the exposition of our dynamic program, combine these trees simply into a single tree T as follows. Introduce for each tree T_t with $t \in [z]$ a dummy job $(n+t)$ with zero weight and cost, i.e., $w_{n+t} = c_{n+t} = 0$, and add it to an arbitrary set in T_t. Finally, define $S_0 = \{n+1, \ldots, n+z\}$ such that the incidence graph forms one single tree, where S_0 can be interpreted as the root. Because the newly added jobs do not have any weight or cost, the optimal solution value does not change.

Furthermore, note that within the incidence graph, the vertices along a path from the root to any other vertex correspond alternately to sets in F and elements in X. For the ease of understanding, assume that this tree is "unfolded" in order to draw the bipartite graph in a layered manner, as illustrated in Figure 3.4.

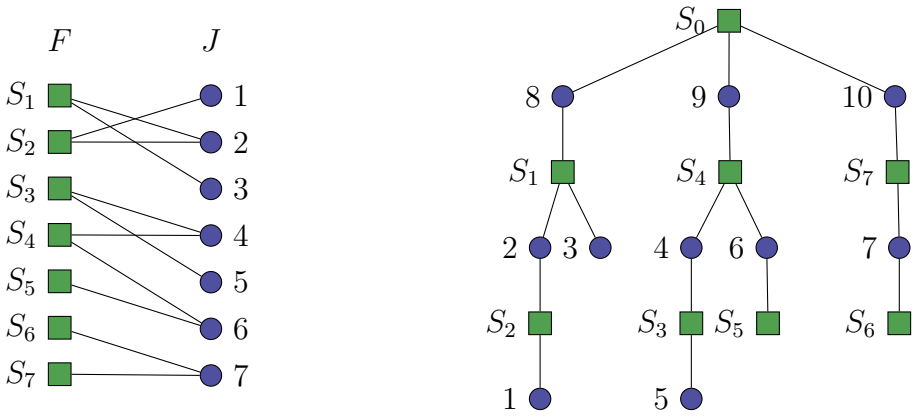

Figure 3.4: Example of an acyclic incidence graph (left) and its "unfolded' tree (right).

The dynamic program proposed in this section processes the unfolded tree T in a bottom-up manner. It consists of two separate dynamic programs: one for the subsets $S_0, \ldots, S_m$, and one for the jobs $1, \ldots, n+z$. Before defining these properly, some intuition will be provided first.

Subtrees of subtrees. Consider an arbitrary subtree in T rooted at either a vertex represented by a resource jobset S_i or a job j. Both dynamic programs rely on the fact that a subset of this subtree can be solved to optimality, which immediately can be used to solve a greater subset to optimality. In case the subtree is rooted at a vertex represented by a resource jobset, only the subtree up until the first k children in the subtree of the resource jobset are considered. Once the optimal solutions (minimum required cost to obtain a specific weight, if possible)

are known for every possible weight (upper bounded by nW), it is possible to find optimal solutions for the subtree until the first $k+1$ children by linear enumeration. A similar, but slightly adapted method works in case the subtree is rooted at a job.

To this aim, introduce the following notation. Given a resource jobset $S_i \in F$, represented by a (circled) vertex in the tree T, let $T(S_i)$ refer to the subtree of T rooted at S_i. Define $\delta_T(S_i) \subseteq J$ as the set of children of S_i in the tree T. For simplicity, the same notation $T(j)$ and $\delta(j) \subseteq F$ is adopted for the subtree rooted at a job j and the set of children of the vertex associated to job j, respectively.

The dynamic program considers even more specific subtrees. Furthermore, define $T(S_i, k)$ with $k \in \delta_T(S_i)$ as the subtree of $T(S_i)$ induced by the union of the vertex associated to S_i and the subtrees of the children of S_i with job index at most k, i.e.:

$$T(S_i, k) = \{k\} \cup \bigcup_{j \in \delta_T(S_i): j \leq k} T(j).$$

Finally, define $T(j, S_\ell)$ as the subtree induced by the union of the subtrees of the children of S_i with resource subset index at most ℓ. However, while $T(S_i, k)$ contains the root job k, the subtree dynamic program for a job does not include the vertex representing that job.

$$T(j, S_\ell) = \bigcup_{S_i \in \delta_T(j): i \leq \ell} T(S_i).$$

This distinction is crucial to make the dynamic program work. The reason is that whenever a resource set S_i is selected in a solution, all jobs $j \in S_i$ have to be processed as well. Therefore, the optimal solution for all subtrees rooted at those jobs need to be taken into account. But conversely, when a job j is selected in a solution, this does not imply that all subsets S_i for which $j \in S_i$ need to be selected in that solution; *at least* one of them has to be selected. The subtrees which the two dynamic programs will consider, are visualized in Figure 3.5.

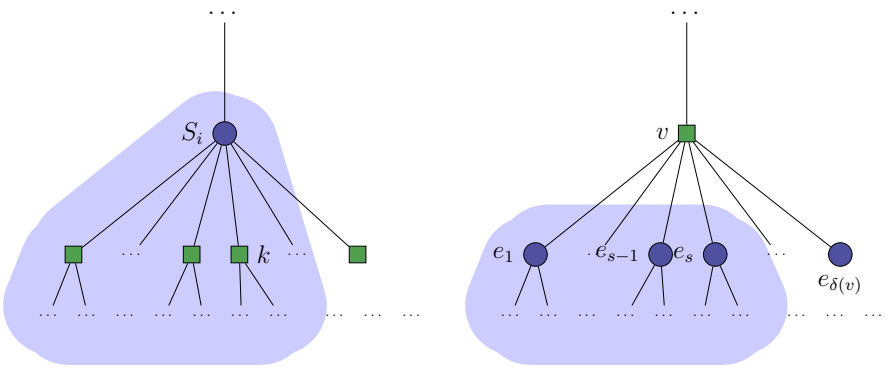

Figure 3.5: Illustration of $T(S_i)$ and $T(S_i, k)$ (left), and $T(j)$ and $T(j, S_\ell)$ (right).

These concepts provide a basis for proposing a dynamic program for the acyclic BJCP.

Proof of Theorem 3.3. Consider the following dynamic program for the resource jobsets. For each $j \in \delta(S_i)$, $x \in \{0,1\}$, and $w = 0, \ldots, nW$, define:

$$f(S_i, k, x, w) = \text{minimum total cost solution in the subtree } T(S_i, k) \text{ such that the total weight is exactly } w \text{ and resource jobset } S_i \text{ is selected in the solution } (x = 1) \text{ or not } (x = 0).$$

The dynamic program for the subtrees rooted at a vertex represented by a job is slightly more complex. For every $s \in \delta(v)$, $x \in \{0,1\}$ and $w = 0, \ldots, nW$, define:

$$g(j, \ell, x, w) = \text{minimum total cost solution in the subtree } T(j, S_\ell) \text{ such that the total weight is exactly } w \text{ and } \textit{at least} \text{ one of the children resource subsets in } \Delta(v) \text{ is selected in the solution } (x = 1) \text{ or not } (x = 0).$$

The unfolded tree will be computed f and g layer by layer in a bottom-up manner, where the layer of a vertex is the distance to the root, until eventually f for the root node 0 is determined. In particular, the total weight of an optimal solution can then be determined by:

$$OPT = \max\{w \in [nW]_0 : f(0, z, 0, p) \leq B\},$$

i.e., the maximum total weight possible in the entire tree.

The formal definitions of the dynamic programs are as follows. For a resource subset $S_i \in F$, $k \in \delta(S_i)$, and $w \in [nW]_0$, the recurrence is:

$$f(S_i, k, x, p) = \min_{y \in \{0,1\}, r \in [W]_0} \big\{ f(S_i, k', x, w - r - \max\{x, y\} \cdot w_k) + g(k, S_{m(k)}, y, r) + \max\{x, y\} \cdot c_k \big\}.$$

where k' is the last processed child of S_i in the dynamic program before child k, i.e., $k' = \arg\max\{j \in \delta(S_i) : j < k\}$, and $m(k)$ is the resource set index of last child of k, i.e., $m(k) = \arg\max \delta(k)$. If k is the first processed child of S_i, there exists no k and the term containing k' drops.

To see why this recurrence holds, fix some $w \in [W]_0$. Then, depending on whether job k is chosen or not, the minimum cost to achieve profit r on the subtrees in $T(k, S_{m(k)})$ is either $g(k, m(k), 0, r)$ or $g(k, m(k), 1, r)$.

Further, job k itself contributes a cost of $\max\{x, y\} \cdot c_k$ to the objective function, since the costs c_k only needs to be added if $x = 1$ (parent resource jobset of k is chosen) or $y = 1$ (at least one child resource jobset of k is chosen), or both. The minimum cost to obtain the remaining weight of $w - r - \max\{x, y\} \cdot w_k$ on the subtree $T(S_i, k')$ is $f(S_i, k', x, w - r - \max\{x, y\} \cdot w_k)$ by definition.

For the dynamic program a job $j \in J$, $S_\ell \in \delta(j)$, and $w \in [nW]_0$, two cases are distinguished. In case $x = 0$ (job j is not selected), obtain a similar recurrence as for the resource subsets:

$$g(j, S_\ell, 0, w) = \min_{r \in [w]_0} \big\{ g(j, S_{\ell'}, 0, w - r) + f(S_\ell, n(\ell), 0, r) \big\}.$$

where $S_{\ell'}$ is the last processed child of j in the dynamic program before child S_ℓ, i.e., $\ell' = \arg\max\{i \in \Delta(j) : i < \ell\}$, and $n(\ell)$ is the job index of the last child of ℓ,

i.e., $n(\ell) = \arg\max \Delta(S_\ell)$. If S_ℓ is the first processed child of j, there exists no ℓ' and the term containing ℓ' drops.

To see why this recurrence holds, fix some $r \in [p]_0$. If $y = 0$, thus if all resource sets in $\Delta(j)$ are not selected in the solution, then the case has to be considered where all previously processed children before S_ℓ are not selected (first term), combined with the case that child S_ℓ is also not selected (second term).

However, there are three possibilities to ensure that at least one child of job j is selected in the optimal solution, i.e., to ensure $y = 1$:

- at least one child in the previously processed children of job j is selected, while S_ℓ is not selected,
- none of the previously processed children of job j is selected, while S_ℓ is selected, or,
- at least one previously processed child of job j is selected, and S_ℓ is selected.

These three cases are described in the dynamic program as follows:

$$g(j,\ell,1,w) = \min_{r\in[w]_0} \min\big\{ g(j,S_{\ell'},0,p-r) + f(S_\ell,n(\ell),1,r), \\ \min_{x\in\{0,1\}} g(j,S_{\ell'},1,p-r) + f(S_\ell,n(\ell),\min\{x,y\},r\big\}.$$

The last two possibilities are combined in the last minimum term.

The proof concludes with an analysis of the running time. Computing the value of $f(S_i,k,x,w)$ for fixed parameters takes time at most $\mathcal{O}(nW)$, since $w \in [nW]_0$. This has to be done at most n^2W times for every hyperedge; thus time $\mathcal{O}(mn^3W^2)$ in total. Similarly, computing the value of $g(j,S_\ell,x,w)$ for fixed parameters takes time at most $\mathcal{O}(nW)$ and this has to be done mnW times for every vertex; thus time $\mathcal{O}(mn^3W^2)$ in total. □

3.4.2 FPTAS

Furthermore, standard techniques (truncation) can be employed to turn the above pseudo-polynomial time algorithm into an FPTAS, i.e., an algorithm that takes an error parameter $\varepsilon > 0$ and computes in time polynomial in the input size and $1/\varepsilon$ a $(1-\varepsilon)$-approximation to the optimal solution.

Theorem 3.4. *There exists an FPTAS for BJCP-Forest that runs in $\mathcal{O}(mn^5/\varepsilon^2)$ time.*

Let $x_j = 1$ if job j is selected in a solution, and $x_j = 0$ otherwise. The optimal selection of jobs to BJCP-Forest is then denoted by $x^* = \{x_1^*,\ldots,x_m^*\}$. The idea is to truncate the values that the weight function takes on, i.e., for all $j \in [n]$:

$$w_j = \left\lfloor \frac{w_j}{10^t} \right\rfloor,$$

after which a solution is determined via dynamic programming. With these truncated profits, a solution can be determined in $\mathcal{O}(mn^3(\frac{W}{10^t})^2)$ instead of $\mathcal{O}(mn^3W^2)$.

However, the optimal selection of jobs using the truncated profits, x', could be sub-optimal compared to the optimal selection x^* with the original profit functions. Yet, a useful relationship holds between the two solutions, which we use in order to derive an FPTAS, namely,

$$\sum_{j=1}^{n} w_j x'_j \geq \sum_{j=1}^{n} 10^t w'_j x'_j \geq \sum_{j=1}^{n} 10^t w'_j x^*_j$$

The first inequality follows directly from the definition of w'_j, while the second inequality follows from the fact that x' is optimal in the problem with truncated profits. Note that x' corresponds to a feasible solution, since the cost functions remain the same; only the weights are potentially reduced. Furthermore, due to rounding down, it must be that $10^t w_j x^*_i \geq w_j \cdot x^*_j - 10^t$. Combining all mentioned (in)equalities gives:

$$\begin{aligned}
\sum_{j=1}^{n} w_j \cdot x^*_j &\geq \sum_{j=1}^{n} (w_j x^*_j - 10^t) \\
&= \sum_{j=1}^{n} w_j x^*_j - n \cdot 10^t \\
&= OPT - n \cdot 10^t \\
&= OPT \left(1 - \frac{n \cdot 10^t}{OPT}\right) \\
&\geq OPT \left(1 - \frac{n \cdot 10^t}{W}\right),
\end{aligned}$$

where the last inequality holds because $OPT \geq W$. Now fix $t = \log_{10}(\varepsilon W/n)$ for some given $\varepsilon > 0$, such that $n \cdot 10^t/W = \varepsilon$. Then, by combining all mentioned (in)equalities, conclude that

$$\sum_{j=1}^{n} w_j \cdot x'_i \geq (1 - \varepsilon) OPT,$$

meaning that an $(1 - \varepsilon)$-approximate solution for BJCP-Forest is obtained. Since the running time of the dynamic program using truncated profits is $\mathcal{O}(mn^3 W^2/10^{2t})$ and because $n \cdot 10^t/W = \varepsilon$, the running time of the new algorithm is $\mathcal{O}(mn^5/\varepsilon^2)$, which directly results in Theorem 3.4.

3.5 The feedback vertex set bounded BJCP

In Section 3.4 it is shown that the BJCP-Forest can be solved in pseudo-polynomial time and that there is an FPTAS. This implies that the inapproximability of the general problem is caused by the cycles in the incidence graph. Following up on that analysis, this section provides a simple fixed parameter tractability result that allows to handle incidence graphs with cycles to a certain extent. Some simple and well-known concepts are used.

Definition 3.6. *A **feedback vertex set** of a graph $G = (V, E)$ is a set of vertices $Z \subseteq V$ such that G with the vertices from Z deleted is cycle-free.*

The problem of finding a minimum feedback vertex set is NP-hard in general, but it is fixed parameter tractable. An $\mathcal{O}(3.83^{\alpha}\alpha n^2)$ time algorithm is given in [13] to solve the FVSP, where n here refers to the number of vertices in the graph. This result is used for the following theorem.

Theorem 3.5. *BJCP is solvable in $\mathcal{O}(mn^3W^2 2^{\alpha} + 3.83^{\alpha}\alpha m^2)$ time.*

Proof. Given a BJCP-instance $I = (F, X, c, w, B)$, the feedback vertex set can be found by reducing the incidence graph $G = (F \cup J, E)$ to a undirected graph $G' = (F, E')$, where:

$$E' = \{(i, i') : (\exists j \in J : j \in S_i \wedge j \in S_{i'})\}.$$

That is, every resource subset is represented by a vertex, and two vertices are connected if there exists at least one job in both of their corresponding resource jobsets. It is now easy to see that there is a one-to-one correspondence between the cycles in the incidence graph G and the cycles in G', and each cycle in G corresponds to a cycle in G' on the same set of vertices. See Figure 3.6 for an example. Basically, all vertices in J are removed from the graph, but the edge are not removed, but rather extended.

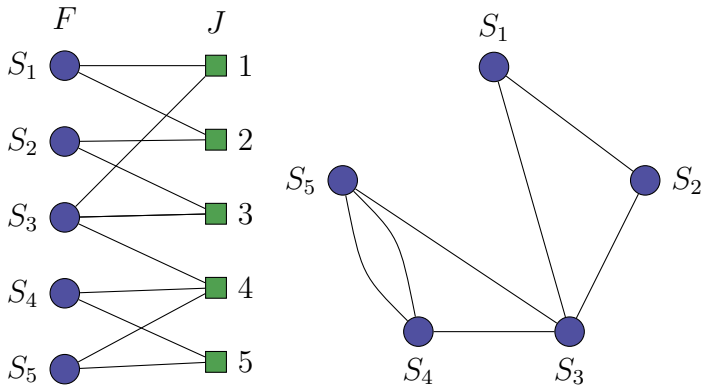

Figure 3.6: Example of a transformation to the feedback vertex set problem

Using the result from [13], a feedback vertex set $Z \subseteq F$ can be found in $\mathcal{O}(3.83^{\alpha}\alpha n^2)$.

The remainder of the proof is straightforward, as it is based on complete enumeration. For each possible subset $Z' \subseteq Z$ (i.e., 2^{α} combinations), consider the subproblem of instance I where the resource jobsets in Z' are already selected, i.e., the instance:

$$I' = \left(F \setminus Z, X \setminus \left(\bigcup_{S_i \in Z'} S_i \right), c, w, B - \sum_{i \in \bigcup\{S_i \in Z'\}} c_i \right)$$

Since the incidence graph of I' is acyclic, the dynamic program used to prove Theorem 3.3 can be used to solve I'. Solving this problem for fixing every possible Z', and taking the best solution, yields a solution to the general BJCP. The running time follows directly from this procedure and Theorem 3.3. □

3.6 Conclusions and future work

This chapter considered the Budgeted Job Coverage Problem, where the goal is to select a subset of a given family of resource jobsets over a set of jobs with costs and weights. The goal is to maximize the total weight while not exceeding a budget. Because costs are assigned to jobs (or elements) instead of sets, the analyses from related work, particularly on greedy approaches for maximum coverage problems, cannot be applied. Instead, special cases of the BJCP have been considered.

In case each resource jobset has cardinality 2, a $\frac{1}{2}(1 - \frac{1}{\sqrt{e}})$-approximation algorithm can be obtained (see Section 3.3). Although this algorithm crucially exploits several characteristics of the problem, this approach is hard and tedious to generalize for fixed cardinality k.

Moreover, it has been shown that in case the corresponding incidence graph is acyclic, the problem can be solved in pseudo-polynomial time using a bi-level dynamic program. This result suggest that cycles increase the computational complexity of the problem significantly. However, there exist no straightforward way to remove cycles from the graph, as there exists no efficient algorithm to find and process a feedback vertex set.

Future work. Clearly, the most interesting open problem that remains to be solved is whether there exists a constant factor approximation algorithm for the general BJCP that runs in polynomial time. Such a result would form a very interesting contrast with the inapproximability result for the problem of submodular function maximization with a submodular budget constraint as mentioned in Section 3.2.3.

Alternatively, an interesting and challenging intermediate goal would be to find a constant factor approximation algorithm for the case that the resource subsets have a fixed size k, i.e., to generalize the result in Section 3.3. Algorithm $\mathcal{B}$ and Theorem 3.2 might serve as a useful tool for achieving this goal. Another option is to research the possibities to find a feedback vertex set for the BJCP efficiently, but this requires also to find a efficient way to enumerate through this set, in order to find a polynomial time algorithm for the BJCP.

Chapter 4

Crew Scheduling

This chapter introduces the European Crew Scheduling Problem (ECSP), where the goal is to assign trips with fixed departure and arrival times to bus drivers while minimizing the sum of all duty times. The additional complexity comes from constraints on the driving, resting and duty times that apply to bus drivers in the European Union, because the complexity of these constraints make the majority of the known models in the literature less suitable for this problem.

After discussing the background of crew scheduling problems in Section 4.1, the problem under consideration will be defined in Section 4.2. Subsequently, a subset of the constraints from the ECSP will be considered such that a column generation approach can be proposed in Section 4.3. Afterwards, some simple heuristics will be proposed for the full ECSP in Section 4.4. Most approaches have been tested on experimental data, for which the results are reported in Section 4.5. The chapter ends with a summary of its conclusions and suggestions for future work in Section 4.6.

4.1 Background

The Crew Scheduling Problem (CSP) is a general problem of assigning a set of tasks to a group of workers (a *crew*), usually with the objective to minimize costs. This problem can appear in many transportation contexts, such as bus and rail transit, truck and rail freight transport, and passenger air transportation. Therefore, crew scheduling is a common problem within the applied Operations Research that comes in many variants, as each company may have its own set of constraints and objectives. This chapter will consider one of such variants.

In practice, schedules for vehicle operators are constructed either manually or automatically, but this may lead to very exhausting schedules. This especially has been the case for bus drivers, which is the type of vehicle operator that is focused on within this chapter. Research conducted in 2001 by the European Transport Safety Council has shown that driver fatigue is a significant factor in approximately 20% of commercial road transport crashes [22]. Hence, to improve road safety and the working conditions for all drivers of road haulage and passenger transport vehicles, the driving and resting times of drivers in the European Union are since April 2007

controlled by regulation (EC) No. 561/2006. The regulations are meant for vehicles carrying goods with a maximum permissible weight of 3500 kg [2] and passenger-carrying vehicles that are constructed to transport more than nine people [3]. These rules ensure sufficiently long breaks and resting times within a duty or between two consecutive duties, and will be elaborated upon in the next section.

4.2 The European Crew Scheduling Problem

4.2.1 Motivation

The aim of this chapter is to propose algorithms that output driver schedules that fulfill the European regulations, in order to provide sufficient resting times for bus drivers. The resting times are not only needed within a duty, but also between duties. Here, a duty is defined as a sequence of trips driven by the same driver, starting and ending at a home depot, including idle time and breaks (formalized in Section 4.2.2). Another goal of this chapter is to give insight in which constraints imposed by the regulations complicate the scheduling process significantly. For this reason, some theoretical or experimental parts in this chapter only consider a subset of the constraints.

Even though there are separate documentations for vehicles carrying goods [2] compared to vehicles carrying passengers [3], the imposed constraints that are relevant for schedule optimization are identical. The regulations from these documentations place constraints on the duration and frequency of resting periods that need to be taken, and consist of the following:

1. **Full breaks**: After at most 4.5 hours of driving, a break of at least 45 minutes has to be taken. This may be split into two breaks of at least 15 and 30 minutes.

2. **Daily driving time**: At most 9 hours of driving time during one daily duty may be driven. This may be extended to 10 hours, but at most twice per week.

3. **Daily resting time**: Between two daily duties, the normal resting period is at least 11 hours. This may be reduced to a short resting period of at least 9 hours, for at most 3 times per week.

4. **Weekly driving time**: At most 56 hours in one weekly duty may be driven, and 90 hours in two consecutive weekly duties.

5. **Weekly resting time**: After at most six daily duties, a weekly rest needs to be taken of 45 uninterrupted hours. This may be reduced every second week to 24 hours.

Despite not explicitly stated in the European regulation, this chapter also considers a sixth constraint that is very common within transport companies in Europe:

6. **Duty time**: A daily duty may not exceed a given maximum duty time λ_{daily}^{duty}. The value of this parameter differs per country and sometimes per company.

These constraints provide the basis for the European Crew Scheduling Problem (ECSP) that will be defined in the next section. An important difference of this problem compared to most related research in the literature is the distinction between duty and driving times, for which two different bounds are used. Another notable characteristic is that every job has a fixed departure and arrival time and location, while some CSP's assume that jobs are done at the locations. For this reason, jobs are referred to as *trips*. These trips have to be driven "on demand", which often is the case for, e.g., touring car companies.

As objective, the usual choices are to either minimize the number of required drivers or the total sum of duty times. This chapter focuses on the latter, as it is more oriented on the efficiency of the total time spent by the crew. It is further justified because transport providers have a fixed number of drivers under contract. This objective function is also the focus of the collaborating touring car and software companies that made this research possible. Additionally, the focus is put on passenger transport in a private setting, meaning that a vehicle during a trip will not pick up passengers of another trip. This is similar to the case of goods transport that cannot pick up other goods during one trip, which is often the case for trucks.

Discussion. As mentioned, the regulations give options to adjust the limits of some constraints, such as splitting a break of 45 minutes into two parts, extending daily driving times from 9 to 10 hours for twice per week, and reducing the daily resting times from 11 to 9 hours for three times per week. This clearly gives more possibilities to optimize a schedule better, but also brings risks. Schedules in practice are prone to delays, e.g., due to traffic jams or bad weather. Thus, if the breaks, driving and resting times are scheduled near its limits, small delays may turn the schedule infeasible.

In other words, if one would generate a schedule without using these such options, a certain level of robustness is guaranteed. For this reason, in the rest of this chapter will be assumed that these options (splitting breaks, shortening resting periods or extending driving periods) should not be used. As a positive side-effect, this makes the upcoming model more manageable and potential solving methods quicker due to fewer, complicated constraints.

4.2.2 Definition

Let $T = [n]$ be a given set of *trips* that have to be driven by a given set of *drivers* $D = [k]$. Every driver $d \in D$ starts and ends its duty at the single depot, also referred to as *home*. Every trip $t \in T$ has a starting $S(t)$ and ending time $E(t)$ that are upper bounded by a maximum trip ending time L^{max}, i.e., $0 \leq S(t) < E(t) \leq L^{max}$. Also, in practice, every trip has a location of departure and arrival, but this requires no notation in the upcoming mathematical models.

A matrix M is given that specifies the travel time between all pairs of trips, where $M(t, u) \geq 0$ is the travel time from ending location of trip t to the starting location of trip u. Moreover, $M(0, t)$ denotes the travel time from home to the starting location of trip t, and $M(t, 0)$ the travel time from the ending location from trip t

to home. Without loss of generality, assume that trips are non-decreasingly ordered by starting time, i.e., for every pair of trips $t, u \in T$, if $t < u$ then $S(t) \leq S(u)$.

Definition 4.1. *A pair of trips $t, u \in T$ with $t < u$ is **compatible** if it is possible to drive trip u after trip t, i.e., if $E(t) + M(t, u) \leq S(u)$. Otherwise, the pair is **incompatible**.*

Let Q denote the set of pairs of trips that are incompatible, i.e.:

$$Q = \{(t, u) \in T \times T \mid E(t) + M(t, u) > S(u), t < u\}.$$

The constants on the driving and resting times described in Section 4.2.1 will be defined by means of the following parameters:

- $\lambda^{drive}_{breakless}$ = maximum breakless driving time before a full break is required,
- $\tau^{rest}_{breakless}$ = resting time required to reset the breakless driving time (full break),
- λ^{drive}_{daily} = maximum daily driving time before a daily resting period is required,
- τ^{rest}_{daily} = daily resting time required to reset the daily driving time,
- λ^{drive}_{weekly} = maximum weekly driving time before a weekly resting period is required,
- τ^{rest}_{weekly} = weekly resting time required to reset the weekly driving time, and
- λ^{duty}_{daily} = maximum daily duty time before a daily resting period is required.

These definitions allow to define a solution. Informally, a solution consists of k schedules, that in hierarchical order consists out of one or multiple weekly duties, daily duties, breakless duty parts and trips. A more formal definition of a solution will be given below.

Definition 4.2. *A **breakless duty part** $b = (t_1, \ldots, t_x)$ is a sequence of trips where $t_j \in T$ for $j \in [x]$. It is considered **feasible** if every subsequent pair of trips is compatible, i.e., if $(t_j, t_{j+1}) \notin Q$ for $j = 1, \ldots, x - 1$. Define the breakless duty's:*

- *trip starting time as $S(b) = S(t_1)$,*
- *trip ending time as $E(b) = E(t_x)$,*
- *trip driving time as $D(b) = \sum_{j=1}^{x}(E(t_j) - S(t_j))$, and*
- *trip duty time as $Y(b) = E(b) - S(b)$.*

The starting and ending trip of b are referred to as $t_1(b)$ and $t_x(b)$, respectively.

Definition 4.3. *A **daily duty** d is a sequence of breakless duty parts $d = (b_1, \ldots, b_y)$. Define the daily duty's:*

- *starting time $S(d) = S(b_1) - M(0, t_1(b_1))$,*

- *ending time* $E(d) = E(b_y) + M(t_x(b_y), 0)$,
- *driving time* $D(d) = M(0, t_1(b_1)) + \sum_{j=1}^{y} D(b_j) + \sum_{j=1}^{y-1} M(t_x(b_j), t_1(b_{j+1})) + M(t_x(b_y), 0)$, *and*
- *duty time* $Y(d) = E(d) - S(d)$.

and is ***feasible*** *if:*

- *every breakless duty part* b_j *is compatible for* $j = 1, \ldots, y$.
- $D(d) \leq \lambda^{drive}_{daily}$,
- $Y(d) \leq \lambda^{duty}_{daily}$,
- *between two subsequent breakless duty parts, a break of at least* $\tau^{rest}_{breakless}$ *is taken, i.e.,* $S(t_1(b_{j+1})) - E(t_x(b_j)) - M(t_x(b_j), t_1(b_{j+1})) \geq \tau^{drive}_{breakless}$, *for* $j = 1, \ldots, y-1$, *and*
- *the driving time of every breakless duty part* b_j *for* $j \in [y]$ *does not exceed* $\lambda^{drive}_{breakless}$. *It is assumed that the driving time from home to the starting location of* $t_1(b_1)$ *is included in the* b_1*'s driving time. Also, the driving time from the ending location of* $t_x(b_y)$ *back to home is included in the driving time of* b_y. *Finally, it is assumed that for two subsequent breakless duty parts* b_j *and* b_{j+1}, *the travelling time between the duties,* $M(t_x(b_j), t_1(b_{j+1}))$, *is included in* b_j*'s duty time. This means that the driver drives to the starting location of the next trip first, before taking a break. This means that the following should hold:*
  - $M(0, t_1(b_1)) + D(b_1) + M(t_x(b_1), t_1(b_2)) \leq \lambda^{drive}_{breakless}$,
  - $D(b_j) + M(t_x(b_j), t_1(b_{j+1})) \leq \lambda^{drive}_{breakless}$, *for* $j = 2, \ldots, y-1$, *and*
  - $D(b_y) + M(t_x(b_y), 0) \leq \lambda^{drive}_{breakless}$.

Note that the driving time between trips within the breakless duty part, as well as the driving time from and to home is not included in Definition 4.2. This is because it depends on whether the breakless duty is the first or last breakless duty (or neither) within the daily duty. It is therefore more convenient to include these driving times in Definition 4.3.

Definition 4.4. *A* ***weekly duty*** w *is a sequence of daily duties* $w = (d_1, \ldots, d_p)$, *with weekly:*

- *starting time* $S(w) = S(d_1)$,
- *ending time* $E(w) = E(d_p)$,
- *driving time* $D(w) = \sum_{j=1}^{p} D(d_j)$, *and*
- *duty time* $Y(w) = \sum_{j=1}^{p} Y(d_j)$,

and is ***feasible*** *if:*

- d_j *is feasible for* $j = 1, \ldots, p$,
- $S(d_{j+1}) - E(d_j) \geq \tau^{rest}_{daily}$ *for* $j = 1, \ldots, p-1$, *and*
- $D(w) \leq \lambda^{drive}_{weekly}$.

Definition 4.5. *A* ***schedule*** *s is a sequence of weekly duties $s = (w_1, \ldots, w_q)$, with total:*

- *driving time* $D(s) = \sum_{i=1}^{q} D(w_i)$, *and*
- *duty time* $Y(s) = \sum_{i=1}^{q} Y(w_i)$,

and is ***feasible*** *if:*

- w_i *is feasible for* $i = 1, \ldots, q$, *and*
- $S(w_{i+1}) - E(w_i) \geq \tau^{rest}_{weekly}$ *for* $i = 1, \ldots, q-1$.

Note that there is no upper bound on the driving time of a schedule, as there is no such rule on the maximum driving time per year.

Definition 4.6. *A* ***solution*** *$\sigma = (s_1, \ldots, s_k)$ is a set of k schedules (drivers) with total driving time $D(\sigma) = \sum_{j=1}^{k} D(s_j)$ and total duty time $Y(\sigma) = \sum_{j=1}^{k} Y(s_j)$. A solution σ is feasible if all k schedules are feasible, and every trip is contained in exactly one schedule.*

These definitions suffice to formalize the problem considered in this chapter.

European Crew Scheduling Problem (ECSP)	
Given:	A set of trips $T = [n]$ with corresponding starting and ending times, $0 \leq S(t) \leq E(t)$, a traveling time matrix M and a number of drivers k.
Goal:	Find a feasible solution σ that minimizes $Y(\sigma)$.

MILP formulation. A new Mixed Integer Linear Programming (MILP) formulation for the ECSP will be introduced here. For this formulation to function, it is necessary to define the following given parameters:

- β = maximum number of breakless duty parts in a daily duty,
- δ = maximum number of daily duties in a weekly duty, and
- ω = maximum number of weekly duties in a schedule,

with $\beta, \delta, \omega \in \mathbb{N}$. Within the upcoming MILP, the following variables will be used:

$$x_{tuhijd} = \begin{cases} 1 & \text{if trip } u \text{ is driven directly after } t \text{ in the } h\text{-th breakless duty part of the } i\text{-th daily duty of the } j\text{-th weekly duty of driver } d, \text{ and} \\ 0 & \text{otherwise,} \end{cases}$$

for every $t, u \in [n]_0$, $h \in [\beta]$, $i \in [\delta]$, $j \in [\omega]$ and $d \in [k]$. Here, $[n]_0$ is defined as $\{0\} \cup [n]$, where the artificial trip $t = 0$ represents home, i.e., if $x_{0u1ijd} = 1$, then trip u is the first trip of the i-th daily duty of the j-th weekly duty of driver d. Furthermore, introduce:

$$y_{hijd} = \begin{cases} 1 & \text{if the } h\text{-th breakless duty part of the } i\text{-th daily duty of the } j\text{-th weekly duty of driver } d \text{ contains a trip, and} \\ 0 & \text{otherwise,} \end{cases}$$

for every $h \in [\beta]$, $i \in [\delta]$, $j \in [\omega]$ and $d \in [k]$. Finally, the following supporting variables are introduced:

- s_{hijd} = starting time of the h-th breakless duty part of the i-th daily duty of the j-th weekly duty of driver d, and

- e_{hijd} = ending time of the h-th breakless duty part of the i-th daily duty of the j-th weekly duty of driver d,

for every $h \in [\beta]$, $i \in [\delta]$, $j \in [\omega]$ and $d \in [k]$. If there are fewer than β breakless duty parts used, then $e_{\beta ijd}$ will become equal to the ending time of the last breakless duty part for all $i \in [\delta], j \in [\omega], d \in [k]$. Also, let $B = \max_{t \in [n]} S_t$, which will act as a constant to model some inequality constraints (also known as the "Big M method").

Finally, for convenience of the presentation of the upcoming models, some notation is slightly abbreviated. Define $S_t = S(t)$ and $E_t = E(t)$ for the starting and ending trip of trip $t \in [n]$, and $S_0 = E_0 = 0$. The driving time is referred to as $D_t = E_t - S_t$, while the driving time from t to u is slightly abbreviated to $M_{tu} = M(t, u)$, for $t, u \in [n]_0$. The ECSP can now be formulated as follows.

$$\min \sum_{d=1}^{k}\sum_{j=1}^{\omega}\sum_{i=1}^{\delta}(e_{\beta ijd} - s_{1ijd}) \tag{4.0}$$

$$\text{s.t.} \sum_{d=1}^{k}\sum_{j=1}^{\omega}\sum_{i=1}^{\delta}\sum_{h=1}^{\beta}\sum_{t=0}^{n} x_{tuhijd} = 1 \qquad \forall u \tag{4.1a}$$

$$\sum_{d=1}^{k}\sum_{j=1}^{\omega}\sum_{i=1}^{\delta}\sum_{h=1}^{\beta}\sum_{u=0}^{n} x_{tuhijd} = 1 \qquad \forall t \tag{4.1b}$$

$$\sum_{u=0}^{n} x_{0u1ijd} = 1 \qquad \forall d, i, j \tag{4.2a}$$

$$\sum_{t=0}^{n}\sum_{h=1}^{\beta} x_{t0hijd} = 1 \qquad \forall d, i, j \tag{4.2b}$$

$$\sum_{d=1}^{k}\sum_{j=1}^{\omega}\sum_{i=1}^{\delta}\sum_{h=1}^{\beta} x_{tuhijd} = 0 \qquad \forall (t,u) \in Q \tag{4.3}$$

$$\sum_{u=0}^{n} M_{0u} \cdot x_{0u1ijd} + \sum_{t=1}^{n}\sum_{u=0}^{n}(D_t - M_{tu}) \cdot x_{tu1ijd} \leq \lambda^{drive}_{breakless} \qquad \forall i, j, d \tag{4.4a}$$

$$\sum_{t=0}^{n}\sum_{u=0}^{n}(D_t - M_{tu}) \cdot x_{tuhijd} \leq \lambda^{drive}_{breakless} \qquad \forall h > 1, i, j, d \tag{4.4b}$$

$$\sum_{u=0}^{n} M_{0u} \cdot x_{0u1ijd} + \sum_{h=1}^{\beta}\sum_{t=1}^{n}\sum_{u=0}^{n}(D_t - M_{tu}) \cdot x_{tuhijd} \leq \lambda^{drive}_{daily} \qquad \forall i, j, d \tag{4.4c}$$

$$\sum_{i=1}^{\delta}\sum_{u=0}^{n} M_{0u} \cdot x_{0u1ijd} + \sum_{i=1}^{\delta}\sum_{h=1}^{\beta}\sum_{t=1}^{n}\sum_{u=0}^{n}(D_t - M_{tu}) \cdot x_{tuhijd} \leq \lambda^{drive}_{weekly} \qquad \forall j, d \tag{4.4d}$$

$$s_{hijd} \leq e_{hijd} \qquad \forall h, i, j, d \tag{4.5a}$$

$$e_{hijd} \leq s_{(h+1)1ijd} \qquad \forall i \neq \beta, j, d \tag{4.5b}$$

$$e_{\beta ijd} \leq s_{1(i+1)jd} \qquad \forall i \neq \beta, j, d \tag{4.5c}$$

$$e_{\beta\omega jd} \leq s_{11(j+1)d} \qquad \forall j \neq \gamma, d \tag{4.5d}$$

$$s_{hijd} \leq (S_t - M_{0t}) \cdot x_{tuhijd} + B \cdot (1 - x_{tuhijd}) \qquad \forall t, h = 1, i, j, d \tag{4.6a}$$

$$s_{hijd} \leq S_t \cdot x_{tuhijd} + B \cdot (1 - x_{tuhijd}) \qquad \forall t, h > 1, i, j, d \tag{4.6b}$$

$$e_{hijd} \geq (E_u + M_{u0}) \cdot x_{tuhijd} \qquad \forall t, h = \beta, i, j, d \tag{4.6c}$$

$$y_{hijd} \geq x_{thijd} \qquad \forall t, h, i, j, d \tag{4.7}$$

$$s_{(h+1)ijd} - e_{thijd} \geq \tau^{rest}_{breakless} \cdot y_{(h+1)ijd} \qquad \forall h \neq \beta, i, j, d \tag{4.8a}$$

$$s_{h(i+1)jd} - e_{thijd} \geq \tau^{rest}_{daily} \cdot y_{h(i+1)jd} \qquad \forall h, i \neq \delta, j, d \tag{4.8b}$$

$$s_{hi(j+1)d} - e_{thijd} \geq \tau^{rest}_{weekly} \cdot y_{hi(j+1)d} \qquad \forall h, i, j \neq \gamma, d \tag{4.8c}$$

$$e_{\beta ijd} - s_{1ijd} \leq \lambda^{duty}_{daily} \qquad \forall i, j, d \tag{4.9}$$

$$x_{tuhijd} \in \{0, 1\} \qquad \forall t, u, h, i, j, d \tag{4.10}$$

$$y_{hijd} \in \{0, 1\} \qquad \forall h, i, j, d \tag{4.11}$$

Unless otherwise stated, $\forall t$, $\forall u$, $\forall h$, $\forall i$, $\forall j$ and $\forall d$ refer to $\forall t \in [n]$, $\forall u \in [n]$, $\forall h \in [\beta]$, $\forall i \in [\delta]$, $\forall j \in [\omega]$ and $\forall d \in [k]$, respectively.

The objective function is simply the sum of the daily duty times of all drivers. The duty time of a (daily or weekly) duty is the difference between end of the last breakless duty part and the start of the first breakless duty part, including driving times from and to the home depot. An explanation of the constraints is given below:

- (4.1): Every trip in $[n]$ has a preceding (4.1a) and successive (4.1b) event in $[n]_0$ (either another trip or arrival/departure at/from home), meaning that every trip is driven exactly once.

- (4.2): For every daily duty, home is departed from and arrived to exactly once. For every *first* breakless duty part ($h = 1$) in any daily duty, the number of departures from home equals 1 (4.2a). The total number of arrivals in all breakless duty parts at home equals 1 (4.2b). Note that if the daily and weekly duty i and j for driver d contains no trip, then $x_{001ijd} = 1$.

- (4.3): Every pair of successively driven trips by the same driver is compatible.

- (4.4): For every breakless duty part, the driving time may not exceed $\lambda^{drive}_{breakless}$. Recall that only for the first breakless duty part, the driving time from the home depot to the next destination will to be added (4.4a). Note that at most one term with such a driving time is added due to constraint (4.2a). For all breakless duty parts, including the first, the driving time of all trips, and between subsequent trips (and home) is also added (4.4a and 4.4b). For every daily duty, the total driving time of all its breakless duty parts may not exceed λ^{drive}_{daily} (4.4c). Similarly, for every weekly duty, the total driving time of all corresponding daily duties may not exceed λ^{drive}_{weekly} (4.4d).

- (4.5): The starting time of a breakless duty part is at most its ending time (4.5a). This prevents negative duty times. Also, for two subsequent breakless duty parts, the starting time of the successive breakless duty part is no earlier than the ending time of the preceding breakless duty part (4.5b). This ordering holds similarly for two subsequent daily duties (4.5c) and two subsequent weekly duties (4.5d). This prevents overlapping breakless duty parts, daily duties and/or weekly duties of the same driver. In other words, these constraints ensure for every driver d a complete ordering on all values of s_{hijd} and e_{hijd}.

- (4.6): The starting time of a breakless duty part is the minimum of all its trip's starting times minus their driving time from the starting point of the trip. In case $h = 1$, this means that this is the start of the daily duty, meaning that the travel time from the depot to the first trip the duty needs to be subtracted (4.6a). For $h > 1$, this is not required (4.6b). The term containing B ensures that the upper bound restriction on s_{hijd} is activated only if $x_{thijd} = 1$, and properly deactivated when $x_{thijd} = 1$. Similarly, the ending time of a breakless duty part is the maximum of all its trip's ending times, plus the driving time from the ending point of the trip to the next location (4.6c).

- (4.7): If at least one trip is assigned to a breakless duty part, the corresponding variable y_{hijd} that indicates whether the breakless duty part is used is set equal to 1.
- (4.8): The total resting time of any breakless duty part, daily duty and weekly duty is at least $\tau^{drive}_{breakless}$ (4.8a), τ^{drive}_{daily} (4.8b) and τ^{drive}_{weekly} (4.8c), respectively.
- (4.9): The total duty time of any daily duty is at most λ^{duty}_{daily}.
- (4.10) and (4.11): Standard integrality constraints.

Graph formulation. To make use of graph algorithms for several analyses in this chapter, the problem is represented as a graph. Introduce a directed acyclic graph (DAG) $D = (N, A)$, where every node corresponds to an *event* at a location. If the event is the departure or arrival of a trip, an exact time is also associated to the node, as these times are fixed. For every driver $d \in [k]$, let n^{out}_d and n^{in}_d be the nodes corresponding to the events that driver d respectively starts and ends its duty at the depot. Furthermore, for every trip t, let n^{start}_t and n^{end}_t be the nodes corresponding to departure and arrival for trip t. Define:

$$N^{out}_D = \left\{n^{out}_d : d \in [k]\right\}, N^{in}_D = \left\{n^{in}_d : d \in [k]\right\},$$

and

$$N^{start}_T = \left\{n^{start}_t : t \in [n]\right\}, N^{end}_T = \left\{n^{end}_t : t \in [n]\right\},$$

such that the entire node set equals:

$$N = N^{out}_D \cup N^{in}_D \cup N^{start}_T \cup N^{end}_T.$$

An arc between two nodes exists if the expected traveling time between the locations of the corresponding events is shorter than the time difference of the corresponding events. More specifically, the set of arcs that connect the departure to the arrival of every trip is denoted by:

$$A_T = \left\{(n^{start}_t, n^{end}_t) \mid t \in T\right\}$$

and the set of arcs that connect compatible trips is denoted by:

$$C_T = \left\{(n^{end}_t, n^{start}_u) \mid E_t + M_{tu} \leq S_u, t \in T, u \in T\right\}.$$

Note that the nodes in N^{out}_D and N^{in}_D have no specified time, unlike the nodes in N^{start}_T and N^{end}_T. After all, the starting and ending time of a duty has yet to be determined. For this reason, all nodes in N^{out}_D can be connected to all nodes in N^{start}_T without traveling time constraint. Similarly, all nodes between N^{end}_T and N^{in}_D can be connected.

Finally, there is also a possibility that a driver d will not be used at all, which will be represented by an arc (n^{out}_d, n^{in}_d), represented in a set:

$$U_T = \left\{(n^{out}_d, n^{in}_d) \mid d \in [k]\right\}.$$

To summarize, the complete set of arcs can be defined as:

$$A = (N_D^{out} \times N_T^{start}) \cup A_T \cup C_T \cup (N_T^{end} \times N_D^{in}) \cup U_T.$$

Now every arc has two weights, representing the driving time and the duty time when the two corresponding events are done by the same driver successively. The driving times are defined by:

$$\beta_{tu} := \begin{cases} M_{0u} & (n_d^{out}, n_u^{start}) \in (N_D^{out} \times N_T^{start}) \\ E_t - S_t & (n_t^{start}, n_t^{end}) \in A_T \\ M_{tu} & (n_t^{end}, n_u^{start}) \in C_T \\ M_{t0} & (n_t^{end}, n_d^{in}) \in (N_T^{end} \times N_D^{in}) \\ 0 & (n_d^{out}, n_d^{in}) \in U_T \end{cases}$$

Note that if $(n_t^{end}, n_u^{start}) \in C_T$ that the duty time does not necessarily equal M_{tu} because the driver has to wait for the starting time of trip u before it can be driven. In other words, M_{tu} is the driving time after finishing trip t and arriving at the start of trip u, and $S_u - E_t - M_{tu}$ is the waiting time before trip u can be started. There is thus a small, subtle difference between duty and driving times that only holds if $(n_t^{end}, n_u^{start}) \in C_T$. The duty times are thus defined by:

$$\gamma_{tu} := \begin{cases} M_{0u} & (n_d^{out}, n_u^{start}) \in (N_D^{out} \times N_T^{start}) \\ E_t - S_t & (n_t^{start}, n_t^{end}) \in A_T \\ S_u - E_t & (n_t^{end}, n_u^{start}) \in C_T \\ M_{t0} & (n_t^{end}, n_d^{in}) \in (N_T^{end} \times N_D^{in}) \\ 0 & (n_d^{out}, n_d^{in}) \in U_T \end{cases}$$

A nonlinear integer program to model the constraints based on a similar graph presentation is presented in [45]. This model only uses driving times as weights on the arcs, but duty times can be incorporated similarly. This requires a procedure that labels nodes, but it is impossible to efficiently determine all possible labels on a node. Since this model is very extended and rather tedious, it will not be discussed here since it is not usable. Also, such a model is not necessary in this chapter, as this graph representation will only be used to analyze special cases of the ECSP.

4.2.3 Related work

Since the ECSP is a significant extension of the standard CSP, some similarities and differences with other related crew scheduling problems and its formulations, as well as the relation to other well-known problems will be discussed.

Crew Scheduling. As mentioned, crew scheduling is a general problem of assigning tasks to a crew and typically arises in the transportation industry. Early and recent reviews on crew scheduling can be found in [28, 48, 64, 123] for airlines, [14, 54, 58] for railways, [72] for trucks and [112] for public transportation systems in general. However, there are notable differences between crew scheduling in these areas compared to crew scheduling for buses, with the ECSP in particular. Most

importantly, the rules on driving and resting times as mentioned in Section 4.2.1 are more complex and elaborate for bus drivers. Even though airline crew scheduling problems often consider breaks between duties (see, e.g., [65]), breaks within a single duty are usually not considered for airline crew scheduling. The reason for this is because there is usually enough crew to cover for each other on flights, which is often not the case for bus drivers. Several railway crew scheduling problems do incorporate breaks within a duty, typically by guaranteeing a meal break in the duty after a certain duty time (see, e.g., [39, 73]). However, this guarantees only one break in the duty, of which the starting time generally based on duty time rather than the driving time. Furthermore, the ECSP considers one (home) depot, while airline and railway companies often (but not always) have multiple.

By limiting the scope to crew scheduling for buses, one of the earlier overviews can be found in [124], where some of the most common constraints on duty times and breaks are considered, including solution methods. More recent overviews that include bus crew scheduling can be found in [33, 60, 114], where also studies on more (company-)specific problems and constraints are discussed.

Within the literature, one of the most common approaches to solve the CSP is column generation, introduced by [27]. See also Section 2.8 for an explanation including examples. Usually, the master problem is formulated as a set covering [109] or set partitioning problem [125] and the subproblem as a Resource-Constrained Shortest Path Problem (RCSPP) (see, e.g., [16, 25, 26, 38, 87]). Using this approach, duties are generated by solving the RCSPP while taking several feasibility constraints into account. Note however, that the RCSPP is NP-complete [52, 86]. Subsequently, given a set of feasible duties, the set covering or partitioning problem is solved. This process of generating a feasible duty and solving the master problem is repeated, as long as a duty can be found that can potentially improve the solution value (also known as *branch and price*).

The literature also considers a variety of heuristic approaches to solve the CSP. These include tabu search algorithms [18, 85, 108], genetic algorithms [29, 85] and greedy randomized adaptive search procedures [23, 85]. Even though these works consider different constraints as the ECSP, these methods can provide a good starting point to approach the ECSP heuristically.

Studies on the CSP that distinguish duty from driving times (due to different rules) while incorporating the elaborate European regulations, could not be found in the literature. In this chapter, the distinction can only be made in the MILP formulation in Section 4.2.2. Separating duty from driving times appear to be very difficult in a column generation approach while being within a reasonable computation time. Instead, the line of work on column generation approaches will be extended here, by initially proposing new models and insights for the pricing problem by considering duty times only. In addition, several insights will be given that can help to eventually distinguish duty and driving time constraints.

European regulations. Because the European rules have only been implemented in 2007, research including these rules is relatively young. The earliest work that includes these rules can be found in [45] for which an initial model is proposed.

Methods for vehicle routing problems and truck driver scheduling under European regulations have been proposed in respectively [43] and [44]. Still, these works do not distinguish driver and duty times. Also, most methods do not assume fixed arrival and departure times, but time windows, as the focus lies on transporting freight. These works can therefore be more classified as a Vehicle Routing Problem (VRP) [75, 77, 113], rather than a CSP. One of the few works that does distinguish duty and driving times is [69, 126], where heuristics are proposed. However, this study works with fewer constraints, and does not consider the case that a driver can have multiple duties (i.e., the schedule of only one single day is optimized).

Extensions There are several noteworthy extensions or variations that will not be considered here, but still will be mentioned here to clarify the scope of this chapter. For example, the ECSP assumes that every driver operates one vehicle during its duty (and a vehicle is brought from and to the home depot by the same driver). However, this may not always be the most efficient, as vehicles may be used by multiple drivers who do not always end their duty in at the home depot. Several models, relaxations and algorithms for such an integrated approach are discussed in [9, 38]. Also, all vehicles (and drivers) are considered to be homogenous, meaning that every trip can be driven by every bus and driver. In practice however, vehicles may have different capacities and drivers may have different skills, requiring a formulation of a heterogeneous fleet, such as in [126]. Since such extensions might be relevant for the ECSP, they will be suggested as future work.

Path Covering. Finally, it is worth mentioning that the ECSP can be seen as a variant of the Path Cover Problem (PCP) from a graph theoretical perspective, for which an early analysis of its complexity is given in [92]. This problem has a directed graph $D = (N, A)$ given, and has the goal to find a set of directed paths such that every node is contained in at least one path. Since the theory behind this problem is a good buildup for some of the results in this chapter, the literature review for this specific problem is woven into Section 4.2.5 instead.

4.2.4 Contributions

The contributions of this chapter are as follows. In Section 4.2.2, a MILP formulation has been proposed for this problem. This formulation takes all necessary constraints imposed by European regulations into account, such as different bounds on the breakless, daily and weekly driving time, the daily duty time and the length of daily and weekly breaks (see Section 4.2.1). In particular, a distinction between driving and duty times can be done. Such an MILP formulation could not be found in the literature before.

In Section 4.2.5, some small (graph theoretical) results of more fundamental versions of the ECSP are given. The first result is that the PCP on a transitive DAG (see Definition 4.11) with weight bounds on the paths is NP-complete. This is done through a reduction from 3-Partition. It will be shown that weight bounds on duties make the problem hard, unless the underlying graph is acyclic.

Furthermore, a column generation approach will be proposed for the cases of the ECSP where a driver only has a single duty or a few duties in Sections 4.3.3 and 4.3.4, *without* considering breaks and weekly resting times. New insights are given how the duty time can be constrained in the pricing problem, by exploiting the time structure of the underlying graph. Subsequently, extensions of the column generation model will be proposed that is able to include breaks and weekly resting times. Also, a direction will be given to how to distinguish duty and driving times, by formulating the pricing problem as a RCSPP. However, these appear to be too time-consuming using realistic instances. For this reason, experimental results only include a comparison of algorithms (including some simple heuristics) where breaks and weekly resting times are disregarded.

4.2.5 Complexity

Even though basic formulations of the CSP are NP-complete, it is not straightforward that the ECSP is hard due to some practical concepts. The analysis will be provided by starting with analysis of the PCP described in Section 4.2.3, after which it is gradually extended with characteristics from the ECSP.

Some basic graph theory concepts will be formalized first. Throughout this thesis, the terms "vertex" and "edge" are used to refer to an undirected graph, while the terms "node" and "arc", respectively, are used to refer to a directed graph. This is only to clarify the context of the problem, but essentially the terms are the same.

Definition 4.7. *Given a graph $G = (V, E)$, a **path** is a sequence of vertices $p = (v_1, \ldots, v_n)$ for which all corresponding edges $(v_i, v_{i+1}) \in E$, for $i \in [n-1]$. A path can also be defined as the sequence of the corresponding edges, i.e., $p = (\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\})$. The **length** of a path is equal to the number of corresponding edges, $n-1$. If G is a weighted graph, with weights c_{uv} for every $(u, v) \in E$, then the **weight** of a path is equal to the sum of the weights of the corresponding edges.*

Definition 4.8. *Given a directed graph $D = (N, A)$, a **path cover** is a set of paths $P = \{p_1, \ldots, p_k\}$ of size k such that every node is contained in at least one path. A path cover is **node-disjoint** if for any p_i, p_j with $1 \leq i < j \leq k$, $p_i \cap p_j = \emptyset$.*

Now consider the following feasibility problem.

Path Cover Problem	
Given:	A directed graph $D = (N, A)$ and a parameter k.
Goal:	Determine whether there exists a path cover containing at most k paths.

The optimization variant of the problem disregards the parameter k and simply has the goal to find a path cover that minimizes the number of paths used.

The complexity of the PCP (both its decision and optimization variant) is well understood. For a general (directed) graph, the problem is NP-complete, as finding a path cover with $k = 1$ is equivalent to the (directed) Hamiltonian Path problem,

which is well-known to be NP-complete [41]. However, if the graph is acyclic, one can use a classical result in graph theory to solve the problem efficiently, for which some definitions and theorems need to be given first.

Definition 4.9. *Given a graph $G = (V, E)$, a **matching** is a subset of edges $M \subseteq E$ such that for every vertex $v \in V$, at most one edge in M contains v. A **maximum matching** is a matching of maximum cardinality, i.e., a matching $M \subseteq E$ for which $|M| \geq |M'|$ for any matching $M' \subseteq E$.*

Definition 4.10. *Given a graph $G = (V, E)$, a **vertex cover** is a subset $U \subseteq V$ such that for every $(u, v) \in E$ it must be that $u \in U$, $v \in U$, or both. A **minimum vertex cover** is a vertex cover of minimum cardinality, i.e., a vertex cover $U \subseteq V$ for which $|U| \leq |U'|$ for any vertex cover $U' \subseteq V$.*

Theorem 4.1 (König's Theorem [70])**.** *Given a bipartite graph $G = (V = L \cup R, E)$, let $\mu(G)$ be the maximum cardinality of a matching of G and let $\tau(G)$ be the minimum cardinality of a vertex cover of G. Then, $\mu(G) = \tau(G)$.*

The original proof is provided in [70], while an alternative proof is provided in [101]. An extended version of the alternative proof is given here.

Proof. Firstly note that $\mu(G) \leq \tau(G)$. To see this, let M be a maximum matching and U be a minimum vertex cover, i.e., $|M| = \mu(G)$ and $|U| = \tau(G)$. Every edge in M contains at least one vertex from U by definition of a vertex cover, and no two edges in M contain the same vertex by definition of a matching.

Thus, it remains to show that $\mu(G) \geq \tau(G)$. For the sake of contradiction, let G be a minimum counterexample graph for which $\mu(G) < \tau(G)$, i.e., for any subgraph $G' \subset G$, $\mu(G) > \mu(G') \geq \tau(G')$. Let m be the number of edges of G. Now note the following characteristics of G:

- G is connected, i.e., there exists a path between every pair of vertices. Suppose G is not connected, i.e., G consists of multiple components. Then there must exist a component $G' \subset G$ for which $\mu(G') < \tau(G')$, but this contradicts to the minimality of G.
- G is not a path, because:
  - if G is path of odd length, it is easy to see that $\mu(G) = \tau(G) = \frac{m-1}{2}$, contradicting the assumption $\mu(G) < \tau(G)$, and
  - if G is path of even length, it is easy to see that $\mu(G) = \tau(G) = \frac{m}{2}$, contradicting the assumption $\mu(G) < \tau(G)$.
- G is not a cycle, because:
  - a cycle of odd length would imply there exists an edge containing both vertices in either L or R, which is by definition not possible in a bipartite graph,
  - if G is a cycle of even length, it is easy to see that $\mu(G) = \tau(G) = \frac{m}{2}$, contradicting the assumption $\mu(G) < \tau(G)$.

Because G is connected and is not a cycle nor path, it must be that G has a vertex with degree at least 3, i.e., a vertex contained in at least 3 edges. Let v be such a vertex, and let w be one of the vertices connected to v through an edge. Recall that for the sake of contradiction is assumed that $\mu(G) < \tau(G)$. Consider the following two cases:

- If $\mu(G \setminus \{w\}) < \mu(G)$, then it follows by minimality of G that any minimum vertex cover for $G \setminus \{w\}$, say U', has cardinality $|U'| = \tau(G \setminus \{w\}) \leq \mu(G \setminus \{w\}) < \mu(G)$. Therefore, note that $U' \cup \{w\}$ is a cover of G with cardinality at most $\mu(G)$, for any w connected to v. Hence, $\tau(G) \leq \mu(G)$, which is a contradiction by assumption.

- If $\mu(G \setminus \{w\}) = \mu(G)$, this means that there exists a maximum matching M of G that does not contain w. Let e be an edge of $E \setminus M$ that contains v but not w. Such an edge must exist, since the degree of v is at least 3. After all, the removal of M from G can only remove one edge containing v by definition of a matching. Let U'' be a cover of $G \setminus e$ with $|U''| = \mu(G \setminus \{w\}) = \mu(G)$. Since no edge in M contains w, U'' also does not contain w. Therefore, U'' contains v and is a cover of G with cardinality $\tau(G) = \mu(G)$, which is a contradiction by assumption.

In conclusion, the contradiction occurs in both cases, meaning that $\mu(G) \geq \tau(G)$. Since also was noted that $\mu(G) \leq \tau(G)$, it must be that $\mu(G) = \tau(G)$. □

This theorem provides the basis for the next theorem, which can be seen as an equivalent of Dilworth's Theorem [31] on bipartite graphs.

Theorem 4.2. *Let $D = (N, A)$ be a DAG with corresponding bipartite graph $G = (V = V_{out} \cup V_{in}, E)$, with:*

- $V_{out} = \{v_{out} \in N \mid \exists u \in N \text{ for which } (v_{out}, u) \in A\}$
- $V_{in} = \{v_{in} \in N \mid \exists u \in N \text{ for which } (u, v_{in}) \in A\}$
- $E = \{(u_{out}, v_{in}) \in V_{out} \times V_{in} \mid (u, v) \in A\}$

Then, D has a node-disjoint path cover $P = (p_1, \ldots, p_k)$ of size k if and only if G has a matching of size $n - k$, where $n = |V|$.

Proof. Both directions of the proof will be shown.

- If $P = \{p_1, \ldots, p_k\}$ is a node-disjoint path cover of D, define a matching M in G by:
$$M = \{\{u_{out}, v_{in}\} \mid (u, v) \in p_i, \forall\ 1 \leq u < v \leq k\}.$$
M is shown to be a feasible matching by contradiction. Suppose there exists a vertex u_{out} in G for which $\{u_{out}, v_{in}\} \in M$ and $\{u_{out}, w_{in}\} \in M$. Then this implies that both (u, v) and (u, w) are in a path in the path cover P, which contradicts to the assumption that P is node-disjoint. The same argument can

be used to show there exists no $v_i n$ for which both $\{u_{out}, v_{in}\}$ and $\{w_{out}, v_{in}\}$ are in M.

Let $|p_i|$ be the number of vertices in path p_i. Then, the number of edges in p_i is equal to $|p_i| - 1$. Since P is a path cover, $\sum_{i=1}^{k} = n$. Therefore, the size of the created matching M is equal to:

$$|M| = \sum_{i=1}^{k} (|p_i| - 1) = \sum_{i=1}^{k} |p_i| - k = n - k.$$

- If M is a matching in G of size $n - k$, a node-disjoint path cover for D can be constructed as follows. For any $\{u_{out}, v_{in}\} \in M$, take arc $\{u, v\}$ in the path cover. Furthermore, every any node in N that corresponds to a vertex in G which is not covered by the matching M is a separate path of length 0. Observe that:
  - The selected set of arcs forms a set of paths: for every node there is at most one ingoing and one outgoing edge, meaning the paths are node-disjoint.
  - No cycle can occur, since D is directed and acyclic.
  - All nodes are covered, since any node in N that corresponds to a vertex in G which is not covered by the matching M is a considered as a separate path of length 0.
  - By construction, the number of nodes in the path cover equals n, while the number of arcs equals $k = |M|$. For every node $u \in N$ for which $\{u_{out}, v_{in}\}$ is not selected in M, this must mean that u has no ingoing edge in the path cover, meaning that u acts as the start of a new path. Therefore, the number of paths is equal to $n - |M| = n - (n - k) = k$.

□

This means that the PCP can be solved by finding the maximum cardinality matching instead. It is well known that this can be done in polynomial time, e.g., using the Hopcroft-Karp algorithm [57] or the Ford-Fulkerson algorithm [36].

Theorem 4.3. *The PCP can be solved in polynomial time if $D = (N, A)$ is acyclic.*

Proof. Using the earlier results, one can solve the PCP as follows:

- Construct a bipartite graph $V = (V_{out} \cup V_{in}, E)$ as described in Theorem 4.2.
- Find a maximum matching.
- Select the edges in $D = (N, A)$ that correspond to the matching M. These correspond to a path cover as argued in the proof of Theorem 4.2.

By Theorem 4.2, if m is the cardinality of a maximum matching, the number of paths for the described path cover, $n - m$ is minimal. Thus if $n - m \leq k$, there exists a path cover containing at most k paths.

Also note that all of the above three steps run in polynomial time. Constructing the bipartite graph runs in $\mathcal{O}(|V| + |E|)$, the Hopcroft-Karp algorithm [57] can be used to find a maximum matching in $\mathcal{O}(|E|^{2.5})$, while selecting the path cover corresponding to the maximum matching M can be done in $\mathcal{O}(|E|)$. □

Some extensions will be added to the PCP next that are motivated by the ECSP. Recall that trips in the ECSP have specified starting and ending times and can cause the driver to change location. This implies a transitive characteristic of the DAG.

Definition 4.11. *A directed graph $D = (N, A)$ is **transitive** if for every triple of nodes $x, y, z \in V$ with $(x, y) \in A$ and $(y, z) \in A$, then also $(x, z) \in A$.*

After all, if event x can be followed by event y, and event y can be followed by event z, then x can also be followed by z. Some variants of the PCP under transitivity are considered.

Motivated by the ECSP, we also represent a weight bound W for every path, representing a maximum on the duty time in the ECSP. This problem will be referred to as the Weight-Bounded Path Cover Problem (WBPCP).

Weight-Bounded Path Cover Problem	
Given:	A transitive, directed graph $D = (N, A)$, a weight bound W and a parameter k.
Goal:	Determine whether there exists a path cover containing at most k paths, such that every path in the path cover has weight at most W.

Theorem 4.4. *The WBPCP is strongly NP-complete.*

To prove Theorem 4.4, consider the following decision problem which is known to be NP-complete.

3-Partition	
Given:	A set of $3n$ integers, $a_1, \ldots, a_{3n}$.
Goal:	Determine whether there exist n disjoint subsets $S_1, \ldots, S_n \subset \{1, \ldots, 3n\}$ such that $\sum_{j \in S_i} a_j = \frac{\sum_{j=1}^{3n} a_j}{n}$ and $\lvert S_i \rvert = 3$ for $i = 1, \ldots, n$.

Proof of Theorem 4.4. The reduction will be done from the strongly NP-complete problem 3-Partition. For the ease of notation, let $\alpha = \sum_{j=1}^{3n} a_j$. Given an instance of 3-Partition, construct the following instance for the WBPCP:

- $N = \{1, \ldots, 4n\}$,
- $A = \{(x, y) \mid 1 \leq x < y \leq 4n, x \leq 3n\}$,
- $c_{xy} = a_x + \alpha$, for $x = 1, \ldots, 3n$ and $x < y \leq 4n$,

- $W = \frac{\alpha}{n} + 3\alpha$, and
- $k = n$.

In other words, N contains $3n$ nodes that correspond to the integers in 3-Partition and n additional dummy nodes. Every directed arc (x, y) has weight equal to the corresponding integer a_x, plus a constant. See Figure 4.1 for an illustration.

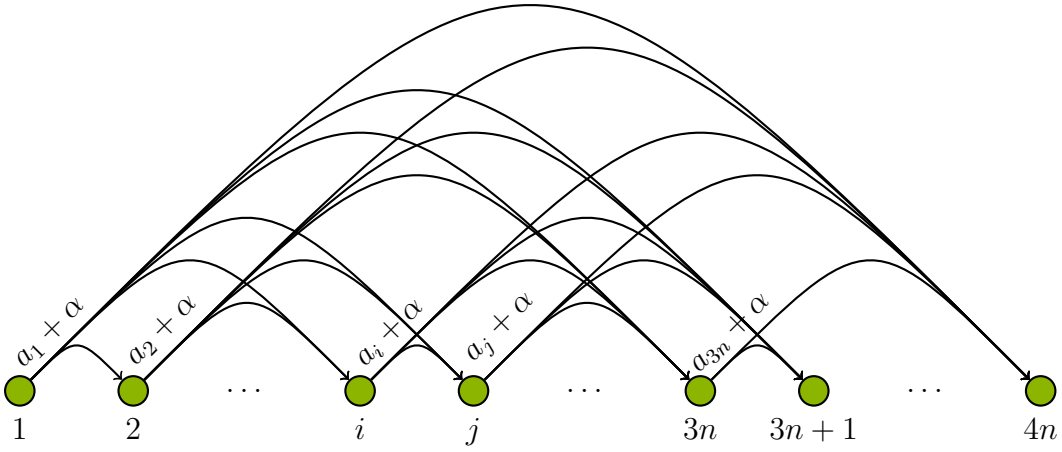

Figure 4.1: Illustration of a ECSP-instance reduced from 3-Partition.

If the instance for 3-Partition is a Yes-instance, then there exist n disjoint subsets $S_1, \ldots, S_n$ such that $\sum_{j \in S_i} a_j = \frac{\alpha}{n}$ and $|S_i| = 3$ for $i = 1, \ldots, n$. A feasible instance for the WBPCP is constructed as follows. For every $S_i = \{a_x, a_y, a_z\}$, select path $P_i = (x, y, z, n + i)$. By construction, such a path always exists and its value equals $\sum_{j \in S_i} a_j + 3\alpha$. Since the instance for 3-Partition is a YES-instance, $\sum_{j \in S_i} a_j = \frac{\alpha}{n}$ for every S_i, meaning that every path P_i meets the weight bound $W = \frac{\alpha}{n} + 3\alpha$ exactly. Moreover, every node is covered, because exactly n disjoint paths of length 3 are required to cover all $4n$ nodes.

If the instance for WBPCP is a YES-instance, then there exist $k = n$ disjoint paths $P_1, \ldots, P_n$ covering all nodes in $D = (N, A)$, with $\sum_{(x,y) \in P_i} c_{xy} \leq \frac{\alpha}{n} + 3\alpha$. It is important to see that every path P_i contains at most three edges. After all, the weight of every edge is at least α by construction. If a path contains four or more edges, the weight of the path is at least 4α, which is larger than the weight bound $W = \frac{\alpha}{n} + 3\alpha$ (assuming $n > 1$, as the problem would be trivial otherwise), contradicting to the instance is a YES-instance. Note that in any feasible solution for the WBPCP, exactly $3n$ edges are covered, since n disjoint paths cover $4n$ vertices. Because there every of the n paths contains at most three edges, and because exactly $3n$ edges are covered, it must be that every path contains exactly three edges. Moreover, every path contains exactly one dummy vertex, since two dummy vertices are not connected by construction. Therefore, every path has a total weight of at most $\frac{\alpha}{n} + 3\alpha$. However, if every vertex is covered by one of the n paths, the total

weight of all paths equals $\sum_{j=1}^{3n}(a_j + \alpha) = (3n+1)\alpha$. Combined with the earlier insight that every path has total weight of at most $\frac{\alpha}{n} + 3\alpha$, it must be that every path has weight of exactly $\frac{\alpha}{n} + 3\alpha$. In conclusion, the sets of integers corresponding to the paths are of cardinality 3 and sum to exactly $\frac{\alpha}{n} + 3\alpha$, meaning the solution is feasible for 3-Partition.

Clearly, this reduction can be done in polynomial time, as construction of the instance requires only linear and quadratic operations. Finally note that the WBPCP is also in NP. A certificate can be a set of paths $P_1, \ldots, P_n \subseteq A$. One can easily verify:

- for all nodes whether it is contained any P_i in $\mathcal{O}(|V|^2)$,
- whether the weight for every path P_i is at most W for all $i \in n$ in $\mathcal{O}(k|V|)$, and
- whether $k \leq n$ in $O(1)$.

□

4.3 Column generation approaches

Recall that the main decision variable in the MILP formulation given in Section 4.2.2 was defined as x_{tuhijd}, indicating whether trip t and u are driven successively during the h-th breakless duty part of the i-th daily duty of the j-th weekly duty of driver d. For practical instances, this may result in a model with too many variables to handle for standard MILP solvers. In such cases, column generation is widely considered as a suitable alternative. The standard routine including its motivation and interpretation behind column generation is also provided in Sections 2.8 and 4.2.3, and is the most common approach for a CSP [54].

In this section, several column generation approaches for simplified versions of the ECSP will be considered. In the first algorithm, drivers are only allowed to have one daily duty, where breaks are disregarded. Afterwards, this approach will be extended to multiple daily duties, i.e., a weekly duty. Finally, this section clarifies how also breaks and multiple weekly duties (i.e., a complete schedule) of drivers can be taken into account, although the corresponding implementation may be very time-consuming. For this research, the approach with multiple daily duties will be compared to the MILP proposed in Section 4.2.2.

Note that if breaks do not need to be considered, the maximum number of breakless duty parts in a daily duty, β, can be set to 1. Similarly, if weekly resting times do not need to be considered, then $\omega = 1$. This allows the upcoming column generation approach to be compared in terms of performance with the solving the proposed MILP using optimization software packages.

4.3.1 Master problem

In order to formulate an alternative ILP formulation, also known as the master problem, for the ECSP, define S as the set of all possible feasible schedules for

any driver. The precise definition of a feasible schedule $s \in S$ depends on which constraints are taken into account (e.g., single duty or multiple duty), and will be made more explicit further in this section. As is standard, the following decision variable is introduced for the master problem:

$$y_s = \begin{cases} 1 & \text{if schedule } s \text{ is selected,} \\ 0 & \text{otherwise,} \end{cases}$$

for $s \in S$. Note that the number of variables obtained this way is exponential in the number of trips. Regardless of the constraints, an upper bound on the number of schedules is 2^n, since a schedule can be identified by its trips only. Moreover, introduce an indicator parameter:

$$I_{st} = \begin{cases} 1 & \text{if schedule } s \text{ contains trip } t, \\ 0 & \text{otherwise,} \end{cases}$$

for $s \in S$ and $t \in T$. Finally, recall that $Y(s)$ is the duty time of schedule s and the objective is to minimize the total duty time, to define the master problem.

$$\begin{aligned} \min \quad & \textstyle\sum_{s \in S} Y(s) y_s && && (M.0) \\ \text{s.t.} \quad & \textstyle\sum_{s \in S} I_{st} y_s = 1 && t \in T && (M.1) \\ & \textstyle\sum_{s \in S} y_s \leq k && && (M.2) \\ & y_s \in \{0, 1\} && s \in S && (M.3) \end{aligned}$$

Constraint (M.1) simply implies that every trip is driven exactly once, while constraint (M.2) ensures that the maximum number of used drivers is not exceeded.

As also described in Section 2.8, the difficulty for the master problem is to find a subset of schedules $S' \subseteq S$ with $|S'| \ll |S|$ efficiently, such that an optimal solution of the master problem is guaranteed to be in S'. The master problem using the subset S' instead of S is called the *restricted master problem* (RMP).

4.3.2 Shadow prices and reduced costs

The challenge is to find a small subset of schedules $S' \subseteq S$ efficiently, such that the optimal solution of the entire problem can be found in this subset S'. To this aim, column generation requires an initial feasible solution for the *LP-relaxation* of the master problem, after which schedules can be found that potentially may improve the solution.

Shadow prices. Recall from basic linear programming theory that after having solved any linear program, one can directly identify the *shadow prices* of the constraints, also known as the values of the dual variables. In fact, a solver typically outputs these shadow prices along with the solution to LP-problems. The shadow price associated with a constraint indicates the rate the objective function would increase (or decrease) if the amount of this resource would increase (or decrease) by one.

In this context, let θ_t be the shadow price associated with constraint (M.1) for trip t. Then, θ_t is equal to the rate that the objective function (M.0) would decrease if trip t is not required to be driven at all, using the solution that has been found after solving the LP. Similarly, constraint (M.2) also outputs exactly one shadow price γ, defining the rate at which the objective function (M.0) would decrease if the number of drivers would decrease.

Reduced costs. Consider a new schedule $s^* \notin S$. Suppose s^* would be used in the solution, the objective function changes at the rate of:

$$r_c(s^*) = Y(s^*) - \sum_{t \in T} I_{s^*t}\theta_t - \gamma. \tag{4.1}$$

This term is referred to as the reduced costs. Note that the γ term is included, because an extra driver is needed to drive schedule s^*.

It is crucial to note that if we have solved the master problem with respect to the current set S of variables and there exists no schedule s^* for which $r_c(s^*)$ is negative, then an optimum solution has been found. On the other hand, if there exists a schedule s^* with $r_c(s^*) < 0$, then adding s^* to S might improve the objective value of the master problem. Thus, we add s^* to S (which corresponds to adding a column to the master problem) and solve the master problem again. This procedure is repeated until no schedule with negative reduced costs can be found, after which the original, non-relaxed master problem is solved using these variables.

Next, we introduce the pricing problems for two special cases of the ECSP: the daily duty case and the weekly duty case, to be defined in the next subsections. As a preliminary remark, the pricing problem for the daily duty case is differently formulated than in other works, but essentially contains no novelty. However, this adjustment in formulation allows an easier and more understandable explanation of the pricing problem of the weekly duty case, where new ideas are used by exploiting the time structure in the graph, using a quick lookup for the required information.

4.3.3 Pricing problem: daily duty excluding breaks

The pricing problem in this subsection considers the daily duty case (or single duty case). It has the goal to return a schedule s^* that contains only one daily duty with minimum reduced costs. The maximum duty time L will be taken into account. A distinction between driving times, as well as breaks within a duty, are initially not considered here. The method to incorporate breaks is clarified in Section 4.3.5. In this subsection will be shown how to find the schedule with minimum cost. First, introduce the following definition.

Definition 4.12. *A $(t, \dots, u)$-**schedule** is a daily duty for a driver that has trip $t \in T$ and $u \in T$ as first and last trip, respectively. The corresponding total duty time is defined as $Y(t, u) = M_{0t} + E_u - S_t + M_{u0}$.*

It is crucial to note that the duty time of a *daily* duty starting with trip t and ending with trip u does not depend on the actual trips driven between t and u. This

occurs due to the time structure from practice: all trips have a fixed starting and ending time and may not be executed at any other time. This property of the graph is characteristic for some scheduling problems, including the ECSP. This does not hold for every scheduling problem, in particular for problems where trips may be driven at any time of choice or need to be driven within specific intervals.

This allows to determine the duty time of a daily duty that starts in y and ends in u in constant time, and therefore also whether the maximum duty time L is not exceeded. Note that this characteristic does not hold for a *weekly* duty that starts in t and ends in u, because there may be daily resting periods within this sequence that do not account for the total duty time of a weekly duty. Fortunately, no maximum weekly duty time is considered (only a maximum weekly driving time), so this will not be a problem.

Additionally, introduce the following notation for the pricing problem:

$$\theta(t,u) = \text{minimum sum of shadow prices of all } (t,\ldots,u)\text{-schedules}$$

After all, there can be many duties that start in t and end in u with different sums of shadow prices.

Lemma 4.1. *Computing all values of $\theta(t,u)$ can done in $\mathcal{O}(mn + n^2 log(n))$ time.*

Proof. Construct the following DAG $D = (N, A)$, where $N = \{1,\ldots,n\}$. Two nodes $t, u \in N$ are connected if their corresponding trips are compatible, i.e.:

$$A = \{(t,u) \mid E_t + M_{tu} \leq S_u, t, u \in T\}.$$

Every arc $(t,u) \in A$ has weight $c_{tu} = -\theta_u$, obtained from the shadow prices of the LP.

A path $p = (t,\ldots,u)$ in D starting in node t and ending at node u thus represents a duty of a driver $d = (t,\ldots,u)$. By definition, the sum of the weights of arcs in p is equal to the amount that the objective function would change if all trips in p excluding trip t do not have to be driven anymore. Note that exclusion of the shadow price of trip t is by construction, as the costs are on the arcs instead of nodes. The number of selected arcs in the path is one less than the number of selected nodes, i.e., the arc going into t (with weight $-\theta_t$) is missing.

Since the goal is to find the path with minimum reduced costs, this reduces to finding the shortest path in D with a small modification. Let $f(t,u)$ be the shortest path in D from node t to node u. Note that there is a subtle difference between $f(t,u)$ and $\theta(t,u)$. As mentioned, by construction of the graph, there is no edge with weight $-\theta_t$ included in the graph D. In other words, $\theta(t,u) = f(t,u) - \theta_t$. If no path exists between t and u, $f(t,u) = \infty$.

Finding $f(t,u)$ for all $t, u \in N$ is also known as the all-pairs shortest path problem in a DAG. This can be solved in polynomial time by, e.g., the Floyd-Warshall algorithm that runs in $\mathcal{O}(n^3)$. A lower asymptotic complexity of $\mathcal{O}(mn + n^2 log(n))$ can be obtained using Fibonacci heaps (such as in [37]) when m is significantly smaller than n^2. Once all values for $f(t,u)$ are determined, $\theta(t,u)$ can subsequently be determined easily by computing $\theta(t,u) = f(t,u) - \theta_t$ in $\mathcal{O}(n^2)$ for all pairs of $t, u \in T$. □

With this lemma, it is a small step towards finding the sequence of trips that corresponds to the daily duty with minimum reduced costs.

Theorem 4.5. *Finding the schedule $s^* \in S$ with minimum reduced costs $r_c(s^*)$ for the pricing problem with daily duties can be done in $\mathcal{O}(mn + n^2 \log(n))$ time.*

Proof. Let s^* be a daily duty starting in t and ending in u, i.e., s^* is a (t,u)-schedule. Then, the reduced costs $r_c(s^*)$, also referred to as $r_c^{daily}(t,u)$, as specified in Equation 4.1 is given by:

$$\begin{aligned} r_c^{daily}(t,u) &= Y(s^*) - \sum_{t \in T} I_{s^*t}\theta_t - \gamma \\ &= \left(M_{0t} + E_u - S_t + M_{u0}\right) + \theta(t,u) - \gamma. \end{aligned}$$

where:

- $\left(M_{0t} + E_u - S_t + M_{u0}\right)$ is the duty time of the schedule, i.e., the rate at which the objective function increases if the corresponding schedule is added,
- $\theta(t,u)$ is the rate at which the objective function, increases if the (t,u)-schedule with minimum shadow prices is added to the solution, and
- γ can be interpreted as the cost of replacing another schedule in the solution (by s^*).

Thus, the variable to be added to the restricted master problem is the (t,u)-schedule for which $r_c^{daily}(t,u)$ is minimal and the length bound is not exceeded, i.e.:

$$\arg \min_{1 \le t \le u \le n} \left\{ r_c^{daily}(t,u) \mid \left(M_{0t} + E_u - S_t + M_{u0}\right) \le L \right\}.$$

Based on t and u with the minimum value of $r_c^{daily}(t,u)$, the corresponding path and therefore schedule s^* (being a daily duty) can be reconstructed using a standard backtracking method in $\mathcal{O}(m)$ time [30]. □

4.3.4 Pricing problem: weekly duty excluding breaks

The pricing problem to determine the minimum reduced costs of a weekly duty is an extension of the pricing problem for the daily duty case. The difficulty of the problem increases because not only needs to be ensured that the length of every daily duty within the weekly duty does not exceed a given bound L, but now additionally also needs to be ensured that the resting time between every subsequent pair of daily duties is at least λ_{daily}^{rest}. Such resting times were not considered in the previous subsection.

Recall that δ is equal to the maximum number of daily duties that a single driver may drive in a weekly duty. Furthermore, if t is the trip with latest ending time for a weekly duty $w = (d_1, \ldots, d_p)$, then the ending time of w is equal to $E(w) = E_t + M_{t0}$.

For the sake of the upcoming proof, assume without loss of generality that all trips are sorted on their ending time plus their driving time to the home depot, i.e., if $t < u$, then $E_t + M_{t0} \leq E_u + M_{u0}$. For every $u \in T$ and $y \in [\delta]$, define:

$g(u, y) =$ minimum reduced costs of all weekly duties consisting of at most y daily duties, with ending time at most $E_u + M_{u0}$ containing only trips in $\{1, \ldots, u\}$.

It is crucial to note that the weekly duty corresponding to the minimum reduced costs of $g(u, y)$ does not necessarily end with trip u. It may end with a different trip that ends earlier than $E_u + M_{u0}$, but not later.

Theorem 4.6. *Finding the schedule $s^* \in S$ with minimum reduced costs $r_c(s^*)$ for the pricing problem for a weekly duty can be done in $\mathcal{O}(mn + n^2 \log(n) + \delta n^2)$ time.*

Clearly, the idea is to find the u and y for which $g(u, y)$ reaches its minimum, and backtrack the corresponding schedule subsequently.

Proof. The following initialization values will be given for $g(u, y)$.

- $g(1, 0) = -\gamma$, because this corresponds to an empty weekly duty because $y = 0$. However, suppose this empty schedule would be added to the solution for the master problem, an extra driver will be assigned to this empty schedule, resulting in a change in objective value at the rate of $-\gamma$.
- $g(1, y) = \min\left\{-\gamma, r_c^{daily}(1, 1)\right\}$ for $y \geq 1$, as under these parameters, either:
  - the duty consists only of trip 1 for which the reduced costs are equal to $r_c^{daily}(1, 1) = \big((E_1 + M_{1,0}) - (S_1 - M_{0,1})\big) - \theta_1 - \gamma$, or,
  - the duty contains no trip, for which the reduced costs are argued earlier.

If the correctness of $g(u - 1, y)$ for $y \in [\delta]$ is assumed, then $g(u, y)$ for $y \in [\delta]$ can be determined as follows. First, introduce the following concept. For every $t \in T$, identify a unique trip $\pi(t) \in T$ for which $E_{\pi(t)} + M_{\pi(t),0}$ is maximum and:

$$S_t \geq E_{\pi(t)} + M_{\pi(t),0} + \lambda_{daily}^{rest} + M_{0t}.$$

In other words, $\pi(t)$ is the trip with the latest ending time at the home depot that can act as the final trip of the duty that precedes the duty that has t as its first trip, while satisfying the resting time constraint between the two duties. The total time needed to identify $\pi(t)$ for every $t \in T$ can by done by a standard binary search in $O(n \log(n))$.

Recall from the proof of Theorem 4.5 that the minimum reduced costs $r_c^{daily}(t, u)$ for all combinations of t and u can be determined in $\mathcal{O}(mn + n^2 \log(n))$ time. These values will be used to determine $g(u, y)$. For every $u \in \{2, \ldots, n\}$ and $y \in [\delta]$, define the dynamic programming function:

$$g(u, y) = \min\left\{g(u - 1, y), \min_{t \in [u]: Y(t,u) \leq L}\left\{g(\pi(t), y - 1) + r_c^{daily}(t, u) + \gamma\right\}\right\}.$$

The correctness of this recursion can be argued as follows. Consider the weekly duty w that corresponds to the minimum reduced costs of all weekly duties consisting of at most y daily duties, with ending time at most $E_u + M_{u0}$, considering only the trips $\{1, \ldots, u\}$. Then there are two possible cases:

- The final trip of w is not u, i.e., w ends in a trip t for which $E_t + M_{t0} \leq E_u + M_{u0}$, which means that $g(u, y) = g(u-1, y)$.

- The final trip of w is u. If so, it is required to find the trip t that acts as the starting trip of the daily duty ending in u. Note that $t = u$ is possible (daily duty would then contain one trip only). The reduced costs of the daily duty is equal to $r_c^{daily}(t, u)$.

  The possible candidates for t are all trips for which a $(t, \ldots, u)$-schedule fulfills the length bound, thus for which $Y(t, u) \leq L$. Also, u needs to be reachable in the graph from t through a path, but then $r_c^{daily}(t, u) = \infty$, meaning that this will never result in a weekly duty with negative reduced costs anyway. For any given t, the time structure of the graph allows the minimum reduced costs to be calculated separately from two parts.

  - The minimum reduced costs of the daily duties in the weekly duty preceding t. Since t is the start of a new daily duty, the latest time a preceding daily duty can end is $S_t - M_{0t} - \lambda_{daily}^{rest}$. By definition, $\pi(t)$ is the trip that has the latest ending time at the home depot in the preceding duty. Thus, $g(\pi(t), y-1)$ is the minimum reduced costs of the daily duties preceding t.
  - The minimum reduced costs of the daily duty starting in t and ending in u, which is $r_c^{daily}(t, u)$ by definition.

  Finally, a term of γ needs to be *added*, as γ is subtracted in both $r_c^{daily}(t, u)$ and $g(\pi(t), y-1)$. Since the same driver will now drive these two duty parts, the term $-\gamma$ needs to be only subtracted once with regard to the reduced costs.

This shows the validity of the above recursion.

Determining the minimum of the above recurrence takes time $O(n)$ for a fixed u and y, as this involves an iteration over all $t \leq u$. The overall time needed to compute all relevant entries $g(u, y)$ for $u \in T$ and $y \in \delta$ is thus $O(\delta n^2)$. By Theorem 4.5, the preparatory computations of all $r_c^{daily}(t, u)$ is done in $\mathcal{O}(mn + n^2 log(n))$. After $g(u, y)$ is determined, the corresponding feasible set of duties with minimum reduced cost can be extracted by backward induction, similar to the single duty case in $\mathcal{O}(m)$. This concludes the total running time of $\mathcal{O}(mn + n^2 \log(n) + \delta n^2)$. □

To conclude, this formulation allows multiple duties to be incorporated, while the running time hardly increases. This is the consequence of using the explicit time structure of the graph such that $\pi(t)$ can be found in constant time, and the use of $r_c^{daily}(t, u)$ allows for an efficient way to determine $g(u, y)$.

4.3.5 Pricing problem: extensions and implementation

In Section 4.3.4, a two-leveled pricing problem is solved, with on the lower level for daily duties, while on the higher level multiple daily duties are combined into a weekly duty. In the ECSP however, there are two more levels that need to be taken into account, because breaks within a duty are not included, and weekly resting times are also not considered.

Multiple weekly duties To allow multiple weekly duties in the schedule of one driver, while respecting the weekly resting time between two subsequent weekly duties, one can introduce:

$$h(u,y) = \text{minimum reduced costs of all schedules where the last weekly duty contains at most } y \text{ daily duties, with ending time at most } E_u + M_{u0} \text{ containing only trips in } \{1,\dots,u\}.$$

for which the recursive identity is given by:

$$h(u,y) = \min \left\{ \begin{array}{l} h(u-1,y), \\ \min\limits_{t\in[u]:Y(t,u)\leq L} \left\{ h(\pi(t),y-1) + r_c^{daily}(t,u) + \gamma \right\}, \\ \min\limits_{t\in[u]:Y(t,u)\leq L} \left\{ h(\phi(t),\delta) + r_c^{daily}(t,u) + \gamma \right\} \end{array} \right\},$$

where $\phi(t) \in T$ is a unique trip for which $E_{\phi(t)} + M_{\phi(t),0}$ is maximum and:

$$S_t \geq E_{\phi(t)} + M_{\phi(t),0} + \lambda^{rest}_{weekly} + M_{0t}.$$

The third term of the recursive function represents the possibility that a new weekly duty is started, with the (t,u)-schedule as its first duty. The choice of $\phi(t)$ ensures enough weekly resting time between the preceding weekly duty. Note that asymptotic running time of $\mathcal{O}(mn + n^2\log(n) + \delta n^2)$ does not increase compared to the approach described in Section 4.3.4, as the dynamic programs are run independently after each other.

Maximum driving times Despite being a very interesting topic, the addition of maximum driving times will be suggested as a direction for future research, but several directions are given here. Incorporating maximum driving times is significantly more difficult than incorporating maximum duty times. Recall that this approach exploits the time structure of the graph in multiple ways: to determine the duty length of a (t,u)-schedule in constant time for a given t and u, as well as to determine $\pi(t)$ in constant time.

This time structure cannot be used to determine the driving time in constant time, as this depends on the actual chosen path between t and u. It is not clear whether this even is possible within polynomial time, since the difficulty lies in finding an (all pairs) shortest path in the DAG D (as described in the proof of Lemma 4.1) where not only the sums of shadow prices is minimized, but also where the maximum driving time is respected. In other words, every edge (x,t) in the D has two weights, $-\theta_t$ and $M_{xt} + E_t - S_t$, of which the sum of the former need to be

minimized, while the sum of the latter may not exceed a specific bound, depending on t and u.

As also mentioned in 4.2.3, to distinguish the driving and duty time, the pricing problem may be formulated as a RCSPP, which is known to be NP-complete [52]. This problem consists of finding a shortest path among all paths that start from a source node, end at a sink node and satisfy a set of constraints defined over a set of resources [25]. For our problem, the driving times can be seen as a separate resource. The duty times however, due to the time structure from the graph that we exploit, does not have to be seen as a separate resource. After all, we can check in constant time whether a duty is feasible in terms of duty time. To solve the RCSPP, common methods are dynamic programming, Lagrangean relaxation, constraint programming and heuristics, for which an overview can be found in [25]. However, since the model so far already comes with high running times for realistic instances (as can be seen in the experimental results), this interesting problem has been omitted in this thesis.

Breaks Breaks within a daily duty are also not straightforward to incorporate, as they depend on the driving time instead of the duty time. Alternatively, we propose a method to include an upper bound on the breakless duty time instead of the breakless driving time to make use of the time structure. Since the driving time cannot exceed the duty time, fulfilling this constraint on the duty time fulfills the constraint on the driving time as well. This indeed however may lead to suboptimal results.

Let L^b be the maximum breakless duty time. If $L^b = L$ and the maximum breakless duty time is never exceeded, then the maximum breakless driving time is also never exceeded since the driving time is at most the duty time. To implement this, the idea is to use a similar dynamic program as in Section 4.3.4, where daily duties are combined in a weekly duty. Here, breakless duty parts will be combined in a daily duty instead, but this comes with another difficulty. Combining multiple daily duties into a weekly duty can be done easily since for every individual daily duty, the duty time is known. For breakless driving parts, this is not true, as it depends on whether the breakless driving part starting in t and ending in u is at the start, middle or end of the duty. This determines whether M_{0t} or M_{u0} needs to be added or not.

Incorporating breaks within daily duties can be achieved by using the values of $\theta(t, u)$. Note that these may also be considered as the minimum sum of shadow prices of a breakless duty part starting in t and ending in u. However, a different, more complex recursion is necessary that distinguishes the mentioned cases.

Similar as in the proof of Theorem 4.6, define $\xi(t) \in T$ as the trip with latest ending time that can act as the final trip of the breakless duty part that precedes the breakless duty part that has t as its final trip. That is, $\xi(t) \in T$ for which $E_{\xi(t)} + M_{\xi(t),t}$ is maximum and:

$$S_t \geq E_{\xi(t)} + M_{\xi(t),t} + \lambda^{rest}_{breakless}.$$

Then the recursive function can be given by:

$a(u, x) =$ minimum reduced costs of all feasible sequences of breakless duty parts with ending time at most $E_u + M_{u0}$ considering only trips in $\{1, \ldots, u\}$, and:

- $x = 1$, if the final breakless duty part of the sequence neither starts nor ends at home.
- $x = 2$, if the final breakless duty part of the sequence starts, but does not end at home,
- $x = 3$, if the final breakless duty part of the sequence ends, but does not start at home, and
- $x = 4$, if the final breakless duty part of the sequence starts and ends at home.

Initialize:

$$a(1,x) = \begin{cases} \infty, & \text{if } x = 1, \\ M_{0t} + E_u - S_t + \theta(1,1) - \gamma, & \text{if } x = 2, \\ \infty, & \text{if } x = 3, \text{ and} \\ M_{0t} + E_u - S_t + M_{u0} + \theta(1,1) - \gamma, & \text{if } x = 4. \end{cases}$$

After all, since only trip 1 is considered for $a(1, x)$, there is no feasible sequence of trips that does not start at home. If the correctness of $a(u-1, x)$ is assumed for all x, then $a(u, x)$ can be determined by the function:

$$a(u,x) = \min\{a(u-1,x), a'(u,x)\}$$

where $a(u-1, x)$ represents the case that u will not be selected as the final trip of the new sequence. This means that $a'(u, x)$ represents the case that u will be selected, for which different possibilities are available:

$$a'(u,1) = \min_{t \in [u]: E_u - S_t \leq L^b} \{E_u - S_t + \theta(t,u) + \min\{a(\xi(t),1), a(\xi(t),2)\}\},$$

$$a'(u,2) = \min_{t \in [u]: M_{0t} + E_u - S_t \leq L^b} \{M_{0t} + E_u - S_t + \theta(t,u) - \gamma\},$$

$$a'(u,3) = \min_{t \in [u]: E_u - S_t + M_{u0} \leq L^b} \{E_u - S_t + M_{u0} + \theta(t,u) + \min\{a(\xi(t),1), a(\xi(t),2)\}\},$$

$$a'(u,4) = \min_{t \in [u]: M_{0t} + E_u - S_t + M_{u0} \leq L^b} \{M_{0t} + E_u - S_t + M_{u0} + \theta(t,u) - \gamma\}.$$

Finally note that the only a sequence of breakless duty parts is a proper daily duty, if it ends at the home depot. Thus, the minimum reduced costs daily duty is the daily duty corresponding to the minimum reduced costs:

$$r_c^{daily} = \min\{a(n,3), a(n,4)\}.$$

Used approach and implementation. For this research is chosen to implement the pricing problem Section 4.3.4 without further extensions. As will be seen in the upcoming result section, the reason for this is because the computation times

already become unmanageable for realistic instances, for both the MILP and the column generation approach.

For completeness, a Branch-and-Price procedure is implemented around the mentioned column generation approach. The procedure summarized: if the solution to the master problem with generated columns (from the pricing problem) is not integral, we branch on non-integral solution values and continue branching until the solution is integral. In case no feasible solution is found within a reasonable amount of time, a standard rounding procedure is applied. See Section 2.8 or [25] for an elaborate description.

4.4 Heuristics for the ECSP

The two approaches so far, solving through integer linear programming and column generation, will find an optimal solution when the procedure completes. However, this may take a large amount of time. For practitioners, heuristic approaches may be more interesting due to the significantly shorter running time. Heuristics may be necessary for larger instances, in case an MILP solver or the Branch-and-Price procedure do not terminate within a reasonable amount of time. Also, even for smaller instances, heuristics may be preferred when a short computation time and a suboptimal solution is preferred over an optimal solution after a long computation time.

A very large amount of research has been done on heuristics on problems that are similar to ECSP. We shortly list notable heuristics for which variants have been considered or inspired for heuristics for this chapter.

- A heuristic using large neighborhood search heuristic is presented in [102], consisting of a number of competing subheuristics that are used with a frequency corresponding to their historic performance.
- A two-staged heuristic is proposed in [8], where the first stage aims to minimize the number of routes using simulated annealing, while the second stage uses Large Neighborhood Search to decrease travel time (comparable to duty time).
- A genetic algorithm is used in [96], providing competitive results.
- A tabu-embedded simulated annealing metaheuristic is proposed in [79], which restarts a simulated annealing procedure from the previous best solution after several non-improving iterations.

For more background behind these methods and heuristics in general, we also refer to [32] and [88].

The upcoming heuristics used for this chapter are intentionally not too advanced or complicated; they are intended to be simple (e.g., greedy algorithms, earliest departure time first). The reason for this is to have a comparison with any straightforward scheduling method, which is assumed to be comparable as the solution that a human planner could generate.

4.4.1 Initializing solutions

In this research, we have considered the following four types of basic, initial solutions as the basis for a heuristic:

- The **Empty** solution drives no trips at all.
- The **Random** solution is a random assignment of trips to drivers.
- The **Earliest Departure Time First (EDTF)** solution schedules trips in order of departure time to the first available driver which feasibly can drive the trip.
- The **Greedy** solution searches for the best driver-trip combination for which this assignment is feasible and increases the objective function the least. This is done iteratively until all trips are assigned.

Note that all four solutions, particularly the random solution, may not be feasible. The next step for our heuristic, is to improve any (starting) solution.

4.4.2 Improving solutions

The most straightforward and known procedure that is used to improve solutions is *Local Search*. This concept takes the current solution, and tries all possible small adjustments to see whether this gives a local improvement. The two used possible permutations to a schedule are simply the *move*-operator (moving a trip from a driver's schedule to another) and the *swap*-operator (swapping two trips from two different drivers), if the permutation results in a feasible schedule and the objective function improves. Choice of the trip(s) is done randomly until no further improvement can be found.

In practice, planners compute the planning manually and generally use a combination of ideas from the EDTF and Greedy solution, and some planners might make use of manual local search to insert new trips in an existing solution.

4.4.3 Hybrid beam-search heuristic

The heuristic proposed in this chapter is based on beam search. This technique uses breadth-first search using a search tree. This tree will iteratively be explored by expanding the most promising nodes, while discarding the least promising nodes, such that the width of the search tree stays constant.

More precisely, in the context of the ECSP, beam search initializes the search tree with the empty solution. Throughout the process, the heuristic stores at most b solutions for the next iteration, being the maximum width of the search tree. In every of the n iterations, a new trip is considered (in increasing order of departure time), and at iteration t when scheduling trip t, the algorithm simply extends every of the stored b solutions (which all have exactly all trips until $t-1$ scheduled) k times, by creating a new solution including trip t to every driver's schedule. This leads to at most bk solutions.

Subsequently, for every of the bk solutions, we apply local search. For every trip in every solution σ, we reconsider whether swapping to another driver improves the solution, in addition to considering the swap of two trips of two different drivers. These two operators run in $\mathcal{O}(bk^2n)$ and $\mathcal{O}(bkn^2)$, respectively.

To prevent the search tree to grow exponentially in this way, the best b solutions are kept afterwards, meaning that at most $b \cdot (k-1)$ solutions are discarded. See Algorithm 2 for the method in pseudocode. We:

- defined Σ as the set of all solutions,
- defined $LS(\sigma)$ as applying the local search procedure on solution σ, and
- defined $BS(\Sigma)$ is abbreviated as keeping the best b solutions in Σ and discarding the other solutions. No duplicate solutions are allowed.

Algorithm 2 Hybrid beam-search heuristic

$\sigma_1 \leftarrow (s_1, \dots s_k) = (\emptyset, \dots, \emptyset)$
$\Sigma = \{\sigma_1\}$
for $t = 1, \dots, n$ **do**
  for all $\sigma = (s_1, \dots, s_k) \in \Sigma$ **do**
    for $i = 1, \dots, k$ **do**
      $\sigma' \leftarrow (s_1, \dots, s_i \cup \{t\}, \dots, s_k)$
      $\sigma'_{LS} \leftarrow LS(\sigma')$
      $\Sigma \leftarrow \Sigma \cup \{\sigma'_{LS}\}$
    end for
  end for
  $\Sigma \leftarrow BS(\Sigma)$
end for
return $\arg\min_{\sigma \in \Sigma} Y(\sigma)$

Note that it is of high importance that the trips are iterated upon in increasing order of departure time. After all, if many trips that have overlap are considered first, a larger variety in the solutions in Σ occur, allowing a broader exploration of the total state space of solutions.

The choice of b completely depends on the size instance, and is chosen in such a way that the computation time stays below 1 minute (arbitrarily chosen) per instance. Also note that, when the choice of b increases, the computation time only increases linearly.

Preliminary experiments have shown that this beam-search based heuristic always outperforms EDTF and Greedy solutions, even after improving these solutions using local search. For this reason, this heuristic will be considered for comparison with column generation in the next section.

4.5 Experimental results

Solving the proposed master problem for column generation has been done in IBM ILOG CPLEX Optimization Studio (12.7.1), integrated in Java.

4.5.1 Experimental data

For the research purpose of this thesis, multiple datasets are provided from Dutch coach (or touring car) companies that are active throughout Europe, and within the Netherlands in particular. From these datasets, realistic daily patterns have been extracted, such as the demand pattern (i.e., the number and proportion of trips that need to be driven at every hour of the day). Using these patterns, new datasets can be generated while maintaining realistic proportions.

The generated datasets comprise 60 locations, the 60 cities with the highest population in the Netherlands, using Amsterdam as the home depot. Traveling times are based on the straight line distances ("as the crow flies"), based on coordinates of the cities. Trips are generated randomly, but for a more realistic structure regarding demand, the chance for a city to be requested within a trip is proportional to their population. A city with twice as higher population therefore has exactly a twice as bigger chance to be a location of departure or arrival for a specific trip. With these settings, the further used parameters are:

	Single duty case	Multi duty case
Number of days (timespan)	1	7
Max. # of daily duties (δ)	1	6
Number of drivers (k)	$\{1, 2, \ldots, 50\}$	$\{5, 10, \ldots, 50\}$
Number of trips (n)	$2k$	$10k$

Table 4.1: Parameters for experimental data generation

Thus, for the single duty case we consider scheduling for a single day only. For the multi duty case, we consider a weekly schedule, where drivers can have at most 6 daily duties, in line with the European regulations. Using these parameters, we are interested in how the methods compare as the instance grow.

Furthermore, we will only use instances for which we know a feasible solution exists for all methods, to provide a fair comparison regarding the objective function of the methods. Finally, for every k, we generate 25 instances and take the average objective value and computation time to provide more precise results. Also, a maximum computation time of 1 hour per 25 instances is chosen (approximately 2 and a half minutes per instance). If this time is exceeded, the best solution so far is returned.

4.5.2 A note on the running time

Before analyzing the column generation approach and heuristics, we comment on the computation time of solving the MILP as described in Section 4.2.2 for the single duty. If enough time is available, the column generation solution is always equal to the MILP solution value as they both should eventually find an optimal solution. However, a MILP solution is provided in significantly more time, even for small instances (see Figure 4.2). This clearly should hold for multiple duties as well. Since column generation is superior to solving the MILP formulation in terms

of computation time, we disregard the MILP, and focus mainly on the alternative approaches.

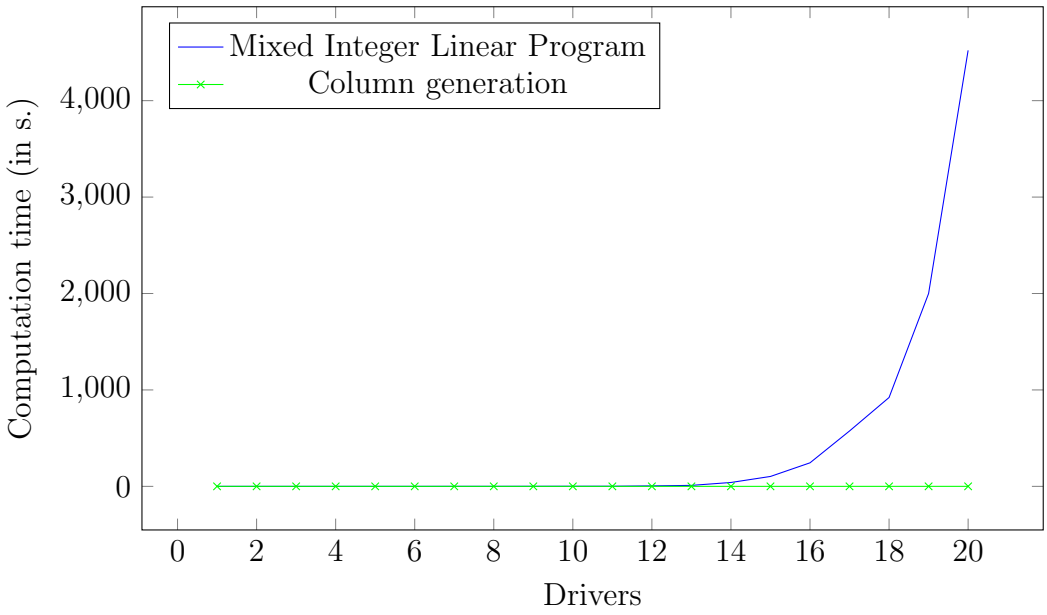

Figure 4.2: Computation times: MILP solvers vs. column generation (single duty)

4.5.3 Comparison of performance

To compare the performance of algorithms, we evaluate the objective values (sum of duty times) as the instances grow. However, an instance with twice as many drivers clearly approximately has a twice as high objective value, given the same approach. For this reason, the objective function is divided by the number of drivers to have a more comparable performance measure in the average duty time.

Furthermore, we compare results of the approaches to a lower bound on the optimal integer solution value. This bound is equal to the optimal solution value for the LP-relaxation of the master problem; it is well known that as soon as the pricing problem outputs a variable with positive reduced costs, the optimal solution (value) is found for the LP-relaxation. Since we consider a minimization objective, the optimal solution value for the LP-relaxation is always lower than the optimal solution for the ILP.

Daily duty case. Considering the daily duty case first, we note that column generation produces always the optimal result given our range of drivers, making the approach superior (See Figure 4.3). Note that the heuristic is not too far off, but clearly suboptimal.

Note that optimal objective values (or rather, averages per driver) decrease as the number of drivers and trips grow, since this enables more trip combinations in a duty to reduce idle time between trips.

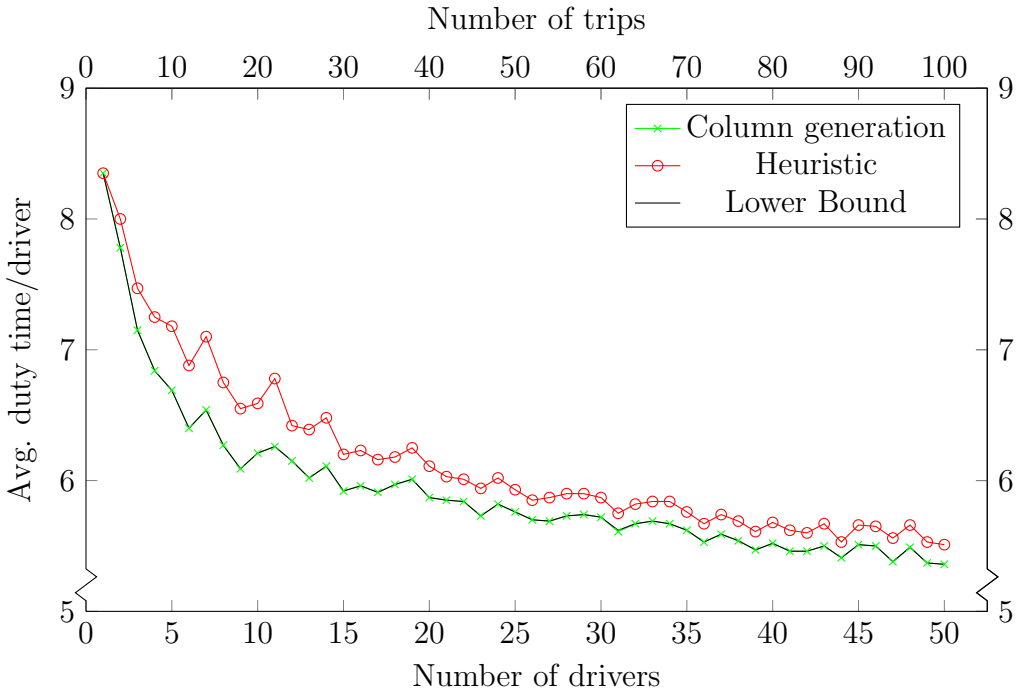

Figure 4.3: Optimality results for the single duty ECSP

Weekly duty case. The weekly duty case shows a different pattern, since the parameters are chosen in such a way that also column generation takes too long to terminate when the number of drivers is 15 or higher (See Figure 4.4). Thus, the solution values from the column generation are (in contrast to the single duty case) not optimal.

With this in mind, note that both the column generation and other heuristics are always within 10% of the lower bound, and therefore also of the optimal solution value, indicating an acceptable performance for practice. Moreover, we observe that there is no superior approach among the two. In fact, column generation outperforms (albeit only slightly) heuristics in approximately 50% of the individual instances, making an OR-based approach such as column generation very competitive to heuristic approaches for ECSP.

Another reason why column generation does not perform optimally (unlike the single duty case), is due to the increased number of possible schedules a driver can drive, given the same number of trips. More pairs of trips are compatible if they are spread over a week. As a consequence, the possible number of schedules in the master problem increased significantly, including the computation time to solve it. Nevertheless, if the instance does not grow larger than these parameters, column generation is certainly a viable practical alternative for smaller transport companies that want an optimal schedule.

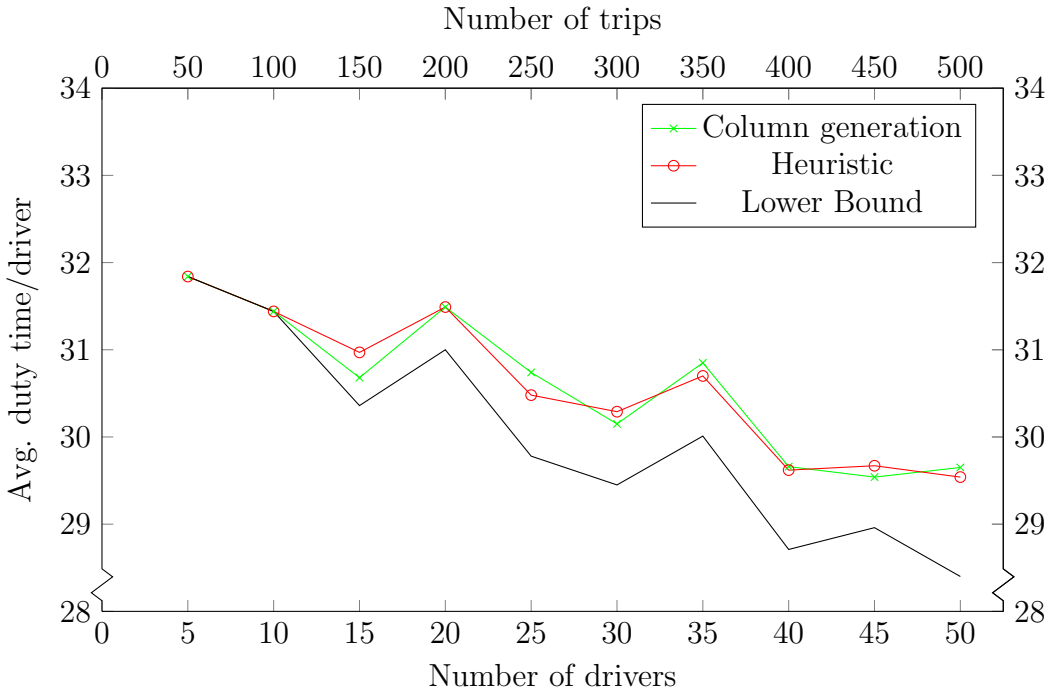

Figure 4.4: Optimality results for the multi duty ECSP

4.6 Conclusions and future work

This chapter introduced and studied the ECSP, where a given set of trips need to be driven by a given set of drivers. Motivated by practice (European regulations), the uniqueness of the ECSP comes from the large variety of constraints on the maximum driving and duty times, as well as minimum resting times between duty parts, daily duties and weekly duties. A new MILP formulation for the ECSP has been proposed in this chapter, which can incorporate all constraints.

From a more theoretical point of view, it has been shown that a very fundamental version of the problem, Path Cover with weight bounds, is NP-hard. This means that the ECSP is also strongly NP-hard. For this reason, alternative approaches were proposed that considered special cases of the ECSP first (single daily and single weekly duty case, without breaks). The first approach concerned a column generation approach, where all constraints are bounded to duty times rather than driving times. Other methods included heuristics that improve the solution based on local search or beam search. While the MILP and heuristics are standard, the column generation distinguishes itself by its explicit use of the time element in the model input. This allows the incorporation of multiple duties with resting times in between, without significant increase in the running time. It is also argued this how this method can be extended such that breaks can be incorporated, and how multiple weekly duties for a driver can be combined to fulfill all constraints.

Experimental results show that solving the column generation performs better than the standard MILP with a solver in terms of running time. In the daily duty case, the optimal solution value is always obtained within a reasonable amount of

time. For the weekly duty case, the instance size as well as the running time grows significantly. The MILP was not able to find a solution for such instances within a reasonable amount of time, while the column generation approach finds a feasible solution, but also may not have enough time to find an optimal solution. As such, it is compared to a heuristic which also is not guaranteed to find an optimal solution. It is shown that both the column generation and the heuristic perform well, being both consistently within at most 10% of the lower bound of the optimal solution, and competitive to each other.

Future work. As mentioned, the column generation approach is able to take resting times and duty times into account. Duty times could be incorporated in a structured and efficient way because the time structure of the graph allowed the algorithm to determine in constant time whether a duty starting in trip t and ending in trip u is feasible.

For driving times, this is not the case. This time structure cannot be used to determine the driving time of a duty starting in trip t and ending in trip u in constant time, as this depends on the actual chosen path between t and u. It is not clear whether this even is possible within polynomial time, since the difficulty lies in finding an (all pairs) shortest path in the DAG D (as described in the proof of Lemma 4.1) where not only the sums of shadow prices is minimized, but also where the maximum driving time is respected. In other words, every edge (x, t) in the D has two weights, $-\theta_t$ and $M_{xt} + E_t - S_t$, of which the sum of the former need to be minimized, while the sum of the latter may not exceed a specific bound, depending on t and u.

Therefore, the addition of maximum driving times is suggested as the main direction for future research. In Section 4.3.5, a direction for this has been given by formulating the pricing problem as a RCSPP, where the driving times are seen as a separate constraint. Common methods to solve a RCSPP are dynamic programming, Lagrangean relaxation, constraint programming and heuristics.

Chapter 5

Hospital Planning

This chapter introduces the Room and Ward Planning Problem (RWPP), where the goal is integrate the planning of patients on operating rooms and wards. To deal with the various stochastic variables of this problem (number of available patients, procedure times and hospitalization times), an Integer Linear Program using linearization methods will be proposed.

After discussing the background of hospital planning in Section 5.1, the problem under consideration will be defined in Section 5.2. An Integer Linear Programming (ILP) method will be proposed in Section 5.3, for which its effectiveness will be experimented on a case study of the VU University Medical Center in Section 5.4. Finally, several conclusions will be given in Section 5.5.

The results in this chapter are based on [117] and require basic knowledge on integer linear programming and probability theory.

5.1 Background

Operating Rooms (ORs) form a key facility of all major hospitals, but are also one of the most expensive resources of a hospital [49], as their operational costs can account for up to 40% of the total resource costs [4]. For a good utilization of these ORs, advanced scheduling plays a crucial role [84]. However, scheduling of interventions is complicated due to many uncertain factors, including variability in the number of patients, urgency of patients, and the duration of such interventions (see, e.g., [50, 78, 89, 97, 104, 110]). Next to the OR, the availability of a hospital bed at a proper ward is necessary for the aftercare of a patient, making scheduling even more involved.

For this chapter, the focus lies on catheterization laboratories (also known as cath labs or cardiac catheterization rooms) due to the corresponding case study within this research. A cath lab is a special type of hospital room in which doctors (mainly cardiologists) perform invasive cardiovascular procedures to diagnose, visualize and treat cardiovascular diseases. Patients undergoing a treatment in a cath lab generally also require preparatory care and aftercare at one of the hospital's wards, which adds an extra layer to the already complicated planning problem.

In practice, scheduling of procedures for cath labs and ORs is mainly done by

hand. The disregard of automated scheduling methods for cath lab and ORs can be explained by the fact that many models make simplistic assumptions that ignore the actual practical complexity of the problem, such as the inherent uncertainty of operating and hospitalization times for patients. The work in this chapter aspires to incorporate this stochasticity into Integer Linear Programming methods to make automated methods more viable.

From an Operations Research perspective, the optimization of the schedules of cath labs may be considered to be a special case of Operating Room scheduling, for which already many studies and surveys exist [15, 17, 51, 59, 100]. Some relevant similarities and differences in the context of the work in this chapter are discussed here.

Similarities with ORs. An important similarity is that both ORs and cath labs may operate a combination of elective and urgent patients. This means that the operating sessions cannot be planned fully to anticipate on the possible arrival of urgent procedures. Note that it is an optimization problem on itself to determine how much time should be reserved for urgent patients every session. Moreover, the majority of the procedure times are very hard to estimate, as the required actions become known no sooner than during that procedure. Also, different patient categories can only be operated on specific cath labs due to the available equipment per room. There are more aspects that the optimization of cath labs and ORs share, but these are the most relevant ones that will be taken into account for the models within this chapter.

Differences with ORs. The main characteristics that distinguish the planning of cath labs from ORs are as follows. In planning and scheduling for ORs, the challenges can be categorized in several levels of a decision hierarchy [53]: strategic, tactical, and operational, where the operational level can be distinguished in an offline (planning in advance) and online (reacting and monitoring) level. At the tactical level, OR-days (or blocks) are allocated to specialties in a master surgery schedule, such that the strategic allocation is met. However, while ORs are used by many medical disciplines involving different departments, cath labs are dedicated to cardiology, meaning that the tactical level is smaller.

Therefore, cardiology has the option to adjust the schedule's structure as they desire, such as the starting time or total duration of a session. This flexibility is a feature for cath labs in the case study later in this research, which adds to the potential complexity of the optimization problem. Furthermore, ORs typically first divide their sessions over the different disciplines (or specialties). This intermediate step is not necessary for cath labs, which simplifies the problem.

Another notable difference for cath labs as opposed to ORs is simply the size. While many large hospitals manage at least 10 to 20 ORs, the number of cath labs is typically only a few. This makes fixed-parameter tractable algorithms, i.e., algorithms that work efficiently for problems with small and fixed parameters, more suitable. For instance, the hospital in the case study considered in this chapter, the VU University Medical Center (VUmc), contains three cath labs.

The fact that cath labs are fully dedicated to one medical discipline allows to make an interesting special case of OR scheduling. However, many of the ideas and methods in this chapter can be applied to ORs as well. Therefore, a cath lab or operating room is within this chapter simply referred to as a "room", unless stated otherwise.

5.2 The Room and Ward Planning Problem

5.2.1 Motivation

Due to (semi-)urgent patients and the uncertainty in the procedure lengths, planners have complications to estimate the number of patients that can be treated at a specific day. When too few patients are scheduled, the available time of the staff and rooms are not used efficiently, which decreases the profit and increases the length of the waiting list. On the contrary, when too many patients are scheduled, staff is forced to work longer than desired, or patients need to be rescheduled on another day, which not only is a very negative experience for the patient, but also can have negative effects on the patient's health.

Types of patients. The choice of patients that need to be scheduled strongly depends on the procedure type, but also depends on the availabilities of surgeons and the degree to which urgent surgeries need to be taken into account. More importantly, all patient types require their own preparation and aftercare at wards that have a limited capacity. This means that the schedules at the rooms are limited by the available capacity at the wards. Typically, the capacity constraints at the wards are more stringent towards the end of the week (Thursdays and Fridays) than during the weekend and early in the week (see, e.g., [5]). This can be explained by the admission pattern, as elective procedures are not scheduled during weekends leading to peaks in bed demand towards the end of the week. Thus, scheduling of patients/procedures at rooms requires an integral approach involving both rooms and the beds at the wards.

5.2.2 Definition

Define R as the number of rooms for procedures, and W as the number of wards that can provide the required preparation and aftercare for patients. Without loss of generality, we assume that every patient always undergoes a procedure in a room and is assigned to a ward (if not, a model can easily deal with this using dummy rooms or wards with infinite capacity).

The schedule will be created for a fixed time horizon, comprising D days. In practice, it is natural to set this value to 7 to create a weekly schedule. Let C be the number of patient categories (or procedure types), and each category has its own probability distribution with respect to its procedure time on a room and hospital-

ization time on a ward. Therefore, the following random variables are introduced:

$$\begin{aligned} H_c &= \text{required hospitalization time (in days) for category } c \text{ at a ward,} \\ P_c &= \text{required procedure time (in hours) for category } c \text{ at a cath lab,} \\ Z_{cd} &= \#\text{ urgent patients of category } c \text{ on day } d. \end{aligned}$$

We assume that the mean and variance of the above random variables are known. For the case study in this chapter, these are based on an extensive data analysis.

As mentioned before, the creation of a schedule is simply the assignment of procedures to time slots. For this reason, the following key decision variable is used:

$$x_{cdrw} = \#\text{ patients of category } c \text{ scheduled on day } d, \text{ room } r \text{ and ward } w,$$

where $c \in [C]$, $d \in [D]$, $r \in [R]$ and $w \in [W]$. Note that a distinction has been made between elective and urgent patients. On a specific day d, the number of scheduled patients of a category c, $\sum_{r=1}^{R} \sum_{w=1}^{W} x_{cdrw}$, has to be at least the number of urgent patients of that day and category, Z_{cd}. The surplus represents the number elective patients. In practice, this may have an upper bound (i.e., the waiting list is finite). This has been omitted for simplicity, but easily can be incorporated using an upper bounds on the number of patients per category that can be scheduled, say $X_{c,max}$. Additionally, the following auxiliary variables are introduced:

$$\begin{aligned} \tau_{dr} &= \text{available time for procedures on day } d \text{ at room } r \text{ (in hours),} \\ \upsilon_{dr} &= \text{expected overtime on day } d \text{ at room } r \text{ (in hours),} \\ \beta_{dw} &= \text{available number of beds on day } d \text{ at ward } w, \\ \lambda_{dw} &= \text{expected overload on day } d \text{ at ward } w \text{ (in beds).} \end{aligned}$$

Depending on the practical situation, τ_{dr} and β_{dw} can either be decision variables or given constants. The former would be the case if the tactical decision would involve the opening times of the cath labs in addition to the blue print. Still, there exists an upper bound on τ_{dr}, say $T_{r,\max}$, which within this context could be up to 24 hours. For β_{dw}, a similar maximum per ward, say $\beta_{w,\max}$, can be argued by the fact that there is physical limited room capacity. Moreover, υ_{dr} and λ_{dw} are used to obtain the expected overtime and expected ward overload as performance indicators. These will be used in the upcoming ILP in Section 5.3.

Due to the expensive (and therefore limited) equipment at the rooms, patients of specific categories cannot be treated at every room, and even not on every day, due to, e.g., the availabilities of surgeons. Also, not every patient can be hospitalized at every ward, primarily due to the specialized skills of the nurses at the corresponding ward. In other words, for the rooms, wards and also days, there is only a set of categories of patients that can be treated. To account for these so-called "compatibilities", the following sets are introduced and assumed to be known:

$$\begin{aligned} \delta_d &= \text{set of patient categories that can be treated on day } d, \\ \rho_r &= \text{set of patient categories that can be treated at room } r, \\ \omega_w &= \text{set of patient categories that can be hospitalized at ward } w. \end{aligned}$$

Note that $\delta_d, \rho_r, \omega_w \subseteq [C]$ for $d \in [D]$, $r \in [R]$ and $w \in [W]$. The following constants are also required to be part of the input, and are closely related to the performance indicators. In particular, these constants represent the relative weights for each of the five different performance measures. Such a weighted combination of performance measures is a standard mathematical approach for comparing the quality of different schedules for multiple criteria. Hence, we require the following five constants for our model:

$$\begin{aligned} V_{\tau,r} &= \text{costs per opened hour of room } r \text{ (including staffing costs)}, \\ V_{\upsilon,r} &= \text{costs per hour overtime at room } r, \\ V_{\beta,w} &= \text{costs per reserved bed at ward } w \text{ (including staffing costs)}, \\ V_{\lambda,w} &= \text{costs per overloaded bed at ward } w, \\ R_c &= \text{profit of treating a patient of category } c. \end{aligned}$$

Using these constants, the following weighted objective function is introduced:

$$\Pi(x, \tau, \upsilon, \beta, \lambda) = \pi_\tau(\tau) + \pi_\upsilon(\upsilon) + \pi_\beta(\beta) + \pi_\lambda(\lambda) - \pi_P(x),$$

of which the individual terms are defined as follows:

- $\pi_P(x)$ is the expected profit obtained from all elective procedures that are performed during the scheduling horizon. Since R_c is defined as the profit obtained for treating a patient of category c, this simply implies that:

$$\pi_P(x) = \sum_{c=1}^{C} \sum_{d:c\in\delta_d} \sum_{r:c\in\rho_r} \sum_{w:c\in\omega_w} R_c \cdot x_{cdrw}.$$

- $\pi_\tau(\tau)$ is the expected costs from staffing/operational costs due to the opening time of the rooms, i.e.:

$$\pi_\tau(\tau) = \sum_{d=1}^{D} \sum_{r=1}^{R} V_{\tau,r} \cdot \tau_{dr}.$$

- $\pi_\upsilon(\upsilon)$ is the expected costs from overtime at rooms, i.e.:

$$\pi_\upsilon(\upsilon) = \sum_{d=1}^{D} \sum_{r=1}^{R} V_{\upsilon,r} \cdot \upsilon_{dr}.$$

- $\pi_\beta(\beta)$ is the expected costs from staffing/operational costs due to the available beds at the wards, i.e.:

$$\pi_\beta(\beta) = \sum_{d=1}^{D} \sum_{w=1}^{W} V_{\beta,w} \cdot \beta_{dw}.$$

- $\pi_\lambda(\lambda)$ is the expected costs from overloaded wards, i.e.:

$$\pi_\lambda(\lambda) = \sum_{d=1}^{D} \sum_{w=1}^{W} V_{\lambda,w} \cdot \lambda_{dw}.$$

Observe that the objective function is clearly linear in all its decision variables. The decision on the relative values of the weights R_c and vector V is typically a managerial decision. These definitions suffice to define the problem under consideration extensively, but precisely.

Room and Ward Planning Problem (RWPP)	
Given:	A number of rooms R with maximum opening time $T_{r,\max}$ for every $r \in [R]$, wards W with maximum capacity β_w for every $w \in [W]$, patient categories C and days D, vectors of random variables H (hospitalization time at ward) and P (procedure time at room), expected urgent patient matrix Z, compatibility vectors δ (days), ρ (rooms) and ω (wards), a cost vector V and patient profit vector R.
Goal:	Find an assignment of patients to rooms, wards and days that minimizes $\pi(x, \tau, \upsilon, \beta, \lambda)$.

5.2.3 Related work

Scheduling of ORs is a well-studied area in Operations Research, see e.g. [17, 59, 100] for some literature reviews in health care. For reviews specifically related to operating room planning and scheduling, the reader is referred to [15, 24]. For more background on integral capacity management in hospitals is referred to [103]. For scheduling in cath labs in particular, no other references could be found; cath labs are also not mentioned at all in the above mentioned reviews.

References considering the planning of ORs taking the bed occupancy of the wards into account has been done very extensively. A good variety of methods and examples are given in [1, 6, 7, 40, 55, 120, 115]. These works are focused on surgery scheduling, or scheduling of time blocks or elective admissions of the Operating Theater (OT), with the objective that the resulting bed occupancy is balanced, while the scheduled procedures fit in the available OR time. For the occupation of the OR and the wards, only the mean is considered, such that the scheduling problem can essentially be formulated as an ILP model. In this research, the expected overtime and the expected excess at the wards is included in the ILP.

The stochasticity in bed demand as a result of the surgical schedule is fully analyzed in [121]. Yet, this work does not contain an optimization algorithm and improvement is accomplished by trial and error.

In this chapter, a linearization method will be used to approximate some of the constraints, such as the expected overtime on a room. More recently and closely related to this research, [104] introduced a two single step approach for scheduling a very similar problem as the RWPP. It extends the model of [121] and takes the

overtime constraint into account, while maximizing the OR utilization and minimizing variation of bed usage. It proposes an alternative version of the MILP including linearization methods compared to the model that is discussed in this chapter.

5.2.4 Contributions

The contributions of this chapter are the following. First, a proof of strongly NP-hardness of the RWPP will be given in Section 5.2.5 by a reduction from the Bin Packing Problem. This will show that the RWPP is already NP-hard without stochasticity and when either the ORs or the wards are disregarded.

However, the main contribution of this thesis consists of the presentation of a linearization method for a model where all constraints and preferences for our case study can be taken into account. After all, the model presented in Section 5.2.2 does more than simply integrating ORs with wards, which has been considered very often before in the literature. Additionally, this model incorporates stochasticity of procedure times, hospitalization times and urgent arrivals, while minimizing the weighted sum of important performance measures for ORs and wards that could not have been found in the literature simultaneously. These measures concern the costs per opened hour of a room and bed, the costs of the expected overtime and overload of the beds, and on the other side the profit per treated patient. This means that the model implicitly is able to make the trade off between these performance measures, while the user of the model can assign weights to the performance measure.

As an example, note that this does not require that the costs per hour overtime or profit per treated patient to be expressed in funds, but allows to incorporate the weight of the satisfaction. For example, if working overtime is undesired by the employees, a hospital can simply increase the weight of the overtime costs. On the other hand, patients from a category c with higher priority can be assigned a higher profit R_c such that the model values the scheduling of such patients higher.

This model was solved using a linearization method. Even though this linearization is an approximation, one can increase the quality of the approximation, but this comes with the addition of more constraints to the model. Although linearization of objective functions are not uncommon, an application to approximate the expected overtime of a room could not have been found before in the literature. The reason why this now is possible is due to a new assumption, which informally implies for every procedure performed within one specific cath lab holds that whenever the expected procedure time is longer, the variance is also longer. For rooms where only one specialty operates (in this case, cardiology), this assumption is much more justifiable than in a general room setting with multiple specialties. However, we argue that this assumption is not necessary, but that would make the use of the model significantly slower.

Finally, a case study (VU University Medical Center) will be presented, where some insights in data regarding procedure times are given, and where the model is applied. This results in a planning for the hospital that can be used as a blueprint.

5.2.5 Complexity

The computational complexity of the RWPP can be shown by a simple proof.

Theorem 5.1. *RWPP is strongly NP-hard.*

Proof. The proof is done by a reduction from the decision variant of the Bin Packing Problem (BPP). This problem is given sets of bins $S_1, S_2, \ldots$ with equal size B, and a list of n items with size $a_1, \ldots, a_n$. The question is whether all items can be packed within K bins. For the decision variant of the RWPP, the question is simply whether a feasible schedule exists (i.e., whether all urgent patients can be scheduled).

Given an instance of the decision variant of the BPP, construct the following instance for the decision variant of the RWPP:

- $R = K$,
- $C = n$,
- $W = D = 1$
- $P_c = a_c, H_c = 0$, for $c \in [C]$,
- $Z_{cd} = 1$, for $c \in [C]$, $d \in [D]$,
- $\delta_d = \rho_r = \omega_w = [C]$, for $d \in [D], r \in [R], w \in [W]$,
- $T_{r,\max} = B$, for $r \in [R]$, and
- all profits and costs are 0.

In other words, the patients and rooms represent the items and bins respectively, while all other aspects of the RWPP such as the costs, days and even the wards are disregarded. Since $W = D = 1$, the decision variable x_{cr} instead of x_{crdw} is considered. Every patient has to be scheduled since they are all urgent. Now the remainder of the proof should be straightforward due to this one-to-one correspondence.

If the instance for the BPP is a YES-instance, then this means that $\sum_{i \in S_j} a_i \leq V$ for $j \in [K]$. Now schedule on every room $j \in [K]$ every patient i for which $i \in S_j$. This gives a total operating time on room j equal to $\sum_{i:i \in S_j} P_i = \sum_{i \in S_j} a_i \leq B = T_{r,max}$, meaning that the room schedule is feasible and all (urgent) patients are scheduled by construction. As a result, the instance for the RWPP is also a YES-instance.

Conversely, if the instance for the RWPP is a YES-instance, a similar argument can be made to show that the instance for the decision variant of the BPP is a Yes-instance. As a result, the RWPP is NP-hard. □

Note that a similar proof can be given where the wards represent the bins, and the rooms are disregarded instead.

For this reason, creative techniques are required to optimize the problem. Yet, to make the mathematical models relevant for practice, it is essential to incorporate realistic assumptions. This chapter aims to make an important step towards that direction.

5.3 An ILP method for the RWPP

For the reader's convenience, a model without variability will be presented first, to illustrate the basic ideas of the model in a simple setting. Afterwards, stochastic procedure and hospitalization times are incorporated in the model.

5.3.1 Model without variability

In the first model formulation, stochasticity of procedure and hospitalization times will not yet be taken into account, i.e., $Var(H_c) = 0$ and $Var(P_c) = 0$ for every $c \in [C]$. Thus, given an assignment of procedures to days, rooms and wards, the required procedure time for day d, room r and ward w in this model can simply be determined by:

$$\sum_{c=1}^{C} \sum_{w:c\in\omega_w} \mathbb{E}(P_c) \cdot x_{cdrw}.$$

The ward occupancy is slightly more difficult to determine, as admissions of previous days need to be taken into account as well. To illustrate, a patient admitted at day $d-i$ is still present at day d with probability $\mathbb{P}(H_c \geq i)$. Summing over all previous days, the mean occupancy of ward w at day d is:

$$\sum_{c=1}^{C} \sum_{r:c\in\rho_r} \sum_{i=0}^{d} \mathbb{P}(H_c \geq i) \cdot x_{c(d-i)rw}.$$

This model also gives the possibility to make the blueprint periodic, meaning that $x_{cdrw} = x_{c(d-kD)rw}$ for some $k \in \mathbb{N}$, but it is not required. After all, in practice, many hospitals prefer a (bi-)weekly schedule for consistency.

Given these definitions, one can formulate an ILP without variability as follows:

$$\min \Pi(x, \tau, \upsilon, \beta, \lambda) \tag{5.0}$$

$$\text{s.t.} \sum_{c=1}^{C} \sum_{w:c\in\omega_w} \mathbb{E}(P_c) \cdot x_{cdrw} \leq \tau_{dr} + \upsilon_{dr} \qquad d \in [D], r \in [R] \tag{5.1}$$

$$\sum_{c=1}^{C} \sum_{r:c\in\rho_r} \sum_{i=0}^{d} \mathbb{P}(H_c \geq i) \cdot x_{c(d-i)rw} \leq \beta_{dw} + \lambda_{dw} \qquad d \in [D], w \in [W] \tag{5.2}$$

$$\upsilon_{dr} \geq 0 \qquad d \in [D], r \in [R] \tag{5.3}$$

$$\lambda_{dw} \geq 0 \qquad d \in [D], w \in [W] \tag{5.4}$$

$$B_{min,w} \leq \beta_{dw} \leq B_{max,w} \qquad d \in [D], w \in [W] \tag{5.5}$$

$$E_{min,c} \leq \sum_{w:c\in\omega_w} \sum_{d:c\in\delta_d} \sum_{r:c\in\rho_r} x_{cdrw} \leq E_{max,c} \qquad c \in [C] \tag{5.6}$$

$$T_{min,r} \leq \tau_{dr} \leq T_{max,r} \qquad d \in [D], r \in [R] \tag{5.7}$$

$$x_{cdrw} \in \mathbb{N}_0. \tag{5.8}$$

See Section 5.2.2 for a definition of the objective function. The constraints can be justified as follows:

- Constraint (5.1) implies that the sum of the scheduled procedures on every room per day, must be smaller than the opening time plus the overtime of that room. The overtime of the room, υ_{dr}, is a decision variable that adds to the penalty. But since there is no upper bound on υ_{dr}, this constraint is a soft constraint which always can be fulfilled.

- Constraint (5.2) implies that the expected occupation on every ward per day must be smaller than the number of reserved beds (the capacity) plus the excess demand (also referred to as *overload*).

- Constraints (5.3) and (5.4) imply that the expected overtime per room and expected overload per ward has to be at least 0, such that underload is not rewarded.

- Constraints (5.5), (5.6), and (5.7) state that the available number of beds, the number of admitted patients over the time horizon, and the opening hours of the rooms, respectively, should be between their upper and lower bound.

- Constraint (5.8) is the standard requirement that the number of elective procedures per day cannot be negative and must be integer.

In constraint (5.1), urgent patients are not yet taken into account, which clearly can affect the optimal opening time of the rooms. A simple way to account for this group is to replace x_{cdrw} by $\left(x_{cdrw} + \frac{\mathbb{E}(Z_{cd})}{|\{r:c\in\rho_r\}|}\right)$. Here, $\mathbb{E}(Z_{cd})$ is the expected number of (semi-)urgent patients of category c on day d, and $|\{r : c \in \rho_r\}|$ is the number of rooms at which these patients can be treated. The additional term represents the number of (semi-)urgent patients that on average need to be treated per day, assuming that these patients are spread evenly among the rooms.

Likewise, (semi-)urgent patients should also be taken into account for the wards in constraint (5.2), by replacing $x_{c(d-i)rw}$ by $x_{c(d-i)rw} + \frac{\mathbb{E}(Z_{cd})}{|w:c\in\omega_w|}$. Also, the model can account for the preparatory care for patients, for which the number of days is usually fixed per patient category c, say $q_c \in \mathbb{N}$. This can be done by adding the term $\sum_{i=1}^{q_c} x_{c(d+i)rw}$ on the left-hand side of the inequality-sign (which is not added for simplicity). After all, if a patient is scheduled for operation on day $(d+i)$ and its preparatory care requires at least i days, the patient is present at the ward on day d.

5.3.2 Overtime and excess demand

This section considers the overtime and excess demand (or overload) if stochasticity in procedure times and occupancy is taken into account. Specifically, let μ_{dr} and σ^2_{dr} denote the mean and variance of the total procedure time (sum of all procedures) at

day d for room r. These first two moments depend on the decision variables x_{cdrw} and are directly given by:

$$\mu_{dr} = \sum_{c=1}^{C} \sum_{w=1}^{W} \mathbb{E}(P_c) \cdot x_{cdrw} \qquad d \in [D], r \in [R] \tag{5.9}$$

$$\sigma^2_{dr} = \sum_{c=1}^{C} \sum_{w=1}^{W} Var(P_c) \cdot x_{cdrw} \qquad d \in [D], r \in [R] \tag{5.10}$$

for $d \in [D]$ and $r \in [R]$. The assumption here is that the **total** procedure time on a day follows a normal distribution. This assumption is motivated by the Central Limit Theorem stating that the scaled sum of independent and identically distributed random variables converges to a normal distribution. Although the duration of a single procedure is typically not normally distributed, the sum of enough procedures often is.

The expected overtime υ_{dr} at any room and day can now be determined using the overshoot over a fixed level of the normal distribution. Let T denote the opening time and drop the indices from the notation of υ_{dr}, μ_{dr}, and σ_{dr} for now for notational convenience. Then using a standard integral to calculate the overshoot of a normal distribution, its expectation can be calculated by:

$$\upsilon = \int_T^\infty (t - T) \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2} dt.$$

A similar expression can be obtained for the expected overload at a ward, λ_{dw}. Observe that the occupied beds at day d consists of patients that arrived at some day $d - i$ for $i \geq 0$. A patient arriving at day $d - i$ is still present at day d with probability $\mathbb{P}(H_c \geq i)$. This means that for each patient there is a Bernoulli random variable that can indicate whether the patient is present at day d or not. The total occupancy may thus be represented by a sum of Bernoulli distributions (with different parameters), naturally leading to an approximation of a normal distribution for the ward occupancy as well.

To obtain the first two moments of the ward occupancy, note that the variance of a Bernoulli random variable with parameter p equals $p(1 - p)$. Thus, the mean (μ_{dw}) and variance (σ^2_{dw}) at day d for ward w are:

$$\begin{aligned}
\mu_{dw} &= \sum_{c=1}^{C} \sum_{r:c\in\rho_r} \sum_{i=0}^{\infty} \mathbb{P}(H_c \geq i) \cdot x_{c(d-i)rw}, \\
\sigma^2_{dw} &= \sum_{c=1}^{C} \sum_{r:c\in\rho_r} \sum_{i=0}^{\infty} (1 - \mathbb{P}(H_c \geq i)) \cdot \mathbb{P}(H_c \geq i) \cdot x_{c(d-i)rw}
\end{aligned}$$

A key issue for the ILP is the non-linear expression for the expected overtime υ; the expressions for μ_{dr} and σ^2_{dr} are linear in the decision variables, but these terms also appear in the exponent within the integral. To overcome this issue, a piecewise linear approximation of the expected overtime will be proposed in the next section.

5.3.3 Linear approximation of overtime

In the ILP model, the expected overtime υ is part of the minimization, whereas it contains non-linear expressions in terms of the decision variables. We first analyze the expecte μ, σ and T are given, also. Using standard calculus, the expected overtime can be rewritten as:

$$\begin{aligned}\upsilon(\mu,\sigma^2,T) &= \int_T^\infty (t-T)\cdot\frac{1}{\sqrt{2\pi\sigma^2}}\cdot e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2}dt \\ &= \int_T^\infty t\cdot\frac{1}{\sqrt{2\pi\sigma^2}}\cdot e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2}dt - T\cdot\int_T^\infty \frac{1}{\sqrt{2\pi\sigma^2}}\cdot e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2}dt.\end{aligned}$$

Using a change of variable for the first term, we have:

$$\begin{aligned}\upsilon(\mu,\sigma^2,T) &= \int_{T-\mu}^\infty (t+\mu)\cdot\frac{1}{\sqrt{2\pi\sigma^2}}\cdot e^{-\frac{t^2}{2\sigma^2}}dt - T\cdot\left(1-\Phi\left(\frac{T-\mu}{\sigma}\right)\right) \\ &= \int_{T-\mu}^\infty t\cdot\frac{1}{\sqrt{2\pi\sigma^2}}\cdot e^{-\frac{t^2}{2\sigma^2}}dt + \mu\int_{T-\mu}^\infty \frac{1}{\sqrt{2\pi\sigma^2}}\cdot e^{-\frac{t^2}{2\sigma^2}}dt \\ &\quad -T\left(1-\Phi\left(\frac{T-\mu}{\sigma}\right)\right) \\ &= \left[-\frac{\sigma^2}{\sqrt{2\pi\sigma^2}}\cdot e^{-\frac{t^2}{2\sigma^2}}\right]_{t=T-\mu}^\infty + \mu\cdot\left(1-\Phi\left(\frac{T-\mu}{\sigma}\right)\right) \\ &\quad -T\cdot\left(1-\Phi\left(\frac{T-\mu}{\sigma}\right)\right) \\ &= \frac{\sigma}{\sqrt{2\pi}}\cdot e^{-\frac{(T-\mu)^2}{2\sigma^2}} + (\mu-T)\cdot\left(1-\Phi\left(\frac{T-\mu}{\sigma}\right)\right),\end{aligned}$$

where $\Phi(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\cdot e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$ is the cumulative distribution function of the normal distribution. The first expression can also be numerically determined using a Riemann sum, which might be practically preferable.

Linearization assumption. The function $\upsilon(\mu,\sigma^2,T)$ represents the expected overtime, which is part of the objective function and constraint (5.1) in the MILP. However, the function is not linear, while that is a requirement for all objective functions and constraints in an MILP. To solve this issue, a common method is to replace a non-linear function by tangent lines that approximate the function, as for example done in [104] within an operating room scheduling context. Adding more tangent lines will increase the approximation of the function, but also the computation time.

The extra complication for this model is that the overtime function depends on multiple variables. It depends on the sum of expected times of the scheduled procedures, sum of variances of the scheduled procedure times and the opening time of the room. Within our case study, the opening times of the rooms are fixed, meaning that τ_{dr} is not a (decision) variable and may be given by a constant T, but the function would still depend on multiple variables.

A solution would be to not consider tangent lines, but tangent areas to approximate the overtime function. However, this would increase the models' complexity and computation time enormously. Alternatively, the overtime function v can be simplified or approximated such that it only depends on one variable. To achieve this, we could assume that the ratio between the variance and the expectation of a procedure team of every category (on a room) is similar. In other words, the assumption is that the variance of a procedure time increases almost linear with the expected procedure time. More formally, this means that for all rooms $r = 1, \ldots, R$, it must be that $Var(P_{c_1})/\mathbb{E}(P_{c_1}) \approx Var(P_{c_2})/\mathbb{E}(P_{c_2})$ for every $c_1, c_2 \in \rho_r$. In other words, $\sigma^2_{dr} = Q_r \cdot \mu_{dr}$ for $d \in [D], r \in [R]$. The mentioned assumption allows the linearization function to be expressed as $v(\mu)$.

This assumption that a higher μ implies a higher σ is not without question though. The underlying idea is that the longer a procedure takes on average, the more actions are required that each have their own variance, increasing the total variance. However, other research considering benchmarks in health care scheduling [76] shows that this is not necessarily the case. In some cases, longer procedures are in fact easier to predict.

Within this research, the assumption will be made in this chapter as the idea that procedures with a higher expected procedure time also have a higher variance is confirmed by by the hospital where our case study is held (see Section 5.4). Additionally, the intention was to increase the complexity of the model further to reduce the computation time.

It is however, not required make this assumption to make the proposed method still work by the use of tangent areas, but the complexity (number of constraints) would increase significantly in size. We leave this option as future work, to find ways to formulate tangent areas that approximate the overtime function well. That would be an extension of the method we propose in the following paragraph.

Tangent lines Consider a room r on day d, and denote μ_{dr} as the expected procedure time and σ^2_{dr} as its corresponding variance. Due to the earlier mentioned reason, we can express σ^2_{dr} as a linear function of μ_{dr}, i.e., $\sigma^2_{dr} = Q_r \cdot \mu_{dr}$, where Q_r is the (weighted) average ratio $Var(P_c)/\mathbb{E}(P_c)$ for every category $c \in \rho_r$. Under this assumption, v_{dr} is a function that only depends on μ_{dr}, which is linear in the decision variables.

To incorporate $v_{dr}(\mu_{dr})$ in our ILP, we adopt a piecewise linear approximation using tangent lines of the function $v_{dr}(\mu_{dr})$ (which is now only a function of μ_{dr}). Let $N_{dr} \in \mathbb{N}$ be the number of tangent lines that touch $v_{dr}(\mu_{dr})$. Then, every tangent line $n \in [N_{dr}]$ must be of the form:

$$y_n(\mu_{dr}) = \alpha_{drn} \cdot \mu_{dr} + \beta_{drn},$$

where the constants α_{drn} and β_{drn} for day d, room r and the nth tangent line should be chosen such that

$$v_{dr}(\mu_{dr}) = y_n(\mu_{dr}) \quad \text{and} \quad v'_{dr}(\mu_{dr}) = \alpha_{drn}.$$

The use of these tangent lines is crucial for the model to become linear again. For the ILP, the constants α_{drn} and β_{drn} should be determined first. Then Equation (2)

should be replaced by

$$\upsilon_{dr} \geq \alpha_{drn}\mu_{dr} + \beta_{drn} \quad n \in [N_{dr}]; d \in [D]; r \in [R],$$

where μ_{dr} is given by Equation (10). Clearly, if N_{dr} increases, the accuracy increases, but also the computation time. The same can be done for overloads at wards, where μ_{dr} and σ_{dr} should be replaced by μ_{dw} and σ_{dw} given above.

An illustration of this procedure is given in Figure 5.1. The blue line represents the expected overtime υ for given planned operating time μ. However, because this line is not linear, it cannot be processed using standard ILP techniques. Instead, this overtime is replaced and approximated by a fixed number of lines, that can be taken into account in an ILP. The more tangent lines are used, the better the approximation.

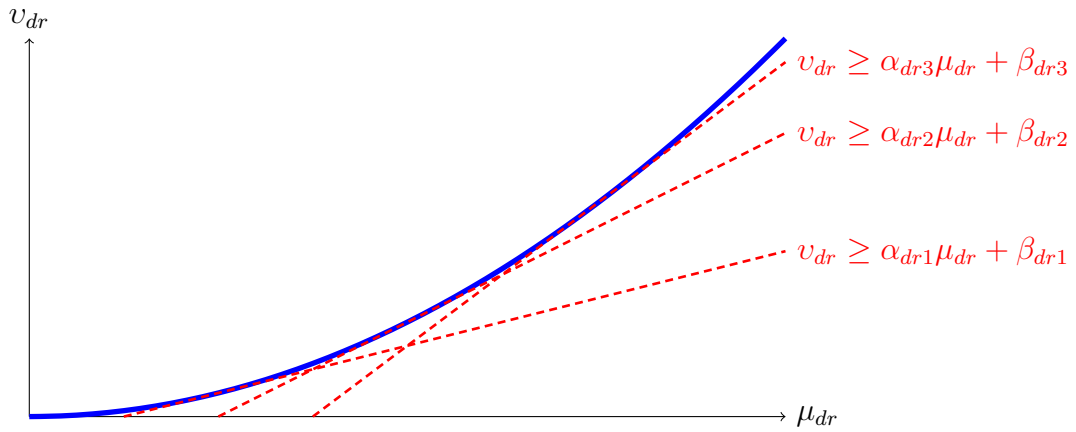

Figure 5.1: Illustration of the linearization method

5.4 Case study: VU University Medical Center

The research presented in this chapter is motivated by the scheduling difficulties faced at the Department of Cardiology of the VU University medical center (VUmc) [117]. VUmc had three cath labs at its disposal where different types of cardiological procedures can be performed (heart catheterizations, implant placements, etc.). Prior to such procedures, patients require a specific preparation (which generally takes hours to days) at a suitable ward, where usually also the required aftercare is provided for the patients. Due to specialized equipments and personnel, not every patient can be hospitalized at every ward or be treated at every cath lab. The subsets of suitable wards and cath labs per patient depend not only on the patient's procedure, but also on the urgency of the patient, since specific wards are primarily intended for urgent or elective patients.

So far, patient scheduling has been done entirely manually, which has regularly led to last-minute or ad hoc improvisations in order to accommodate incoming patients properly. Even though no severe accidents have resulted from manual scheduling, there has been a strong conjecture at VUmc that improvement could be

obtained with patient scheduling, particularly because the scarce number of beds at the five wards are poorly taken into account during the scheduling process. Ideally, the utilization at each ward is as stable as possible to balance the workloads, but arguably more important, to make sure that there is room available for (semi-)urgent patients as often as possible.

5.4.1 Input data

The available wards and cath labs can be found in Tables 5.1 and 5.2.

Ward	# Beds	Urgencies	Procedures
5B	14	Elective, semi-urgent, urgent	CAG/PCI's, Implants
5C	4	Elective, semi-urgent	EFO/ablations, Implants
CCU	6	Urgent, semi-urgent	CAG/PCI's, Implants
EHH	6	Urgent, semi-urgent	CAG/PCI's, Implants
SCAR	4	Elective, semi-urgent	CAG/PCI's

Table 5.1: Wards cardiology VUmc

Room	Starting time	Ending time	Procedures
1	08:15	16:30	EFO/ablations, Implants
2	09:30	16:30	CAG/PCI's, Implants
3	08:15	16:30	CAG/PCI's

Table 5.2: Cath labs cardiology VUmc

To admit patients at the cath lab, there is also a bed required at a ward, requiring synchronization during the scheduling process. In this section, we give an impression of patient flow at the wards due to patients at the cath lab. It is important to plan patients at wards such that the load is stabilized, thereby increasing the buffer for accepting (semi-urgent or urgent) patients and to balance the workloads of nurses at the wards.

To plan this appropriately, the scheduler should have knowledge regarding the distribution of the hospitalization times of patients. To provide an intuition for this duration, the data analysis of the hospitalization times of all patients combined is given in Figure 5.2.

This multi-modal pattern is due to two reasons. First of all, it is very unusual that patients are transferred or sent home from the hospital during the evening at night. Patients are generally received during the morning, meaning patients are unlikely to have a hospitalization time between 16 or 20 hours. Also, most of the patients visit the hospital for a CAG/PCI-procedure, for which the total hospitalization time lasts between 4 and 10 hours. Moreover, there are groups of patients that remain slightly over 1 (24h) or 2 (48h) days. For the model, the same has analysis has been done for every patient group category $c \in [C]$, which is used as a sampled distribution for H_c.

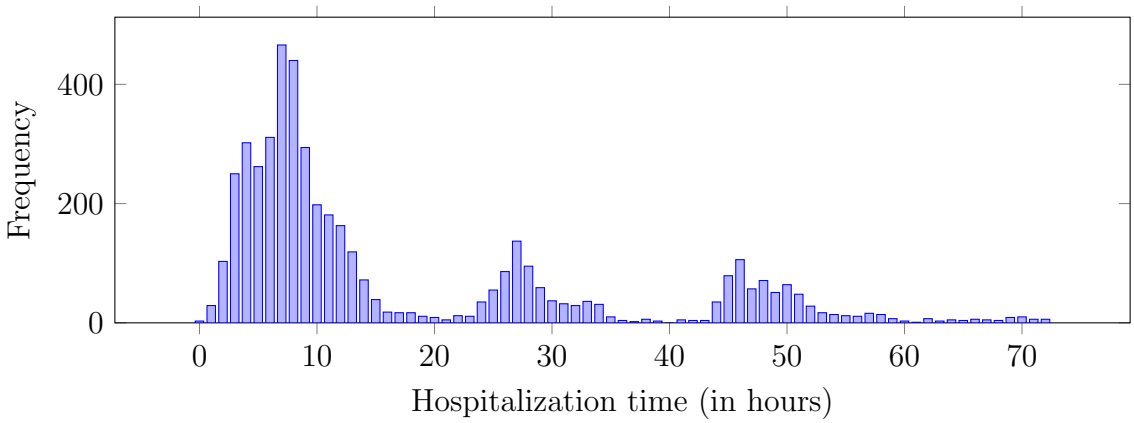

Figure 5.2: Distribution of hospitalization times

When creating a schedule of patients, one naturally can only work with information that is known prior to the procedure. The inconvenience of the acquired dataset for this research is that only the information after the procedure is known. For example, a specific procedure may be planned, but during the procedure, this may turn out to become a significantly other procedure, which last two to three times as long. Hence, we cannot derive by our data what the original diagnosis was. For this reason, we will work only with categories for which we know that the patient belongs to this category. We have chosen to only work with categories for which enough procedures/measurements in the data could be found, i.e., at least 40 procedures per year.

This has led to the following input data regarding the procedures in Table 5.3. The procedure times are in **hours**, while the hospitalization times are expressed in **days**.

Category	#	μ_v	σ_v	μ_p	σ_p	μ_n	σ_n
CAG/PCI	18	0	0	1.33	0.5	0.2	0.1
CAG/PCI 5B	1	0	0	1.5	0.6	0.8	0.5
SwanGanz5C	1	1	0.5	1.25	0.6	0.5	0.6
Implant	5	*	*	2	0.75	0.7	0.7
Short Ablation	2	*	*	2.5	0.9	0.75	0.4
Long Ablation	3	*	*	3.5	0.8	0.5	0.6
Semi-urgent PCI on ward 5B	5	2	2.3	1.25	0.5	1.1	1.1
Semi-urgent PCI on ward CCU	5	0.7	1.2	1.2	0.3	0.8	1.8
Semi-urgent PCI on ward EHH	1	0.5	0.4	1.05	0.5	0.2	0.2
Semi-urgent PCI on ward SCAR	5	0.1	0.1	1.2	0.5	0.2	0.1
Semi-urgent implant	1	0.5	0.4	1.5	0.6	2.6	4.4
Urgent PCI CCU	4	0.1	0.2	1.0	0.4	0.6	1.1
Urgent PCI EHH	2	0.1	0.2	1.0	0.5	0.2	0.2

Table 5.3: Average number of procedures per week

For simplicity, the numbers in Table 5.3 have been rounded or corrected where

required. Whenever a star is mentioned at the preparation time, the hospitalization time depends on the time of the day where the procedure takes place. If a patient for an implant is scheduled in the morning on a cath lab, he or she will be hospitalized the day prior to the procedure at a ward. When the patient is scheduled in the afternoon, there is enough preparation time for the patient to be hospitalized in the morning of the same day.

As a minor remark, prior to every procedure, the room needs to be prepared and the preceding patient needs to depart from the room, which in total takes approximately 15 minutes. For this reason in our implementation, approximately 15 minutes have been added to every procedure.

Choice of weights. In addition to the numbers in Table 5.3, the following weights have been chosen to complete the required input. Of course, this is very subjective and may have affect results significantly. In dialogue with the VU University Medical Center, the penalties have been set on $V_{\tau,r} = 3, V_{\upsilon 1,r} = 5, V_{\upsilon 2,r} = 9, V_{\beta,w} = 0.25, V_{\omega 1,w} = 3$ and $V_{\omega 2,w} = 9$. The rewards per procedure, R_c, have been set equal to 5 for CAG/PCI and SwanGanz procedures, 10 for implants and 20 for ablations.

5.4.2 Results

We illustrate the quality of the solution of our case study by depicting the plan that is derived on an average day, as these are more clarifying than objective values. In other words, the model outputs a list of instructions in an average week. This provides fundamental insights to the scheduler that can also be used in irregular weeks. The next three figures illustrate the order and moment at which the procedures from Table 5.3 need to be planned.

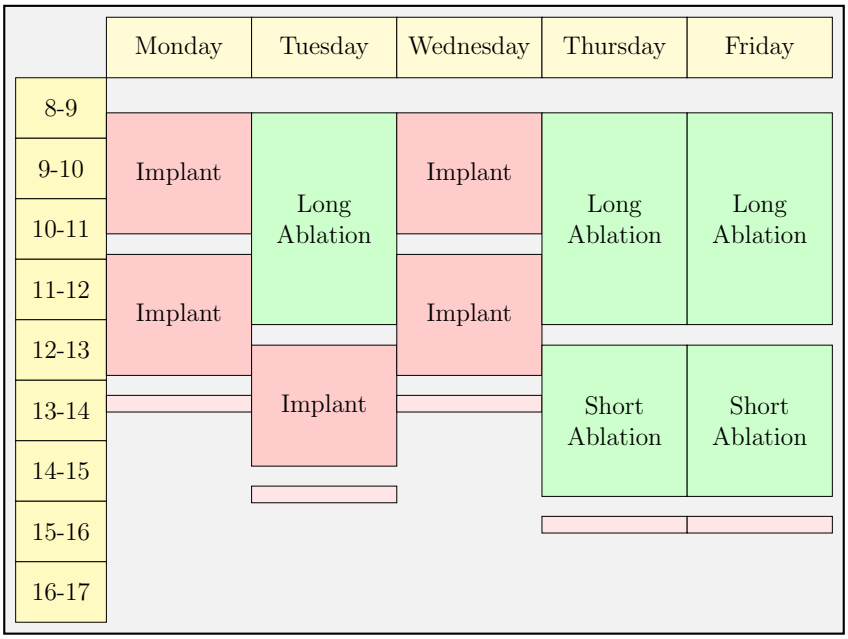

Figure 5.3: Blueprint room 1

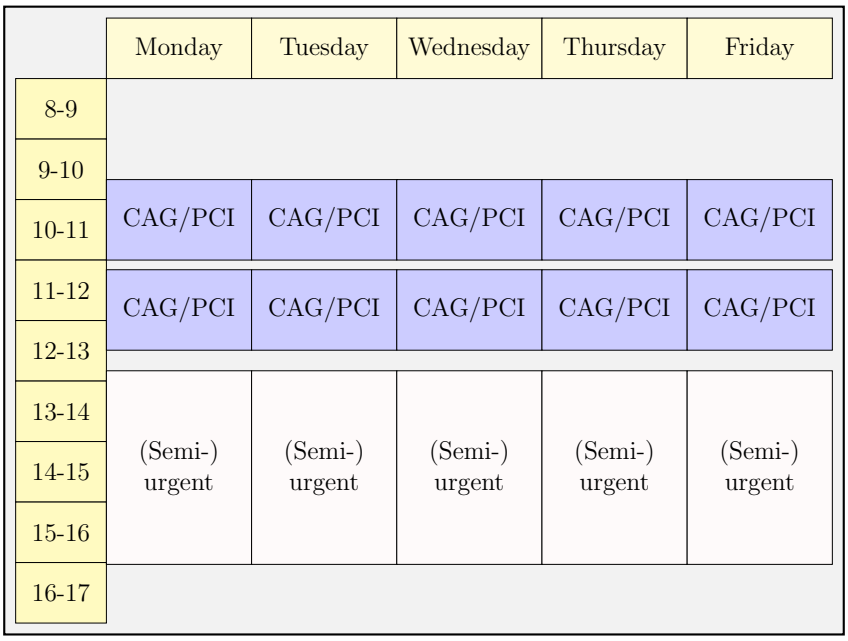

Figure 5.4: Blueprint room 2

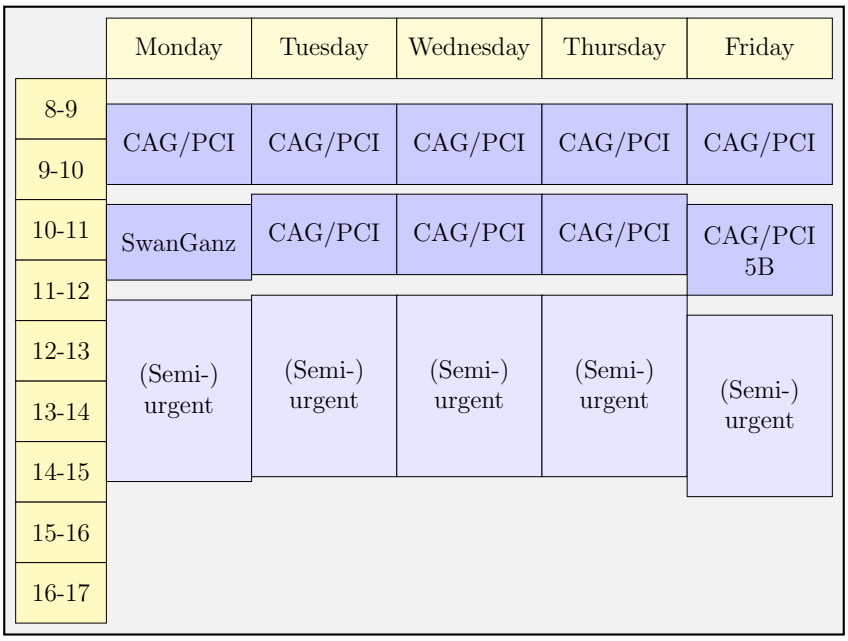

Figure 5.5: Blueprint room 3

As a clarification regarding the blueprint of room 1: the small, red thin blocks represent the average duration per day that the room is busy with a (semi-)urgent patient for an implant. Depending on the urgency, it is sensible to treat these at Monday or Wednesday because these days contain more free space. The blueprint for room 2 only consists of two PCI's per day and requires no explanation.

Regarding the blueprint of room 3, it must be noticed that the Swan Ganz procedure and Dotter5B (a PCI with a much longer aftercare) on respectively Monday

and Friday are scheduled. After all, a Swan Ganz procedure generally has a much longer procedure time, while a PCI-procedure often results in an extra day aftercare at ward 5B. This stabilizes the bed occupancy on 5B and 5C as beds are more often used in the weekend, creating more space throughout the week.

A rule of thumb is therefore clearly to plan the procedures with the longest preparation on the Monday (possibly on Tuesday, since the Monday is usually not crowded), and that procedures with the longest aftercare are to be scheduled at Friday. Note that the occupation at 5B on Monday according to this blueprint can be equal to 4; 2 implants and a Swan Ganz procedures are treated on Monday, while a patient with a long ablation on Tuesday also needs to be hospitalized that Monday.

Occupancy rate 5C. Using these blueprints, we consider the occupancy rate of ward 5C. This ward has shown in practice to have the most fluctuations. By simulating the working process on the ward using a simulation model, a comparison can be made between utilization using the current scheduling methods, and the utilization using the blueprints following from the model, as visualized in Figure 5.6.

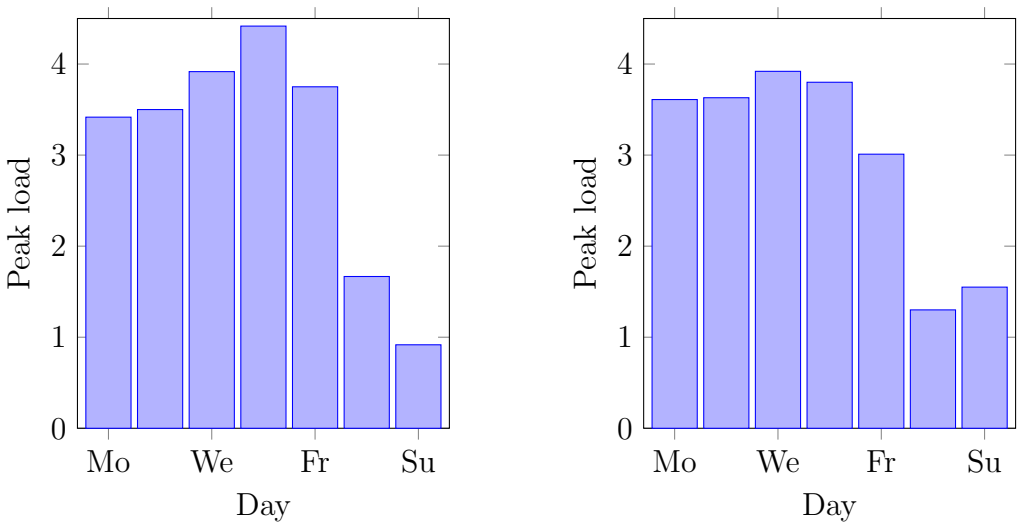

Figure 5.6: Average peak occupation 5C current (left) vs. optimal (right)

For ward 5C, a notable improvement can be seen as the peak on the Thursday has been reduced, and a much more stable pattern throughout the whole week can be seen. This pattern is mainly due to the fact that procedures with a long preparation time have been scheduled on the Monday (such that the Sunday is used optimally).

Running time The model has been solved using the commercial optimization software CPLEX. For the running time we remark that it depends on the number of tangent lines used in the ILP (see end of Section 5.3.2). In this research, three tangent lines per constraint have been chosen, which empirically have shown a good enough accuracy for our research. As a result, the running time of the model was in the order of hours.

5.4.3 Scenarios

Experiments have been done with the use of this mathematical model by developing an extensive simulation model that replicates the planning process of VUmc as good as possible. With such a simulation model, the performance measures of possible scenario's can be analyzed. Using the blueprints that follow from the model, the current situation is once more simulated, after which a variety of other scenarios are simulated. Within this research, a scenario is referred to as a specific change in structure or size regarding the logistic process. A different distribution of the beds among the ward is therefore an example of such a scenario, if one wants to investigate whether this can reduce the overload at a ward.

These simulations were requested by the VU University Medical Center, as specific changes are very realistic to happen such as a significant increase in patients in a specific category, as new hospitals were planning to redirect their patients to VUmc. An overview of these scenarios are explained below, after which the corresponding results can be found at the end of this section.

Current situation Although simulations are meant to research changes in the process, it is of vital importance to simulate the current situation to gain a fair zero measurement. In this way, the simulation model can be validated to see whether it actually simulates the correct process and to see whether the results are in correspondence with the practice (i.e., the data analysis). In other words, by simulation the current situation, one gains insight in how close the model lies to reality and a fair comparison with simulating the scenario's.

In case the utilization rate of a specific cath lab lies a few percent below the percentage in the data analysis, one should take at the interpretation of the simulation results (of other scenarios) into account that the simulated utilization rate also should be a few percentages lower.

Growth scenario A very realistic scenario for VUmc is the scenario where the number of patients increases significantly, because VUmc aims to take over the cardiological group of patients of a different hospital. In expectation, this leads to an increase of 240 CAG/PCI's (1 per day), 40 implants and 80 EFO/ablations per year. To facilitate this growth, the idea is to increase the number of beds from the Special Care from 4 to 7 beds, while department 5C gains one extra bed.

Different starting times cath labs The growth scenario is very likely to force extra capacity on the wards, but in case there is also too little capacity on the cath labs, the hospital considers to change the starting time of room 2 from 09:30 to 08:15. This scenario is abbreviated as "ExtR2" in the upcoming results.

Four days EFO/ablations Currently, EFO/ablations are performed on only three days (Wednesday, Thursday and Friday) due to the availability of the cardiologists. However, when the number of EFO/ablations increases with 80 per year (2 per week), the cardiologists may consider to extend their availability to four days.

The results for the simulation of specific scenarios are given in Table 5.4. For completeness, a simulation has been performed of one run, representing 100 years of the exact same setting.

	Data analysis	Simulation current	Growth	Growth +ExtR2	Growth +ExtR2 +4DayEFO
Utilization					
Room 1	66.4%	64.5%	75.8%	73.7%	74.1%
Room 2	70.3%	69.3%	76.9%	74.5%	75.3%
Room 3	73.9%	70.5%	78.5%	76.7%	77.0%
Undertime					
Room 1	69.1 m	86.4 m	38.2 m	45.0 m	46.5 m
Room 2	43.9 m	58.4 m	39.1 m	50.7 m	46.8 m
Room 3	46.9 m	50.6 m	43.5 m	51.8 m	50.1 m
Overtime					
Room 1	9.9 m	11.2 m	21.2 m	18.2 m	14.8 m
Room 2	18.5 m	19.1 m	32.6 m	17.3 m	16.7 m
Room 3	30.8 m	29.0 m	38.5 m	22.4 m	22.2 m

Table 5.4: Overview simulation results of different scenarios

This table contains a large variety of information, but it is important to verify the simulation model first (i.e., to see whether the simulation model is close to the realistic process), which can be seen in the first two columns. The simulation model produces utilizations (of the cath labs) that lie approximately two percent lower than in practice (i.e., the utilization that follows from the data analysis). As a rough estimation to correct the results, one could simply add two percent to the utilization rate to fix the gap between the simulation model and the realistic process. These extra two percents clearly have not been included in any of the last four columns for consistency, but will be referred to in the remainder of this section.

An explanation for this difference is that the schedulers of the hospitals most likely can solve unexpected difficulties in a more efficient way. The overtime from the model is very close to the results from the data analysis, but the undertime shows a significant, but not extreme, difference of approximately 5 to 20 minutes.

With this information, one has a better understanding on how to interpret the results of these three scenarios. Clearly, the utilization rate will increase due to the increase in patients (in all scenarios). In this case study, an increase of the utilization of 10% is very realistic to happen for room 1 using this model, while room 2 and 3 are expected to have an extra 5 to 10% increase. For the scenario with all three possible changes (growth + extension room 2 + 4 days EFO), an average utilization of respectively 79%, 77% and 77% is obtainable (after the simulation correction of 2%). For the department of cardiology, this would a very positive development, as more efficient usage of their rooms is of high importance.

Finally, the scenario is considered in which cardiologists that can perform EFO and ablations should be available four days per week (instead of three). Minor improvements are visible (an utilization increase of about 0.5%), but it is questionable

whether this improvement is significant. For the cardiologists at room 1, overtime decreases with 6 minutes on average per day, which is considerable, but the cardiologists are free to choose whether this is worth for them.

5.5 Conclusions and future work

In this chapter, we proposed a technique that integrates two optimization problems: the scheduling of patients at the cath labs, and the logistics of patients at the wards where patients need preparative and aftercare. By linearizing the overshoot at cath labs and wards, we can formulate an Integer Linear Program that outputs blueprints within a reasonable amount of time, that can guide planners to where and when specific procedures should be scheduled ideally. We also note that linearization of overtime is usable in practice since the accuracy can be increased as long as the computation time permits.

However, the assumption that is needed to linearize the overtime function remains questionable. It is assumed that whenever the expected procedure time increases, the variance increases similarly, i.e., $Var(P_{c_1})/\mathbb{E}(P_{c_1}) \approx Var(P_{c_2})/\mathbb{E}(P_{c_2})$ for all $c_1, c_2 \in \rho_r$ for any $r \in [R]$. This allowed a simplification of the overshoot function υ, such that it depends on only one variable, which allows an easier linearization. Even though this assumption has been discussed and agreed upon with the cardiologists in our case study, more recent research has shown this is not necessarily the case. Procedures that have a longer average procedure time, may in some cases have a smaller variance, as they are more predictable. This may lead to incorrect results of the model.

For this reason, finding an alternative to this linearization assumption is the main suggestion for future research. It is possible to linearize a function that consists of two or multiple variables, by using tangent areas rather than tangent lines. Within this research, an attempt has been done to incorporate such tangent areas within the model, but initial attempts have made the model too time-consuming. It would be interesting and relevant to find a way to incorporate tangent areas without increasing the computation time not too much, in order to drop the questionable linearization assumption. Alternatively, an interesting question is whether there is another way to let (the approximation of) the overtime function depend on one variable only, without making assumptions on the relationship between the mean and variance of the procedure times.

Chapter 6

Train Timetable Generation

This chapter considers the Periodic Event Scheduling Problem (PESP), where the goal is to schedule a given set of events to times within a cyclic framework, subject to constraints that impose upper or lower bounds on the time difference of pairs of events. Even though determining feasibility is already hard, a supplementary weighted slack objective function is added, which in practical settings can be interpreted as minimizing the waiting times of passengers.

After discussing the background of train timetable generation in Section 6.1, the problem under consideration will be defined in Section 6.2. Several simple, but useful state and search space reduction techniques are discussed in Section 6.3. By solving an easier special case of the problem, explained in Section 6.4, the used tree decomposition heuristic will be elaborated upon in Section 6.5. This approach is tested on online benchmarks, for which its performance is reported in Section 6.6 and concluded in Section 6.7.

The results in this chapter are based on [116] and require basic knowledge on combinatorial optimization.

6.1 Background

In many countries with an advanced transport network, the planning process of railway transport providers is an extremely complicated and time-consuming procedure. Most railway transport providers apply a hierarchically structured planning process that consists of multiple stages, where the focus of timetabling starts macroscopic towards microscopic feasibility. From a high-level point of view, the planning process for train networks can be divided into the following tasks [11]:

1. Network planning: constructing the infrastructure of the railway network, such as tracks, usually based on technical constraints as well as on an economical analysis.

2. Line planning: determining the routes (and frequencies) of trains within the railway network.

3. Train timetable generation: determining the arrival and departure times of trains, including their routes through the infrastructure/stations.

4. Rolling stock and personnel planning: assigning the available rolling stock and personnel to the trips.

5. Real time traffic: ensuring the realization of the planning by solving irregularities (e.g., delays) on an operational level.

The focus of this chapter lies on the third step within this hierarchy, the design of train timetables, although the proposed algorithms are not restricted to this setting. However, routing through the infrastructure lies beyond the scope of this chapter as this complicates the problem significantly. In practice, a separate routing algorithm is used that can verify whether the generated timetable is also feasible with respect to its railway network within stations.

6.2 The Periodic Event Scheduling Problem

6.2.1 Motivation

Constructing a high quality railway timetable is a complex and time consuming task, since typically all lines in the entire traffic network are connected. In other words, the departure and arrival times of all trains directly or indirectly depend on each other. Due to the additional numerous constraints that are involved in a timetable, it is undesirable and/or practically impossible to construct an optimal timetable manually, which motivates the research for automated timetable generation.

It is preferable to have a cyclic timetable, i.e., a timetable where every event occurs exactly once every period of time. The length of such a cycle is typically set to one hour. This also makes the generation of a timetable more tractable, as the timetable needs to be designed for one hour only, rather than an entire day. Moreover, this makes it for travelers easier to recall the departure time of the required train in case they need to traverse the same route on a different time. The hourly pattern may have some exceptions where more trains could be required to fulfill the demand (e.g., during rush hours), but this simply can be solved creating a separate timetable, where the second timetable is a subset or variant of the first.

Usually, the constraints of a railway timetable can be defined as putting a time difference between two events. For example, two trains should depart at least a minimum amount of time after each other to prevent possible overlap. To incorporate these constraints and preferences within a cyclic framework, the Periodic Event Scheduling Problem (PESP) is widely used within railway timetabling. This model is initially proposed by [107] and will be defined in Section 6.2.2. For a more extensive motivation for the usage of the PESP the reader is referred to [81], where also is argued how this model can be used in the entire planning process.

6.2.2 Definition

The PESP aims to schedule a number of events within a cyclic framework of length T. In a railway timetabling context, examples of such events can be the departure, pass-through or arrival of a train at a specific station.

Let n be the number of events that need to be scheduled. Furthermore, introduce a set of decision variables V, where $v_i \in [0, T)$ is the time at which event i takes place for all $i \in [n]$. A constraint in the PESP considers a pair of events, $(i, j) \in [n] \times [n]$, and applies a time window using a lower and upper bound, $L_{ij}, U_{ij} \in \mathbb{Z}$, on the scheduled time difference of this pair of events, i.e., $v_j - v_i$. Let $A \subseteq [n] \times [n]$ be the pairs of events for which a constraint exists. This means that for $(i, j) \in A$, there exists a time window $[L_{ij}, U_{ij}]$ for which the constraint:

$$(v_j - v_i) \bmod T \in [L_{ij}, U_{ij}], \tag{6.1}$$

must hold. For example, if events i and j represent the departure of two different trains from the same track, safety regulations could require the trains to depart at least 3 minutes after each other. In this case, $L_{ij} = 3$ and $U_{ij} = 57$ to prevent trains (from possibly different cycles) to coincide, assuming $T = 60$.

A PESP instance can be transformed and visualized in a directed graph $D = (V, A)$, where $n = |V|$ is the number of vertices/variables, and $m = |A|$ is the number of arcs/constraints. For every constraint $(i, j) \in A$ for which a constraint as in Equation 6.1 exists, an arc $i \to j$ is introduced and labeled with $[L_{ij}, U_{ij}]$. For convenience, vertices and variables are used as synonyms throughout this chapter. The same is done for constraints and arcs. See Figure 6.1 for a simple example with only three constraints.

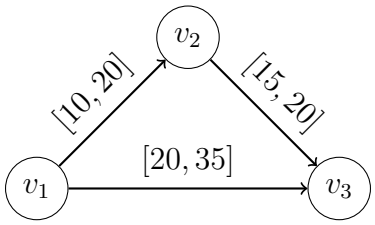

Figure 6.1: Example of a PESP instance visualized in a graph ($T = 60$)

As a notational remark: $x \bmod T$ is abbreviated to $(x)_T$. Here, x can be a number, but also an interval that will be scaled within the interval $[0, T)$. Since the graph formulation is slightly preferred in the literature, this chapter adopts the same notation, which allows the problem to be formally defined as follows.

Periodic Event Scheduling Problem (PESP)	
Given:	A directed graph $D = (V, A)$, a time window $[L_{ij}, U_{ij}]$ for every $(i, j) \in A$ with $L_{ij}, U_{ij} \in \mathbb{R}$ and a cycle time T.
Goal:	Find a $v \in [0, T)^n$ such that $(v_j - v_i)_T \in [L_{ij}, U_{ij}]$ for every $(i, j) \in A$, or state infeasibility.

Trivially, it is assumed that $L_{ij} \leq U_{ij}$ as the instance is infeasible otherwise, and that $U_{ij} - L_{ij} < T$, since the constraint would be redundant otherwise. Moreover, all L_{ij} and U_{ij} are assumed to be integer, which is practically justified because timetables are usually published in minutes (integers). Using this assumption, it has been proven in [93] that every feasible PESP-instance then has an integer solution.

Note that by the cyclicity of PESP, the orientation of the arcs can be reversed by "mirroring" the corresponding interval with $T/2$ as the center, i.e., constraints of the type in Equation 6.1 is equivalent to

$$(v_j - v_i)_T \in [T - U_{ij}, T - L_{ij}]\,. \tag{6.2}$$

Handling the modulo operator. Even though the modulo operator follows naturally from the cyclicity of the model, most standard mathematical optimization techniques (such as Branch and Bound) are unable to handle this operator. For this reason, constraints of the type as in Equation 6.1 are alternatively in the literature formulated as:

$$L_{ij} \leq v_j - v_i + T \cdot p_{ij} \leq U_{ij}, \tag{6.3}$$

at the cost of one extra integer variable p_{ij} per constraint (in similar other models, p_{ij} can also be a binary variable). Here, $p_{ij} \in \mathbb{Z}$ indicates the cycle difference between i and j. In these constraints, p_{ij} is also referred to as the modulo parameter of the constraint. The model now has become suitable for Mixed Integer Linear Programming (MILP) methods.

A single constraint as in (6.1) defines a convex time window on the time difference between two events, i.e., if $x, y \in [L_{ij}, U_{ij}]$ then $\alpha x + (1-\alpha)y \in [L_{ij}, U_{ij}]$ for any $\alpha \in [0, 1]$. However, using this integer variable p_{ij}, one can implicitly define non-convex intervals, even though the interval $[L_{ij}, U_{ij}]$ for every constraint is convex. This follows from the possibility in the model to allow multiple constraints between a pair of events, and because L_{ij} and U_{ij} do not necessarily need to be in $[0, T)$. For instance, if $T = 60$, the two constraints:

$$(v_j - v_i)_T \in [0, 45] \text{ and } (v_j - v_i)_T \in [30, 72]$$

result in a feasible difference interval between v_i and v_j of $[0, 12] \cup [30, 45]$, by the cyclicity of the model.

Remark 6.1. *In the graph setting, this implicitly means that there must be multiple edges $(i, j) \in A$ with different, which may be notationally inconvenient and confusing. To overcome this issue, it is possible to introduce a dummy variable k and set $v_j = v_k$ and add an edge (i, k) instead to mimic (i, j).*

Cost optimization. Although the PESP is originally formulated as a feasibility problem, an objective function can be added without complications. One of the easiest, but also practically most useful, objective functions can be deduced from the constraints. In many cases, for a constraint $(v_j - v_i)_T \in [L_{ij}, U_{ij}]$, the optimal value $(v_j - v_i)_T$ is the minimum L_{ij} from a time-efficiency perspective.

For example, if arc $a = (i, j)$ corresponds to the constraint that the changeover time between two trains (that correspond to variables i and j) should lie in $[L_{ij}, U_{ij}]$, the waiting time is minimized if $v_j - v_i = L_{ij}$. If w_{ij} denotes the cost of every time unit that all travellers need to wait longer at the changeover corresponding to the constraint, the term:

$$z_{ij}(v_i, v_j) = w_{ij}\left((v_j - v_i)_T - L_{ij}\right) \tag{6.4}$$

can be used to quantify the costs. The objective function or total costs, referred to as the weighted slack function, can then be expressed as $z(v) = \sum_{(i,j)\in A} z_{ij}(v_i, v_j)$.

The focus in this chapter is on this weighted slack function. Other objective functions are discussed in [98] and [83], such as minimization of passenger travel time, required rolling stock, or the number of violated constraints (in case of an infeasible instance), while maximization functions include the profit or robustness.

6.2.3 Related work

This chapter focuses for a large part on heuristics, but will use efficient combinatorial optimization algorithms to solve subproblems if possible. One of the earlier and more influential solution methods in a railway timetabling context is found in [106], which will be briefly described further. Moreover, an overview of the Operations Research of railway timetabling can be found in [74], while an overview for the PESP in particular (including extensions) can be found in [81].

The PESP was originally formulated in [107], where also several algorithms were proposed. These are primarily searching methods where the modulo parameters are solved first. To this end, a minimum spanning tree is initially constructed, where the the number of possible values (cardinality of the time window) are used as weights on the arc. The idea is that a solution is found that satisfies the $n-1$ tightest (and therefore expected to be the hardest to fulfill) constraints beforehand, but similar techniques might lead to a brute-force algorithm at an early stage.

Exact methods. A large part of the methods in the current literature focuses on the PESP as a feasibility problem, rather than an optimization problem. One of the first solution methods in a railway timetabling context has been implemented by [106] by solving the Mixed Integer Linear Program (MILP) using constraints of the type as in Equation 6.3. With the aid of the commercial optimization software package CPLEX, solutions for practical railway timetabling instances can be found with the aid of searching algorithms and adjustable parameters within the software package. Other works that focus on solving the MILP can be found in [94], [95], [83] and [98], using cutting planes and similar other mathematical optimization techniques.

Heuristics. A few heuristics already exist that output only very few violated constraints for real-world instances, for example in [80], where cuts and/or local improvements are used to improve the original heuristic from [107]. Although the performance may be relatively good in practice, many of the currently known heuristics struggle with the task of restoring an infeasible solution, without using brute-force early.

The work presented in [80] is similar to the modulo simplex algorithm, firstly presented in [91], and improved by [47], by exploiting advanced methods in the modulo simplex tableau and larger classes of cuts to escape from local optima.

This method currently performs best on many benchmarks that are also used for this research. Still, more ways to backtrack a solution and escape local optima are searched for in the current literature. This work aspires to contribute to this concept from a different perspective.

6.2.4 Contributions

The scientific contributions presented in this chapter are the following. First, a special case of the PESP will be considered in Section 6.4 where the graph $D = (V, A)$ is cycle-free and every variable v_i for which $i \in V$ is potentially bound to a specific subset of $\{0, 1, \ldots, T-1\}$. A dynamic program will be introduced that solves this problem in $\mathcal{O}(nT^2)$ time, even when an objective function is considered.

This dynamic program is used as part of the contribution in Section 6.5, which consists of a tree decomposition heuristic that solves the PESP. Even though solving individual components of a problem may be easy, the difficulty lies in merging these components such that the problem as a whole is solved as well. The idea is that once the main problem is split into subproblems. Once a one subproblem is solved, this puts restrictions on other subproblems that need to be fulfilled to solve the main problem. A procedure to recognize and solve this issue will be propo. that decomposes a PESP problem into trees that are solved independently and merged into a feasible solution.

These techniques are primarily based on dynamic programming, which allows the usage of a smart objective function that heuristically maximizes the possibility that a solution for a component can be extended to a solution for all other components.

6.2.5 Complexity

For $T = 2$, the PESP can be solved in polynomial time, for which an algorithm is given in [98]. However, the PESP is strongly NP-complete for $T \geq 3$. At least three proofs are currently known, being reductions from the Linear Ordering Problem [82], the Hamiltonian Cycle Problem [90] and the Graph k-Colorability Problem [93]. A reduction using the last-mentioned problem is included below.

Graph k-Colorability Problem	
Given:	An undirected graph $G = (V, E)$ and an integer $k \leq \|V\|$..
Goal:	Determine whether there exists an assignment of k colors to vertices $c : [k] \to V$ such that for every edge $(i, j) \in E$, $c(i) \neq c(j)$.

Theorem 6.1. *PESP is strongly NP-complete.*

Proof. Given an instance $G' = (V', E)$ and k of the Graph k-Colorability Problem, construct the following instance of the PESP.

- $T = k$,
- $V = V'$,

- $A = E$ with arbitrary direction, and
- $[L_{ij}, U_{ij}] = [1, k-1]$, for all $(i,j) \in A$.

If the instance for the Graph k-Colorability Problem is a Yes-instance, then $c(i) \neq c(j)$ for all $(i,j) \in E$. Now in the corresponding PESP-instance, set $v_i = c(i) - 1$, because the minimum value for v_i is 0, while the minimum value for $c(i)$ is 1. Note then that therefore $(v_j - v_i)_T \in [1, k-1]$ since $v_i \neq v_j$, meaning that the instance for the PESP also is a Yes-instance.

Conversely, if the instance for the PESP is a Yes-instance, recall that it has been proven in [93] that there exists a feasible integer solution, since k is integer. Since the time interval is $[1, k-1]$, it also must mean that for every $(i,j) \in A$, the two corresponding events have different values assigned, i.e., $v_i \neq v_j$. This must mean that also different colors can be assigned in the Graph k-Colorability Problem and that it also is a Yes-instance.

Finally note that this reduction clearly can be done in polynomial time as all transformations can be done in linear time, and that PESP is in NP since verifying whether $(v_j - v_i)_T \in [L_{ij}, U_{ij}]$ for all $(i,j) \in A$ also can be done in linear time. □

Hence, no (pseudo)polynomial time algorithm can be found to solve the PESP, unless P = NP.

6.3 State- and search-space reduction

From a practical point of view, it may be computationally very beneficial to reduce the state- and search space without excluding feasible solutions. This usually can be achieved fairly simple indeed, especially within a railway timetabling context. In the following sections, several state- and search space reduction techniques are discussed, of which most are also (partially) noted in [83]. Even though most of these methods are straightforward, it is useful to mention these methods to provide an intuition for the complexity of the reduced problem.

6.3.1 Intersecting feasible intervals

As explained in Remark 6.1, multiple constraints between a pair of events i and j can be constructed to implicitly define a constraint with a non-convex feasible interval. This may require additional variables as mentioned in Remark 6.1.

When using MILP methods, it is essential that a single constraint induces a convex interval. However, the heuristics explained in this chapter are not MILP methods, and are not affected by whether these intervals are convex or not. This allows to combine all constraints between a specific pair of variables, into one constraint. To elaborate the possibilities, the following simple definition is introduced for notational convenience.

Definition 6.1. *The **feasible interval** Δ_{ij} between variables i and j are the values $v_j - v_i$ for which $(v_j - v_i)_T \in [L_{ij}, U_{ij}]$ for every $(i,j) \in A$.*

Initializing Δ_{ij} can simply be done as follows. For every constraint, scale the feasible interval $[L_{ij}, U_{ij}]$ within the cycle $[0, T)$ and call this new interval Δ_{ij}. For example, if $T = 60$, $[30, 75]$ will be scaled to $[0, 15] \cup [30, 59]$. Then, let $\Delta_{ij} = \cap_{(i,j)\in A}\Delta_a$. In Section 6.2 it was argued that the orientation of arcs can simply be redirected, which implies that at most $\frac{1}{2}n(n-1)$ constraints have to be considered.

6.3.2 Eliminating variables

There exist two straightforward ways to eliminate variables:

1. For every constraint (i, j, Δ_{ij}) where $|\Delta_{ij}| = 1$, either variable v_i or v_j does not have to be considered for optimization, as its value completely depends on the other variable. Let δ_{ij} be the only value in Δ_{ij}. Assuming v_j will be disregarded, all constraints of the type (j, k, Δ_{jk}) can be replaced by:

$$(i, k, (\Delta_{jk} + \delta_{ij}) \mod T)\,.$$

   A similar shift can be done for constraints of the type (k, j, Δ_{kj}). After solving the model without x_j, its value can easily be determined by $v_j = (v_i + \delta_{ij}) \mod T$.

2. If a variable v_i is contained in only one constraint (i, j, Δ_{ij}) and $|\Delta_{ij}| > 1$, the constraint always can be satisfied. After all, consider the problem without v_i. Once v_j is determined, one can afterwards choose $|\Delta_{ij}|$ different values for v_i such that the constraint is satisfied.

6.3.3 Propagating constraints

Constraint propagation refers to the method of tightening the feasible interval between variable i and j, Δ_{ij}, by combining a series of $\Delta_{ik}, \ldots, \Delta_{k'j}$, where $i \to k \to \ldots \to k' \to j$ is a path from i to j in the PESP graph. To see this, note that if $(v_k - v_i)_T$ must be in $[L_a, U_a]$ and if $(v_j - v_k)_T$ must be in $[L_b, U_b]_T$, that $(v_j - v_i)_T$ must be in $[L_a + L_b, U_a + U_b]_T$.

To describe the method informally, let $P_{ij} \subseteq A$ be the set of all possible paths from i to j. To reduce the feasible interval Δ_{ij}, consider all possible paths P between i and j. The indirect feasible interval between i and j according to path P and verify whether $\oplus_{a\in P}\Delta_a$ reduces the feasible interval Δ_{ij}. Indeed, the number of possible paths between i and j may be exponential, but a precise description on how to propagate constraints efficiently can be found in [83].

To illustrate, reconsider the example in Figure 6.1. There is one direct constraint which initializes Δ_{13} to $[20, 35]$. However, using constraints $(1, 2, [10, 20])$ and $(2, 3, [15, 20])$, it is easy to see this sequence induces a constraint between variable 1 and 3 with feasible interval $[10 + 15, 20 + 20] = [25, 40]$. The existence of this indirect feasible interval implies that the already existing interval $\Delta_{13} = 20, 35]$ can be reduced to $[20, 35] \cap [25, 40] = [25, 35]$.

6.4 The Restricted PESP

This section defines and analyzes a special case of the PESP, the so-called Restricted Periodic Event Scheduling Problem (RPESP), which provides the basis for heuristic methods for the PESP in this chapter. Even though these heuristics will be explained in detail in the next section, it is helpful to provide a motivation for the upcoming heuristics in a later section, in order to understand the intuition behind the problem considered in this section.

6.4.1 Motivation

The proposed algorithms in this chapter are based on the concept of decomposing a PESP instance into components that each contain a subset of the variables (and therefore also a subset of the constraints), which will be solved separately. Trees are large components, for which it will be shown that these can be efficiently solved, and even optimized. To clarify the concept, a few definitions will be introduced first.

Definition 6.2. *A PESP instance $C_x = (V_x, A_x)$ is a component of PESP instance $D = (V, A)$ if $V_x \subset V$ and $A_x = \{(i,j) \in A : i, j \in V_x\}$.*

A component can also be seen as a subproblem. It is important to see that whenever a PESP instance $D = (V, A)$ is decomposed into k disjoint subproblems $C_1, \ldots, C_k$ with $\cup_{x=1}^{k} V_x = V$, that A is not necessarily equal to $\cup_{x=1}^{k} A_x$. After all, constraints/arcs that connect two components in the original instance D are not included in $A_1, \ldots, A_k$. In fact, these connecting constraints will turn out to be the ones most difficult to fulfill.

Definition 6.3. *The bridging constraints B_{xy} between two disjoint components $C_x = (V_x, A_x)$ and $C_y = (V_y, A_y)$ with respect to $D = (V, A)$ are all constraints $(i,j) \in A$ for which $i \in V_x$ and $j \in V_y$.*

With this definition, note that $A = \left(\cup_{x=1}^{k} A_x\right) \cup \left(\cup_{x=1}^{k} \cup_{y=x+1}^{k} B_{xy}\right)$. In particular, given two components (or subproblems) C_x and C_y with respect to D, the combined subproblem is denoted by $C_{xy} = (V_x \cup V_y, A_x \cup A_y \cup B_{xy})$.

When two components are solved separately, it is likely that the combined solution does not correspond to a feasible solution with respect to D, because the bridging constraints cannot be satisfied. If so, one prefers to make as few adjustments as possible to the components, such that two solutions can be integrated. This idea provides the basis for the heuristics in this chapter, and also motivates the consideration of trees because of the following concept.

Suppose that the solution values of the variables in a component C_x are fixed, and one wants to integrate this component with another component $C_y = (V_y, A_y)$. The solution within C_x might induce several constraints on the values in C_y (the bridging constraints). Basically, these bridging constraints induce restrictions on the exact values of the variables in C_y, alongside the constraints that already were in C_y. See below for an example.

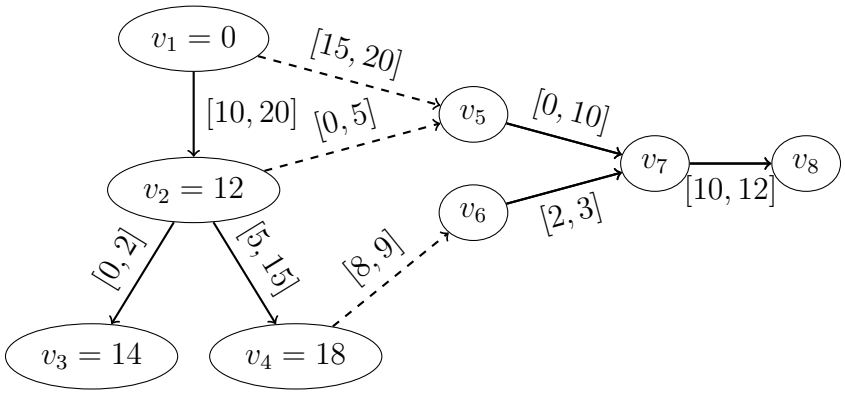

Figure 6.2: Example of restrictions while integrating components.

Example 6.1. The graph in Figure 6.2 contains eight variables and nine constraints. An already solved component C_x is the subgraph containing variables v_1 to v_4. The dashed lines correspond to the bridging constraints, which are not considered when the components are solved individually.

Based on these values, an algorithm needs to determine whether the fixed solution $(v_1, \ldots, v_4)$ for C_x can be extended to a feasible solution $(v_1, \ldots, v_8)$ for D. To do so, the algorithm needs to solve C_y based on the values $v_1, \ldots, v_4$ and the bridging constraints. In this case, one can easily see that the chosen solution for C_x limits the possible values for v_5 to $[15, 20] \cap [12, 17] = [15, 17]$ and for v_6 to $[26, 27]$. These constraints need to be taken as a starting point for solving C_y, in order to determine whether a solution for the entire problem can be found with the starting solution for C_x. Such constraints are referred to as exact variable restrictions X_i for variable v_i. This concept motivates the subproblem defined in the following subsection.

Finally, note that bridging arcs from C_x can always point to C_y since arcs can always be mirrored, as explained using Equation 6.2.

6.4.2 Problem description

Lemma 6.1. *A PESP instance for which the underlying graph $D = (V, A)$ is a tree can be solved in linear time.*

Proof. To see the correctness of this lemma, take an arbitrary vertex $i \in V$ and fix v_i with any value (e.g., $v_i = 0$). The possible values from the adjacent variables can be determined directly from the constraints corresponding to the arc. This procedure can be repeated for unfixed variables adjacent to fixed variables, until all variable values are fixed. □

As argued in the motivation, so-called variable restrictions will be added to the problem, meaning that every variable v_i might be bound to a specific set of values X_i. This notation allows the RPESP to become formulated as follows.

Restricted Periodic Event Scheduling Problem (RPESP)	
Given:	A directed, cycle-free graph $D = (V, A)$, a cycle time T, a feasible interval $\Delta_{ij} \subseteq \{0, \dots, T-1\}$ for all $(i,j) \in A$ and variable restrictions $X_i \subseteq \{0, \dots, T-1\}$ for all $i \in V$.
Goal:	Find a $v \in [0,T)^n$ such $v_j - v_i \in \Delta_{ij}$ for all $(i,j) \in A$ and $v_i \in X_i$ for all $i \in V$, or state infeasibility.

Note that due to the addition of variable restrictions, the problem has become non-trivial and a different algorithm is required.

6.4.3 Optimizing RPESP

Theorem 6.2. *RPESP can be optimized in $\mathcal{O}(nT^2)$ time.*

Proof. Theorem 6.2 is fundamental for the heuristic in this chapter, and will be proven using dynamic programming. To this aim, label a vertex of choice as the root r of the tree. A vertex j is a child of i if there exists an arc between i and j and i is closer to the root than j in terms of number of arcs. Similarly, i is the parent of j, which is denoted by $p(j) = i$. Note that $p(r) = \emptyset$.

The dynamic program starts with the vertices at the bottom of the tree (i.e., the leaves), and proceeds in a bottom-up fashion by considering in every iteration a vertex of which all children have been considered earlier. Because the graph contains no cycles, such a vertex always exists.

At vertex i, the dynamic program enumerates all feasible solution values for $v_i \in X_i$ and determines for which of these values a feasible solution exists, considering *only* the constraints and variables in the subtree rooted at i (i.e., a subproblem is considered). Additionally, an objective function where potentially every constraint has a cost $z_{ij}(v_i, v_j)$ as described in Subsection 6.2.2 can be added, although this is not required. Using the mentioned model and definitions, the dynamic program will use the following function:

$$f(i,x) = \begin{array}{l}\text{minimum cost of a feasible solution of the subproblem} \\ \text{rooted at vertex } i, \text{ while } x \in X_i \text{ and } v_i = x,\end{array}$$

with initialization for the leaves as:

$$f(i,x) = \begin{cases} 0 & \text{if } x \in X_i, \\ \infty & \text{otherwise.} \end{cases}$$

In other words, the subproblem rooted at vertex i using $v_i = x$ is infeasible if and only if $f(i,x) = \infty$. The recursive identity that solves the dynamic program is:

$$f(i,x) = \sum_{j:p(j)=i} \min_{v_j \in [T-1]} \left(f(j, v_j) + z_{ij}(v_i, v_j) \right),$$

for $x_i \in X$.

The correctness of the recursion of the dynamic program can be inductively argued as follows. The goal is to determine the optimal solution value of the subproblem rooted at i, when v_i is fixed at x. Prior to this stage, the dynamic program has determined for every child j of i what the optimal value $f(j, v_j)$, for every possible value $v_j = 0, \ldots, T-1$ of the individual subproblems rooted at child j. Whenever vertex i is added to the subproblem, more terms in the objective function need to be considered. However, since the graph is a tree, only terms to the objective function are added between i and its children, i.e., the terms $z_{ij}(v_i, v_j)$ for all j. Since a fixed $v_i = x$ is considered for evaluating $f(i, x)$ and the subproblems rooted at the children of i can be optimized independently of each other, one can simply iterate in linear time what the optimal value for v_j is, including also the terms in $z_{ij}(v_i, v_j)$.

The running time of this dynamic program is as follows. Let c_i be the number of children of vertex i. Note that $\sum_{i \in V} c_i = n-1$, because every vertex, apart from the root, is a child of exactly one other vertex. Computing one value for $f(i, x)$ takes $\mathcal{O}(c_i T)$ time, because for every child $j = 1, \ldots, n_i$ of i, for exactly $|\Delta_{ij}| = \mathcal{O}(T)$ values need to be verified whether there exists a v_j such that $(v_j - x) \bmod T \in \Delta_{ij}$. Since $f(i, x)$ needs to be calculated for at most T values for every vertex $i \in V$, the running time concludes to $\mathcal{O}\left(T \cdot \sum_{i \in V} c_i T\right) = \mathcal{O}(\left(\sum_{i \in V} c_i\right)^2) = \mathcal{O}(nT^2)$. This proves Theorem 6.2.

□

Finally note that the dynamic program can be terminated earlier if it detects for a vertex i that there exists no $f(i, x) < \infty$, as this implies there is no solution for the subproblem rooted at i (and therefore the RPESP instance).

6.5 Tree decomposition heuristics

Decomposing the PESP into trees is the key technique for heuristics used to solve PESP instances. The intuition behind this method has been explained in Section 6.4.1: the problem is decomposed in subproblems which are solved independently, and integrated afterwards. If integration is not possible, it is desirable to make as few changes as possible to enable integration. This is elaborated in the next subsections.

6.5.1 Decomposing a PESP graph into trees

An important part of the algorithm concerns the decomposing of the original graph D into trees. Clearly, this can be done in numerous ways for realistic instances. For this research, a simple greedy heuristic has been applied based on the feasible intervals Δ_{ij}. To describe the method intuitively, a component C will be initialized by adding the two vertices i and j that correspond to the arc with minimal $|\Delta_{ij}|$. Subsequently, a vertex is added to C if its addition will not lead to a cycle within the component.

The resulting tree graphs, which by definition are components, are denoted as $C_1, \ldots, C_k$. As mentioned earlier, the original graph D is not equal to $\cup_{i=1}^{k} C_i$, since

the bridging constraints are not considered. Indeed, when all trees are optimized individually, the bottleneck lies in satisfying the bridging constraints.

6.5.2 Requirements for partial solutions

Before specifying the requirements of a partial solution, we note the following.

Theorem 6.3. *Given two components C_x and C_y with respect to D, a given solution v^x can be **extended** to a feasible solution for the (merged) component $C_{xy} = (V_x \cup V_y, A_x \cup A_y \cup B_{xy})$ if and only if there exists a solution to the RPESP instance C_y with variable restrictions $X_j = \cap_{(i,j)\in A: i\in V_x} ((v_i \oplus \Delta_{ij}) \mod T)$, for all $v_j \in V_y$.*

To emphasize the difference, v^x is a **partial solution** to D, but a complete solution to C_x. It is of interest whether v^x can be extended to a feasible solution for the merged subproblem C_{xy}, including the bridging constraints.

Proof of Theorem 6.3. To see the correctness of Remark 6.3, note that by definition, all constraints in A_x are satisfied by definition of v^x. Moreover, by construction of X_i, the bridging constraints B_{xy} are fulfilled if the variable restrictions are satisfied. Hence, the remaining constraints A_y are fulfilled if there exists a solution to the RPESP instance using these variable restrictions.

Note that the dynamic program explained in the proof of Theorem 6.2 can determine whether a partial solution v^x can be extended to a feasible solution for C_{xy}. Moreover, optimization of an objective can be taken into account to retrieve the best solution for C_{xy} given v^x. This justifies more formally the consideration of the RPESP. Indeed, the next step is to integrate a feasible solution for C_{xy} to a solution for a larger component.

Using this concept, one needs to find partial solutions $v^1, \dots, v^k$ such that $v^x \cup v^y$ is a feasible solution for C_{xy} for all $x = 1, \dots, k$ and $y = x+1, \dots, k$.

Clearly, a prerequisite for every partial solution v^x with respect to D is that it can be extended to a solution for the merged subproblem C_{xy} for all $y = 1, \dots, k$. If not, then v^x clearly cannot be extended to a solution for the original problem $D = (V, A)$. One can verify in $\mathcal{O}(knT^2)$ time whether a solution can be extended to a solution for merged subproblems, using the dynamic program. □

6.5.3 Identifying non-extendable partial solutions

The idea will firstly be illustrated informally by reconsidering the example in Figure 6.2. Given the solution $v^1 = (0, 12, 14, 18)$ for C_1, the bridging constraints impose variable restrictions $X_5 = \{15, 16, 17\}$ and $X_6 = \{26, 27\}$. It turns out that, given the solution v^1 for C_1, C_2 in fact has become infeasible. After all, the constraint a_{57} demands that $v_7 \in \{15, \dots, 27\}$, while a_{67} demands that $v_7 \in \{28, 29, 30\}$, making the feasible region for v_7 equal to $\{15, \dots, 27\} \cap \{28, 30\} = \emptyset$. Even though the full PESP-instance is feasible, e.g., if

$$v = (0, 10, 10, 15, 15, 23, 25, 35),$$

no feasible solution v^2 for C_2 can be found given the variable restrictions imposed by solution v^1. This clearly means that a different solution for C_1 needs to be found. While attempting to solve C_2, the dynamic program will note this as well, since $f(7, x)$ will be FALSE for all x. Informally, the dynamic program needs to send feedback to C_1 on how to find a feasible solution (that can be extended to a feasible solution for C_2), by imposing additional constraints on finding a solution for v^1 for C_1.

In this specific example, note that a change has to be made in the subset (v_1, v_2, v_4); a feasible value for v_3 can instantly be found due to the tree structure. Thus, one needs to analyze the possible values for (v_1, v_2, v_4) and identify which combinations of values can never lead to a feasible solution for C_2. This procedure will be formalized in the next section.

6.5.4 Fixing non-extendable partial solutions

To solve the problem mentioned in the previous subsection, we introduce the following notion.

Definition 6.4. *Given a PESP-instance, a* ***subset ban*** $(Y_i, \dots, Y_k)$*, with* $Y_j \subseteq \{0, \dots, T-1\}$ *for* $j = i, \dots, k$*, is a set of variable values for which any combination* $(v_i, \dots, v_k) \in Y_i \times \cdots \times Y_k$ *can never extend to a feasible solution.*

Subset bans basically form an administration of combinations of variables from which the dynamic program already concluded that this leads to guaranteed infeasibility. In this way, an earlier found partial solution for a component C_x can never be considered again, if it has been proven to be non-extendable to another component. When finding a feasible solution from the dynamic program described in Section 6.2, one can easily determine a value that fulfills these bans by picking a value x for a variable i for which $f(i, x) < \infty$ and $v_i \notin X_i$. Note that there can be multiple subset bans on the same subset.

To complete the heuristic, suppose v^x can be extended to a solution $v^x \cup v^y$ for C_{xy}, and v^x can also be extended to a solution $v^x \cup v^z$ for C_{xz}, where v^y and v^z can be deduced from the dynamic programs. Having found these solutions, this does not necessarily mean that $v^y \cup v^z$ is a solution for C_{yz} (the constraints in B_{yz} have not been considered). This directly implies that $v^x \cup v^y \cup v^z$ is not necessarily a solution to C_{xyz}. This is indeed where exponentiality theoretically can occur. Once multiple trees are integrated in a component C, but are not able to be integrated with another tree C_x, there may be subset bans in C spanning multiple trees. Note that this problem occurs more if the trees are connected to each other, which occurs less in a railway timetabling framework due the railway network (variables/trains in a specific part of the country are less related to variables/trains at the far other end of the country).

6.6 Experimental results

For this research, the 16 railway timetabling instances from publicly available PESP benchmark library PESPlib [46] have been used. The upper bound for the running

time has been set to 1 hour, though if a possible solution can be found, it is usually done within minutes. The remainder of the running time is spent on optimizing the objective function. The results are summarized in Table 6.1.

Table 6.1: Results of the tree decomposition using the PESLlib datasets

Dataset	# Var.	% Cons.	Trees	Sol. value	Best value	% Diff.
R1L1	3664	6385	5	36.1	31.1	+16.0%
R1L2	3668	6543	4	38.3	31.7	+20.8%
R1L3	4184	7031	5	35.0	30.5	+14.8%
R1L4	4760	8528	4	31.9	27.9	+14.3%
R2L1	4156	7361	4	48.8	42.5	+14.8%
R2L2	4204	7563	5	50.1	43.1	+16.2%
R2L3	5048	8286	4	42.9	39.9	+7.5%
R2L4	7660	13173	4	40.1	33.0	+21.5%
R3L1	4516	9145	5	55.4	45.4	+22.0%
R3L2	4452	9251	5	54.7	46.2	+18.4%
R3L3	5724	11169	5	56.5	43.0	+31.4%
R3L4	8180	15657	5	N/A	35.5	N/A
R4L1	4932	10262	5	61.2	51.7	+18.3%
R4L2	5048	10735	5	64.6	52.0	+24.4%
R4L3	6368	13238	6	N/A	45.8	N/A
R4L4	8384	17754	4	N/A	38.8	N/A

All experiments were conducted on a PC with an AMD Ryzen 5 1600 Six-Core Processor (3.20 GHz) with 16 GB of RAM. The source code was written in Java. To clarify Table 6.1:

- **Trees** is the minimum number of trees to which the variables can be decomposed for the tree decomposition heuristic.
- **Sol. value** is the solution value when using the tree decomposition heuristic in millions. If no feasible solution could be found within the time bound, N/A is given.
- **Best value** is the best found solution value at the time of publishing this research (also in millions), generally by [46].
- **% difference** is the percentual difference between sol. value and best value.

Although the tree decomposition heuristic does not improve the current best, the performance on these datasets can still be practically useful and at least offer perspective for improvements. Particularly, the short duration of the tree decomposition method, for an entire timetable with constraints of an entire country, is one of the key contributions. To the best of the knowledge of the author, there exists no method that can solve large instances (after data reduction) within such a short amount of time.

Unfortunately, three of the datasets could not be solved by the tree decomposition heuristic. This may be due to the higher number of constraints, or possibly a structure within the constraints where the heuristic cannot deal properly with. Nevertheless, the other 13 datasets could be solved, although the performance is about 20% worse on average than the currently best found solutions. The actual time that can be saved is hard to estimate, as the running time for the best solutions are not published. Still, since this method is a heuristic from a new perspective, there is room for improvements and perhaps potential to improve the currently known approaches.

6.7 Conclusions and future work

The PESP is a difficult problem for which the current literature is seeking more practical methods to escape local optima, without applying brute force in an early stage. This chapter has proposed techniques for heuristics that decompose a PESP problems into trees. These techniques are primarily based on dynamic programming, which allows the usage of a smart objective function that heuristically maximizes the possibility that a solution for a component can be extended to a solution for all other components. Experiments are performed using online benchmarks, and even though the heuristic performs on average about 20% worse in terms of objective function, feasible solutions can still be found quickly. The amount of time that can be saved is currently hard to estimate, as the running time for the best solutions are not published.

Future research should be done in improving this method to find feasible and better solutions in a fast way. Other future work concerns the incorporation of heuristics for the PESP into parallel problems. Current research includes the routing of trains through stations in parallel to the optimization of the PESP. Due to the highly complex structure of both problems, it is plausible that heuristics are likely to be more suitable than exact optimization techniques.

Chapter 7

Conclusions and future work

In this thesis, both the theory and applications of four scheduling problems from different areas have been studied. Even though every of the individual chapters has its own conclusion, an overview and comparison of the contributed approaches and gained insights is given in this concluding chapter, based on the goals and research questions mentioned in Section 1.3.

Models and algorithms To reflect on the primary goal, this thesis has proposed a variety of new perspectives, models and algorithms for four scheduling problems, including for special cases thereof to achieve the secondary goal. The novelty and suitability of these approaches is discussed here.

Algorithms based on dynamic programming are introduced for the acyclic BJCP in Section 3.4 and the restricted PESP in Section 6.4. These both run in pseudo-polynomial time. Dynamic programming turns out to be an adequate approach for these problems, because the cycle-free property ensures that the (optimal) solution of a subproblem can be used to solve a slightly bigger subproblem efficiently containing the already solved subproblem.

Moreover, for both problems, an algorithm for the general problem is proposed that uses the dynamic program for the acyclic version as a subroutine. However, one might conclude that the proposed algorithm for the general BJCP is less suitable. After all, the running time of the most straightforward algorithm using this subroutine includes two exponentially growing terms (see Theorem 3.5). On the other hand, the proposed tree decomposition heuristic for the PESP (see Section 6.5) is argued to be at least viable for practical purposes. As seen in the experimental results in Section 6.6, despite not being able to solve every instance, the heuristic can solve several real-life instances within a reasonable amount of time. The difference between these approaches is that the algorithm for the BJCP contains complete enumeration for which no timesaving methods or exploitable insights are available so far. For the PESP, a heuristic is designed that maximizes the size of solvable, acyclic components. Since this minimizes the number of different components that need to be merged, fewer integration steps are needed.

Yet, it is important to note that the use of dynamic programming in graphs is not limited to acyclic graphs, especially for scheduling problems. This is shown in Chapter 4 for the ECSP in the case where the constraints on the duty times

(rather than driving times) are considered only. A column generation approach is explained in Section 4.3 for which a standard pricing problem is solved to find a daily duty (without breaks, but with a maximum bound on the duty time) with lowest reduced costs. The dynamic program becomes more complicated when a weekly duty of a driver with lowest reduced costs needs to be found, but the computational complexity hardly increases. In this case, not only needs to be ensured that the length of the individual daily duties does not exceed a length bound L, but now additionally also needs to be ensured that the resting time between every subsequent pair of daily duties is at least λ^{rest}_{daily}.

In order to solve this special case of the ECSP more efficiently, it is crucial to note that the duty time of a daily duty starting in trip t and ending in trip u does not depend on the trips driven between t and u. In other words, the daily duty time can be determined in constant time when the starting and ending trip is known. This property is characteristic to scheduling problems, where the jobs/trips (and therefore the corresponding graph) contain such a time structure. The algorithms proposed in Section 4.3.4 (weekly duty) make use of this time structure, which allows a polynomial running time algorithm to solve the pricing problem. However, column generation is still a part from the branch-and-price procedure, which in theory runs in exponential time.

Finally, the RWPP is characterized by the stochastic nature of procedure and hospitalization times for patients. This complicates the design of an Integer Linear Program for which some performance measures in the objective function depend on stochastic factors. As shown in Section 5.3.3, the expected overtime on a room is based on the mean and the variance, but does not grow linearly with these two parameters, making Integer Linear Programming theoretically unsuitable. Such a non-linear expression can be approximated by a combination of linear expressions with the use of tangent lines. For this problem, an assumption is made that whenever the expected procedure time increases, the corresponding variance increases similarly. Despite this assumption being questionable in general, it does allow the model to incorporate stochasticity of procedure times, hospitalization times and urgent arrivals, while minimizing the weighted sum of important performance measures for ORs and wards that could not have been found simultaneously in the literature before.

Complexity and structure All four problems were shown to be NP-complete by a reduction from another NP-complete problem. As explained in Section 2.4, this means that the problem is very unlikely to be solvable efficiently (as otherwise P = NP). To reflect on the goal to identify what structural aspects may make a scheduling problem hard to solve, note the following.

Although both the BJCP and PESP were shown to be NP-complete, but these problems can be solved in pseudo-polynomial time in case the underlying graph was cycle-free (see Theorem 3.3 and 6.2, respectively). This implies that cycles of the underlying graph is one of the reasons that such (scheduling) problems are hard.

To handle graphs with cycles to a certain extent, one could isolate a set of vertices for which its removal would make the graph cycle-free, also known as a feedback

vertex set (see e.g., Theorem 3.5). However, this comes with several downsides. First, an algorithm that finds such a feedback vertex set runs in exponential time. Second, this still requires to completely enumerate over all possible values in the feedback vertex set, which also may be an exponential number of (acyclic) instances. As an alternative, the graph may be decomposed in acyclic instances that can be solved independently. However, there is no efficient way to guarantee that these solved instances can be integrated in a feasible (and therefore optimal) solution to the original instance.

Future work The primary goal of this thesis was to provide new perspectives, models and algorithms to solve scheduling problems. The proposed methods at some point reached its limitation regarding computation time, performance, or some constraints could not easily be incorporated in the model or solution method at all. Cyclicity in graphs has been identified as a main characteristic why the considered scheduling problems cannot be solved easily.

An interesting, overarching future research direction would be to identify what characteristics of scheduling problems make some techniques more suitable than others. Such characteristics may include the structure of the problem (e.g., whether the jobs are fixed to a specific time or not, or whether the problem can be modeled as a directed acyclic graph) or a specific type of constraint. To illustrate this idea, consider the following comparison. Most vehicle routing problems can be modeled using a graph since it is usually assumed that every resource (driver) can perform every task (trip). This enables the use of classical results in graph theory, where nodes and paths often represent the tasks and the assignments to resources, respectively. On the other hand, hospital planning problems are rarely formulated as a graph, which may be because not every task (operation) can be done by every resource (room and/or doctor). When both scheduling problems are modeled as a graph, a path in the graph for the vehicle routing problem results in a feasible duty, while this is not necessarily the case for the hospital planning problem. This difference in problem structure might mean that graph algorithms are generally more suitable for vehicle routing problems compared to hospital planning problems.

To give insight in why some approaches are more suitable than others for the same scheduling problem, an enormous amount of additional research would be required. This is not only due to the numerous types of scheduling problems, but also because of large number of solution methods in Operations Research. To be complete in achieving the goal of this thesis for one scheduling problem, one would need to consider all possible approaches to the limit for this specific scheduling problem, and compare the approaches by performance (computation time and solution quality) on sufficiently problem instances. However, this clearly would require too much research time. Hence, only a subset of the possible approaches for scheduling problems has been considered in this thesis.

For the BJCP, ECSP, PESP and RWPP, an approximation algorithm, ILP model, dynamic program, graph algorithm and/or heuristic has been proposed. But it would have been interesting to attempt to implement more approaches for every problem, and compare the performance of such algorithms with the ILP model and

heuristics. And in case such an algorithm cannot be designed, it is interesting to understand which constraint or characteristic makes this too difficult. In this way, researchers that consider a new scheduling problem with similar constraints or characteristics immediately have better insight in why a specific method is suitable or not to the problem.

Summary

Scheduling problems belong to the classical optimization problems in Operations Research due to their practical significance and theoretical elegance. This thesis focuses on both the theory and applications of scheduling and contributes new methodologies and insights in both aspects. With this aim, four scheduling problems from different areas have been considered in this thesis. For all four problems is proven that the problem is difficult, meaning that it is very unlikely that an algorithm exists that can solve the problem efficiently. For this reason, alternative approaches are required that are proposed for every problem.

Chapter 3 considered the Budgeted Job Coverage Problem (BJCP), where the goal is to select a subset of a given family of resource jobsets over a set of jobs with costs and weights. The goal is to maximize the total weight while not exceeding a budget. Because costs are assigned to jobs (or elements) instead of sets, analyses from related work, particularly on greedy approaches for maximum coverage problems, cannot be applied. Instead, special cases of the BJCP have been considered. In case each resource jobset has cardinality 2, a $\frac{1}{2}(1-\frac{1}{\sqrt{e}})$-approximation algorithm can be obtained (see Section 3.3). Moreover, it has been shown that in case the corresponding incidence graph is acyclic, the problem can be solved in pseudo-polynomial time using a bi-level dynamic program. This result suggests that cycles increase the computational complexity of the problem significantly. To that end, a method has been proposed to detect and remove those cycles from the graph by finding a so-called feedback vertex set, but this algorithm comes with a computation time of exponential order.

In Chapter 4, the European Crew Scheduling Problem (ECSP) is studied, where the goal is to assign trips to drivers, subject to bounds on breaks, driving and duty times of the drivers that are imposed by European regulations. It has been shown that a special case, Path Cover with length bounds on a transitive graph, is strongly NP-hard using a reduction from 3-Partition. A new MILP formulation has been introduced, which can take practically all European regulations into account (breaks, maximum duty times, maximum daily and weekly driving times and minimum daily and weekly resting times). However, this may take too time-consuming to solve for solvers due to its size and complexity. As an alternative, a two-level column generation approach has been proposed for a special case of the problem, where only one or multiple daily duties (excluding breaks) are considered. This method crucially exploits the time structure of the underlying graph of the pricing problem. Additionally, algorithms and research directions for the ECSP with more constraints (multiple weekly duties, maximum driving times, breaks) have been given, which

include some extensions of the model for a weekly duty. For the single duty case, this approach is very suitable for realistic instances, but when scheduling over a longer period (e.g., a week) the column generation approach may fail to deliver the optimal solution within a reasonable amount of time. As an alternative, a beam-search heuristic is proposed. Results show that this method is competitive with the column generation approach, but runs much faster.

Chapter 5 considered the Room and Ward Planning Problem (RWPP), where the goal is to optimize a planning of cath labs (comparable to operating rooms) and the underlying wards, subject to a large variety of realistic constraints. A new model is proposed that incorporates stochasticity of procedure times, hospitalization times and urgent arrivals, while minimizing the weighted sum of important performance measures for operating rooms and wards. This model is solved using a linearization method. Even though this linearization is an approximation, one can increase the quality of the approximation by simply adding more tangent lines, but this comes with an increase of the computation time. Although linearization of objective functions is not uncommon in the literature, an application to approximate the expected overtime of a room subject to the constraints in this chapter could not have been found before. This was possible in this research due to an additional assumption, which informally implies that a procedure with a higher expected procedure time also has a higher variance. While this assumption remains questionable, for rooms where only one specialty operates (in this case, cardiology), this assumption is more justifiable than in a general operating room setting with multiple specialties. Finally, a case study (VU University Medical Center) has been presented, where some insights in data regarding procedure times are given, and where the model is applied. This resulted in a planning for the hospital that can be used as a blueprint.

Chapter 6 provided a different view on the classical Periodic Event Scheduling Problem (PESP). In this problem, the goal is to find a cyclic timetable, given a set of events and constraints on the time difference between pairs of events. While the PESP normally is a feasibility problem, a variant with an objective function has been considered instead. A heuristic that decomposes a PESP into trees is given, where the variables in the trees are bound to specific subsets of values. In order to solve such trees, an algorithm is proposed that can solve this restricted variant using dynamic programming. The objective function in this dynamic program leaves some room for optimization, to heuristically maximize that a solution for a component can be extended to a solution for all other components. Experiments are performed using online benchmarks, and even though the heuristic performs on average about 20% worse in terms of objective function, feasible solutions for many instances can still be found within a reasonable amount of time.

Bibliography

[1] Ivo Adan, Jos Bekkers, Nico Dellaert, Jan Vissers, and Xiaoting Yu. Patient mix optimisation and stochastic resource requirements: A case study in cardiothoracic surgery planning. *Health Care Management Science*, 12(2):129–141, 2008.

[2] Vehicle & Operator Services Agency. *Rules on Drivers' Hours and Tachographs: Goods vehicles in GB and Europe.* Vehicle & Operator Services Agency, 2011.

[3] Vehicle & Operator Services Agency. *Rules on Drivers' Hours and Tachographs: Passenger-carrying vehicles in GB and Europe.* Vehicle & Operator Services Agency, 2011.

[4] Simone Barbagallo, Luca Corradi, Jean de Ville de Govet, Marina Iannucci, Ivan Porro, Nicola Rosso, Elena Tanfani, and Angela Testi. Optimization and planning of operating theatre activities: an original definition of pathways and process modeling. *BMC Medical Informatics and Decision Making*, 15(38), 2015.

[5] René Bekker and A. M. de Bruin. Time-dependent analysis for refused admissions in clinical wards. *Annals of Operations Research*, 178(1):45–65, 2010.

[6] René Bekker and Paulien M. Koeleman. Scheduling admissions and reducing variability in bed demand. *Health Care Management Science*, 14(3):237—-249, 2011.

[7] Jeroen Beliën and Erik Demeulemeester. Building cyclic master surgery schedules with leveled resulting bed occupancy. *European Journal of Operational Research*, 176:1185–1204, 2005.

[8] Russell Bent and Pascal Van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, 33(4):875–893, 2006.

[9] Vincent Boyer, Omar J. Ibarra-Rojas, and Yasmín Á. Ríos-Solís. Vehicle and crew scheduling for flexible bus transportation systems. *Transportation Research Part B: Methodological*, 112:216–229, 2018.

[10] Peter Brucker. *Scheduling Theory.* Springer-Verlag, 2007.

[11] M. R. Bussieck, T. Winter, and U. T. Zimmermann. Discrete optimization in public rail transport. *Mathematical Programming*, 79(1):415–444, 1997.

[12] Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

[13] Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set new measure and new structures. In *Algorithm Theory - SWAT 2010*, pages 93–104. Springer Berlin Heidelberg, 2010.

[14] Alberto Caprara, Matteo Fischetti, Paolo Toth, Daniele Vigo, and Pier Luigi Guida. Algorithms for railway crew management. *Mathematical Programming*, 79:125–141, 1997.

[15] Brecht Cardoen, Erik Demeulemeester, and Jeroen Beliën. Operating room planning and scheduling: A literature review. *European Journal of Operational Research*, 201(3):921–932, 2010.

[16] Paolo Carraresi, Maddalena Nonato, and Leopoldo Girardi. Network models, lagrangean relaxation and subgradients bundle approach in crew scheduling problems. In Joachim R. Daduna, Isabel Branco, and José M. Pinto Paixão, editors, *Computer-Aided Transit Scheduling*, pages 188–212, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[17] Tugba Cayirly and Emre Veral. Outpatient scheduling in health care: A review of literature. *Production and Operations Management*, 12(4):519–549, 2003.

[18] Mingming Chen and Huimin Niu. A model for bus crew scheduling problem with multiple duty types. *Discrete Dynamics in Nature and Society*, 2012, 09 2012.

[19] Reuven Cohen and Liran Katzir. The generalized maximum coverage problem. *Information Processing Letters*, 108(1):153–22, 2008.

[20] Stephen Cook. The complexity of theorem proving procedures. *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[21] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[22] European Transport Safety Council. *The Role of Driver Fatigue in Commercial Road Transport Crashes*. European Transport Safety Council, 2001.

[23] Renato De Leone, Paola Festa, and Emilia Marchitto. A bus driver scheduling problem: a new mathematical model and a grasp approximate solution. *Journal of Heuristics*, 17(4):441–466, 2011.

[24] Erik Demeulemeester, Jeroen Beliën, Brecht Cardoen, and Michael Samudra. Operating room planning and scheduling. *Handbook of Healthcare Operations Management*, 184:121–152, 2013.

[25] Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon. *Column Generation*. Springer, 2006.

[26] Martin Desrochers, Jacques Desrosiers, and Marius M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, 1992.

[27] Martin Desrochers and Francois Soumis. A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23(1):1–13, 1989.

[28] Muhammet Deveci and Nihan Çetin Demirel. A survey of the literature on airline crew scheduling. *Engineering Applications of Artificial Intelligence*, 74:54–69, 2018.

[29] Teresa Dias, Jorge Pinho de Sousa, and João Falcão e Cunha. Genetic algorithms for the bus driver scheduling problem: a case study. *Journal of the Operational Research Society*, 53, 03 2002.

[30] Edsger W. Dijkstra. A note on two problems in connexion with graph. *Numerische Mathematik*, 1(1):269—-271, 1959.

[31] Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

[32] Agoston E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2013.

[33] Andreas Ernst, Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27, 2004.

[34] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[35] Jon Feldman, S. Muthukrishnan, Martin Pál, and Clifford Stein. Budget optimization in search-based advertising auctions. In *Proceedings of the 8th ACM Conference on Electronic Commerce*, pages 40–49. ACM, 2007.

[36] Lester Randolph Ford Jr. and Delbert Ray Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5(1):97–101, 1958.

[37] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[38] Richard Freling, Dennis Huisman, and Albert P. M. Wagelmans. Models and algorithms for integration of vehicle and crew scheduling. *Journal of Scheduling*, 6(1):63–85, 2003.

[39] Richard Freling, Ramon Lentink, and Michiel Odijk. Scheduling train crews: A case study for the dutch railways. *Compter-Aided Scheduling of Public Transport*, pages 153–165, 2001.

[40] Stephen Gallivan and Martin Utley. Modelling admissions booking of elective in-patients into a treatment centre. *IMA Journal of Management Mathematics*, 16(3):305–315, 2005.

[41] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[42] William Gasarch. Guest column: The third P=?NP poll. *ACM SIGACT News*, 50:38–59, 2019.

[43] Asvin Goel. Vehicle scheduling and routing with drivers' working hours. *Transportation Science*, 43(1):17–26, 2009.

[44] Asvin Goel. Truck driver scheduling in the European Union. *Transportation Science*, 44(4):429–441, 2010.

[45] Asvin Goel and Volker Gruhn. Drivers' working hours in vehicle routing and scheduling. In *Transportation Science*, volume 43, pages 1280–1285. IEEE, 2006.

[46] Marc Goerigk. PESPlib, A benchmark library for periodic event scheduling. http://num.math.uni-goettingen.de/~m.goerigk/pesplib/, 2019.

[47] Marc Goerigk and Anita Schöbel. Improving the modulo simplex algorithm for large-scale periodic timetabling. *Computers & Operations Research*, 40(5):1363–1370, 2013.

[48] Balaji Gopalakrishnan, Ellis Johnson, et al. Airline crew scheduling: State-of-the-art. *Annals of Operations Research*, 140(1):305–337, 2005.

[49] Francesca Guerriero and Rosita Guido. Operational research in the management of the operating theatre: A survey. *Health care management science*, 14:89–114, 03 2011.

[50] Diwakar Gupta, Madhu Kailash Natarajan, Amiram Gafni, Lei Wang, Don Shilton, Douglas Holder, and Salim Yusuf. Capacity planning for cardiac catheterization: A case study. *Health Policy*, 82(1):1–11, 2007.

[51] Şeyda Gür and Tamer Eren. Application of operational research techniques in operating room scheduling problems: Literature overview. *Journal of Healthcare Engineering*, 2018, 2018.

[52] Gabriel Y. Handler and Israel Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10:293–309, 1980.

[53] Erwin W. Hans, Mark van Houdenhoven, and Peter J. H. Hulshof. *A Framework for Healthcare Planning and Control*, pages 303–320. Springer US, Boston, MA, 2012.

[54] Julia Heil, Kirsten Hoffmann, and Udo Buscher. Railway crew scheduling: Models, methods and applications. *European Journal of Operational Research*, 283(2):405–425, 2020.

[55] Jonathan E. Helm and Mark P. Van Oyen. Design and optimization methods for elective hospital admissions. *Operations Research*, 62(6):1265–1282, 2014.

[56] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Co., 1997.

[57] John E. Hopcroft and Richard M. Karp. A $n^{5/2}$ algorithm for maximum matchings in bipartite. In *12th Annual Symposium on Switching and Automata Theory (SWAT 1971)*, pages 122–125, 1971.

[58] Dennis Huisman, Leo Kroon, Ramon Lentink, and Michiel Vromans. Operations research in passenger railway transportation. *Statistica Neerlandica*, 59:467–497, 02 2005.

[59] Peter J. H. Hulshof, Nikky Kortbeek, Richard J. Boucherie, Erwin W. Hans, and Piet J. M. Bakker. Taxonomic classification of planning decisions in health care: a structured review of the state of the art in or/ms. *Health Systems*, 1(2):129–175, 2012.

[60] Omar Ibarra-Rojas, Felipe Delgado, Ricardo Giesen, and Juan Muñoz. Planning, operation, and control of bus transport systems: A literature review. *Transportation Research Part B: Methodological*, 77:38–75, 2015.

[61] Rishabh Iyer and Jeff Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, pages 2436–2444. Curran Associates Inc., 2013.

[62] Narendra Karmarkar. A new polynomial-time algorithm for linear programmingg. *Combinatorica*, 4(4):373—395, 1984.

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

[64] Atoosa Kasirzadeh, Mohammed Saddoune, and François Soumis. Airline crew scheduling: models, algorithms, and data sets. *EURO Journal on Transportation and Logistics*, 6(2):111–137, 2017.

[65] Atoosa Kasirzadeh, Mohammed Saddoune, and François Soumis. Airline crew scheduling: models, algorithms, and data sets. *EURO Journal on Transportation and Logistics*, 6(2):111–137, 2017.

[66] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer-Verlag, 2004.

[67] Leonid Khachiyan. A polynomial algorithm for linear programming. *Doklady Akademii Nauk SSSR*, 224(5):1093—1096, 1979.

[68] Samir Khuller, Anna Moss, and Joseph (Seffi) Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

[69] Leendert Kok, Erwin W. Hans, Marco Schutten, and Willem Zijm. A dynamic programming heuristic for vehicle routing with time-dependent travel times and required breaks. *Flexible services and manufacturing journal*, 22:83–108, 2010.

[70] Dénes König. Über graphen und ihre anwendung auf determinantentheorie und mengenlehre. *Mathematische Annalen*, 77(4):453–465, 1916.

[71] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2018.

[72] M. Koubaa, Souhail Dhouib, Diala Dhouib, and Abderrahman El Mhamedi. Truck driver scheduling problem: Literature review. *IFAC-PapersOnLine*, 49(12):1950–1955, 2016.

[73] Leo Kroon and Matteo Fischetti. Scheduling train drivers and guards: the dutch "noord-oost" case. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, pages 10 pp. vol.2–, 2000.

[74] Leo Kroon, Dennis Huisman, and Gábor Maróti. Optimisation models for railway timetabling. *Railway Timetable and Traffic*, pages 135–154, 2008.

[75] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992.

[76] Gréanne Leeftink and Erwin Hans. Case mix classification and a benchmark set for surgery scheduling. *Journal of Scheduling*, 21, 02 2018.

[77] Jan Karel Lenstra and Alexander H. G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(1):221–227, 1981.

[78] Osnat Levtzion-Korach, Arthur Reitman, Pat Jansen, and Susan Madden. Doing more with less overtime: Improving patient flow through the cath lab. *Cath Lab Digest*, 16(10), 2008.

[79] Haibing Li and Andrew Lim. A metaheuristic for the pickup and delivery problem with time windows. In *Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence*, pages 160–167. IEEE Computer Society, 2001.

[80] Christian Liebchen. A cut-based heuristic to produce almost feasible periodic railway timetables. In *Proceedings of the 4th International Conference on Experimental and Efficient Algorithms*, pages 354–366. Springer-Verlag, 2005.

[81] Christian Liebchen and Rolf H. Möhring. The modeling power of the periodic event scheduling problem: Railway timetables — and beyond. In *Algorithmic Methods for Railway Optimization*, pages 3–40. Springer Berlin Heidelberg, 2007.

[82] Christian Liebchen and Leon Peeters. Some practical aspects of periodic timetabling. In *Operations Research Proceedings 2001*, pages 25–32. Springer Berlin Heidelberg, 2002.

[83] Thomas Lindner. *Train Schedule Optimization in Public Rail Transport*. PhD thesis, Braunschweig University of Technology, 2000.

[84] Eugene Litvak and Michael C. Long. Cost and quality under managed care: Irreconcilable differences? *The American Journal of Managed Care*, 6(3):305–312, 2000.

[85] Helena R. Lourenço, José P. Paixão, and Rita Portugal. Multiobjective metaheuristics for the bus driver scheduling problem. *Transportation Science*, 35(3):331–343, 2001.

[86] Kurt Mehlhorn and Mark Ziegelmann. Resource constrained shortest paths. In Mike S. Paterson, editor, *Algorithms - ESA 2000*, pages 326–337, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[87] Marta Mesquita and Ana Paias. Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem. *Computers & Operations Research*, 35(5):1562–1575, 2008.

[88] Zbigniew Michalewicz. *How to Solve It: Modern Heuristics*. Springer-Verlag, 2010.

[89] Srimathy Mohan, Qing Li, Mohan Gopalakrishnan, John Fowler, and Antonios Printezis. Improving the process efficiency of catheterization laboratories using simulation. *Health Systems*, 6(1):41–55, 2017.

[90] Karl Nachtigall. Cutting planes for a polyhedron associated with a periodic network. Technical report, 1996.

[91] Karl Nachtigall and Jens Opitz. Solving periodic timetable optimisation problems by modulo simplex calculations. In *8th Workshop on Algorithmic*

Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'08), volume 9, pages 1–:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2008.

[92] Simeon Ntafos and Teofilo Gonzalez. On the computational complexity of path cover problems. *Journal of Computer and System Sciences*, 29(2):225–242, 1984.

[93] Michiel A. Odijk. Construction of periodic timetables, part i: A cutting plane algorithm. Technical report, Delft University of Technology, 1994.

[94] Michiel A. Odijk. Construction of periodic timetables, part ii: An application. Technical report, Delft University of Technology, 1994.

[95] Michiel A. Odijk. A constraint generation algorithm for the construction of periodic railway timetables. *Transportation Research Part B: Methodological*, 30:455–464, 1996.

[96] Giselher Pankratz. A grouping genetic algorithm for the pickup and delivery problem with time windows. *OR Spectrum*, 27(1):21–41, 2005.

[97] Jennifer Papin. A suggested approach for improving flow in the cardiac catheterization laboratory. *Cath Lab Digest*, 21(7), 2013.

[98] Leon W. P. Peeters. *Cyclic Railway Timetable Optimization*. PhD thesis, 2003.

[99] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2016.

[100] Abdur Rais and Ana Viana. Operations research in healthcare: a survey. *International Transactions in Operational Research*, 18(1):1—31, 2010.

[101] Romeo Rizzi. A short proof of könig's matching theorem. *Journal of Graph Theory*, 33, 03 2000.

[102] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[103] A. J. (Thomas) Schneider. *Integral Capacity Management & Planning in Hospitals*. PhD thesis, University of Twente, 2020.

[104] A. J. (Thomas) Schneider, J. Theresia van Essen, Mijke Carlier, and Erwin W. Hans. Scheduling surgery groups considering multiple downstream resources. *European Journal of Operational Research*, 282(2):741–752, 2020.

[105] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume A. Springer, 2003.

[106] Alexander Schrijver and Adri Steenbeek. Spoorwegdienstregelingontwikkeling. Technical report, Centrum voor Wiskunde en Informatica, 1993.

[107] Paolo Serafini and Walter Ukovich. A mathematical for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989.

[108] Yindong Shen and Raymond Kwan. Tabu search for driver scheduling. *Computer-Aided Transit Scheduling*, 505:121–135, 01 2001.

[109] Barbara M. Smith and Anthony Wren. A bus crew scheduling system using a set covering formulation. *Transportation Research Part A: General*, 22(2):97–108, 1988.

[110] Pieter S. Stepaniak, Mohamed A. Soliman Hamad, Lukas R.C. Dekker, and Jacques J. Koolen. Improving the efficiency of the cardiac catherization laboratories through understanding the stochastic behavior of the scheduled procedures. *Cardiology Journal*, 21(4):343–349, 2014.

[111] Zoya Svitkina and Lisa Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. *SIAM Journal on Computing*, 40(6):1715–1737, 2011.

[112] Dušan Teodorović and Milan Janic. *Public Transportation Systems*, pages 405–522. 01 2022.

[113] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2002.

[114] Jorne Van den Bergh, Jeroen Beliën, Philippe Bruecker, Erik Demeulemeester, and Liesje De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(4):550–581, 1989.

[115] Marc B. V. Rouppe van der Voort, Arvid J. Glerum, and Erwin W. Hans. *Minimizing Variation in Hospital Bed Utilization by Creating a Case Type Schedule for the Operating Room Planning*, pages 231–247. Springer International Publishing, Cham, 2021.

[116] Irving I. van Heuven van Staereling. Tree decomposition methods for the periodic event scheduling problem. In *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018)*, volume 65, pages 6:1–6:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018.

[117] Irving I. van Heuven van Staereling, René Bekker, and Cornelis P. Allaart. Stochastic scheduling techniques for integrated optimization of catheterization laboratories. In *Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling*, pages 313–329, 2018.

[118] Irving I. van Heuven van Staereling, Bart de Keijzer, and Guido Schäfer. The ground-set-cost budgeted maximum coverage problem. In *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58, pages 50:1–50:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016.

[119] Irving I. van Heuven van Staereling and Guido Schäfer. A branch-and-cut algorithm for driver scheduling under European regulations. *Manuscript*.

[120] Jeroen M. van Oostrum, Mark van Houdenhoven, Johann. L. Hurink, Erwin. W. Hans, Gerhard Wullink, and Geert Kazemier. A master surgical scheduling approach for cyclic scheduling in operating room departments. *OR Spectrum*, 30(2):355–374, 2008.

[121] Peter T. Vanberkel, Richard J. Boucherie, Erwin W. Hans, Johann L. Hurink, Wineke A. M. van Lent, and Wim H. van Harten. An exact approach for relating recovering surgical patient workload to the master surgical schedule. *Journal of the Operational Research Society*, 62(10):1851–1860, 2011.

[122] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[123] Xin Wen, Xuting Sun, Yige Sun, and Xiaohang Yue. Airline crew scheduling: Models and algorithms. *Transportation Research Part E: Logistics and Transportation Review*, 149:102304, 2021.

[124] Anthony Wren and Jean-Marc Rousseau. Bus driver scheduling — an overview. In Joachim R. Daduna, Isabel Branco, and José M. Pinto Paixão, editors, *Computer-Aided Transit Scheduling*, pages 173–187, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[125] Shangyao Yan and Yu-Ping Tu. A network model for airline cabin crew scheduling. *European Journal of Operational Research*, 140(3):531–540, 2002.

[126] Hande Öztop, Ugur Eliiyi, Deniz Eliiyi, and Levent Kandiller. A bus crew scheduling problem with eligibility constraints and time limitations. *Transportation Research Procedia*, 22:222–231, 12 2017.

Appendix A

Acronyms & notation

A.1 Acronyms

Acronym	Description
BJCP	Budgeted Job Coverage Problem
BMCP	Budgeted Maximum Coverage Problem
CSP	Crew Scheduling Problem
CTP	Class-Teacher Problem
DAG	Directed Acyclic Graph
DTBPCP	Driving-Time-Bounded Path Cover Problem
ECSP	European Crew Scheduling Problem
FPTAS	Fully Polynomial Time Approximation Scheme
FVSP	Feedback Vertex Set Problem
MP	Master Problem
MILP	Mixed Integer Linear Programming
PCP	Path Cover Problem
PESP	Periodic Event Scheduling Problem
RMP	Restricted Master Problem
RWPP	Room and Ward Planning Problem
UCTP	University Course Timetabling Problem
VRP	Vehicle Routing Problem
VRPDWH	Vehicle Routing Problem with Driver's Working Hours
WBPCP	Weight-Bounded Path Cover Problem

A.2 Notation

Chapter 3: Job Covering

The following notation is used in Chapter 3. Unless explicitly stated, the notation can be used for both the BJCP and the BMCP.

Notation	Description
$\mathcal{A}$	$\frac{1}{2}(1-\frac{1}{e^{\alpha}})$-approximation algorithm for the BMCP
B	Budget
c_j	Costs of processing/performing job j
$c(S)$	Costs of a subset $S \in F$ (only applies for the BMCP)
$c(\sigma)$	Costs of the jobs of solution σ
$f_I(Y)$	α-approximate cost-efficiency oracle with input Y (see Def. 3.3)
F	Family of subsets $\{S_1, \ldots, S_m\}$ (or resource jobsets)
$G(F)$	Incidence graph of a family of subsets F (see Def. 3.1)
i	Index of a subset $S_i \in F$ (corresponding to e.g., a resource)
j	Index of an element (or job) $j \in F$
J	Set of jobs $\{1, \ldots, n\}$
$J(\sigma)$	Union of all jobs of solution σ $J = \cup_{S \in \sigma} S \subseteq X$
k	Index for the iteration of Step 2 of Algorithm $\mathcal{A}$
ℓ	Total number of iterations of Step 2 of Algorithm $\mathcal{A}$
m	Number of subsets (or resource jobsets)
n	Number of elements (or jobs)
$r(I)$	Instance I of the BJCP reduced to an instance of the BMCP
S	Subset of jobs (or elements) of J
$S_k^{\mathcal{A}}$	Subset output by f_I during iteration k of Step 2 of Algorithm $\mathcal{A}$
w_j	Weight of element (or job) j
$w(\sigma)$	Weight of the jobs of solution σ
X	Set of n elements (or jobs) $X = \{1, \ldots, n\}$
$X(\sigma)$	Union of elements of the sets in σ
Y	Subset of elements $Y \subseteq X$
σ	Solution $\sigma \subseteq F$
$\sigma_k^{\mathcal{A}}$	Partial solution at the end of iteration k of Step 2 of Algorithm $\mathcal{A}$

Chapter 4: Crew scheduling

Notation	Description
$D(d)$	Driving time of daily duty d
$D(s)$	Driving time of schedule s
$D(w)$	Driving time of weekly duty w
$D(\sigma)$	Driving time of solution σ
$E(d)$	Ending time of daily duty d
$E(t)$	Ending time of trip t
$E(w)$	Ending time of weekly duty w
$S(d)$	Starting time of daily duty d
$S(t)$	Starting time of trip t
$S(w)$	Starting time of weekly duty w
$Y(d)$	Duty time of daily duty d
$Y(w)$	Duty time of weekly duty w
$Y(s)$	Duty time of schedule s
$Y(\sigma)$	Duty time of solution σ
$M(t,u)$	Travel time from ending and starting location of resp. trip t and u
T	Set of trips
D	Set of drivers
k	Number of drivers
n	Number of trips
p	Number of daily duties
q	Number of weekly duties
s	Schedule, sequence of weekly duties $s = (w_1, \ldots, w_q)$
w	Weekly duty, sequence of daily duties $w = (d_1, \ldots, d_p)$
$\lambda^{drive}_{breakless}$	Maximum breakless driving time before a break
λ^{drive}_{daily}	Maximum daily driving time before a daily resting period
λ^{duty}_{daily}	Maximum daily duty time before a daily resting period
λ^{drive}_{weekly}	Maximum weekly driving time before a weekly resting period
$\tau^{rest}_{breakless}$	Time required to reset the breakless driving time
τ^{rest}_{daily}	Time required to reset the daily driving time
τ^{rest}_{weekly}	Time required to reset the weekly driving time
σ	Solution, set of schedules $s_1, \ldots, s_k$

Chapter 5: Hospital planning

Notation	Description
R	Number of rooms
W	Number of wards
D	Number of days
C	Number of patient categories (or procedure types)
H_c	Required hospitalization time for category c at a ward
P_c	Required procedure time for category c at a cath lab
Z_{cd}	# (semi-)urgent patients of category c on day d
x_{cdrw}	# patients of category c scheduled on day d at room r and ward w
τ_{dr}	Available time for procedures on day d at room r
υ_{dr}	Expected overtime on day d at room r
β_{dw}	Available number of beds on day d at ward w
λ_{dw}	Expected overload on day d at ward w
$T_{r,\max}$	Upper bound on the available time for procedures on at room r
$\beta_{w,\max}$	Upper bound on the capacity of ward w
δ_d	Set of patient categories that can be treated on day d
ρ_r	Set of patient categories that can be treated at room r
ω_w	Set of patient categories that can be hospitalized at ward w
$V_{\tau,r}$	Costs per opened hour of room r (including staffing costs)
$V_{\upsilon,r}$	Costs per hour overtime at room r
$V_{\beta,w}$	Costs per reserved bed at ward w (including staffing costs)
$V_{\lambda,w}$	Costs per overloaded bed at ward w
R_c	Profit of treating a patient of category c
Π	Objective function, $\pi_\tau(\tau) + \pi_\upsilon(\upsilon) + \pi_\beta(\beta) + \pi_\lambda(\lambda) - \pi_P(x)$
$\pi_P(x)$	Expected profit obtained from all elective procedures
$\pi_\tau(\tau)$	Expected costs due to the opening time of the rooms
$\pi_\upsilon(\upsilon)$	Expected costs from overtime at rooms
$\pi_\beta(\beta)$	Expected costs due to the available beds at the wards
$\pi_\lambda(\lambda)$	Expected costs from overloaded wards

Chapter 6: Train Timetable Generation

Notation	Description
a	Index of a constraint $a = (i, j) \in A$
A	Set of pairs of events for which a constraint exists, $A \subseteq [n] \times [n]$
B_{xy}	Bridging constraints between components C_x and C_y
C	Component of the PESP
D	Directed graph $D = (V, A)$
i	Index of an event $i \in V$
L_{ij}	Lower bound on the scheduled time difference of events (i, j)
m	Number of arcs / constraints, $m = \|A\|$
n	Number of vertices / variables / events, $n = \|V\|$
T	Length of the cyclic framework
U_{ij}	Upper bound on the scheduled time difference of events $a = (i, j)$
Δ_{ij}	Feasible interval between variables i and j

Appendix B

Overview of used problems

An overview of formal definitions of all used combinatorial optimization problems within this thesis is given here. Section B.1 lists the main five scheduling problems that form the core of this thesis, ordered in appearance (i.e., per chapter). Section B.2 lists all other combinatorial optimization problems, that are used for reductions, subroutines, etc. For the meaning of the used notation for the main scheduling problems is referred to the corresponding chapters and/or Appendix A.

B.1 Main scheduling problems

Budgeted Job Coverage Problem (BJCP)

Given: A family of subsets $F = \{S_1, \ldots, S_m\}$ over a set of jobs $J = [n]$ with cost $c_j > 0$ and weight $w_j > 0$ for each $j \in J$, and a budget B.

Goal: Find a collection of subsets $\sigma \subseteq F$ that maximizes $\sum_{j \in J(\sigma)} w_j$ such that $\sum_{j \in J(\sigma)} c_j \leq B$.

European Crew Scheduling Problem (ECSP)

Given: A set of trips $T = [n]$ with corresponding starting and ending times, $0 \leq S_t \leq E_t$, a traveling time matrix M and a number of drivers k.

Goal: Find a feasible solution σ that minimizes $Y(\sigma)$.

Room and Ward Planning Problem (RWPP)

Given: A number of rooms R with maximum opening time $T_{r,\max}$ for every $r \in [R]$, wards W with maximum capacity β_w for every $w \in [W]$, patient categories C and days D, vectors of random variables H (hospitalization time at ward) and P (procedure time at room), expected urgent patient matrix Z, compatibility vectors δ (days), ρ (rooms) and ω (wards), a cost vector V and patient profit vector R.

Goal: Find an assignment of patients to rooms, wards and days that minimizes $\pi(x, \tau, \upsilon, \beta, \lambda)$.

PERIODIC EVENT SCHEDULING PROBLEM (PESP)

Given: A directed graph $D = (V, A)$, a time window $[L_{ij}, U_{ij}]$ for every $(i, j) \in A$ with $L_{ij}, U_{ij} \in \mathbb{R}$ and a cycle time T.

Goal: Find a $v \in [0, T)^n$ such that $(v_j - v_i)_T \in [L_{ij}, U_{ij}]$ for every $(i, j) \in A$, or state infeasibility.

B.2 Used combinatorial optimization problems

3-PARTITION

Given: A set of $3n$ integers, $a_1, \ldots, a_{3n}$.

Goal: Determine whether there exist n disjoint subsets $S_1, \ldots, S_n \subset \{1, \ldots, 3n\}$ such that $\sum_{j \in S_i} a_j = \frac{\sum_{j=1}^{3n} a_j}{n}$ and $|S_i| = 3$ for $i = 1, \ldots, n$.

ASSIGNMENT PROBLEM

Given: A set of m people $P = \{1, \ldots, m\}$, a set of n jobs $J = \{1, \ldots, n\}$ and a cost c_{ij} for every person $i \in P$ to execute any job $j \in J$.

Goal: Find an assignment $f : P \to J$ such that $\cup_{i \in P} f(i) = J$ that minimizes $\sum_{i \in P} c_{i,f(i)}$.

BIN PACKING PROBLEM

Given: A bin capacity $B > 0$, a set of n items with size $0 < a_i \leq B$ for $i \in [n]$ and a maximum number of bins m.

Goal: Minimize the number of required bins K for which an assignment $f : [n] \to [K]$ exists such that $\sum_{i:f(i)=j} a_i \leq B$ for all $j \in [K]$.

KNAPSACK PROBLEM

Given: A set of n items with corresponding weights $w_i > 0$ and profits $p_i > 0$ for $i \in [n]$ and a knapsack capacity B.

Goal: Find a subset of items $\sigma \subseteq [n]$ that maximizes $\sum_{i \in \sigma} p_i$ such that $\sum_{i \in \sigma} w_i \leq B$.

PARTITION PROBLEM

Given: A set of integers $a_1, \ldots, a_m$.

Goal: Determine whether there exists a subset $\sigma \subseteq [m]$ such that $\sum_{i \in \sigma} a_i = \sum_{i \notin \sigma} a_i$.

PATH COVER PROBLEM

Given: A directed graph $D = (N, A)$ and a parameter k.

Goal: Determine whether there exists a path cover containing at most k paths.

SET COVER PROBLEM

Given: A family of sets $F = \{S_1, \ldots, S_m\}$ over a domain of elements X.

Goal: Find a subcollection $\sigma \subseteq F$ of minimum cardinality such that its union equals X.

TRAVELING SALESMAN PROBLEM

Given: An undirected graph $G = (V, E)$ and a distance $d_e > 0$ for every $e \in E$.

Goal: Minimize the total length of a tour that visits every vertex in G exactly once while starting and ending in the same vertex.

WEIGHT-BOUNDED PATH COVER PROBLEM

Given: A directed graph $D = (N, A)$, a weight bound W and a parameter k.

Goal: Determine whether there exists a path cover containing at most k paths, such that every path in the path cover has weight $\leq W$.