Delft University of Technology

Master Thesis

MSc Applied Mathematics

Department of Electrical Engineering, Mathematics and Computer Science

People Scheduling Service: an Employee Scheduling Algorithm based on Stochastic Workloads

I.R. (Ian) Luik, MSc
Student number: 4655893

Thesis Committee
Dr. D. de Laat (Supervisor)
Dr. ir. L.J.J. Van Iersel (Chair)
Dr. J.L.A. Dubbeldam

Company Supervisor
F. Kulik, MSc
Picnic Technologies

Date final version
June 18, 2023

Abstract

In this thesis, a model and solution approach is proposed for the employee scheduling problem in a very flexible and general setting, encountered in practice at the online supermarket Picnic Technologies. It is called the People Scheduling Service (PSS). The purpose is to make a schedule that minimizes expected costs, where the costs considered are (1) lost revenue in case workload cannot be completed; (2) excess personnel costs in case of overcapacity; (3) employee satisfaction costs incurred when a schedule is unfavorable for an employee,

People are assigned to shifts, based on a workload model called "workload packages", which work as follows: at some known release time, an amount of workload (in the form of small, independently executable tasks) is made available to employees. Employees can work on the workload until some known deadline, after which any uncompleted workload is lost. The release time and deadline time do not have to correspond to the start and end time of a shift, adding flexibility to the workload planning system. On top of that, it is not assumed that perfect information is available: the amount of workload in each package is modeled as a random variable. The model, therefore, minimizes the *expected costs*.

On the scheduling side, very few restrictive assumptions are made. Shifts may overlap and they may vary in length. Employees may have full-time or part-time contracts and even contracts where the number of hours is not fixed. The model is formulated in a manner that makes it very easy to change the scheduling constraints that are applied to a given scheduling instance. This is necessary to support operations in an international environment, where the applicable laws can differ between jurisdictions. All in all, PSS is, compared to many other scheduling solutions considered in other papers, very versatile and widely applicable.

The contributions of this paper are: (1) a mixed integer linear program (MILP) formulation of the scheduling problem explained above, which is much more general and practically applicable than many other papers in the field; (2) a comparison between two solution approaches: direct application of an off-the-shelf MILP-solver and the L-shaped algorithm, which is a specific implementation of Benders' decomposition for two-stage stochastic optimization problems; (3) a comparison between scheduling with and without workload uncertainty taken into account, resulting in a clear practical use case for stochastic optimization; (4) a comparison between the performance of SCIP (an open-source MILP-solver) and Gurobi (a commercial MILP-solver) when applied to the scheduling problem considered in this thesis.

PSS has been implemented and taken into production in practice. Practical experience shows that the proposed model works well and saves a lot of manual work. The experiments in this thesis show firstly the direct MILP-solver approach is much faster than the implementation of the L-shaped algorithm proposed in this thesis when applied to most real-world test instances (on very large instances in combination with SCIP as the solver, the L-shaped algorithm is faster). Secondly, there is real value in taking uncertainty into account: schedules created without uncertainty in the model had expected costs of up to approximately 60% higher compared to the ones created with uncertainty in the model. Thirdly, Gurobi was a lot faster than SCIP. When using a direct MILP-solver approach, Gurobi never took more than five minutes to reach an optimality gap of at most 1% on any of the test instances. SCIP, on the other hand, could not reach a 1% gap in two hours for the larger test cases. In two cases, it did not even find a feasible solution at all.

Further research can be done to extend the model to contain a more general workload model (for example, one where the workload can both be lost or put on a back-log when the deadline is reached), a better uncertainty model modeling more uncertainty in workload more accurately and taking other sources of uncertainty into account as well and doing more practical testing.

Contents

Foreword

This thesis project has been conducted on behalf of Picnic Technologies. This online supermarket went live in 2015 and is active in the Netherlands, Germany and France at the time of writing (2023). The goal has been to write a versatile employee scheduling algorithm that generates a cost-minimizing and retention-maximizing assignment of people to shifts in various settings: the algorithm should work in all countries and all different operational teams of Picnic. Furthermore, it had to allow for a very particular workload structure called "workload packages", enabling the company to move a part of the workload to shifts with the most capacity. Finally, any undercapacity or overcapacity should be well-balanced over different shifts to get a schedule that is robust against workload forecasting errors.

Those goals have been accomplished. At the time of writing, the algorithm is live internationally in several teams: all shoppers in fulfillment centers in Germany and France, and all customer success agents in the Netherlands are scheduled using the algorithm presented in this thesis. This amounts to thousands of people per week. Consequently, planners now need much less time to make schedules for these teams, as the previous solutions for scheduling at Picnic still required many of manual adjustments. In turn, those time savings have made it possible to centralize all people scheduling operations in one team, which has all of the scheduling know-how necessary to maintain and innovate the scheduling solutions within Picnic.

The project has helped me develop a great number of skills outside of just developing a mathematical solution to a real-world problem. On the tech side, I have built the Java service around the scheduling algorithm, called the People Scheduling Service, which will interface with the other relevant Picnic systems. The development of this service required me to learn a lot about modern Java frameworks and programming practices. All code was subject to review from developers, which has helped me become a better programmer. On the business side, I have taken on a substantial part of the alignment with the other business teams and the tech team within Picnic to ensure that the required data is available and to help the business processes get ready for full automation.

I would like to thank Picnic Technologies for trusting me with the challenge of designing, developing, testing, and implementing this algorithm. Dozens of people from different countries and teams have, directly or indirectly, contributed to the success of this project, so any attempt to list everyone would inevitably fall short. Thanks to everyone who contributed! Nonetheless, there are two people whose contribution I would like to highlight.

Firstly, Friederike Kulik has played a major role in the success of this project and my development in the nine months I have worked with her. She trusted and encouraged me to take on the challenge of automating people scheduling, and has been nothing but supportive throughout the process. Furthermore, she was willing to invest considerable time in my personal development goals and adjust our way of working throughout the process to enable maximal growth. Thank you!

Secondly, thanks are due to David de Laat for reviewing and guiding the thesis from a scientific perspective. His input greatly improved the readability and understandability of the document. Moreover, he gave useful guidance on what to include in the thesis to make it interesting from a scientific perspective.

Ian Luik
Amsterdam, 26 May 2023

Overview of Common Notation and Conventions

Table 0.1 explains notation and conventions used in this thesis which may not be considered standard by the reader or may be ambiguous if left undefined.

Table 0.1: *A list of the most common notations and conventions used in this thesis.*

Notation	**Meaning**
$\mathbb{N}$	The natural numbers, starting at one. So $\mathbb{N} = \{1, 2, \ldots\}$
$\mathbb{Z}_+$	The non-negative integers, so $\mathbb{Z}_+ = \mathbb{N} \cup \{0\}$
$\mathbb{R}_+$	The non-negative real numbers $[0, \infty)$
$[n]$	$\{1, 2, \ldots, n\}$, assuming $n \in \mathbb{N}$
$\lvert X \rvert$	The cardinality of set X
$\mu(X)$	The Lebesgue measure of set X

1 Introduction

The aim of this thesis is to propose an employee scheduling algorithm for Picnic Technologies (or "Picnic", for short). Picnic is an app-only online supermarket that delivers customers' groceries to their homes. The company was founded in 2015 in the Netherlands. At the time of writing, Picnic is a company with thousands of operational employees, active in the Netherlands, Germany, and France.

All of these operational employees work in shifts, meaning they should receive a schedule. For example, "Shoppers" pick groceries from storage locations and place them into crates which are sent out to customers. "Runners" use electric vehicles to deliver these groceries from hubs to the customers. In contrast to some office jobs, where people may be able to determine their own working hours, the operational types of work described above are subject to a variety of time and equipment availability constraints. As a consequence, these employees must be assigned to shifts in order for operations to run smoothly.

In this chapter, we introduce the reader to the general problem of employee scheduling, the operations of Picnic, and the current state of scheduling at Picnic, with the purpose of explaining the relevance of the problem covered in this thesis and covering the background information required to understand the formal problem description presented in chapter 2.

1.1 Introduction to employee scheduling

The main decision considered by the General Employee Scheduling problem (defined for example in Kletzander and Musliu, 2020) is the assignment of working times to employees. The goal is to do this in a "legal" and "optimal" way. Reformulated in the language of mathematical optimization, one must create a roster that is feasible to a set of constraints and optimal with respect to some objective function. Both the set of constraints and the objective function are highly context-dependent. Following Kletzander and Musliu (2020), this section provides a brief overview of the main considerations which are relevant to employee scheduling (in no particular order of importance). These considerations demonstrate the complexity of the employee scheduling algorithm and therefore illustrate the relevance of automated scheduling solutions.

Compliance with labor laws and contractual agreements An important set of "hard" constraints, i.e., constraints that may not be violated in any situation, is induced by labor laws and contractual agreements. Labor laws are typically formulated by the legislatures which govern the territories in which the employees work. They apply usually to all people working in that territory and serve to protect the rights and interests of workers. For some sectors or jobs, the legislature may formulate additional labor laws. Contracts contain further, more specific agreements between the employer and employee.

Compliance with labor laws and contractual agreements can be enforced by government agencies, the judiciary, and labor unions. Typically, these parties aim to make the costs of violations so high that such violations are unacceptably detrimental to the interest of the employer. In addition, violations of laws or contracts can cause employee dissatisfaction and negative publicity. Therefore, labor laws and contractual agreements are typically considered hard constraints.

Alignment between planned hours and expected workload A major consideration for any employer is to align the roster with the expected workload as closely as possible. For example, an e-store may have 1200 hours of order-picking work available on a given day. Overplanning, i.e., scheduling more than 1200 man-hours would lead to wasted hours (which typically still have to be paid). Underplanning can lead to delays or lost sales. Especially in industries where

employee costs are a high part of the operational costs or in industries where profit margins are tight, minimizing the costs associated with under and overplanning is a high priority.

Operational considerations Further considerations for employee scheduling follow from operational requirements. Employee capabilities are an example of such considerations. The total workload in a shift can consist of a variety of different tasks, some of which may require special training or even certificates. Left unconsidered, such requirements may cause a situation where enough people are scheduled, but not enough people with the proper qualifications are scheduled to execute the workload.

Moreover, buildings have limited space. This may limit the number of people that can work in that building simultaneously.

Employee satisfaction Finally, employee satisfaction factors are important to take into account. High employee satisfaction leads to higher retention rates, which translates into (i) less recruitment work and (ii) a more experienced pool of workers. Examples of decisions improving employee satisfaction are: assigning employees to shifts that they prefer, honoring time-off requests, and assigning tasks that interest them.

The list of examples of scheduling considerations given above is by no means exhaustive for the general scheduling problem. An exact specification of constraints and objectives for the version of the scheduling problem considered in this thesis is given in chapter 2. However, the examples listed above already illustrate that scheduling can be a time-intensive and difficult process for humans, leading to the need for automated scheduling solutions which can solve the scheduling problem with better outcomes and lower outcomes than human schedulers.

1.2 An overview of Picnic's operations

This thesis focuses on employee scheduling for Picnic's operations (although we expect the results to be applicable or easily generalizable to other settings). Therefore, we use this section to briefly introduce the most important components of these operations so that the relevance of requirements introduced in chapter 2 may be properly understood by a reader unfamiliar with Picnic.

The operations of Picnic may be split into two categories: "supply chain" and "customer success". The first one is by far the largest and most complex in terms of employee scheduling.

The supply chain of Picnic can be described by following the goods from the end customer all the way back to the supplier. Grocery orders are delivered to the customer by so-called *runners* from *hubs*. The hub is supplied by a large truck carrying the crates containing the customers' orders. The crates are loaded into small electric vehicles, which the runners use to deliver the groceries.

The orders are picked in *fulfillment centers*. Here, so-called *shoppers* pick groceries from storage locations (such as shelves) containing usually a small amount of inventory. They place the groceries in the crates that go out to customers. In other words: before picking, items are grouped by the article (e.g. "Picnic 1.5L biological milk"), and after picking, the items are grouped by customer (e.g. the Tuesday-morning order of Jan de Wit). Many auxiliary processes (such as shelf replenishment, clearing out-of-date products, and storage location reassignments) take place to facilitate efficient picking.

The inventory in the fulfillment center is delivered in different ways, depending on the article. Some articles (mostly fresh articles) are supplied from the supplier directly, but many articles are supplied from a *distribution center (DC)* operated by Picnic. Here, the goods are received

from the supplier in bulk and stored in larger quantities. In the DC, products are stored by article and regrouped into carts or pallets of items headed for a specific fulfillment center.

The three locations above, "hubs", "fulfillment centers" and "distribution centers" are the primary locations on which supply chain operations take place. Next to these teams, there is a "customer success" team that handles customer feedback.

There are thus four different main types of operations for which employees must be scheduled, each with overlapping, but also slightly different needs. Moreover, at the time of writing, Picnic is active in three different countries, which have different labor laws. It follows that now and in the future, there will be at least twelve different settings in which the employee scheduling problem must be solved.

1.3 State of employee scheduling at Picnic

Currently, there is no standardized way of scheduling at Picnic. There are not just differences in the scheduling procedures between hubs, fulfillment centers, distribution centers, and customer success, but within each operation, there are also distinct ways of working between countries. The differences between the types of operation stem mostly from different operational requirements and constraints. The distinctions between the way of working between different countries stem mainly from the discrepancies between the labor laws and available contract types between these countries. Moreover, the size of Picnic has led to a split of scheduling responsibilities across different teams, which contributed to the divergence of the scheduling procedures.

In the next subsections, we explain the main scheduling principles in each department of Picnic. After that, the main challenges are summarized.

1.3.1 Employee scheduling in fulfillment

The fulfillment team operates the fulfillment centers and distribution centers. For those locations, most employees' schedules are made weekly, meaning that each week, employees receive their schedules for the next week. Employees in leadership roles are scheduled differently - they are scheduled for two weeks at a time.

The exact moments of scheduling differ per country because of legal reasons, but scheduling happens generally over two different days so any problems in scheduling (such as data errors) can be detected and fixed before the schedules have to be sent out and so workload may be moved between shifts or even fulfillment centers in case there is undercapacity in one and overcapacity in another. The reason for the different timelines in the Netherlands and Germany is due to different legal requirements.

When making the non-leadership schedules for fulfillment, there are two main principles that are used to determine how many people should be scheduled for each shift. Firstly, workload forecasts are mapped to capacity targets per shift. For example, if there are 200 hours of picking to do between 6:00 and 15:30, and if there are two morning shifts (6:00 - 14:30 and 7:00 - 15:30), 100 hours of workload may be mapped to shift one and the other 100 hours of workload may be mapped to shift two. The effective working time in both shifts could be 7.75 hours (excl. breaks), meaning that approximately 13 people would be required in each shift to fulfill the picking work. All other workload is mapped to shifts in a similar way, leading to a total number of people required per shift (the "capacity targets", or "required capacities").

The planning teams then create legal schedules and aim to match the capacity targets as closely as possible. It is often not possible to match them perfectly: if the employee pool size is too small or if availabilities for certain shifts are lacking, there may be underplanning in some shifts. If the pool size is too big or employees' availabilities do not match the workload well, Picnic is still

obliged to assign employees their contracted hours, and therefore there may be overplanning in some shifts. Planners aim to minimize underplanning and overplanning, and they aim to make a "balanced" schedule, meaning they aim to minimize the spread of the relative underplanning and overplanning. Examples of an unbalanced schedule and a well-balanced schedule are shown in Tables 1.1 and 1.2 respectively.

Table 1.1: *The relative overplanning per shift in an unbalanced schedule.*

Planned vs required #people	**Monday**	**Tuesday**	**Wednesday**	**Thursday**	**Friday**	**Saturday**
Morning	10%	2%	15%	7%	8%	0%
Evening	5%	12%	5%	1%	5%	4%

Table 1.2: *The relative overplanning per shift in a well-balanced schedule.*

Planned vs required #people	**Monday**	**Tuesday**	**Wednesday**	**Thursday**	**Friday**	**Saturday**
Morning	4%	5%	4%	6%	4%	5%
Evening	4%	4%	5%	5%	5%	6%

The reason that planners at Picnic try to balance the schedules as well as possible is because of uncertainty in the amount of workload. After all, schedules are made based on workload forecasts, well before customers finish placing their orders. The unbalanced overplanning in Table 1.1 would not be a big problem if the workload turns out exactly as expected (see Figure 1.1). But suppose more customers order at Picnic than expected, say 4% more (evenly distributed across the week). Then, the Tuesday morning, Thursday evening, and Saturday morning shifts are underplanned (see Figure 1.2). As a consequence, Picnic cannot serve all customers or employees are required to work overtime. Both are undesirable.

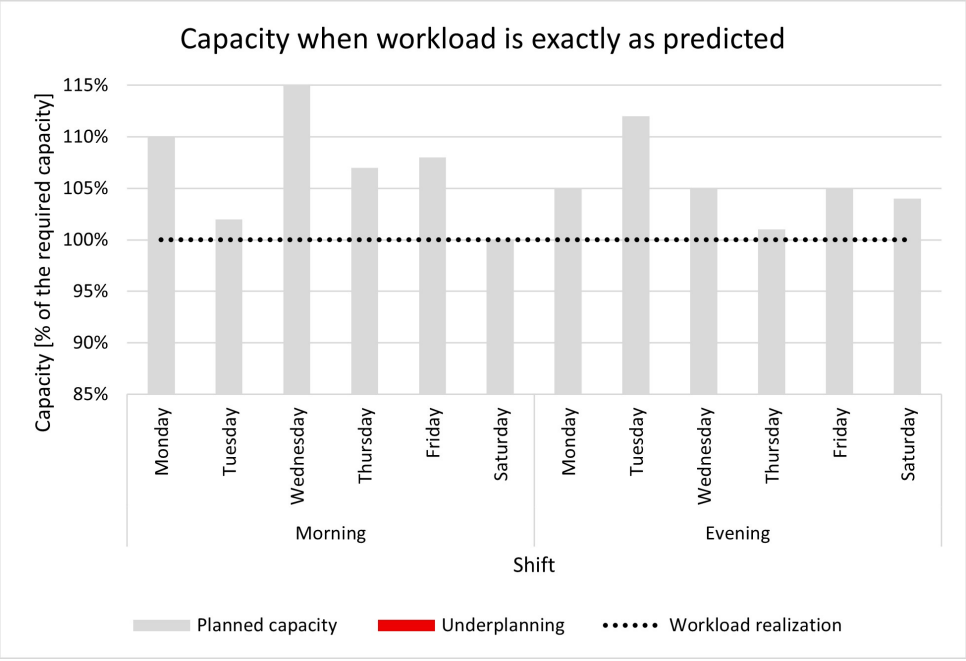


Figure 1.1: *Schedule of Table 1.1 when the workload forecast is correct. In that case, there is no underplanning.*

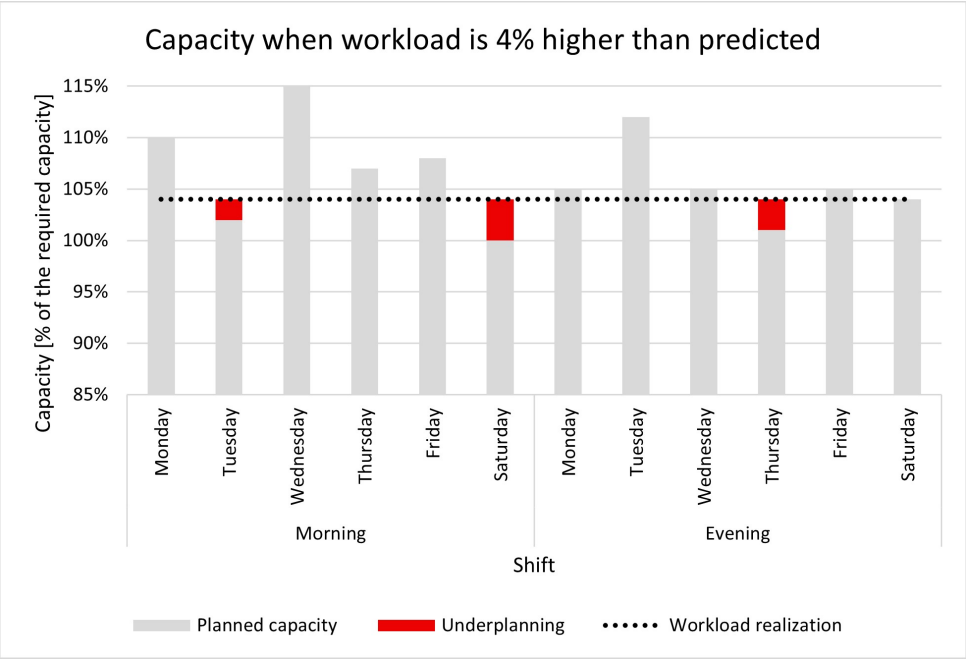


Figure 1.2: *Schedule of Table 1.1 when the workload turns out four percent higher than the forecast. In that scenario, there is underplanning in three shifts.*

In conclusion, planners at Picnic aim to (1) minimize the underplanning and overplanning as much as possible and (2) balance the relative underplanning / overplanning as well as possible.

1.3.2 Employee scheduling in customer success

Employee scheduling in the customer success team works very similarly to scheduling in fulfillment. There are two differences: (1) scheduling in customer success schedules for longer periods, namely 2 weeks; customer success works with up to eight overlapping shifts of different durations per day, whereas fulfillment (in most locations) uses two disjoint shifts of the same length.

1.3.3 Challenges in employee scheduling

Within Picnic, most scheduling procedures are not fully automated or mathematically optimized. The scheduling procedures involve a mix of human work and partial automation. For example, the planning in fulfillment centers in Germany happens using a mixed integer linear program model which is solved to near-optimality (with an off-the-shelf solver). It aims to minimize a weighted sum of relative underplanning in each shift. However, humans are still involved: if employees hand in a list of available shifts for which no legal schedule exists (respecting the contracted number of hours, the minimum rest time between shifts, and the availabilities of the employee), the solver returns "infeasible". Then, human schedulers manually go through the list of employees to find which shoppers have handed in an infeasible set of availabilities, and remove them from the problem. These employees are then scheduled manually, partially outside of their availabilities. The solver is run again for the remaining employees. Usually, manual improvements are made to the schedule, as it is often possible to move people from overplanned shifts to underplanned shifts. This is an indication that the current model does not capture the reality fully.

In the Netherlands, scheduling happens using a heuristic algorithm that constructs a pseudo-feasible schedule and iteratively moves people from the shift with the most overplanning to a

shift with underplanning. The word "pseudo-feasible" refers to the fact that this algorithm does not take into account the Dutch law which stipulates a minimum uninterrupted rest time of 36 hours in each window of seven consecutive days. Any violations of this rule are repaired manually.

Even though the current scheduling procedures are certainly good enough to keep Picnic running and delivering to customers, they are unlikely to produce profit maximizing and employee satisfaction maximizing schedules, they are time-intensive (leading to a lack of scalability) and they are hard to maintain. It is for these reasons that there is a wish for a better solution to the scheduling problem.

1.4 Goals and research questions

The goal of this thesis project is to build a scheduling algorithm that can accommodate the needs of all different operations in all the countries in which Picnic is active. Specifically, the most important goals of the new scheduling system are given below.

1. The scheduling system must work as a "platform"; it should be configurable to accommodate the different operational demands and legal requirements of all teams in all countries within Picnic.
2. The system should be able to optimize for maximal profit and employee satisfaction.
3. The system should be able to make schedules based on "workload packages" (i.e., hours of workload that are released at some point in time and have a deadline at a later point in time) that do not necessarily coincide with the start and end time of shifts.

The goals described above lead to the main research question of this thesis: how should Picnic automate the employee scheduling process? The subquestions are as follows.

1. How should the scheduling problem be modeled?
2. How should the most important future uncertainties (such as the future workload) be taken into account in the model?
3. Which algorithm should be used to solve the model to near-optimality (a relative optimality gap of at most 1%) in two hours on the largest real-world Picnic instances?

The primary motive for this thesis project is to automate scheduling for fulfillment centers, however, it would be greatly beneficial for Picnic to have a more general platform that can accommodate the scheduling needs of all different teams in all the countries in which Picnic is operational - now and in the future.

We, therefore, attempt to model and solve the employee scheduling problem in the most general setting which can reasonably be expected to be useful for Picnic, while at the same time exploiting the limitations on the scope that the focus on the restriction to Picnic's operation places on the scheduling problem to gain the best algorithmic speed and solution quality possible.

1.5 Results and business impact

The model proposed in this thesis meets the goals presented in the previous section, and it has already been taken into production under the name "People Scheduling Service" (PSS). PSS is used operationally at the time of writing for the fulfillment teams in Germany and France, and the Customer Success team in the Netherlands. This amounts to thousands of people per week being scheduled using PSS.

The model proposed in this thesis and implemented in PSS is a two-stage stochastic mixed integer linear program, where the first stage consists of all scheduling constraints (legal requirements,

employee satisfaction factors, and operational limitations are included here), and the second stage evaluates and minimizes the cost of the schedule.

The first goal was that the scheduling system should be designed as a platform, so that it may be used across different countries and different teams. This objective was accomplished by making the first-stage problem completely configurable: the scheduling service contains a list of abstract constraints, which can be parameterized using a location-specific configuration file. For example, there is a "maximum working time" constraint, which ensures that no employee works more than some specified number of hours are worked in some specified period. The parameters can be configured per location so that PSS can be used in the Netherlands, Germany, and France without any country-specific code. Only scheduling people on locations from the distribution team (responsible for the last-mile delivery of groceries) cannot be done with the code base that exists at the time of writing, but Appendix C contains a proposal to extend the model to enable this.

The second goal was to optimize for maximal profit and employee satisfaction. The most important factors for employee satisfaction are supported by some of the first-stage constraints specified in Chapter 2. The profit maximization is handled by the second stage problem. This problem models, given some legal schedule, the total amount of work that can be completed from each workload package and computes the total cost related to lost workload and overplanning. It minimizes those costs, which is equivalent to maximizing profit. The model of the People Scheduling Service even optimizes the expected profit under uncertainty in the workload.

The third goal was to schedule based on workload packages, whose release and deadline times do not necessarily correspond to the start and end times of shifts. The scheduling model presented in this thesis allows for this. During the project, it was discovered that this model does not work well in the last-mile delivery context, but Appendix C contains a proposal for an extension of the model to accommodate this setting as well.

In this thesis, we compare the run times and objective values corresponding to a simple MILP-solver solution approach to those corresponding to a more advanced approach: the L-shaped algorithm. In practice, the MILP-solver (even the open source solver tested in Chapter 6) is fast enough for practical use, and usually, it is faster than our implementation of the L-shaped algorithm. Because of the speed and simplicity of the MILP-solver, the scheduling model is solved using a simple MILP-solver in the operational version of PSS.

Even though the specific implementation of the L-shaped algorithm will not be used operationally within Picnic, its exploration is still scientifically valuable: firstly, in certain situations (large problem instances in combination with certain MILP-solvers), the L-shaped algorithm is faster and could be the preferable option; secondly, because the experiments expose vulnerabilities in the direct application of the L-shaped algorithm to a mixed integer linear program which other researchers should be aware of when considering the use of the L-shaped algorithm and thirdly because different implementations of the L-shaped algorithm may have a better performance and may be worth looking further into.

Within the nine months of this project, the requirements for the People Scheduling Service have been compiled, and a solution has been developed, programmed, tested, and deployed. It required substantial effort, beyond just designing and programming a model, to create a solution that could be rolled out operationally. It required converging business processes that used to be distinct across countries and teams, making lots of data available in a clean and standardized format, and building interfaces to different systems that must interact with PSS.

The business impact of all of this work is substantial. Most directly, the manual planning time per location has been reduced substantially, allowing a small planning team to plan more locations in a day. As a consequence, it was possible to start centralizing the scheduling operations in a

single team that has the experience and know-how to schedule efficiently, improve tooling and streamline operational procedures. Furthermore, it is now possible to grow the company without requiring a lot of extra planners. Finally, since PSS has some advanced features, such as planning based on workload packages, it paves the way for future initiatives that improve efficiency: the scheduling service is ready for it.

The thesis outline is as follows. Chapter 2 contains a formal description of the problem, with all the requirements that must be met. In Chapter 3, we present a survey of the available literature to get a sense of the state of the art and to identify literature gaps that can be filled by this thesis. Chapters 4 and 5 describe respectively the mathematical model and the algorithm used to solve the model to (near-)optimality in a reasonable time. In chapter 6, the real-world performance of the algorithm presented in Chapter 5 is evaluated. Finally, Chapter 7 gives conclusions, recommendations, and directions for future research.

2 Problem Description and Requirements

In this section, the version of the employee scheduling problem studied in this thesis is specified. To this end, we first introduce some definitions which will be used to specify the problem considered in this thesis. Then, we specify what it means for a schedule to be "allowed" and how we quantify the desirability of a schedule.

2.1 Definitions

The goal of employee scheduling is to assign employees to shifts for a given planning period. Therefore, we first define the terms "planning period" and "shift".

Definition 2.1 (Planning period)**.** Let $\mathcal{T} = \mathbb{R}$ represent the axis of time. We call a subset $[0, T) \subset \mathcal{T}$ with $T > 0$ a *planning period*.

Remark 2.2. We shall throughout this text use the datetime format yymmdd-hh:mm (e.g. 221004-15:30) or other intuitive notation to represent time. With some abuse of notation, we may write a planning period (or other time intervals) in the form [221003-00:00, 221010-0:00), with the understanding that the boundaries are implicitly mapped to points in $\mathbb{R}$.

Definition 2.3 (Shift)**.** A *shift* s is a subset of time $[t_s^{start}, t_s^{end}) \subset \mathcal{T}$. It has the interpretation of "an interval of time to which an employee can be assigned for work". We say that a shift *lies within* the planning period if $s \subset [0, T)$, a shift *lies outside* of the planning period if $s \cap [0, T) = \emptyset$ and a shift *crosses the boundary of the planning period* in the other cases. We use S to denote a set of shifts and $S^T \subset S$ to represent the subset of shifts that lie within the planning period.

Now that the concepts "planning period" and "shift" have been defined, they can be used to define the notion of "(employee) schedules".

Definition 2.4 (Employee schedule)**.** Let S be a set of shifts and define S^T as in definition 2.3. An *employee schedule* is a set of pairwise disjoint shifts in S^T. We denote the set of all employee schedules given a shift set S and a planning period $[0, T)$ by $R(S^T)$, or R for short.

Example 2.5. Suppose $S = \{s_1, s_2, s_3\}$ with $T = [221004\text{-}07{:}00, 221005\text{-}07{:}00)$ and

- $s_1 = [221004\text{-}06{:}30,\ 221004\text{-}15{:}00)$;
- $s_2 = [221004\text{-}07{:}30,\ 221004\text{-}16{:}00)$;
- $s_3 = [221005\text{-}06{:}30,\ 221005\text{-}15{:}00)$.

Then $S^T = \{s_1, s_2\}$ and $R(S^T) = \{\emptyset, \{s_1\}, \{s_2\}\}$. Not that $\{s_1, s_2\} \notin R(S^T)$, since $s_1 \cap s^2 \neq \emptyset$.

Remark 2.6. Note that the empty set is also an employee schedule. This is a logical choice, since employees can, for example, have vacations or suffer from illness.

Definition 2.7 (Schedule)**.** Given a set of employees E, a planning period $[0, T)$ and a set of shifts S, we define a *schedule* as a mapping from employees to employee schedules, i.e., a function $\mathcal{R} : E \to R(S^T)$.

The most important goal in employee scheduling is to cover the workload that needs to be completed. Within the context of the thesis, we represent the workload by a set of "workload packages", as defined in definition 2.8.

Definition 2.8 (Workload package)**.** A *workload package* is a tuple

$$\begin{bmatrix} t^{\text{available}} \\ t^{\text{deadline}} \\ t^{\text{work}} \\ \text{TYPE} \\ \text{PRIORITY} \end{bmatrix} \tag{1}$$

representing an amount of work to be done. Work becomes available at $t^{\text{available}}$, must be completed at time t^{deadline}, must be completed at t^{work}, and takes t^{workload} units of working time to complete. We shall refer to t^{workload} as the workload. TYPE refers to the type of activity, such as "picking" or "replenishment" and PRIORITY, represented by a natural number, is used to order different workload packages by importance (priority 1 refers to the most important work). The priority determines the cost of not completing a unit of workload.

Example 2.9. Consider the workload package $(221004\text{-}04{:}00, 221004\text{-}12{:}00, 35\text{ h}, \textit{picking}, 1)$, with $t^{\text{available}}$ and t^{deadline} in datetime format yymmdd-hh:mm. This means that there are 35 hours of picking work (a priority 1 task) to be done between 4:00 and 12:00. This workload may for example be divided over five people working from 4:30 and 12:00, with a 30-minute break in between.

Using the framework of workload packages, the basic aim of employee scheduling can be reformulated as "schedule enough qualified people (i.e. people who can do the work "type" specified in the workload package) to each shift such that the workload of each workload package is completed before its respective deadline." In practice, this may not always be possible: in that case, we say that there is "underplanning". Moreover, it is sometimes possible to have "overplanning" as well: this is the case if (some of) the people who are scheduled to work do not have any high-priority work to do. Underplanning and overplanning lead to economic losses and are therefore undesirable.

2.2 Scheduling process and scope of the problem

In this section, the place of the scheduling algorithm proposed in this thesis within the broader scheduling process is explained. This, in turn, restricts the scope of the project.

We aim towards a sequential scheduling process, consisting of three steps applied in order. This thesis focuses on the second step: scheduling employees.

Step 1: creating workload packages Based on forecasts (such as the customer demand forecast), workload packages are created with a forecasted workload t^{work} and some additional metrics giving information about the amount of uncertainty in the workload.

Step 2: scheduling employees Employees are scheduled based on the forecasted workload packages and forecasted no-show rates (i.e. the rates at which people do not show up even though they were scheduled, for example, due to sickness). The goal is to schedule employees such that the workload is "covered as well as possible" (this notion is clarified later in this chapter), taking into account the need for people with special training and capabilities as well. However, at this stage, people are not yet assigned to work on specific workload packages.

Step 3: creating task assignments In this stage, the work of all the workload packages available in a shift is scheduled to anonymous task schedules. Here, the number of people

scheduled for a shift (output of step 2) is used to determine how many schedules should be made. Only when the shift starts, the schedules are assigned to employees (since at the start of the shift, it is certain which people have shown up and which people have not).

2.3 Restrictions to scheduling

The concepts of "employee schedules" and "schedules" have been defined. However, given a set of shifts S^T, not all schedules are allowed, due to legal or contractual restrictions for example. In this section, we list the legal and contractual requirements that should be supported.

Table 2.1: *A list of the types of legal and contractual obligations that must be supported by the scheduling algorithm proposed in this thesis.*

Requirement Name	**Requirement**	**Example**
MaxWorkingTime	An employee may, in total, work at most H [time units] over a period of W [time units].	(a) An employee may work at most 60 hours per week. (b) An employee may, on average, work at most 48 hours per week over a period of 16 weeks (so: a total of 768 hours in 16 weeks). (c) An employee may not work more than 10 hours a day.
MinNightlyRestTime	An employee must have a rest time of H [time units] between two consecutive assigned shifts on two different days.	An employee must have at least 11 hours of rest time between shifts.
MinRestTime	Every W [time units], an employee must have at least H [time units] of uninterrupted rest.	Every 7 days, an employee must have an uninterrupted rest period of at least 35 hours.
Availability	An employee may only be scheduled within their available working times. An exception can be made for at most N shifts per W [time units].	An employee is available on Wednesday from 9:00 - 17:00, and scheduling outside of availability is not allowed. The employee may be scheduled on Wednesday from 11:00 - 17:00, but not from 12:00 - 18:00 hours.
MinMaxHours	During a period from t_1 until t_2, the employee works at least (at most) H^{min} (H^{max}) [time units].	During calendar weeks $4n + 1$ to $4n + 4$ ($n = 0, ...12$), the employee must work between 20 and 24 hours per week.
MinMaxShifts	During a period from t_1 until t_2, the employee works at least (at most) N^{min} (N^{max}) shifts.	An employee on a flexible contract has requested to work four shifts in the upcoming week, and may not be assigned more than that.
AssignedShifts	An employee with fixed shifts must be assigned those shifts.	An employee who is promised to be assigned on Monday and Wednesday from 8:00 - 16:30 must be assigned to those shifts.

The requirements in Table 2.1 should be considered building blocks. They are only abstract classes of constraints: dependent on the context of the application, specific instances must be defined. In any given situation, some of the requirements may not apply, and in some cases, several instances of the same requirement may be applicable. The latter case is illustrated in the following example.

Example 2.10. The Dutch labor law stipulates that

- an employee may work no more than 60 hours per working week;
- during any four consecutive working weeks, an employee may work no more than 55 hours per working week;
- during any 16 consecutive working weeks, an employee may work no more than 48 hours per working week,

where a working week starts on Monday at 0:00 and ends on Sunday at 24:00 (all translated from Rijksoverheid, 2022). Observe that in this example, three different instances of MaxWorkingTime apply simultaneously.

2.4 Operational constraints

Next to the legal and contractual restrictions, there are some operational requirements that should be supported. These requirements are divided into hard and soft requirements. Hard requirements must always be satisfied, and violations of soft requirements should be penalized but not prohibited.

Table 2.2: *A list of the types of operational requirements that should be supported.*

Requirement Number	**Requirement**	**Example**
BuildingCapacity	No more than N people may work on a given location at the same time.	No more than 50 people may be assigned to work at the same time.

2.5 Employee Satisfaction Factors

Furthermore, there is a variety of employee satisfaction factors that should be taken into account.

Table 2.3: *A list of the types of employee satisfaction factors that should be supported by the scheduling algorithm proposed in this thesis.*

Requirement Name	**Requirement**	**Example**
MinMaxHourSoft	Employee E gets assigned at least H [time units] per W [time units] (where H is higher than H^{min} in LC5a)	Bert gets assigned at least 10 hours of work per week.
MinMaxShiftSoft	Employee E gets assigned at least N shifts per W [time units] (where N is higher than N^{min} is LC5b)	Ernie gets assigned at least 2 shifts of work per week.
PreferredShift	Employee E can hand in preferred shifts, in which he/she would like to work. Assigning preferred shifts to this employee is incentivized.	Bert prefers to work on the Monday morning shift and the Wednesday evening shift. If Bert is not needed harder on different shifts, he will receive the Monday morning and Wednesday evening shifts in his schedule.
RequestedHourFairDivision	Each employee can request a certain number of hours to work. If employees are assigned fewer hours than they request, these shortages must be divided fairly.	If Elmo and Bert both want to work 24 hours in some given week, a schedule where they both work 16 hours is better than one where Elmo works 24 hours and Bert only 8.

2.6 Objective

The main objective is to minimize the total expected cost associated with under- and overplanning (including under- and overplanning of people with specific skills). In the cost of underplanning, a distinction must be made between underplanning on different priorities of workload packages. Underplanning on high-priority work is worse than underplanning on low-priority work. Moreover, any of the penalties related to soft constraints should be included in the objective as well.

In this thesis, we shall consider two scheduling concepts, based on two different sets of assumptions. The main difference will be in the definition of underplanning. The first scheduling concept is based on workload packages, explained in the next section. The main idea behind the notion of workload packages is that workload in one package can be seen as a big stack of infinitesimal activities which can be executed independently of each other by different people. When this assumption holds and we make good use of it in a scheduling model, it can result in a great amount of flexibility: workload can be processed whenever people are available, as long as the workload is processed before the deadline. The workload that is not processed before the deadline is assumed to be lost. In this context, "Underplanning" is defined exactly as the number of hours of workload not processed before the deadline.

An example where this assumption holds reasonably well is picking orders for a specific truck in a fulfillment center. A truck has a planned departure time, which induces a deadline for the picking work. But the orders that have to be shipped by this truck can be done by several

workers in parallel and may be done well before the departure of the truck. An example where the assumption behind workload packages does not hold is in the last-mile delivery of groceries. Each scheduled employee drives one or more routes with a vehicle. These routes can last several hours. Suppose there are shifts from 9:00 - 11:00, 10:00 - 12:00, and 11:00 - 13:00, where each employee spends the full shift driving one route. An employee scheduled for the 10:00 - 12:00 shift cannot take over work from people in the 9:00 - 11:00 shift or the 11:00 - 13:00 shift. This is the main contrast with workload structured according to workload packages consisting of infinitesimal activities.

To cover the last-mile delivery use case, we include a second scheduling concept in the appendix of this thesis. Here, we plan based on "capacity targets". The capacity target for a shift is simply the number of people required per shift. The exact setting is described in more detail in Appendix C.1.

2.7 Scheduling based on workload packages

In this section, we formalize the assumptions behind scheduling based on workload packages. The first two assumptions formalize the consequences of the intuitive idea that "workload consists of a large stack of small (infinitesimal) activities". The third assumption concerns what happens to the workload that is not processed before the deadline.

Assumption 2.11. For any workload package $(t^{available}, t^{deadline}, t^{work}, type, priority)$, workload t^{work} can be arbitrarily divided among employees who are qualified to perform the type of work specified in the workload package.

Example 2.12. Suppose a workload package has three hours of work, consisting of four indivisible tasks of 45 minutes. Furthermore, suppose there are two employees available for this workload. Then we still assume that two hours of work can be assigned to employee A and one hour of work can be assigned to employee B.

Assumption 2.13. At any moment during a shift, an employee can switch from working on one workload package to working on another workload package.

Furthermore, we make some assumptions about the distribution of breaks. For each shift, employees must take a given number of breaks with a given duration per break. As a consequence, a shift generally has a different gross and net number of hours, and the total workforce available during a shift fluctuates over time. The assumption for the division of people over the breaks is as follows. Each break has a few possible "break slots" during which an employee might be on break. Employees are divided equally over the different break slots. See the example below for the intuition, see Appendix B for a rigorous treatment of the assumptions calculations around break times.

Example 2.14. During a shift from 6:30 - 15:00, employees must have a 30-minute break between 10:00 and 11:00. Breaks may start at 10:00, 10:15, and 10:30. To this shift, 62 employees are assigned. Then 21 employees may have their breaks at 10:00, another 20 start their breaks at 10:15, and the final 21 may have their breaks at 10:30.

Assumption 2.15. Given a workload package, any hours of work that are not completed on time (some number in $[0, t^{\text{work}}]$) are lost.

This assumption immediately gives a natural definition of "underplanning": the total number of hours of workload lost within the planning period is the total underplanning in that period. The corresponding lost profits are the associated costs. Minimizing those costs, along with the costs of overplanning, is a major priority for the scheduling model introduced in this thesis.

3 Literature Review and Mathematical Background

In this chapter, we summarize the results of earlier research regarding employee scheduling. The goal is to gather any existing knowledge that could be relevant to the problem specified in Chapter 2 and to identify gaps in the literature that could be filled by this thesis. Furthermore, in this chapter, we give already some additional results that may be common knowledge to many researchers but are, to the best of the author's knowledge, not explicitly written down in the literature.

Firstly, it should be noted that the employee scheduling problem is well-studied. Just the literature review of Van Den Bergh et al. (2013), which covers papers published between 2004 and July 2012, considers 293 papers in the field of employee scheduling. Furthermore, Van Den Bergh et al. (2013) refers to another literature review by Ernst et al. (2004), which considers approximately 700 papers on employee scheduling published before 2004. From just this information, we can conclude that an exhaustive review of the literature related to personnel scheduling would be time intensive and of limited use: after all, much information synthesis has been done already by others in the research community.

The approach used in this literature review is therefore three-step process. First, we use existing literature reviews to determine in which settings the employee scheduling problem has been considered. Then, we survey the papers referenced in these literature studies which consider the employee scheduling problem in a similar setting to the one sketched in chapter 2. Finally, we include papers that consider the same setting published after the most recent literature review. This process should yield most of the relevant insights that are theoretically available while keeping the scope of the review realistic.

3.1 Segmenting the employee scheduling problem

The first step is to segment the different settings for which the employee scheduling problem has been studied. To this end, we follow Van Den Bergh et al. (2013). They classify the employee scheduling problem among four different dimensions:

1. "personnel characteristics, decision delineation and shift definitions";
2. "constraints, performance measures and flexibility";
3. "solution method and uncertainty incorporation";
4. "application area and applicability of research".

In this section, we summarize the main distinctions made in the first three dimensions and compare them with the setting of this thesis. We skip the application area dimension as we found very few insights of interest to this project in this section of Van Den Bergh et al. (2013).

Personnel characteristics, decision delineation, and shift definitions Under the "personnel characteristics" umbrella, Van Den Bergh et al. (2013) distinguish between full-time and part-time employees, skill requirements for specific tasks, and grouping of employees. The authors state that the majority of the papers surveyed in their paper consider a setting with full-time employees only. However, still, a few dozen of them study the case with a mix of full-time and part-time employees. Skill requirements for specific tasks are also considered by dozens of different papers. "Grouping of employees" refers to scheduling teams of people instead of individual employees. Van Den Bergh et al. (2013) only studies eight papers where the team scheduling setting is considered.

The "decision delineation" category refers to the type of decisions that need to be made. Possible decisions include "the assignment of tasks (e.g. employee A is assigned to job K), groups (e.g.

multiple workstations), shift sequence (e.g. employee A works the night shift on Monday), time (e.g. employee A is busy in time periods 1 - 4)" (Van Den Bergh et al., 2013).

"Shift definitions" refer to the different degrees of flexibility that the shift structure may have: the main distinctions are on this basis of overlap (allowed / not allowed), and shift start times and lengths (flexible / fixed).

In the thesis, we consider a general setting where for each employee, a lower bound and an upper bound on the number of working hours can be specified. This includes full-time and part-time employees and employees with fixed and variable-hour contracts. Furthermore, we also consider the setting where certain workload packages require special skills. The main decision is to produce shift sequences. We wish to abstain from imposing any restrictive assumptions on the shift setup: in particular, we want to allow shifts to overlap and have mutually differing shift lengths.

Constraints, performance measures and flexibility Alternatively, papers can be split up among constraints and objectives. The most important constraint identified by Van Den Bergh et al. (2013) is the coverage constraint. They make the distinction between hard and soft coverage constraints and they distinguish based on allowed / non-allowed under and overplanning. Almost three-quarters of the papers surveyed consider hard coverage constraints: most often, they prohibit underplanning in general or they prohibit underplanning for people with specific skills. Imposing such hard coverage constraints could be problematic when developing an algorithm that is intended to be used operationally.

For academic purposes, studying a problem with hard coverage requirements can be interesting, but in reality, underplanning cannot always be avoided: a company may deal suffer from staff shortages (on specific days, or in general), which may be hard or even undesirable to solve. To robustly cover all workloads on all days, a company may need an unreasonably high pool size of employees, which can be quite costly depending on the applicable labor law, contracts, and the job market. On the other hand, overplanning can also not always be avoided: employees may have a guaranteed minimum number of hours of work, which have to be paid regardless of the available workload. Therefore, within the context of this thesis, we consider underplanning and overplanning performance measures with associated costs that should be minimized instead of quantities that should be limited by hard constraints.

An additional complicating factor in the coverage constraints is the scheduling of breaks, which "...are often omitted from personnel scheduling problems" according to Van Den Bergh et al. (2013). Scheduling breaks in advance can be relevant if the workload is time-dependent, i.e., if a part of the workload becomes available after the start of the shifts or has a deadline before the end of the shift. An example where this is the case is nurse rostering: nurses may have to administer medicine to a patient within a strict time window. On the other hand, if one considers an IT service desk where employees start the day with a stack of tickets that have to be handled before the end of the day, the exact break times may not matter as much. In this thesis, we consider the setting where portions of the workload may have deadlines during the shift, and so we do take break scheduling into account.

In terms of performance measures, many of the papers surveyed in the review by Van Den Bergh et al. (2013) use financial costs. Compared to only optimizing for over- and underplanning in terms of time, it gives the advantage of flexibility. In financial cost metrics, factors such as differences in salary between more and less skilled employees, Sunday bonuses, overtime costs, and travel costs can be taken into account.

The papers also consider many different employee satisfaction and fairness factors, ranging from distributing overtime evenly and limiting the number of undesirable shifts assigned to a person to

guaranteeing sufficient rest time between different shifts. Any of these many employee satisfaction and fairness factors have been modeled in some papers as hard constraints (for example because they may be required by law) or as soft constraints (i.e., using penalties in the objective function). An elaborate comparison of all of these constraint types with the ones relevant to this thesis may not be useful at this point, however, it is clear that for many of the constraints listed in chapter 2, there should be some inspiration for modeling in the literature.

Solution method and uncertainty incorporation Van Den Bergh et al. (2013) have analyzed which solutions were the most important (at least between 2004 and 2012) for solving employee scheduling problems. Moreover, they have categorized the types of uncertainty that have been taken into account.

According to Van Den Bergh et al. (2013), mathematical programming techniques have been very popular for solving scheduling problems. These include (mixed integer) linear programming, goal programming, column generation, branch-and-price, and Lagrangian relaxation techniques. The other major category of solution methods consists of construction and improvement heuristics, such as tabu search and genetic algorithms.

The uncertainty is divided by Van Den Bergh et al. (2013) into three different categories: stochastic demand, which refers to uncertainty in the workload; stochastic arrival, which refers to uncertainty in the time at which the workload becomes available and stochastic capacity, which covers the possibility of employee no-show (e.g. due to sickness). Most papers surveyed by Van Den Bergh et al. (2013) do not take into account any of these factors of uncertainty. From the papers that do take into account uncertainty, most focus on stochastic demand and/or arrival. Taking into account the stochastic no-show rate is done in only five of the papers cited.

In this thesis, we consider a setting in which all of these uncertainties are present in practice. We focus, however, on just the stochastic workload and capacity, since these are the largest sources of uncertainty in the operations for which we intend to optimize scheduling.

In summary, after following the classifications of Van Den Bergh et al. (2013), we can properly characterize the position of this thesis in the broader literature.

- Personnel characteristics: mix of full-time and part-time, fixed and variable hours. Some tasks require special skills.
- Decisions: shift sequencing, break assignment.
- Shift definition: overlapping shifts, pre-defined (but configurable and changeable) start- and end-times.
- Constraints: preventing under and overplanning are soft constraints.
- Performance measures: cost minimization, employee satisfaction.
- Uncertainty: the amount of workload, rates of no-show.

Some of the aspects of our employee scheduling problem are not mentioned in Van Den Bergh et al. (2013), most notably the rolling planning system which makes the scheduling problem a multi-stage stochastic optimization problem.

Using the preceding alignment of the problem considered in this thesis with the available literature, we can select literature that is likely to contain the most important literature.

3.2 State of the art in employee scheduling

The paper that covers, to the best of the author's knowledge, the setting most similar to the one considered in this paper is Restrepo et al. (2017). This paper proposes a two-stage stochastic

programming algorithm for multi-activity scheduling where the first-stage decision is the shift assignment and the second-stage decision is the assignment of activities and breaks within the shifts. They use a context-free grammar system proposed by Côté et al. (2011) to implicitly define valid activity sequences (shifts). These are then translated into linear constraints. A combination of the L-shaped algorithm (a Benders decomposition-based algorithm for two-stage stochastic programming) and column generation is used to find good solutions.

Restrepo et al. (2017) report low average optimality gaps (the worst test case has an average optimality gap of 1.31%, most of the other test cases' gaps were substantially lower) and run times in the order of thousands of seconds. The number of different activity types varied from one to five, and the number of employees varied from 13 to 128. The run time seems to increase in the number of activities and the number of employees, but the authors increased those values simultaneously. They did not vary those parameters independently, so it cannot be determined from their text which of the two factors contribute most to the increase in run time.

Moreover, they make some assumptions that are not necessarily met within the context of this thesis. For example, they work with a discontinuous anonymous version of the scheduling problem, where shifts are not allowed to span from one day to the next (discontinuous) and all employees have the same skills, preferences, and availabilities (anonymous). On the other hand, they schedule on a "higher resolution" than required within our context, as they also construct task schedules within the shifts.

Nevertheless, the paper contains some really interesting ideas that could be applicable within the context of this thesis, such as the combination of an L-shaped algorithm for dealing with the stochasticity in the workload, the column generation algorithm for generating feasible employee schedules, and the context-free grammar framework to capture the rules applying to employee schedules.

Eitzen et al. (2004) also consider a setting that is similar to the one of this thesis but in a different way. They do not solve a two-stage scheduling problem but instead consider a multi-skill workforce planning problem with equity constraints. They assume that each worker has a subset of some set of available skills, which are not necessarily hierarchical (i.e., it assumes different specializations, not different levels of seniority where the senior person has at least all of the skills of a more junior person). Moreover, they assign to each schedule an attractiveness score and put a hard limit on the maximum difference in schedule attractiveness between employees. They do assume that an employee works at one skill level during an entire shift, and assign this skill level in advance.

Eitzen et al. (2004) propose and compare three different solution methods that deal with a large number of feasible schedules per employee. The first is called "column expansion", which is a multi-phase method where (i) optimal employee schedules (which are the columns of this problem) are generated for the problem without skill considerations, i.e., the problem where all the workload is aggregated and each employee can do each task and (ii) the optimal schedule of each employee is expanded into multi-skill schedules, i.e., schedules are generated with the same assigned shifts, where a skill/task is added to each shift. The second technique is called the "reduced column subset method". Here, a predetermined number of random feasible schedules (with skill assignments) is generated per employee. Then, an exact optimization over this reduced subset of schedules is done. The final technique proposed is branch-and-price, with a longest-path problem as the subproblem of the column generation routine.

The authors note that branch-and-price with a time limit of four hours yields in general the best solutions (branch-and-price is also the only method of the three that can reach global optimality). The other methods have no gap with branch-and-price on the uni-skill problem instance, but for multi-skill problems, there are gaps that increase as a function of the number of skills. For

instances with nine skills, some column expansion and reduced column subset yield solutions with more than double the amount of underplanning that the solution returned by branch-and-price. In terms of run-time, column expansion, and reduced column subset are very good for problem instances with a low number of employees. The run-time increases relatively fast in the number of employees, and relatively slow in the number of skills considered. Branch-and-price shows the reverse trend. On test instances with large numbers of employees (80 - 110), the run times of all three algorithms are of the same order of magnitude.

The learnings from the paper of Eitzen et al. (2004) give several important lessons: firstly, it is important to take the skills of employees into account when determining which employees should be scheduled in each shift. If this is not done (which is the case when using the column expansion method), substantial losses can be incurred due to local labor shortages on specific pieces of work requiring special skills. Secondly, even finding locally optimal solutions using integer linear programming has scale limitations, which is to be expected given the NP-hardness of integer linear programming in general (see for example Nemhauser and Wolsey, 2014, Proposition 6.1, for a proof that the binary linear programming feasibility is already NP-complete). Thirdly, it is possible to implement branch-and-price for scheduling in the setting considered in Eitzen et al. (2004) using a longest-path problem in a graph, where nodes represent shifts and edges connect shifts which are allowed to follow each other in an employee schedule. If the problem considered in this thesis could be formulated in a similar way, this may prove beneficial as longest-path problems in directed acyclic graphs can be solved using Dijkstra's algorithm (Eitzen et al., 2004).

3.3 State of the art in multi-stage stochastic optimization

In this section, we summarize the approaches available to solve multi-stage stochastic optimization problems. The reader is assumed to be familiar with the field of optimization and probability theory at the level of undergraduate mathematics courses.

3.3.1 Introduction to multi-stage optimization

The general optimization problem has the following form:

$$\min_{\boldsymbol{x} \in X} f(\boldsymbol{x}). \tag{2}$$

Here, X is the *feasible region* (usually closed, but not necessarily bounded), $\boldsymbol{x} \in X$ the *decision variables*, and f the *objective function*. The problem may also be a maximization problem, but we restrict ourselves to minimization problems since $\max_{\boldsymbol{x} \in X}\{f(\boldsymbol{x})\} = -\min_{\boldsymbol{x} \in X}\{-f(\boldsymbol{x})\}$.

In many optimization problems in practice, decisions are made before all of the information necessary to make the optimal decision is available. Think of the capacitated lot sizing problem. Here, a producer with a limited production capacity must decide on (i) a sequence of periods in which to produce batches of their product and (ii) the sizes of these batches. A batch may have to be produced before the orders are placed by customers, such that the producer can guarantee a low lead time for customers. So what happens if, after production, there is more demand than expected? The producer then has the choice of refusing the order and losing the demand or producing a small extra batch at relatively high costs.

Problems such as the one described above are called multi-stage problems. A decision is made before all the required knowledge to optimize this decision is available, but the decision-maker has the possibility to make additional decisions after more information is known. We say that the decisions are made in multiple *stages*, or that there are one or more possibilities for *recourse*.

Consider a two-stage optimization problem, with first-stage decisions x and second-stage decisions y. The questions may then be: which first-stage decision minimizes the expected total

costs, given the fact that corrections in the second stage (dependent on the first-stage decisions) are possible?

The classical approach, where uncertainty in parameters is not taken into account, is as follows:

$$\min_{\boldsymbol{x}\in X}(f(\boldsymbol{x}) + Q(\boldsymbol{x}, \mathbb{E}_{\boldsymbol{\xi}}(\boldsymbol{\xi}))), \tag{3}$$

where

$$Q(\boldsymbol{x}, \boldsymbol{\xi}) = \min_{\boldsymbol{y}\in Y(\boldsymbol{x},\boldsymbol{\xi})} g(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{\xi}), \tag{4}$$

with $Y(\boldsymbol{x}, \boldsymbol{\xi})$ representing the feasible region and $g(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{\xi})$ the cost function of the second-stage decision, given a first-stage decision and a realization of the random data. In non-stochastic optimization, expected values or other estimators of random data are simply plugged in as the true, known values for those parameters. This is what happens Problem (3) by plugging in $\mathbb{E}_{\boldsymbol{\xi}}(\boldsymbol{\xi}))$ as realization of $\boldsymbol{\xi}$.

However, the two-stage problem can also be formulated as follows:

$$\min_{\boldsymbol{x}\in X}(f(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{\xi}}(Q(\boldsymbol{x}, \boldsymbol{\xi})). \tag{5}$$

The difference is subtle, but in Problem (5), the expectation of the objective value of the second-stage problem is minimized. This allows for optimization under many different scenarios, whose objective values are weighted by the probabilities of those scenarios.

For any optimal solution $\boldsymbol{x}^*$ to Problem (3),

$$f(\boldsymbol{x}^*) + \mathbb{E}_{\boldsymbol{\xi}}(Q(\boldsymbol{x}^*, \boldsymbol{\xi})) \geq \min_{\boldsymbol{x}\in X}(f(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{\xi}}(Q(\boldsymbol{x}, \boldsymbol{\xi}))), \tag{6}$$

where the left-hand side represents the total expected cost of the solution $\boldsymbol{x}^*$. In other words, taking approach (5) is never worse than approach (3). Of course, we hope that approach (5). In Chapter 6 we show that this is the case for the scheduling problem considered in this thesis.

One should note that Equation 6 does not imply that the first-stage decisions resulting from Problem (5) are robustly optimal. There may be a particular value of $\boldsymbol{\xi}$ where the decision-maker would have profited from a different first-stage decision than the one which is optimal on average. In an uncertain world, such risks are inevitable.

Pflug and Pichler (2014) explain that multi-stage stochastic optimization may be generalized even further, in at least the following two ways. Firstly, it is conceivable that decisions are not made in two stages, but in $n \in \mathbb{N}$ stages. Secondly, instead of minimizing the expected objective value, one may minimize some other risk functional $\mathcal{R}$, such as the α-quantile of some probability distribution. Chapter 3 of Pflug and Pichler (2014) is completely dedicated to an exposition of different risk functionals that may be used.

A big part of the research in this area has focused on two-stage stochastic problems (multi-stage stochastic problems also referred to in the literature as *recourse problems*), where the probability distributions underlying the stochastic data have finite support and the goal is to minimize the expected costs (Birge and Louveaux, 2011). The advantage of the finite support assumption is that the problem

$$\min_{\boldsymbol{x}\in X}(f(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{\xi}\in\Xi}(Q(\boldsymbol{x}, \boldsymbol{\xi}))) \tag{7}$$

is then equivalent to

$$\min_{\boldsymbol{x}\in X}(f(\boldsymbol{x}) + \sum_{k=1}^{|\Xi|} p_k Q(\boldsymbol{x}, \boldsymbol{\xi}_k)), \tag{8}$$

where p_k is the probability associated with the outcome $\boldsymbol{\xi}_k$ and $Q(\boldsymbol{x}, \boldsymbol{\xi}_k)$ is the optimal value of the recourse problem given a first-stage decision $\boldsymbol{x} \in X$ and the realization $\boldsymbol{\xi}_k$ of the random variable $\boldsymbol{\xi}$.

Different solution methods for two-stage stochastic optimization problems exist, depending on the exact form of the problem. Some solution approaches, such as an algorithm called the "L-shaped algorithm", can benefit from an additional property that the problem may have: *relatively complete recourse*.

Definition 3.1 (**Relatively complete recourse**)**.** A two-stage stochastic optimization problem, i.e. a problem of the form (5), is said to have *relatively complete recourse* if any $x \in X$ (a feasible first-stage solution) admits a feasible second-stage solution with objective value $\mathbb{E}_{\boldsymbol{\xi} \in \Xi}(Q(\boldsymbol{x}, \boldsymbol{\xi}))$.

Remark 3.2. Relatively complete recourse is a slightly weaker property than *complete recourse*: if a problem has complete recourse, the second stage also has a solution for any infeasible first-stage solution.

In the case of linear programs with continuous variables, there is an algorithm called the "L-shaped algorithm", which is a version of Benders decomposition. It solves Problem (5) to optimality. The L-shaped algorithm is an iterative cut-generation method that builds up an outer linearization of the primal problem (chapter 5, Birge and Louveaux, 2011). That means that increasingly tight relaxations of the full problem are solved (and therefore, in the case of a minimization problem, the intermediate objective values form a non-decreasing sequence). It is proven to converge in a finite number of steps (Birge and Louveaux, 2011).

To explain how it works, we first explain a technique called "row generation". Then, we discuss a decomposition method called Benders' decomposition, which can be combined with row generation to create an iterative solution algorithm. Finally, we describe the L-shaped algorithm, which is a specific application of Benders' decomposition and row generation to two-stage stochastic optimization.

3.3.2 Row generation

In this section, we introduce a technique called *row generation*, or *constraint generation*, loosely based on the explanation by Ben-Ameur and Neto (2006). Consider an optimization problem whose feasible region is (partially or fully) characterized by a finite set of constraints. Without loss of generality, this has the form:

$$\begin{aligned} \min_{x \in X} \quad & f_0(\boldsymbol{x}) \\ & f_i(\boldsymbol{x}) \leq 0 \quad i \in [n]. \end{aligned} \tag{9}$$

Now suppose the number of constraints n is large with respect to the number of variables, such that the number of constraints is the bottleneck determining (and slowing down) the speed of the solution process.

If the number of constraints is very large, it may happen that there exists a substantial subset of "superfluous" constraints, i.e., a subset of constraints that can be removed without changing the optimal solution. Formally, there may exist a smaller problem

$$\begin{aligned} \min_{x \in X} \quad & f_0(\boldsymbol{x}) \\ & f_i(\boldsymbol{x}) \leq 0 \quad i \in I \subset [n], \end{aligned} \tag{10}$$

with $|I| \ll n$, such that there is a solution $\boldsymbol{x}^*$ which is both optimal for Problem (9) and Problem (10).

In this case, row generation may provide a way of speeding up the solution process. The idea is as follows. Initialize a *reduced master problem* (RMP), which is Problem (9) with a small (or even empty) subset of constraints $I_0 \subset [n]$:

$$\begin{aligned} \min_{x \in X} \quad & f_0(\boldsymbol{x}) \\ & f_i(\boldsymbol{x}) \leq 0 \quad i \in I_0 \subset [n], \end{aligned} \tag{11}$$

Then, solve RMP, find one or more constraints in the master problem (MP, Problem (9)) which are violated by the optimal solution to the RMP, add these constraints to RMP, and repeat. As soon as no more violated constraints are found, the optimal solution to RMP is also optimal for Problem (9), so the procedure can be stopped. The problem of finding an MP constraint that is violated by the optimal solution to RMP is referred to in the literature as the *strong separation problem* (Ben-Ameur and Neto, 2006).

Proposition 3.3. The row generation algorithm converges to the optimal solution in a finite number of iterations.

Proof. Problem (9) has a finite number of constraints. Each iteration of the row generation algorithm adds (at least) one constraint. □

The row generation algorithm is explained schematically in 3.1.

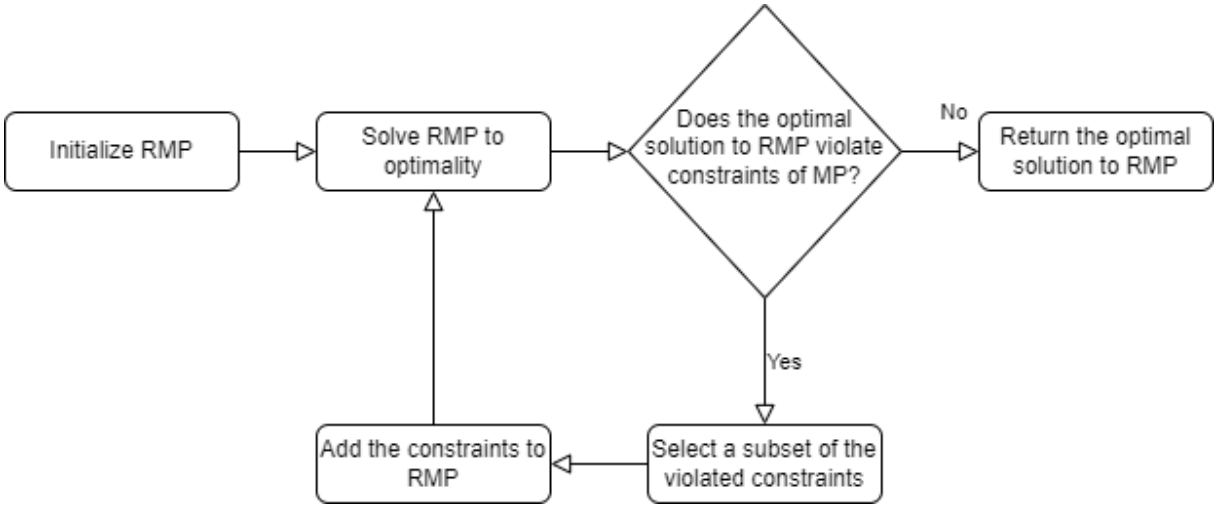


Figure 3.1: *Schematic overview of the Row Generation procedure. RMP is short for "Restricted Master Problem" (Problem* (11)*) and MP is short for "Master Problem" (Problem* (9)*).*

Note that a speedy execution of the row generation algorithm hinges on a few crucial components:

1. a fast routine to identify one or more violated constraints in MP;
2. an efficient way to find an optimal solution to the updated RMP, given a solution to the previous iteration of RMP.

The first problem can be solved in several ways. If the number of constraints in MP is limited (preferably polynomial in some meaningful property of the problem instance), brute-force checking all of the constraints for violations may be a feasible approach. However, since row

generation is typically used for problems with a large (exponential) number of constraints, the trick is often to find the "most violated" constraint by solving

$$\max_{i \in [n]} f_i(\boldsymbol{x}). \tag{12}$$

If the optimal solution to that problem is positive, the optimal solution i^* corresponds to a violated constraint in MP. If Problem (12) admits a polynomial-time special-purpose algorithm or if it can be rewritten as an LP (which can also be solved in polynomial time, Jiang et al. (2020)), it can typically be solved quickly.

Problem (12) can be solved easily in case MP is a linear program. Spliet (2021b) notes that in that case, the optimal basis to some iteration of RMP can be used as a starting basis for the next iteration of RMP: "since the basis remains dually feasible, the dual simplex method can be used" (Spliet, 2021b).

Natural applications of row generation are cutting-plane algorithms for solving mixed integer linear programs and solving problems that have been reformulated using Benders' decomposition.

3.3.3 Benders' Decomposition

In this section, Benders' decomposition (introduced in Benders (1962)) and the corresponding iterative optimization algorithm are derived. The trick is to rewrite an optimization problem in a form that is suitable for row generation. This section closely follows the lecture slides of Spliet, specifically Spliet (2021a) and Spliet (2021b).

The problems of the following form are suitable for Benders' decomposition:

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) + \boldsymbol{q}^T \boldsymbol{y} \\ & \boldsymbol{g}(\boldsymbol{x}) + W\boldsymbol{y} = \boldsymbol{h} \\ & \boldsymbol{x} \in S, \boldsymbol{y} \geq \boldsymbol{0}. \end{aligned} \tag{13}$$

Here, $\boldsymbol{f}$ and $\boldsymbol{g}$ are continuous functions and S is a compact set. Problems of the form (13) have a decision vector consisting of two parts $\boldsymbol{x}$ and $\boldsymbol{y}$, which generally represent different, but linked decisions.

Example 3.4. Suppose an airline has to decide (a) an assignment of airplanes to routes and (b) the amount of fuel to load in each airplane, given a route assignment. Then $\boldsymbol{x}$ could represent decision (a), where $\boldsymbol{x}$ would be a vector of binary variables, and $\boldsymbol{y}$ could represent decision (b). An airline may choose to optimize those simultaneously instead of sequentially, because a smart route assignment (say assigning efficient vehicles to longer routes) may save fuel. This example can also be considered a two-stage decision problem, where uncertainty may be taken into account (weather, for instance, influences fuel consumption).

In general, one can interpret $\boldsymbol{x} \in S$ as the "complicating variables", and $\boldsymbol{y}$ as the "easy variables". This is because S can be more complicated than a polyhedron, and $\boldsymbol{f}$ and $\boldsymbol{g}$ can be much more complicated than linear functions. In this thesis, we will model two-stage stochastic problems as problems of the form (13). Benders' decomposition yields an iterative cutting-plane algorithm to solve Problem (13). Bender's decomposition exploits the fact that, given a solution $\boldsymbol{x}$, Problem (13) is a linear program. This enables using dual information to generate cuts. The full algorithm is derived in this section.

A key step is to find a suitable representation of the feasible region of $\boldsymbol{x}$, i.e., a subset of S such that for all $\boldsymbol{x} \in S$ there exists a $\boldsymbol{y}$ such that $(\boldsymbol{x}, \boldsymbol{y})$ is feasible for the problem above. Observe that the set R, defined as

$$R = \{\boldsymbol{x} \in S \mid \exists \boldsymbol{y} \geq \boldsymbol{0} \text{ such that } W\boldsymbol{y} = \boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x})\}, \tag{14}$$

is precisely this feasible region of $\boldsymbol{x}$.

Proposition 3.5. The set R as defined in Equation 14 is equal to

$$R' = \{\boldsymbol{x} \in S \mid W^T\boldsymbol{u} \leq 0 \implies (\boldsymbol{h} - g(\boldsymbol{x}))^T\boldsymbol{u} \leq 0\}. \tag{15}$$

Proof. Suppose $\boldsymbol{x} \in S$ is given. The condition $\exists \boldsymbol{y} \geq \boldsymbol{0} : W\boldsymbol{y} = \boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x})$ is equivalent to the condition that the following LP must be feasible for a given $\boldsymbol{x} \in S$:

$$\begin{aligned} \min_{\boldsymbol{y} \geq \boldsymbol{0}} \quad & 0 \\ & W\boldsymbol{y} = \boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}). \end{aligned} \tag{16}$$

Note that the feasibility of this problem immediately implies boundedness since the objective function is 0. The dual of this problem is:

$$\begin{aligned} \max_{\boldsymbol{u} \in \mathbb{R}^p} \quad & (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T\boldsymbol{u} \\ & W^T\boldsymbol{u} \leq \boldsymbol{0}, \end{aligned} \tag{17}$$

where $p \in \mathbb{N}$ is the number making the multiplications well defined.

Because linear programs have the property of strong duality, Problem (17)is feasible and has an optimal solution with objective value 0 if Problem (16) is feasible. If Problem (17) is feasible and bounded with objective value 0, we have $(\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T\boldsymbol{u} \leq 0$ for all $\boldsymbol{u} \in \mathbb{R}^p$ satisfying $W^T\boldsymbol{u} \leq \boldsymbol{0}$. This is exactly the condition in the definition of R'.

The other direction of the proof is similar. Suppose $\boldsymbol{x} \in S$ satisfies the condition in the definition of R'. Problem (17) is always feasible ($\boldsymbol{u} = \boldsymbol{0}$ is a solution). From the condition in R', it follows that Problem (17) is also bounded with objective value 0. By strong duality, the primal problem is feasible which means the condition in the definition of R is satisfied.

We conclude that $R = R'$. □

Proposition 3.6. R' as defined in Equation (15) is equal to R'' defined by

$$R'' = \{\boldsymbol{x} \in S \mid (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T\hat{\boldsymbol{v}}^j \leq 0, k \in [n_C]\}, \tag{18}$$

where $\hat{\boldsymbol{v}}^k$, $k \in [n_C]$ are the extreme rays (see Definition A.15) of the cone $C = \{W^T\boldsymbol{u} \leq \boldsymbol{0}\}$.

Proof. Observe that $C = \{W^T\boldsymbol{u} \leq \boldsymbol{0}\}$ is indeed a cone by Proposition A.28. Moreover, by Proposition A.29, C has only the trivial extreme point $\boldsymbol{0}$ and a finite number of extreme rays. By the representation theorem (A.17), any $\boldsymbol{u} \in C$ can be written as

$$u = \sum_{j=1}^{n_C} \lambda_j \hat{\boldsymbol{v}}^j \tag{19}$$

with $\hat{\boldsymbol{v}}^j, j \in [n_C]$ the set of extreme rays of C and $\lambda_j \leq 0$, $j \in [n_C]$. R' can thus equivalently be written as

$$R' = \{\boldsymbol{x} \in S \mid \lambda_j \geq 0, j \in [n_C] \implies (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T \sum_{j=1}^{n_C} \lambda_j \hat{\boldsymbol{v}}^j \leq 0\}. \tag{20}$$

The condition that $(\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{y}))^T \sum_{j=1}^{n_C} \lambda_j \hat{\boldsymbol{v}}^j \leq 0$ for any arbitrary combination of $\lambda \in \mathbb{R}_+^{n_C}$ is met if and only if

$$(\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T \hat{\boldsymbol{v}}^j \leq 0 \quad j \in [n_C]. \tag{21}$$

It follows that $R' = R''$. □

Using the fact that $R = R''$, we can write Problem (13) in the following form:

$$\min_{\boldsymbol{x} \in R''} \{ f(\boldsymbol{x}) + \min_{\boldsymbol{y} \geq \boldsymbol{0}} \{ \boldsymbol{q}^T \boldsymbol{y} \mid W\boldsymbol{y} = \boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}) \} \}. \tag{22}$$

Proposition 3.7. For any $\boldsymbol{x} \in R''$, $\min_{\boldsymbol{y} \geq \boldsymbol{0}} \{ \boldsymbol{q}^T \boldsymbol{y} \mid W\boldsymbol{y} = \boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}) \} = \max_{i \in [n_P]} \{ (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T \hat{\boldsymbol{u}}^i \}$, where $\{ \hat{\boldsymbol{u}}^i, i \in [n_P] \}$ is the set of extreme points of $P = \{ \boldsymbol{u} \mid W^T \boldsymbol{u} \leq \boldsymbol{q} \}$.

Proof. Suppose $\boldsymbol{x} \in R''$ is given. Then $\min_{\boldsymbol{y} \geq \boldsymbol{0}} \{ \boldsymbol{q}^T \boldsymbol{y} \mid W\boldsymbol{y} = \boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}) \}$ is feasible because R was defined as exactly the subset of S for which this problem is feasible. Its dual is

$$\begin{aligned} \max_{\boldsymbol{u} \in \mathbb{R}^p} \quad & (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T \boldsymbol{u} \\ & W^T \boldsymbol{u} \leq \boldsymbol{q}. \end{aligned} \tag{23}$$

In case the primal problem is bounded, the dual problem is feasible and bounded too, where the dual problem has the same optimal objective value. By Theorem A.19, solving this dual problem is equivalent to finding the extreme point of P with the highest objective value.

In case the primal problem is unbounded, the dual problem is infeasible, meaning in this case that $P = \emptyset$. The proposition still holds under the common convention in optimization that $\max_{x \in \emptyset} f(x) = -\infty$. □

Lemma 3.8. Suppose $C = \{ \boldsymbol{u} | W^T \boldsymbol{u} \leq \boldsymbol{0} \}$ and $P = \{ \boldsymbol{u} | W^T \boldsymbol{u} \leq \boldsymbol{q} \}$. Then $\hat{\boldsymbol{v}}$ is an extreme ray of C if and only if $\hat{\boldsymbol{v}}$ is an extreme ray of P.

Proof. From the definition of a ray (Definition A.14), it follows that C is precisely the set of rays of P and of C itself. Since P and C have the same set of rays, they also have the same set of extreme rays. □

Theorem 3.9. Problem (13) is equivalent to the following problem:

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) + \theta \\ & \theta \geq (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T \hat{\boldsymbol{u}}^i \qquad i \in [m_P] \\ & 0 \geq (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T \hat{\boldsymbol{v}}^j \qquad j \in [n_P] \\ & x \in S, \end{aligned} \tag{24}$$

where $\{ \hat{\boldsymbol{u}}^1, \ldots, \hat{\boldsymbol{u}}^{m_P} \}$ is the set of extreme points of $P = \{ \boldsymbol{u} \mid W^T \boldsymbol{u} \leq \boldsymbol{q} \}$ and $\{ \hat{\boldsymbol{v}}^1, \ldots, \hat{\boldsymbol{v}}^{n_P} \}$ is the set of extreme rays of P.

Proof. We have already noted that Problem (13) can be rewritten to Problem (22). By Proposition 3.7, Problem (22) can be rewritten as follows:

$$\min_{\boldsymbol{x} \in R''} \{ f(\boldsymbol{x}) + \min_{\boldsymbol{y} \geq \boldsymbol{0}} \{ \boldsymbol{q}^T \boldsymbol{y} \mid W\boldsymbol{y} = \boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}) \} \} = \min_{\boldsymbol{x} \in R''} \{ f(\boldsymbol{x}) + \max_{k \in [n_P]} \{ (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T \hat{\boldsymbol{u}}^i \} \}, \tag{25}$$

which is equivalent to

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) + \theta \\ & \theta \geq (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T \hat{\boldsymbol{u}}^i \qquad i \in [m_P] \\ & \boldsymbol{x} \in R'' \qquad\qquad\quad j \in [n_P]. \end{aligned} \tag{26}$$

Using the definition of R'' (Equation (18)) and the result of Lemma 3.8, the problem can be rewritten as Problem (24). □

To solve Problem (24), row generation (described in section 3.3.2) can be used. Given a solution $(\boldsymbol{x}, \theta)$ to the restricted master problem, in this context also referred to as the *Benders' master problem (BMP)* (Spliet, 2021b), the separation problem is given by Problem (23). That problem is also referred to in this context as the *Benders' subproblem (BSP)* (Spliet, 2021b).

Row generation applied to Benders' composition can be summarized in the following steps (Spliet, 2021b).

1. Initialize BMP (Problem (24) with some or no constraints i and j).
2. Solve BMP to find $(\boldsymbol{x}^*, \theta)$.
3. Solve BSP (Problem (23)).
4. If BSP is unbounded, get an extreme ray $\hat{\boldsymbol{v}}^j$ corresponding to a direction of unboundedness, add the constraint $0 \geq (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T \hat{\boldsymbol{v}}^j)$ to RMP and go to step 2. Else, go to step 5.
5. If BSP is bounded with extreme point $\hat{\boldsymbol{u}}^k$ and $\theta < (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T \hat{\boldsymbol{u}}^i$, add the constraint $\theta \geq (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T \hat{\boldsymbol{u}}^i$ to RMP and go to step 2. Else, stop.

Remark 3.10. The cuts added in step 4 are called *feasibility cuts*, since they define the feasible region R'', and the cuts in step 5 are called *optimality cuts* because they provide lower bounds on the objective term θ (Spliet, 2021b).

Remark 3.11. Since BMP still depends on quite general functions f and $\boldsymbol{g}$ and a general compact set S, the resulting optimization problem can still be quite hard and slow to solve. The true speed-up comes if one can use some efficient (perhaps special-purpose) algorithm to solve BMP (such as dynamic programming). If BMP is a MILP, efficient (but not polynomial-time) solvers are also available to solve BMP to optimality.

Proposition 3.12. The row-generation-based algorithm for solving Problem (24) terminates in a finite number of steps.

Proof. Polyhedron P has a finite number of extreme points and directions (Proposition A.16). Because of this, the result follows directly from the fact that row generation terminates in a finite number of steps (Proposition 3.3). □

All steps of the algorithm are illustrated in the example below.

Example 3.13. Suppose we wish the following MILP using the Benders' decomposition algorithm:

$$\begin{aligned} \min \quad & x + y_1 + y_2 \\ & x + y_1 = 2.5 \\ & x + y_2 = 5 \\ & x \in \{0, 1, 2, 3\}, \boldsymbol{y} \geq 0 \end{aligned} \tag{27}$$

Note that this is easily written in the form of Problem (13): $f(x) = x$, $\boldsymbol{q} = [1, 1]^T$, $\boldsymbol{g}(x) = [x, x]^T$, $W = I$, $\boldsymbol{h} = [2.5, 5]^T$ and $S = \mathbb{Z}_+$. Quick inspection reveals that $(x, y_1, y_2) = (2, 0.5, 3)$ is optimal with objective value 5.5.

Step 1 We initialize BMP without any constraints:

$$\begin{aligned} \min \quad & x + \theta \\ & x \in \{0, 1, 2, 3\} \end{aligned} \tag{28}$$

Step 2 Solving BMP to optimality yields $(x^*, \theta^*) = (0, -\infty)$.

Step 3 The corresponding BSP for Problem (27), given the current optimal solution of BMP is

$$\begin{aligned} \max \quad & (2.5 - x^*)u_1 + (5 - x^*)u_2 \\ & u_1 \leq 1, u_2 \leq 1 \end{aligned} \tag{29}$$

Since Problem (27) admits a feasible solution with $\boldsymbol{x}^* = 0$, we expect an optimality cut rather than a feasibility cut, so BSP should be bounded for $\boldsymbol{x}^* = 0$. Indeed, the optimal solution to

$$\begin{aligned} \max \quad & 2.5u_1 + 5u_2 \\ & u_1 \leq 1, u_2 \leq 1 \end{aligned} \tag{30}$$

is $(u_1, u_2) = (1, 1)$, which is an extreme point of the feasible region.

Step 4 BSP was bounded, so we proceed to step 5.

Step 5 The objective of BSP is $7.5 > -\infty$, so we add the following optimality cut to BMP and go back to step 2:

$$\theta \geq (2.5 - x) + (5 - x) = 7.5 - 2x. \tag{31}$$

Step 2 BMP is now

$$\begin{aligned} \min \quad & x + \theta \\ & \theta \geq 7.5 - 2x \\ & x \in \{0, 1, 2, 3\}, \end{aligned} \tag{32}$$

which has optimal solution $(x^*, \theta^*) = (3, 1.5)$.

Step 3 Note that Problem (27) does not admit a feasible solution with $x^* = 3$. So we expect BSP to be unbounded now. Indeed, BSP is

$$\begin{aligned} \max \quad & -0.5u_1 + 2u_2 \\ & u_1 \leq 1, u_2 \leq 1, \end{aligned} \tag{33}$$

which is unbounded.

Step 4 Observe that $(u_1, u_2) = (-1, 0)$ is an extreme direction that increases the objective value of BSP arbitrarily. We can add the following feasibility cut and return to step 2:

$$0 \geq -1(2.5 - x) + 0(5 - x) = x - 2.5. \tag{34}$$

Step 2 BMP for $x^* = 2$ is

$$\begin{aligned} \min \quad & x + \theta \\ & \theta \geq 7.5 - 2x \\ & 0 \geq x - 2.5 \\ & x \in \{0, 1, 2, 3\}, \end{aligned} \tag{35}$$

with optimal solution $(x^*, \theta^*) = (2, 3.5)$.

Step 3 BSP is now

$$\begin{aligned} \max \quad & 0.5u_1 + 3u_2 \\ & u_1 \leq 1, u_2 \leq 1, \end{aligned} \tag{36}$$

with optimal solution $(u_1, u_2) = (1, 1)$ and corresponding objective value 3.5.

Step 4 BSP is bounded, so we continue.

Step 5 The optimal solution to BSP has objective value $3.5 = \theta^*$, so the algorithm terminates.

The algorithm returns as optimal solution $(x, \theta) = (2, 3.5)$, giving a objective value of 5.5. This is correct. The corresponding $\boldsymbol{y}$ can easily be calculated by plugging in $x = 2$ into Problem (27) and solving the remaining LP, which can be done in polynomial time. These observations finish the example.

One property of a problem can make the algorithm even simpler.

Proposition 3.14. If Problem (13) has relatively complete recourse, the row-generation method based on Benders' decomposition will never return a feasibility cut.

Proof. Follow directly from the definition of relatively complete recourse: the second-stage problem is always feasible because it is only evaluated for feasible first-stage solutions. The Bender Subproblem (Problem (23)) is therefore bounded or infeasible, but never unbounded (by weak duality). □

To make the algorithm work in practice, we still need to account for the fact that in practice, (1) there are numerical errors and (2) MIPs are usually not solved to optimality but only to near-optimality. We thus want to set a stopping criterion that is more relaxed than $\theta \geq Q(x)$. To this end, we use the following results.

Proposition 3.15. Given Benders' master problem (Problem (24)) with any subset of optimality cuts in $[m_P]$ and feasibility cuts in $[n_P]$ included, we have

$$f(\boldsymbol{x}) + \theta \leq f(\boldsymbol{x}^*) + \theta^* \leq f(\boldsymbol{x}) + \zeta, \tag{37}$$

with $(\boldsymbol{x}, \theta)$ the optimal solution to Benders master problem, ζ the optimal value of the Benders' subproblem (Problem (23)) given $\boldsymbol{x}$ and $(\boldsymbol{x}^*, \theta^*)$ the true optimal solution of Problem (24).

Proof. The first inequality follows immediately from the observation that Benders' master problem is a relaxation of Problem (24). The second inequality must be proven for two different cases. Case 1: assume that $\boldsymbol{x}$ is feasible for Problem (24). Then Benders' subproblem is bounded and $f(\boldsymbol{x}) + \zeta$ corresponds to the objective value of some feasible solution of Problem (24). Case 2: assume that $\boldsymbol{x}$ is not feasible for Problem (24). Then Benders' subproblem (a maximization problem) is unbounded and therefore $\zeta = \infty$ by convention. □

Proposition 3.16. Given Benders' master problem (Problem (24)) with any subset of optimality cuts in $[m_P]$ and feasibility cuts in $[n_P]$ included, a feasible solution to Benders' master problem $(\boldsymbol{x}, \theta)$ with objective $\bar{z}$ and a lower bound on the optimal solution to Benders' master problem $\underline{z}$,

$$\underline{z} \leq f(\boldsymbol{x}^*) + \theta^* \leq \bar{z} - \theta + \zeta, \tag{38}$$

with ζ the optimal solution to Benders' subproblem (Problem (23)) given $\boldsymbol{x}$.

Proof. The first inequality follows trivially from Proposition 3.15. The second inequality follows from the observation that $\bar{z} - \theta = f(x)$, and $f(x) + \zeta$ corresponds to the objective value to Problem (24) if $\boldsymbol{x}$ is feasible for Problem (24) and $f(\boldsymbol{x}) + \zeta = \infty$ otherwise. □

Proposition 3.17. Let $(\boldsymbol{x}^* + \theta^*)$ be the optimal solution to Problem (24) and $(\boldsymbol{x}, \theta)$ be a solution to Benders' master problem, with objectives z^* and $\bar{z}$ respectively. Furthermore, suppose a lower bound $\underline{z}$ to Benders' master problem is known such that the relative gap

$$\frac{\bar{z} - \underline{z}}{\underline{z}} \leq \epsilon^{\text{solver}} \tag{39}$$

for some ϵ^{solver} and ζ is the optimal objective value of Benders' subproblem (Problem (23)) given $\boldsymbol{x}$. Then, the relative error ϵ can be bounded as follows:

$$\epsilon = \frac{f(\boldsymbol{x}) + \zeta - z^*}{z^*} \leq \epsilon^{\text{solver}} + \frac{\zeta - \theta}{\underline{z}}. \tag{40}$$

Proof.

$$\begin{aligned}
\epsilon &= \frac{f(\boldsymbol{x}) + \zeta - z^*}{z^*} \\
&\leq \frac{f(\boldsymbol{x}) + \zeta - \underline{z}}{\underline{z}} \\
&= \frac{f(\boldsymbol{x}) + \theta - \theta + \zeta - \underline{z}}{\underline{z}} \\
&= \frac{\bar{z} - \theta + \zeta - \underline{z}}{\underline{z}} \\
&\leq \epsilon^{\text{solver}} + \frac{\zeta - \theta}{\underline{z}}.
\end{aligned} \tag{41}$$

□

Corollary 3.18. Suppose one wishes to stop the Benders' decomposition algorithm when the relative error is below ϵ^{stop}. If the solver used to solve Benders' master problem is configured such that $\epsilon^{\text{solver}} < \epsilon^{\text{stop}}$, the algorithm will converge to a solution at most a factor ϵ^{stop} away from the true optimal solution.

3.3.4 Benders' decomposition algorithm without extreme points

Getting extreme points and extreme rays can be quite tricky: in case the simplex algorithm is used to solve the LP, an extreme point or ray can be obtained from the basis (Spliet, 2021b). But interior point methods may find an optimal solution that is not an extreme point or a ray that is not extreme. Although, given an optimal solution of a linear program, linear algebra techniques can be used to find an optimal extreme point, we will show in the remainder of this section that such procedures are not necessary.

In this section, we will show that step 5 of Benders' decomposition algorithm also works when general optimal solutions are used instead of extreme points. We will leave finding extreme rays out of scope for now since this thesis uses the L-shaped algorithm which has a special purpose routine for finding feasibility cuts.

To show that step 5 of Benders' decomposition algorithm also works when general optimal solutions are used instead of extreme points, two facts must still be shown: firstly, $\theta \geq (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T \hat{\boldsymbol{u}}$ is a valid cut in the more general setting where $\boldsymbol{u}$ is an optimal solution to Problem (23) instead of an extreme point. Secondly, adding optimal solutions instead of extreme points of Problem (23) also guarantees finite-time convergence of the row generation algorithm.

Proposition 3.19. The cut $\theta \geq (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T \hat{\boldsymbol{u}}$, with $\boldsymbol{u}$ an optimal solution to Problem (23) given some first-stage solution $(\boldsymbol{x}^*, \theta^*)$, is a valid optimality cut for Problem (24).

Proof. Let $\hat{\boldsymbol{u}}$ be an optimal solution of Problem (23) for some point $(\boldsymbol{x}^*, \theta^*)$, with $\theta^* < (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}^*))^T\hat{\boldsymbol{u}}$. Then indeed, adding the inequality $\theta \geq (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T\hat{\boldsymbol{u}}$ cuts of this solution.

Furthermore, suppose $(\boldsymbol{x}', \theta')$ is feasible for Problem (24). Then

$$\theta' \geq \max_{i \in [m_P]} (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}'))^T\hat{\boldsymbol{u}}^i \geq \max_{\boldsymbol{u} \in P} (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}'))^T\boldsymbol{u} \geq (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}'))^T\hat{\boldsymbol{u}} \tag{42}$$

So indeed, no valid solution to Problem (24) is cut off by the cut $\theta \geq (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T\hat{\boldsymbol{u}}$. □

Proposition 3.20. The row generation algorithm for Problem (24) convergences in a finite number of steps if, in step 5, optimality cuts $\theta \geq (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T\hat{\boldsymbol{u}}$ are added where $\boldsymbol{u}$ is an arbitrary optimal solution to Problem (23).

Lemma 3.21. Let P be a polyhedron with extreme points $\{\hat{\boldsymbol{u}}^i, i \in I\}$ and extreme rays $\{\hat{\boldsymbol{v}}^i, j \in J\}$. Furthermore, let F be a face of P. Then F consists precisely of the points $\boldsymbol{x} \in \mathbb{R}^n$ that can be represented as

$$\boldsymbol{x} = \sum_{i \in I'} \beta_i \hat{\boldsymbol{u}}^i + \sum_{j \in J'} \mu_j \hat{\boldsymbol{v}}^{\boldsymbol{j}}, \tag{43}$$

with $\beta_i \geq 0$, $i \in I'$, $\sum_{i \in I'} \beta_i = 1$, μ_j, $j \in J'$, $I' \subset I$ and $J' \in J$. In other words, F can be represented using a subset of the extreme points and rays of P.

Proof. Note that any face F of polyhedron P is itself a polyhedron (Proposition A.23). By Minkowski's representation theorem (Theorem A.17), it can be represented as

$$\boldsymbol{x} = \sum_{i \in I'} \beta_i \hat{\boldsymbol{u}}^i + \sum_{j \in J'} \mu_j \hat{\boldsymbol{v}}^{\boldsymbol{j}}, \tag{44}$$

with $\{\hat{\boldsymbol{u}}^i, i \in I'\}$ extreme points of F and $\{\hat{\boldsymbol{v}}^j, j \in J'\}$ extreme rays of F. The last step of this proof is to show that these extreme points and rays are also extreme points and rays of P.

Suppose the opposite is true (we will derive a contradiction), and one extreme point $\hat{\boldsymbol{u}}^i$ of F is not an extreme point of P. Then there exist $\boldsymbol{x}^1, \boldsymbol{x}^2 \in P$ such that $\hat{\boldsymbol{u}}^i = \frac{1}{2}(\boldsymbol{x}^1 + \boldsymbol{x}^2)$, with $\boldsymbol{x}^1 \notin F$ or $\boldsymbol{x}^2 \notin F$. Without loss of generality, suppose $\boldsymbol{x}^1 \notin F$. Let $\boldsymbol{\pi}^T\boldsymbol{x} \leq \pi_0$ be a valid inequality for P defining F, i.e., $F = \{\boldsymbol{x} \in P \mid \boldsymbol{\pi}^T\boldsymbol{x} = \boldsymbol{\pi_0}\}$. Since $\boldsymbol{x}^1 \notin F$, $\boldsymbol{\pi}^T\boldsymbol{x}^1 \neq \pi_0$. If $\boldsymbol{\pi}^T\boldsymbol{x}^1 \leq \pi_0$, then $\boldsymbol{\pi}^T\boldsymbol{x}^2 \not\leq \pi_0$ and vice versa, meaning that $\boldsymbol{\pi}^T\boldsymbol{x} \leq \pi_0$ is not a valid inequality. Contradiction.

Similarly, suppose an extreme ray $\hat{\boldsymbol{v}}^j$ of F is not an extreme ray of P. Then there are two distinct rays $\boldsymbol{y}^1$, $\boldsymbol{y}^2$ of P such that $\hat{\boldsymbol{v}}^j = \frac{1}{2}(\boldsymbol{y}^1 + \boldsymbol{y}^2)$, where (we assume without loss of generality) $\boldsymbol{y}^1$ is not a ray of F. As a consequence, there exists no $\boldsymbol{x} \in F$ such that $\boldsymbol{x} + \mu\boldsymbol{y}^1 \in F$ for all $\mu \geq 0$. So for any $\boldsymbol{x} \in F$ there exists a $\mu \geq 0$ such that $\boldsymbol{x} + \mu\boldsymbol{y}^1 \notin F$, meaning $\boldsymbol{\pi}^T(\boldsymbol{x} + \mu\boldsymbol{y}^1) \neq \pi_0$. By a similar argument as in the previous argument, $\boldsymbol{\pi}^T\boldsymbol{x} \leq \pi_0$ is not a valid inequality as a consequence. Contradiction.

We conclude that the extreme points and rays of F are extreme points and rays of P. The desired result follows. □

Proof of Proposition 3.20. We will now prove that, if step 5 of the row-generation algorithm is modified such that general optimal solutions (instead of optimal extreme points) of the benders subproblem are used to generate optimality cuts, the algorithm still converges in a finite number of steps.

Note that the polyhedron $W^T\boldsymbol{u} \leq \boldsymbol{q}$ has a finite number of faces (Proposition A.24). Also, note that the extreme points of this polyhedron are contained in the set of faces. It thus suffices to show that each iteration of the row-generation algorithm, which returns an optimal solution, returns an optimal solution from a different face.

As a technicality, we identify for this proof a face F by the minimal subsets $I' \subset I$ and $J' \subset J$ required to represent F in the way of Lemma 3.21. We say that two faces are different if they have different minimal I' or J'.

Suppose we are in some iteration of the row generation algorithm and n inequalities $\theta \geq (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))^T\boldsymbol{u}^i$, $i \in [n]$, with $\boldsymbol{u}^i$ optimal solutions to the benders subproblems (Problem (23)) of previous iterations. Let $(\boldsymbol{x}^*, \theta^*)$ be the solution to the incumbent Benders master problem and $\boldsymbol{u}^{n+1}$ an optimal solution to Problem (23). We will show that the algorithm either terminates (with an optimal solution to Problem (24)) or that $\boldsymbol{u}^{n+1}$ is on a different face than $\boldsymbol{u}^i, i \in [n]$.

If $\theta^* \geq (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}^*))^T\boldsymbol{u}^{n+1}$, the algorithm terminates. The solution is then optimal since, in particular, all extreme points of $W^T\boldsymbol{u} \leq \boldsymbol{q}$ would also satisfy the same inequality.

If $\theta^* < (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}^*))^T\boldsymbol{u}^{n+1}$, then $\boldsymbol{u}^{n+1}$ is on a different face than $\boldsymbol{u}^i$, for any, $i \in [n]$. Indeed, take any $i \in [n]$. Then $\theta^* \geq (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}^*))^T\boldsymbol{u}^i$ while $\theta^* < (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}^*))^T\boldsymbol{u}^{n+1}$. Suppose $\boldsymbol{u}^i$ and $\boldsymbol{u}^{n+1}$ are on the same face F. Then there is a direction of improvement for Problem (23) on F from $\boldsymbol{u}^i$ to $\boldsymbol{u}^{n+1}$. Then F is not an extreme point, since that would imply $\boldsymbol{u}^i = \boldsymbol{u}^{n+1}$. As a consequence, the dimension of F is at least 1 (say m), meaning that the boundary of F consists of a union of different $(m-1)$-dimensional faces. Therefore, $\boldsymbol{u}^{n+1}$ is in the interior of F, meaning there exists an $\epsilon > 0$ such that $\boldsymbol{u}^{n+1} + \epsilon(\boldsymbol{u}^{n+1} - \boldsymbol{u}^i) \in F$. Since $\boldsymbol{u}^{n+1} - \boldsymbol{u}^i$ is a direction of improvement in a maximization problem,

$$(\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}^*))^T(\boldsymbol{u}^{n+1} + \epsilon(\boldsymbol{u}^{n+1} - \boldsymbol{u}^i)) \geq (\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}^*))^T\boldsymbol{u}^{n+1}. \tag{45}$$

This is a contradiction with the assumption that $\boldsymbol{u}^{n+1}$ is optimal.

We conclude that each optimality cut uses an optimal solution $\boldsymbol{u}^i$ from a different face of $\{\boldsymbol{u}|W^T\boldsymbol{u} \leq \boldsymbol{q}\}$. Since the number of faces is finite, the set of faces includes all extreme points and each optimality cut added is valid for Problem (23), the algorithm converges in a finite number of iterations to the optimal solution. □

There is still one theoretical problem. In the more general setting of this section, BSP returns solutions on faces of $W^T\boldsymbol{u} \leq \boldsymbol{q}$ instead of extreme points. The number of faces of a polyhedron is potentially much higher than the number of extreme points. Therefore, the Benders' decomposition algorithm is potentially much slower if we do not restrict BSP to returning extreme points.

However, in practice, this is not expected to be a problem. To see this, we can do some probability calculations from a Bayesian perspective. Let's suppose a computer has some numerical errors in floating-point computations which are not known a priori. Then we can show the probability of an m-dimensional face of $W^T\boldsymbol{u} \leq \boldsymbol{q}$ with $m \geq 1$.

The following theorem and proof require some background knowledge about probability theory, which is summarized for reference in Appendix A.4.

Theorem 3.22. Suppose all the data in BSP is given, $\boldsymbol{u} \in \mathbb{R}^n$, and the calculation of $(\boldsymbol{h} - \boldsymbol{g}(\boldsymbol{x}))$ returns $\boldsymbol{a}$, where $\boldsymbol{a}$ is a random vector for which there exists $\epsilon > 0$ such that $N_\epsilon(\boldsymbol{a}) \subset \text{supp}(\boldsymbol{a})$. Furthermore, suppose the probability measure $\mathbb{P}_{\boldsymbol{a}}$ of $\boldsymbol{a}$ is absolutely continuous with respect to the Lebesgue measure on $\mathbb{R}^n$. Then, with probability 1, $\boldsymbol{a}$ is not orthogonal to any face with a dimension larger or equal to 1.

Proof. Assuming all the assumptions of the theorem, let F be an arbitrary face of $W^T\boldsymbol{u} \leq \boldsymbol{q}$ with $\dim(F) \geq 1$. Note that we $\mathbb{P}(\boldsymbol{a} \perp F) = \mathbb{P}(\boldsymbol{a} \in F^\perp)$ with$F^\perp$ the linear space of vectors which are orthogonal to all vectors between points in F. Note that $\dim(F^\perp) = n - \dim(F) = n - m < n$.

Since $\mathbb{P}_{\boldsymbol{a}}$ is absolutely continuous with respect to the Lebesgue measure μ on $\mathbb{R}^n$, there is a density function $\lambda : \mathbb{R}^n \to [0, \infty)$ (the Radon-Nikodym derivative) which is μ-integrable and satisfies

$$\mathbb{P}_{\boldsymbol{a}}(X) = \int_X \lambda \, d\mu. \tag{46}$$

Since $\dim(F^\perp) < n$, $\mu(F^\perp) = 0$ and therefore $\mathbb{P}_{\boldsymbol{a}}(F^{perp}) = 0$ too. Since there is a finite number of faces, it also holds with probability 1 that $\boldsymbol{a}$ is not perpendicular to any face of $W^T\boldsymbol{u} \leq \boldsymbol{q}$ with dimension at least 1. □

Corollary 3.23. Under the assumptions of Theorem 3.22, there is almost surely no face F of $W^T\boldsymbol{u} \leq \boldsymbol{q}$ with $\dim(F) \geq 1$ where all points are optimal solutions to BSP (Problem (23)). Thus, with probability 1, BSP returns an extreme point, regardless of the method of optimization.

We conclude that generalizing step 5 of Benders' decomposition algorithm to allow for optimality cuts following from general optimal solutions to BSP should not result in a higher number of iterations until convergence.

3.3.5 L-shaped Algorithm

Benders' decomposition can be applied to two-stage stochastic optimization problems of the following form:

$$\begin{aligned} \min \quad & \boldsymbol{c}^T\boldsymbol{x} + \sum_{k=1}^{K} p_k (\boldsymbol{q}^k)^T \boldsymbol{y}^k \\ & A\boldsymbol{x} = \boldsymbol{b} \\ & T^k\boldsymbol{x} + W^k\boldsymbol{y}^k = \boldsymbol{h}^k \qquad k \in [K] \\ & \boldsymbol{x} \geq \boldsymbol{0}, \boldsymbol{y}^k \geq \boldsymbol{0} \qquad k \in [K]. \end{aligned} \tag{47}$$

Indeed, note that we can use the following transformations to rewrite Problem (47) as Problem (13): $S = \{\boldsymbol{x} | A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$, $f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x}$, $\boldsymbol{g}(\boldsymbol{x}) = [T^1, \ldots, T^K]^T\boldsymbol{x}$, $\boldsymbol{y} = [\boldsymbol{y}^1, \ldots, \boldsymbol{y}^K]^T$, $\boldsymbol{h} = [\boldsymbol{h}^1, \ldots, \boldsymbol{h}^K]^T$, $\boldsymbol{q} = [p_1\boldsymbol{q}^1, \ldots, p_K\boldsymbol{q}^K]^T$ and $W = \text{diag}(W^1, \ldots, W^K)$. We could apply Benders' decomposition to this problem directly.

However, we can also use a more specific implementation, tailor-made implementation of Bender's decomposition, called the L-shaped algorithm. This algorithm solves two questions regarding the application of Benders' decomposition that have not been answered yet.

1. How to solve BSP quickly, in case K is large?
2. How to find the extreme rays needed for feasibility cuts?

The explanation below closely follows Birge and Louveaux (2011) and Postek and Kuryatnikova (2021). First, it is good to observe the final form of BMP which we will use:

$$\begin{aligned} \min \quad & \boldsymbol{c}^T\boldsymbol{x} + \theta \\ \text{s.t.} \quad & A\boldsymbol{x} = \boldsymbol{b} \\ & (\boldsymbol{e}^i)^T\boldsymbol{x} + \theta \geq \tilde{e}_i \qquad i \in I' \subset I \\ & (\boldsymbol{d}^j)^T\boldsymbol{x} \geq \tilde{d}_j \qquad j \in J' \subset J \\ & \boldsymbol{x} \geq 0, \boldsymbol{y}^k \geq 0 \qquad k \in [K], \end{aligned} \tag{48}$$

with I and J the full sets of optimality and feasibility cuts. Note that the optimality and feasibility cuts of Problem (24) can easily be written in the form of Problem (48) when the optimization problem considered is Problem (47).

Solving BSP fast The answer to the first question is: exploit the block structure of W. BSP for Problem (47) can be written as:

$$\begin{aligned} \max_{\boldsymbol{u}^k, k\in[K]} \quad & \sum_{k=1}^{K} (\boldsymbol{h}^k - T^k\boldsymbol{x})^T \boldsymbol{u}^k \\ & (W^k)^T \boldsymbol{u}^k \leq p_k \boldsymbol{q}^k \qquad k \in [K]. \end{aligned} \tag{49}$$

The feasible region and the objective function can be decomposed into K independent subproblems. Given $k \in [K]$, subproblem k is defined as

$$\begin{aligned} \max_{\boldsymbol{u}^k} \quad & (\boldsymbol{h}^k - T^k\boldsymbol{x})^T \boldsymbol{u}^k \\ & (W^k)^T \boldsymbol{u}^k \leq p_k \boldsymbol{q}^k. \end{aligned} \tag{50}$$

These subproblems can be solved independently and the sum of their optimal objective values is equal to the objective value of BSP.

This block-based decomposition is computationally advantageous since the fastest LP-solving algorithm known at the time of writing is believed to be approximately $O(n^{2.055}L)$ (Jiang et al., 2020), with n the number of variables and L the number of bits needed to encode the problem. If we let n and L be the number of variables and the number of bits in one subproblem respectively, solving the K subproblems has complexity $O(K(n^{2.055}L))$. If we were to solve BSP as one problem, the complexity would be (assuming no bits are required for the sparse parts of the constraint matrix) $O((Kn)^{2.055}KL) = O(K^{3.055}n^{2.055}L)$, which is higher for $K > 1$.

Finding feasibility cuts The second question, how to find extreme rays of BSP, is slightly more involved. BSP is unbounded if and only if at least one of the subproblems $k \in [K]$ is unbounded. We restrict ourselves to the cases where BSP is feasible. This is a sensible restriction since without it Problem (47) would be infeasible or unbounded and thus have no optimal solution, eliminating the need to find any. The feasibility of BSP can be checked before running any time-costly algorithms.

Taking the dual of Problem (50) (the subproblem for $k \in [K]$), we obtain

$$\begin{aligned} \min_{\boldsymbol{y}^k \geq \boldsymbol{0}} \quad & p_k \boldsymbol{q}^k \boldsymbol{y}^k \\ & W^k \boldsymbol{y}^k = \boldsymbol{h}^k - T^k \boldsymbol{x}, \end{aligned} \tag{51}$$

which is to be expected given the derivation of BSP. This formulation is convenient for two reasons: firstly, we can implement an optimality cut routine using this problem much more cleanly than using the regular BSP formulation, since it doesn't require finding the dual of parts of Problem (47). Instead, we can use the constraints from Problem (47), which are usually more interpretable, directly. The duals required to construct the optimality cut can be queried from any off-the-shelf solver used to solve Problem (51).

Secondly, this problem can easily be checked for feasibility (i.e., unboundedness of BSP under the assumption that BSP is feasible), by modifying it slightly (Birge and Louveaux, 2011):

$$\begin{aligned} \min_{\boldsymbol{y}^k, \boldsymbol{\gamma}^{k+}, \boldsymbol{\gamma}^{k-}} \quad & \mathbf{1}^T (\boldsymbol{\gamma}^{k+} + \boldsymbol{\gamma}^{k-}) \\ & W^k \boldsymbol{y}^k + I(\boldsymbol{\gamma}^{k+} - \boldsymbol{\gamma}^{k-}) = \boldsymbol{h}^k - T^k \boldsymbol{x}. \end{aligned} \tag{52}$$

Problem (52) is, due to the auxiliary variables $\boldsymbol{\gamma}^{k+}$ and $\boldsymbol{\gamma}^{k-}$, always feasible and bounded. Furthermore, Problem (51) is feasible if and only if the optimal objective value of Problem (52) is strictly positive. If the latter condition holds, by strong duality, $0 < (\boldsymbol{h}^k - T^k\boldsymbol{x})^T\hat{\boldsymbol{v}}$ with $\hat{\boldsymbol{v}}$ the optimal duals of the constraints in Problem (52). Any feasible solution to Problem (51) is feasible for Problem (52) with any feasible dual $\hat{\boldsymbol{v}}$ satisfies (by weak duality)

$$(\boldsymbol{h}^k - T^k\boldsymbol{x})^T\hat{\boldsymbol{v}} \leq 0, \tag{53}$$

since 0 is in that case the optimal objective value of Problem (52). It follows that Equation (53) is a valid feasibility cut.

For convenience of notation, we can set $\boldsymbol{d}^j = (T^k)^T\hat{\boldsymbol{v}}$ and $\tilde{d}_j = \boldsymbol{h}^T\hat{\boldsymbol{v}}$ to get the feasibility cut in the form of Problem (48).

Finding optimality cuts Finding optimality cuts is much more straightforward - there is no real difference from the regular Benders' decomposition. All we do in this paragraph is clean up the notation slightly. If $\boldsymbol{x}$ yields a feasible second-stage problem for any $k \in K$, we solve a slightly modified version of Problem (51) to optimality for all $k \in K$, namely:

$$\begin{aligned} \min_{\boldsymbol{y}^k \geq \boldsymbol{0}} \quad & \boldsymbol{q}^k\boldsymbol{y}^k \\ & W^k\boldsymbol{y}^k = \boldsymbol{h}^k - T^k\boldsymbol{x}. \end{aligned} \tag{54}$$

The only difference is the removal of p_k from the objective function, which is nice for interpretation: Problem (54) can now be read as "find the optimal second-stage solution given a first-stage solution and a realization of the random data".

Then, the optimal duals $\hat{\boldsymbol{u}}^k$, $k \in K$ are computed and we can check whether the optimal value of θ from the restricted master problem satisfies

$$\theta \geq \sum_{k=1}^{K} p_k(\boldsymbol{h} - \boldsymbol{T}^k\boldsymbol{x})^T\hat{\boldsymbol{u}}^k. \tag{55}$$

If that is the case, the solution is optimal. Else, we can add Equation 55 to the restricted master problem.

For convenience of notation, we can set $\boldsymbol{e}^i = \sum_{k=1}^{K} p_k(T^k)^T\hat{\boldsymbol{u}}$ and $\tilde{e}_i = \sum_{k=1}^{K} p_k\boldsymbol{h}^T\hat{\boldsymbol{u}}$ to get the optimality cut in the form of Problem (48).

L-shaped algorithm From the preceding discussion, we can derive an iterative algorithm called the L-shaped algorithm (Algorithm 1).

Theorem 3.24. Algorithm 1 finds the optimal first-stage solution to Problem (47).

Proof. See for instance pages 196 - 198 of Birge and Louveaux (2011). □

When using the L-shaped algorithm, there are some additional considerations around the feasibility cuts that may help decrease the run time.

Remark 3.25. If in some iteration of Algorithm 1 multiple $k \in [K]$ are found with $z^k > 0$ (say a subset $K' \subset [K]$) we can add one of multiple of the feasibility cuts corresponding to K'. If solving Problem (52) for all $k \in K$ takes relatively long compared to solving BMP it may be worth adding multiple cuts at the same time to reduce the number of iterations. Alternatively, we may also stop solving Problem (52) after the first $k \in [K]$ with $z^k > 0$ is found.

Algorithm 1: L-shaped algorithm

1 Initialize BMP (Problem (48)) with $I' \subset I$ and $J' \subset J$;
2 optimal := false;
3 **while** *!optimal* **do**
4 Solve BMP, set $(\boldsymbol{x}^*, \theta^*)$ as optimal solution;
5 Solve Problem (52) for $k \in K$ with optimal values z^k and optimal duals $\hat{\boldsymbol{v}}^k$;
6 **if** $\exists k \in K : z^k > 0$ **then**
7 Add feasibility cut (53) with duals $\hat{\boldsymbol{v}}^k$ to BMP for some $k \in K$;
8 **continue**;
9 Solve Problem (54) for $k \in K$ with optimal duals $\hat{\boldsymbol{u}}^k$;
10 **if** *Equation* (55) *is violated by* $(\boldsymbol{x}^*, \theta^*)$ **then**
11 Add optimality cut (55) with duals $\hat{\boldsymbol{u}}^k$, $k \in K$ to BMP;
12 **else**
13 optimal := true;

Definition 3.26 (**Relatively complete recourse**, Birge and Louveaux (2011))**.** We say that Problem (47) has *relatively complete recourse* if for any $\boldsymbol{x} \geq \boldsymbol{0}$ satisfying $A\boldsymbol{x} = \boldsymbol{b}$ there exists a feasible solution $\boldsymbol{y}^k$ for all $k \in K$.

Proposition 3.27. If Problem (47) has relatively complete recourse, no feasibility cuts are generated in Algorithm 1.

Proof. Trivial from the discussion in the paragraph on finding feasibility cuts. □

Corollary 3.28. Lines 5 to 8 may be removed from Algorithm 1 in case Problem (47) has relatively complete recourse.

Note that there are many settings imaginable where two-stage stochastic optimization problems have relatively complete recourse.

Example 3.29. Airlines must decide how many flights to schedule. In case more personnel is sick than expected, they may have to cancel some flights. The problem described here could be modeled as a two-stage optimization problem, where the number of flights to plan is the first-stage decision, the number of flights to cancel is the second decision, and the sickness rate is the random data. This problem has relatively complete recourse since an airline can always cancel all flights (albeit at a very high cost).

The L-shaped algorithm has been extended to many different settings. In chapters 5 and 6 of Birge and Louveaux (2011), a multi-cut version (where several cuts are added in each iteration), a regularized version with a quadratic penalty term, and a multi-stage nested version are given. Laporte and Louveaux (1993) introduce a branch-and-cut procedure based on the L-shaped algorithm for problems where the first stage variables are binary. A large part of their paper is focused on presenting efficient cuts to add in each node of the branching tree. They demonstrate the effectiveness of their algorithm on a problem with 24 first-stage and 6 second-stage binary decision variables and a second-stage problem with a nice closed-form expression: this is a relatively small problem, but the experiment was run in the early nineties on a 16 MHz processor.

Angulo et al. (2016) present two improvements to the integer L-shaped method. They consider two-stage problems where the first stage problem has both a set of binary variables and a set

of mixed-integer variables and the second stage problem depends only on the binary first-stage variables (so a slight generalization of the problem described in the previous paragraph). The first improvement to the integer L-shaped method is the introduction of an alternating cut routine. With this method, they try to prevent the computation of the optimal solution to the mixed-integer second-stage problem as much as possible by first checking if a cut can be generated based on the optimal solution of the LP-relaxation of the second-stage problem. Moreover, they propose a new cut-generating routine, which the authors call CGLP (cut-generating linear program). They test four different methods (combining standard or alternating cut routines with standard or CGLP cuts) on two problems: the stochastic server location problem and the stochastic multiple binary knapsack problem (the details of both problems are irrelevant here). With the first problem, they demonstrate excellent accelerations: the test instances have ten to fifteen first-stage binary variables and a few hundred binary second-stage variables. The alternating cut procedure decreased the solution time by an order of magnitude compared to the standard cutting routine (while the cut type did not make a noticeable difference). With the second problem, the results are less promising: there is no substantial difference in run-time between the standard and alternating cut routines. On the other hand, in both cases, using CGLP cuts reduced the run times by approximately 10 percent on average.

The large difference in the performance of the alternating cut routine between the two different problems is explained by the authors of Angulo et al. (2016) as being caused by the relative complexity of the second stage problems. In the stochastic server location problem, there are many more second-stage binary variables than first-stage binary variables, so evaluating the second-stage problem is the bottleneck here. In the stochastic multiple binary knapsack problem, the first and second-stage problems have the same number of binary variables. The authors argue that the alternating cut routine is more effective for the first problem because it reduces the number of exact second-stage problem evaluations. They conclude that CGLP-cuts are effective when the first-stage problem is relatively difficult to solve.

A few more interesting observations can be made from the paper of Angulo et al. (2016). Firstly, they implemented their proposed algorithm using IBM Cplex with its callback functions to add cuts in each node of the branching tree, demonstrating that the approach of using callback functions in a commercial solver is indeed a viable approach for implementing advanced branch-and-cut strategies. Secondly, the test instances on which the performance of the algorithms was tested were quite small. For example, the stochastic server location problem instance with 10 binary first-stage variables and 500 binary second-stage variables was solved with up to 2000 different scenarios in the probability distribution. On the other hand, the instance with 15 binary first-stage variables and 675 binary second-stage variables was only solved with up to 25 different scenarios. No larger instances were tested. The authors considered only stochastic multiple binary knapsack problem instances with 240 binary first-stage variables and 120 binary second-stage variables. They grouped the instances into "small", "medium" and "hard", and the hard instances took on average approximately half an hour to solve with all algorithms.

Zou et al. (2019) consider multi-stage stochastic integer optimization problems, where the n-th stage problem depends only on binary variables of the $(n-1)$-th stage problems. They propose an algorithm that they call Stochastic Nested Decomposition (SND), where the idea is to generate a finite scenario tree based on the possible realizations of the random variables of each stage and iteratively generate an improved lower bound and statistical upper bound on the optimal value of the problem. In SND, a subset of the possible scenarios is sampled from the tree (i.e., a subset of leaves is selected, and the unique paths from the initial node to these leaves from the scenarios). Using a forward pass, the optimal solution (not just the first stage solution, but the solution at each stage) for each sampled scenario is calculated. From this, a statistical upper bound on the optimal objective value is computed. Then, using a backward pass, cuts are generated and the nodal optimization problems are tightened. Zou et al. (2019) furthermore gives a much

faster algorithm for the special case where the stochasticity in each stage is independent of the stochasticity in the other stages.

Zou et al. (2019) prove almost-sure convergence of their algorithms and list several different types of relaxations of the nodal problems to generate cuts from: Benders'cuts, based on the LP-relaxations of the nodal problems, the integer optimality cuts proposed by Laporte and Louveaux (1993), Lagrangian cuts (newly proposed in Zou et al. (2019)) and strengthened Benders' cuts. They demonstrate experimentally that the use of Lagrangian cuts and strengthened Benders cuts can substantially accelerate the convergence of their algorithms compared to the use of the integer optimality cuts from Laporte and Louveaux (1993). They draw the following conclusions about their algorithm: (a) using strengthened Bender's cuts combined with integer optimality cuts seems to be the best approach; (b) their algorithm for the special case where stochasticities of a stage do not depend on uncertainties in the previous stage can solve problems of sizes much larger than those which could be solved by off-the-shelf solvers in their naive formulation; (c) the number of forward sample paths required is very small: about 1 to 3 seems to be very effective for large scale instances.

3.4 Literature gaps

Van Den Bergh et al. (2013) note in their paper that "many research projects do not make it until the implementation in practice". The first main reason is that "the personnel scheduling problem is hardly ever integrated with other scheduling problems such as operating room scheduling and machine scheduling". Secondly, the authors blame "the low degree of uncertainty incorporation". They say that "most of the researchers do not consider the effect of canceled tasks, unavailable employees, increased workload, ..."

Furthermore, from the segmentations made by Van Den Bergh et al. (2013), one can also see obstacles that stand in the way of practical implementation. For example, a very substantial part of the papers reviewed has hard constraints that prohibit understaffing. In real-world operations, it is not always possible (or economically viable) to guarantee that the employee size is sufficient to satisfy the workload requirements. Furthermore, if we look at individual constraints relating to labor laws, collective labor agreements, and contractual obligations, we see that many such constraints are only considered by a (small) subset of papers. One can think of constraints enforcing a minimum or maximum number of hours, minimum time between assignments, minimum uninterrupted rest time per week, etc. Just to be compliant with labor laws and contractual agreements, it is usually required to incorporate a combination of such constraints into the model.

To the best of our knowledge, this thesis proposes the first model and corresponding algorithm that has the following combination of properties:

- it supports compliance to the complex sets of labor laws and contractual regulations of several Western European countries (in particular: The Netherlands, Germany, and France);
- it takes into account information about uncertainty in the amount of workload and the no-show rates;
- it supports a large degree of flexibility, for example by taking into account individual availabilities and contracted hours of employees and supporting overlapping shifts.

While the scheduling algorithm proposed in this thesis is specifically designed to suit the needs of Picnic, we believe that the same principles will be more widely applicable to creating scheduling solutions in the industry.

4 Problem Formulation

Given a set of employees E and a set of shifts S, a planning period $[0, T)$ the main decision variables are $x_{es} \in \{0, 1\}$ for $e \in E$, and $s \in S^T$. We assume without loss of generality that the shifts are sorted by increasing starting time.

We formulate the problem as a two-stage stochastic mixed integer linear optimization problem, with binary and continuous variables in the first-stage problem and only continuous variables in the second-stage problem. In this chapter, we first introduce the first-stage problem, which covers all of the scheduling decisions and constraints that are introduced in Chapter 2.

Then, we introduce two versions of the second-stage problem, which we will refer to as the "performance model". The performance model computes the "quality" of a given schedule satisfying the first-stage constraints.

4.1 First-stage problem: create a feasible schedule

The first stage problem consists of a set of constraints that specify the set of feasible schedules. To support scheduling operations in different countries and contexts, we do not work with a fixed first-stage problem, but instead develop a set of abstract constraint types which can be selected and parameterized by the user. There is no fixed objective function either, but some of the constraints listed in this section are so-called "soft" constraints, meaning they can be violated at the cost of some penalty. The corresponding penalty terms are included in the objective function.

We first introduce the concept of "time windows", which is a central building block of many of the constraint types that can be included in the scheduling problem. After that, we give a MILP formulation of each constraint type specified in Chapter 2.

4.1.1 Modelling time windows

Many of the constraints that must be considered are time-windowed constraints, in the sense that some conditions must hold for all time windows of a certain form. Take as an example the following instance of requirement MaxWorkingTime: "An employee may, on average, work at most 55 hours per period of four weeks". This requirement is still slightly ambiguous: which periods of four weeks are considered precisely? Consider the following three different interpretations of the requirement:

- an employee may not work more than 55 hours in the windows [Monday 3 October, Sunday 9 October], [Monday 10 October, Sunday 16 October], etc.;
- an employee may not work more than 55 hours in the windows [Monday 3 October, Sunday 9 October], [Tuesday 4 October, Monday 10 October], etc.;
- an employee may not work more than 55 hours in the windows $[t, t + 4\text{weeks}] \in \mathcal{T}$ (so uncountably many windows).

We shall in general restrict ourselves to discrete time windows (not a restrictive assumption, as start and end times of shifts, are scheduled in a discrete-time setting as well) and model time windows in general as a set $\mathcal{W}(t_0, \Delta t, W) = [t_0 + k\Delta t, t_0 + k\Delta t + W), k \in \mathbb{Z}$, where t_0 and Δt are real number defining a discrete time grid, and W and the time window length.

4.1.2 Constraints

There are some basic constraints required to get a physically feasible assignment:

$$x_{es} + x_{es'} \leq 1 \quad e \in E, (s, s') \in S^T \times S, s \cap s' \neq \emptyset \tag{56}$$

$$x_{es} \in \{0,1\} \quad e \in E, s \in S^T \tag{57}$$

MaxWorkingTime Suppose an employee $e \in E$ may on average not work more than H hours per [time unit] over a set of windows $\mathcal{W}(t_0, \Delta t, W)$ with W in the same time units.

$$\sum_{s \in S} x_{es} t^{\text{net}}_{sw} \leq H_e W \quad w \in \mathcal{W}(t_0, \Delta t, W), e \in E \tag{58}$$

where t^{net}_{sw} is the net working time of shift s in window w. This can be computed as $\mu(s \cap w)$ minus the break time that falls within w. Here, and in the remainder of this text, $\mu(X)$ denotes the Lebesgue measure of set X.

MinNightlyRestTime Suppose an employee must have a rest time of H [time units] between two consecutive shifts on different days. We can enforce this requirement by the following constraint.

$$\sum_{s' = s}^{\max\{s' | t^{start}_{s'} - t^{end}_{s} \leq H \wedge s, s' \text{are on different days}\}} x_{es'} \leq 1 \quad s \in S, e \in E \tag{59}$$

MinRestTime This requirement enforces a minimum uninterrupted rest time of H for each window $w \in \mathcal{W}(t_0, \Delta t, W)$. Just as in MinNightlyRestTime, we consider the time between two consecutive shifts. The difference is here that in each window w, there just needs to be *one* uninterrupted rest time of at least H.

We introduce a new decision variable $\xi_{es} \in \{0,1\}$ for each shift (exact conditions to be specified), indicating whether an employee has a rest period of at least H time units after the end of shift s. Indeed, let $\tilde{S}(s) = \{\tilde{s} \in S \mid \tilde{s} \cap [t^{end}_s, t^{end}_s + H) \neq \emptyset\}$. Then we require that

$$\xi_{es} \leq 1 - x_{e\tilde{s}} \quad \tilde{s} \in \tilde{S}(s), s \in S, e \in E. \tag{60}$$

Analogously, we introduce decision variable ξ_{ew} for each window $w \in \mathcal{W}(t_0, \Delta t, W)$ and $e \in E$ and define $\tilde{S}(w) = \{\tilde{s} \in S \mid t^{\text{start}}_{\tilde{s}} \in [w^{\text{start}}, w^{\text{start}} + H)\}$, where $w = [w^{\text{start}}, w^{\text{end}})$ and require that

$$\xi_{ew} \leq 1 - x_{e\tilde{s}} \quad \tilde{s} \in \tilde{S}(w), w \in \mathcal{W}(t_0, \Delta t, W), e \in E. \tag{61}$$

Then, for each window $w \in \mathcal{W}(t_0, \Delta t, W)$, we must guarantee that there is at least one uninterrupted rest of at least H. Given a window $w \in \mathcal{W}(t_0, \Delta t, W)$, a set $S'(w) = \{s \in S | [t^{end}_s, t^{end}_s + H) \subset w\}$ can be constructed. Then, the constraints

$$\xi_{ew} + \sum_{s \in S'(w)} \xi_{es} \geq 1 \quad w \in \mathcal{W}(t_0, \Delta t, W), e \in E \tag{62}$$

enforce that not all ξ_{es} in a given window can be 1, or equivalently: at least one of the ξ_{es} must be 0.

One undesirable property of the proposed formulation for MinRestTime is that it introduces $|E||S|$ binary variables. However, the binarity requirements can be relaxed because of the following proposition.

Proposition 4.1. Given $\boldsymbol{x} \in \{0,1\}^{E \times S}$, there exists $\boldsymbol{\xi} \in \{0,1\}^{E \times S}$ such that (60) and (62) are satisfied if and only if there exists $\boldsymbol{\xi} \in \mathbb{R}^{E \times S}_{+}$ such that (60) and (62) are satisfied.

Proof. Let $\boldsymbol{x} \in \{0,1\}^{E\times S}$ arbitrarily. First, suppose a $\boldsymbol{\xi} \in \{0,1\}^{E\times S}$ exists satisfying Constraints (60) and (62) are satisfied. Then trivially, that same $\boldsymbol{\xi}$ is also an element of $\mathbb{R}_+^{E\times S}$ satisfying Constraints (60) and (62). Now, we prove the reverse implication. Suppose some $\boldsymbol{\xi} \in \mathbb{R}_+^{E\times S}$ is satisfies Constraints (60) and (62). Let $\boldsymbol{\xi'} \in \{0,1\}^{E\times S}$ such that

$$\xi'_{es} = \begin{cases} 0 & \xi_{es} < 1 \\ 1 & \xi_{es} \geq 1 \end{cases} . \tag{63}$$

Indeed, $\boldsymbol{\xi'}$ satisfies Constraint (60) by the binarity of $\boldsymbol{x}$ and it satisfies Constraint (62) because $\xi'_{es} \leq \xi_{es}$ for all $e \in E$ and $s \in S$. □

Corollary 4.2. When including constraints of type MinRestTime into the scheduling problem, it suffices to let $\boldsymbol{\xi}$ be a vector of continuous variables with the conventional domain $\mathbb{R}_+$ instead of binary variables.

MinRestTimeNL We also consider a variant of the previous constraint MinRestTime. It is called MinRestTimeNL because it is a special variant required to comply with the Working Hours Act of the Netherlands. In this constraint, we do not require the uninterrupted rest time of H to be present in each window w in some window set $\mathcal{W}(t_0, \Delta t, W)$, but in the interval $[t_s^{\text{start}}, t_s^{\text{start}}) + W$ for each *assigned* shift s instead. In other words: if shift $s \in S$ is assigned, then there must be an uninterrupted rest with a duration of at least H hours in the interval $[t_s^{\text{start}}, t_s^{\text{start}} + W)$. For this constraint, we can use the variables ξ_{es}, $e \in E$, $s \in S$ again with Constraints (60). Let $S'(s) = \{s' \in S \mid [t_{s'}^{\text{end}}, t_{s'}^{\text{end}} + H) \subset [t_s^{\text{start}}, t_s^{\text{start}} + W)\}$. Then, we require that for each $(e, s) \in E \times S$ for which $x_{es} = 1$,

$$\sum_{s' \in S'(s)} \xi_{es'} \geq 1. \tag{64}$$

The equation above gives the constraint for the case $x_{es} = 1$. If $x_{es} = 0$, there should be no constraint on the ξ_{es} with $s \in S'(s)$. We can combine both cases in one constraint:

$$\sum_{s' \in S'(s)} \xi_{es'} \geq x_{es} \quad e \in E, s \in S. \tag{65}$$

When $x_{es} = 1$, Constraint (64) is recovered. If $x_{es} = 0$, the inequality is always satisfied as desired.

Availability Here, we enforce the requirement that employees are scheduled only within their availabilities, with the exception of at most N shifts per W time units. Let $S_e^{\text{unavailable}} \subset S^T$ be the set of shifts during which employee $e \in E$ is unavailable. Furthermore, let $S_e^{\text{prohibited}} \subset S_e^{\text{unavailable}}$ denote the set of shifts during which assignment is completely prohibited, even if scheduling outside of availabilities is theoretically allowed (for example: if employee e has vacation during shift s, $s \in S_e^{\text{prohibited}}$).

Then, we can add constraints

$$\sum_{s \in S_e^{\text{unavailable}} \setminus S_e^{\text{prohibited}}} x_{es} \leq N \quad \forall e \in E \tag{66}$$

and

$$x_{es} = 0 \quad e \in E, s \in S_e^{\text{prohibited}}. \tag{67}$$

This correctly restricts the assignment outside availabilities according to the specification of requirement type Availability. For $e \in E$, $s \in S_e^{\text{unavailable}} \setminus S_e^{\text{prohibited}}$, a properly priced penalty term $\beta_{es} x_{es}$ must be added to the objective function to discourage assignment outside of availabilities.

MinMaxHours Given and employee t_1, t_2 with $t_1 < t_2$ and a minimum and maximum number of hours H_e^{min}, H_e^{max} that may be worked between t_1 and t_2, requirement MinMaxHours can be enforced by constraint

$$H_e^{min} \leq \sum_{s \in S} x_{es} t_s^{[t_1,t_2)} \leq H_e^{max} \quad e \in E \tag{68}$$

with $t_s^{[t_1,t_2)}$ the net working time of shift $s \in S$ between t_1 and t_2.

MinMaxShifts Similarly, MinMaxShifts can be enforced by constraint

$$N_e^{min} \leq \sum_{s \in S} x_{es} \mathbb{1}_{s \cap [t_1,t_2) \neq \emptyset} \leq N_e^{max}. \tag{69}$$

AssignedShifts This constraint makes sure that employees' fixed shifts are assigned to them. Let f_{es} be a binary parameter that is equal to 1 if and only if employee $e \in E$ must be assigned shift $s \in S^T$. Then we must enforce

$$x_{es} \geq f_{es} \quad s \in S^T, e \in E. \tag{70}$$

BuildingCapacity Given a maximum number of people N, constraint BuildingCapacity can be written as

$$\sum_{s \in S(t)} \sum_{e \in E} x_{es} \leq N \quad t \in [0, T), \tag{71}$$

where $S(t) = \{s \in S \mid t \in [t_s^{start}, t_s^{end})\}$. This constraint does not have to be enforced on all $t \in [0, T)$. One can partition $[0, T)$ into intervals $[t_i, t_{i+1})$ for $i \in \{1, \dots n\}$ with $n \in \mathbb{N}$ such that $t_s^{start}, t_s^{end} \notin [t_i, t_{i+1})$ for all $s \in S$ and $i \in \{1, \dots, n\}$. Then, it suffices to enforce the constraint on one arbitrary time instant per time interval $[t_i, t_{i+1})$.

MinMaxHoursSoft This is a soft constraint encouraging a minimum number of hours H_{ew} to be assigned in a set of time windows. This can be done using the following constraint:

$$H_{ew} - v_{ew} \leq \sum_{s \in S} x_{es} t_{sw}^{net} \quad w \in \mathcal{W}(t_0, \Delta t, W), \tag{72}$$

where $v_{ew} \geq 0$ is a violation variable, to be penalized in the objective function and t_{sw} is the net working time of shift s in windows w.

MinMaxShiftsSoft This is a soft constraint encouraging a minimum number of hours N_{ew} to be assigned in a set of time windows. This can be done using the following constraint:

$$N_{ew} - v_{ew} \leq \sum_{s \in S} x_{es} \quad w \in \mathcal{W}(t_0, \Delta t, W), \tag{73}$$

where $v_{ew} \geq 0$ is a violation variable, to be penalized in the objective function.

PreferredShift If shift $s \in S^T$ is a preferred shift of employee $e \in E$, a term $-\epsilon_{es} x_{es}$, with $\epsilon_{es} > 0$ is added to the objective function.

RequestedHoursFairDivision If each employee $e \in E$ requests a number of hours H for the planning period $[0, T)$, it can be important for employee satisfaction that any hour assignment shortages u_e, defined by

$$u_e = \max\{H_e - \sum_{s \in S^T} t_s^{\text{net}} x_{es}, 0\}, \tag{74}$$

are divided fairly.

Definition 4.3. Given an assignment $\boldsymbol{x} \in \{0,1\}^{E \times S}$ (assuming $E \neq \emptyset$), we call $\boldsymbol{u} \in \mathbb{Z}_+^E$ with u_e given by Equation (74) for $e \in E$ the corresponding *hour assignment shortage distribution* (or *shortage distribution*), for short.

Definition 4.4. We say that shortage distribution $\boldsymbol{u}^1 \in \mathbb{Z}_+^E$ is more *fair* than $\boldsymbol{u}^2 \in \mathbb{Z}_+^E$ if $\text{Var}(\boldsymbol{u}^1) \leq \text{Var}(\boldsymbol{u}^2)$, with $\text{Var}(\boldsymbol{u})$ the population variance, given by

$$\text{Var}(\boldsymbol{u}) = \frac{1}{|E|} \sum_{e \in E} (u_e - \bar{u})^2, \tag{75}$$

with $\bar{u}$ the population average

$$\bar{u} = \frac{1}{|E|} \sum_{e \in E} u_e. \tag{76}$$

It follows directly from Definition 4.4 that maximizing fairness is equivalent to minimizing $\text{Var}(\boldsymbol{u})$. Since $\text{Var}(\boldsymbol{u})$ is a non-linear function, the question is now how this should be done within the MILP framework. We can make use of a few observations. Firstly, it is an elementary result from probability theory (see for instance Grimmett and Welsh, 2014) that

$$\frac{1}{|E|} \sum_{e \in E} (u_e - \bar{u})^2 = (\frac{1}{|E|} \sum_{e \in E} u_e^2) - \bar{u}^2. \tag{77}$$

Moreover, we assume that the total number of assigned shifts $\sum_{e \in E} \sum_{s \in S^T} x_{es}$ is constant since it is primarily determined by the amount of workload that needs to be done.

Proposition 4.5. Given a total number of assigned shifts H and a requested number of shifts H_e, the fairest shortage distribution with a total of H assigned hours is the one that minimizes $\sum_{e \in E} u_e^2$.

Proof. Follows directly from Definition 4.4 and Equation (77). □

Observe that the objective term of minimizing $\sum_{e \in E} u_e^2$ can be approximated arbitrarily closely by linearization with a finite number of equations. Indeed, one can create an ordered list $(\Delta H_0, \Delta H_1, \Delta H_2, \ldots, \Delta H_I)$ with $I \in \mathbb{N}$ and require that

$$\begin{aligned}
\min & \sum_{e \in E} v_e \\
& t_e = \sum_{s \in S^T} t_s^{\text{net}} x_{es} && e \in E \\
& v_e \geq (H_e - \Delta H_i) + g_i(t_e - (H_e - \Delta H_i)), && e \in E, i \in [I] \\
& v_e \geq 0 && e \in E,
\end{aligned} \tag{78}$$

where

$$g_i = \frac{(H_e - \Delta H_i)^2 - (H_e - \Delta H_{i-1})}{(H_e - \Delta H_i)^2 - (H_e - \Delta H_{i-1})}. \tag{79}$$

The set of equations above can be added to the scheduling problem as desired, where the objective term $\sum_{e \in E} v_e$ should be multiplied by some weight (or employee-dependent weights).

Remark 4.6. If $\Delta H_0 = 0$, there will always be a penalty enforced for an hour shortage.

Remark 4.7. If $\Delta H_0 = 0$ and $\Delta H_I \geq \max_{e \in E}\{H_e\}$, the linearization will always overestimate the underlying quadratic penalty function.

Remark 4.8. Finally, if $t_s^{\text{net}} = t^{\text{net}}$ for all $s \in S^T$ and some $t^{\text{net}} \geq 0$, and H_e is an integer multiple of t^{net} for all $e \in E$, one can choose $\Delta H_i = it^{\text{net}}$. Then, on all feasible values of t_e, the linearization is equal to the quadratic penalty function.

4.1.3 Alternative decision variables

Instead of working with decision variables x_{es}, $e \in E$, $s \in S$, one could also work with variables $\chi_{e\sigma}$ with $\sigma \in \mathcal{S}(e)$, where $\mathcal{S}(e)$ is the set of feasible schedules for employee $e \in E$. The advantage is that most of the constraints specified at the beginning of this chapter can then be removed from the problem since they are satisfied by the requirement that σ is a feasible schedule. In the other constraints, x_{es} can then be substituted by $x_{es} = \sum_{\sigma \in \mathcal{S}(e)} \alpha_{s\sigma}\chi_{e\sigma}$ where $\alpha_{s\sigma} = 1$ if a shift s is worked in schedule σ and 0 otherwise.

This substitution would reduce the number of constraints, but increase the number of variables since the number of feasible schedules is $O(2^{|S^T|})$, so in general exponential in the number of shifts. Moreover, it is not easy to characterize all of the feasible shifts (since they have to satisfy a subset of the constraints listed at the beginning of this chapter). However, explicit enumeration of all the feasible schedules is not necessary if the problem is solved using a column generation approach. Here, the problem is initialized with a small subset of the variables and schedules, after which new schedules with the lowest reduced costs are generated and added iteratively.

4.2 Second-stage problem: create an efficient schedule

In this section, we describe how we compute the performance (i.e. the costs associated with underplanning and overplanning) when using the first scheduling concept considered in this thesis: scheduling based on workload packages.

The full second stage problem is more complex and is therefore built up in four steps. First, we construct a model that works under the following simplifying assumptions:

1. there is only one workload package;
2. there is only one skill, i.e. even if there are multiple workload packages, all employees can work on them;
3. all parameters are deterministic.

Then, one by one and in order, the assumptions are removed and the model is generalized accordingly.

4.2.1 One workload package, one skill, deterministic parameters

In this section, the model is explained in the very simple setting of a single workload package that needs to be processed. The idea is to model the problem as a multi-commodity flow problem, in which the workload moves through a graph. Most of the nodes in the graph represent time intervals within the planning period, during which work can be processed. To this end, we must first define a suitable partition of the planning period.

Planning period partition We define a grid of points in time on which a relevant change can happen. Let P be the set of workload package time intervals of the form $[t_p^{\text{release}}, t_p^{\text{deadline}})$ and S be the set of shifts. Let $B(s)$ be the full set of break intervals of shift $s \in S$, i.e.

$$B(S) = \{[t_{sbj}^{\text{break,start}}, t_{sij}^{\text{break,start}} + t_{sb}^{\text{break}}) \mid j \in [n_{sb}], b \in [m_s], s \in S\}. \tag{80}$$

Here, $m_s \in \mathbb{N}$ is the number of breaks an employee takes in shift $s \in S$ (if assigned), and n_{sb} is the number of available starting moments for break b in shift s.

The grid is defined as follows:

$$\mathcal{T}^{grid} = \{\partial s \mid s \in S\} \cup \{\partial p \mid p \in P\} \cup \{\partial b \mid b \in B(s), s \in S\} \cup \{0, T\}, \tag{81}$$

where ∂X denotes the set of boundary points of set X. At each of the points $t \in \mathcal{T}^{grid}$, any of the following events can happen:

- the number of people working may change;
- workload may be released;
- workload may be lost.

However, in the intervals between consecutive points in $\mathcal{T}^{grid}$, there are no changes in the employees present and there is no workload released or due. This is convenient for computing how much workload is processed and how much is lost. In the remainder of this chapter, we index the intervals by $I = \{1, 2, \ldots, |\mathcal{T}^{grid}|\}$: so τ_i is the i-th interval in $\mathcal{T}^{grid}$.

Model In the workload model, we assume that at any time, each (infinitesimal) work unit of the workload package is in one of four states:

- *not ready*, meaning that the workload has not been released yet and nobody can work on it;
- *available*, meaning that the workload is not yet done, but could be processed by employees with the correct set of skills;
- *finished*, meaning that the workload has been done,
- *lost*, meaning that the workload could not be completed before the deadline.

The total workload is conserved throughout time.

Example 4.9. Suppose there is one workload package, which contains 10 hours of work. It is released at 10:00 and has a deadline of 14:00 on the same day. There are 2 employees, who do not have any breaks between 10:00 and 14:00. One of the crucial goals of the scheduling model is the minimize the amount of workload reaching the "lost" state. In Table 4.1, the states are listed for different times of the day assuming the lost workload and overplanning are minimized. Observe that the total workload is indeed conserved.

Table 4.1: *Optimal progression of the workload considered in Example 4.9 through the different states throughout the time range considered in the example. The release time and deadline time of the workload package are marked in bold.*

Time	**Workload in state (hrs)**				**Total workload (hrs)**
	Not ready	**Available**	**Finished**	**Lost**	
09:00	10	0	0	0	10
10:00	0	10	0	0	10
11:00	0	8	2	0	10
12:00	0	6	4	0	10
13:00	0	4	6	0	10
14:00	0	0	8	2	10
15:00	0	0	8	2	10

The model can be formulated as a flow problem (in this section still single-commodity) through a graph. We construct the directed graph $G = (V, A)$ as follows: let $V = \{\tau_i, i \in I\} \cup \{r\} \cup \{f, l\}$. The nodes f and l represent the finished and lost state respectively. The nodes $\{\tau_i, i \in I\}$ represent the time intervals in the planning period. Those can be used to process workload and keep track of the remaining available workload at each relevant point in time. Node r represents the source node of the workload from the workload package. The edge set A is constructed as follows:

$$A = \{(\tau_i, \tau_{i+1}) | i, i+1 \in I\} \cup \{(r, \tau_{\text{release}})\} \cup \{((\tau_i, f), i \in I)\} \cup \{(\tau_{\text{deadline}}, l)\}, \tag{82}$$

with τ_{release} and τ_{deadline} the time intervals starting at the release time and ending at the deadline time of the workload package respectively.

In this graph, the workload flowing from r to τ_{release} is the workload flowing from the state "not ready" to "available" at the release time. The workload flowing from arc τ_i to τ_{i+1} for some $i \in I$ is precisely the workload that is still in the state "available" at the end of interval τ_i. Any workload flowing from τ_i to f is the workload completed by employees during time interval i. The workload flowing from τ_{deadline} to l is the remaining workload that could not be completed by the deadline.

Per arc, a capacity is specified according to the following rules.

- For arcs (τ_i, τ_{i+1}), the capacity is ∞ if $\sup_{t \in \tau_i}\{t\} < t^{\text{deadline}}$ and 0 otherwise.
- For arcs (τ_i, f), the capacity is equal to the expected number of working people during τ_i (to be defined exactly later).
- For the arc into node l, the capacity is ∞.
- For the arc going out of node r, we do not define a capacity: instead, there is a fixed flow equal to the total amount of workload in the workload package.

Example 4.10 (Continuation of Example 4.9). Figure 4.1 shows the graph corresponding to Example 4.9, with a planning period from 9:00 to 15:00. Although the correct partition would be $\{[9{:}00, 10{:}00), [10{:}00, 14{:}00), [14{:}00, 15{:}00)\}$, a partition consisting of intervals of one hour is used for illustrative purposes.

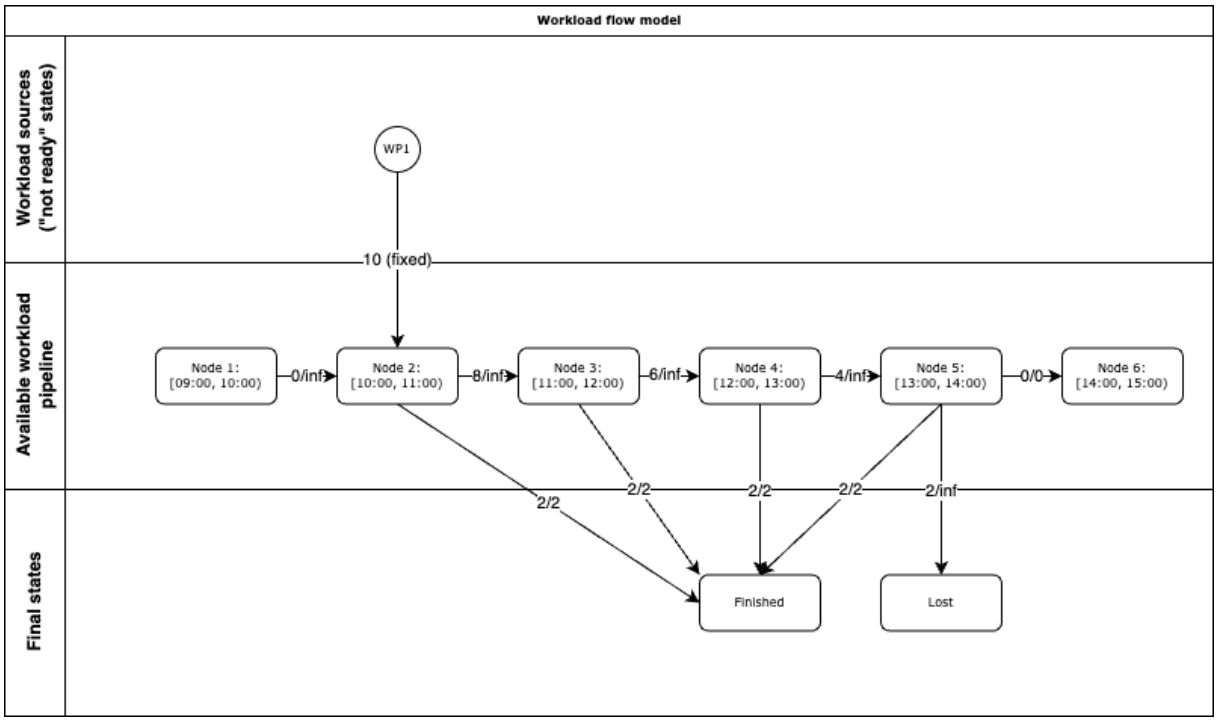


Figure 4.1: *The flow graph corresponding to the workload flow of Example 4.9 / Table 4.1. The node "WP1" corresponds to r, and the nodes "finished" and "lost" correspond to f and l respectively. The notation "x/y" means "flow/capacity", and "inf" means ∞.*

The model described above can be translated into a MILP (or an LP, if the schedule is given), consisting of five sets of constraints and an objective value. They are given below. All decision variables introduced are continuous and have a lower bound of 0 unless specified otherwise.

Firstly, the conservation of workload is modeled through the **conservation constraints**. Let w_i be the remaining available workload after time interval τ_i, $i \in I$. Let r_i be the amount of workload released (i.e. transitioned from the state "not ready" to "available") at the beginning of τ_i. Let f_i be the amount of workload processed during interval τ_i, i.e. the amount of workload flowing from node τ_i to f in the graph. Finally, let d_i be a binary parameter which is 1 if and only if τ_i ends in t^{deadline} and let l be the amount of workload lost. Then the conservation constraints are:

$$w_i = w_{i-1} + r_i - f_i - d_i l \quad i \in I, \tag{83}$$

with $w_0 = 0$ by convention.

Secondly, the capacity restrictions on the amount of workload processed in τ_i are modeled through the **capacity constraints on workload**. Let c_i denote the capacity for processing workload expressed as the number of people working during τ_i. Then the constraints are:

$$f_i \leq \mu(\tau_i) c_i \quad i \in I, \tag{84}$$

with $\mu(\tau_i)$ the Lesbesgue measure of τ_i. Note that the product $\mu(\tau_i)c_i$, $i \in I$ are the capacities referred to in Figure 4.1.

Thirdly, the capacity variables are restricted by the number of people present, modeled through the **capacity constraints by people**. The workload processing capacity in any interval τ_i cannot exceed the number of scheduled people during that time interval, reduced by the expected no-show rate and reduced by the portion of people that are taking a break during τ_i. Let $\pi_s^{\text{no-show}} \in [0, 1]$ denote the forecasted portion of people who will not show up, even though

they are scheduled to work in shift $s \in S$. Let π_{si}^{break} denote the portion of people working in shift $s \in S$ that has a break during interval τ_i. Appendix B explains how this is modeled in the experiments of Chapter 6. The constraints are:

$$c_i \leq \sum_{s \in S} \mathbb{1}_{\tau_i \subset S}(1 - \pi_{si}^{\text{break}})(1 - \pi_{si}^{\text{no-show}}) \sum_{e \in E} x_{es} \quad i \in I. \tag{85}$$

Recall that x_{es} is the binary first-stage decision variable indicating whether employee $e \in E$ is assigned to shift $s \in S$. To abbreviate notation, we will use the notation $\pi_{si}^{\text{working}} = (1 - \pi_{si}^{\text{break}})(1 - \pi_{si}^{\text{no-show}})$ and $S(\tau_i) = \{s \in S \mid \tau_i \subset S\}$. This yields:

$$c_i \leq \sum_{s \in S(\tau_i)} \pi_{si}^{\text{working}} \sum_{e \in E} x_{es} \quad i \in I. \tag{86}$$

One note: it looks like Constraints (84) and Constraints (86) could be combined into one constraint. However, when the model is generalized to a multi-skill model (Section 4.2.3, this is no longer true. Therefore, those constraints are kept separate for now.

Fourthly, there are **overplanning constraints**, modeling how many human hours go wasted. Overplanning is defined as the number of human hours which employees spend without processing workload. It can be modelled through the following constraint:

$$o_i = \mu(\tau_i) \sum_{s \in S(\tau_i)} \pi_{si}^{\text{working}} \sum_{e \in E} x_{es} - f_i \quad i \in I. \tag{87}$$

Fifthly, it is necessary to impose the workload deadline, i.e. to ensure that all workload is either in the state "finished" or "lost" at t^{deadline}. As a consequence of the conservation constraints, it suffices to require that the remaining available workload is 0. The equations modeling this requirement will be referred to as the **deadline constraints**:

$$w_{i^{\text{deadline}}} = 0, \tag{88}$$

where i^{deadline} is the unique $i \in I$ satisfying $\sup\{\tau_i\} = t^{\text{deadline}}$.

Finally, the **objective** is to minimize the costs associated to lost workload and overplanning. Let λ_i^o denote the cost per hour of overplanning in τ_i. Moreover, let λ^l denote the cost per hour of lost workload. Then the objective is:

$$\min \lambda^l l + \sum_{i \in I} \lambda_i^o o_i. \tag{89}$$

In the next section, the model is generalized to work for multiple workload packages.

4.2.2 Multiple workload packages, one skill, deterministic parameters

This section contains a straightforward generalization of the preceding model to a situation with multiple workload packages. Instead of a single-commodity flow problem, a multi-commodity flow problem must be modeled. This simply means keeping track of the individual flows of workload from the different workload packages. Many decision variables and parameters get an additional subscript p, representing the workload package $p \in P$ the variable/parameter corresponds to. Due to the straightforward nature of the generalization, we give the resulting model immediately without further explanation.

$$
\begin{aligned}
\min \quad & \sum_{p\in P} \lambda_p^l l_p + \sum_{i\in I} \lambda_i^o o_i \\
\text{s.t.} \quad & w_{pi} = w_{p(i-1)} + r_{pi} - f_{pi} - d_{pi} l_p && p \in P, i \in I, \\
& \sum_{p\in P} f_{pi} \le \mu(\tau_i) c_i && i \in I, \\
& c_i \le \sum_{s\in S(\tau_i)} \pi_{si}^{\text{working}} \sum_{e\in E} x_{es} && i \in I, \\
& o_i = \mu(\tau_i) \sum_{s\in S(\tau_i)} \pi_{si}^{\text{working}} \sum_{e\in E} x_{es} - \sum_{p\in P} f_{pi} && i \in I, \\
& w_{pi^{deadline}} = 0 && p \in P.
\end{aligned} \tag{90}
$$

In the next section, the model will be generalized further to accommodate a setting where different workload packages require different skills.

4.2.3 Multiple workload package, multiple skills, deterministic parameters

Now, we assume that there is a set of skills K. Each employee has some subset of those skills. We shall first impose constraints on how the available employees can be distributed over the workload requiring skills $1, \ldots, |K|$. Let E_k denote the subset of employees who possess skill $k \in K$.

Assumption 4.11. Each workload package requires precisely one skill.

Remark 4.12. Note that this assumption is not restrictive. Suppose a workload package requires n skills $k_1, k_2, \ldots k_n \in K$, then we can define a new skill k' which represents the union of skills $k_1, \ldots, k_n$. Then $E_{k'} = \bigcap_{i=1}^n E_{k_i}$. Moreover, consider the situation where a workload package requires no special skill. Then we can introduce the skill k_0 which represents the "generalist" skill, with $E_{k_0} = E$.

To compute which work gets processed and which work gets lost, one must know how the human capacity is distributed. Let $E_k(\tau_i)$ denote the subset of employees with skill $k \in K$ who are assigned at the time interval τ_i. We introduce the decision variable $c_k(\tau_i)$ to denote the planned capacity (i.e. the number of employees) assigned in interval τ_i to work requiring skill $k \in K$. The goal is now to know how the total number of employees can be distributed over the work of different skills, without actually having to assign employees to individual tasks or work of a specific skill.

Definition 4.13. Given a set of employees E and a vector $\boldsymbol{c} \in \mathbb{Z}_+^K$, we that E *is* $\boldsymbol{c}$*-distributable* if there exists a subset $\tilde{E} \subset E$ which can be partitioned into sets $\{\tilde{E}_k \subset E_k, k \in K\}$ such that $|\tilde{E}_k| = c_k$ for all $k \in K$. In this case, we call $\boldsymbol{c}$ a *capacity distribution.*

We must have constraints to specify which capacity distributions are feasible given a schedule. Such a set of constraints can be derived using proposition 4.14.

Proposition 4.14. Suppose a set of skills K, and a set of employees E are given. Then, for any $\boldsymbol{c} \in \mathbb{Z}_+^K$, E is $\boldsymbol{c}$-distributable if and only if

$$
\sum_{k\in K'} c_k \le |\bigcup_{k\in K'} E_k| \quad \forall K' \subset K. \tag{91}
$$

Proof. Let K be a set of skills and E be a set of employees. Let $\boldsymbol{c} \in \mathbb{Z}_+^K$. We first prove that System (91) holds if E is $\boldsymbol{c}$-distributable. Assuming $\boldsymbol{c}$ is a capacity distribution, there exists a subset $\tilde{E} \subset E$ and a partition $\{\tilde{E}_k, k \in K\}$ of $\tilde{E}$ such that $\tilde{E}_k \subset E_k$ and $|\tilde{E}_k| = c_k$ for all $k \in K$. It follows that, given $K' \subset K$,

$$\sum_{k \in K'} c_k = \sum_{k \in K'} |\tilde{E}_k| = |\bigcup_{k \in K'} \tilde{E}_k| \leq |\bigcup_{k \in K'} E_k|, \tag{92}$$

where the second equality follows from the fact that different sets in a partition are mutually disjoint. We conclude that $\boldsymbol{c}$ satisfies System (91) if E is $\boldsymbol{c}$-distributable.

Now, we prove the reverse: that $\boldsymbol{c}$ satisfying System (91) implies that E is $\boldsymbol{c}$-distributable. This can be proven by formulating the problem as an integral max-flow problem. Let $G = (V, A)$ be a directed graph with $V = \{s, t\} \cup E \cup K$, with s and t denoting a start and end node of the graph. Let $A = \{(s, e), e \in E\} \cup \{(e, k) \in E_k \times K\} \cup \{(k, t), k \in K\}$. Let the capacity γ_{uv} of arc $(u, v) \in A$

$$\gamma_{uv} = \begin{cases} c_k & u = k, v = t \\ 1 & \text{else} \end{cases}. \tag{93}$$

Note that an integral s-t flow through this graph is equivalent to partitioning some subset $\tilde{E}$ into $\tilde{E}_k \subset E_k$ for $k \in K$: a flow of 1 from s to e means that $e \in \tilde{E}$, and a flow of 1 from e to k means that $e \in \tilde{E}_k$. By the conservation of flow, e is in precisely one set $\tilde{E}_k$ if the flow from s to e is 1. The fact that $\tilde{E}_k \subset E_k$ follows from the construction of the arc set. It now suffices to show that an integral flow exists where $f_{kt} = c_k$ for all $k \in K$. Now observe the following facts (1) the arc capacities are integral, so the integrality requirement of the flow can be relaxed: given a maximum s-t flow (not necessarily integral), there exists an integer s-t flow with the same total value (corollary 6.2.3, Diestel, 2017); (2) since $(V \backslash \{t\}, \{t\})$ is an s-t cut with capacity $\sum_{k \in K} c_k$ (which is an upper bound on the maximum flow), showing that a flow of total value $\sum_{k \in K} c_k$ from s to t exists is enough to prove that there exists a flow with $f_{kt} = c_k$ for all $k \in K$.

We will prove that the maximum s-t flow has value $\sum_{k \in K} c_k$ using the max-flow min-cut theorem (theorem 6.2.2, Diestel, 2017). Since we already found a cut of this capacity, it remains to show that this is there is no cut with a smaller capacity.

Let $\gamma(\delta^{\text{out}}(S))$ denote the capacity of the arcs going out of node subset S. Since an s-t cut consists of a set S and $T = V \backslash S$ with $s \in S$, $t \notin S$, finding the minimum capacity s-t cut then comes down to solving

$$\min_{K' \subset K} \min_{E' \subset E} \gamma(\delta^{\text{out}}(\{s\} \cup E' \cup K')). \tag{94}$$

We first show that $\sum_{k \in K} c_k$ is a lower bound on the inner minimization problem. Let $K' \subset K$ be given. All arcs going out of K' go from S into T and their capacities are $\sum_{k \in K'} c_k$. For each $e \in E$, we can choose to include e in S or not. In case e is not included, $(s, e) \in \delta^{\text{out}}(S)$ which has a capacity of 1. All arcs going out of e are not elements of $\delta^{\text{out}}(S)$. In the other case, where e is included in S, $(s, e) \notin \delta^{\text{out}}(S)$. Furthermore, $(e, k) \in \delta^{\text{out}}(S)$ if and only if $k \notin K'$. So if e is included in S, the contribution to the cut capacity is $\sum_{k \in K \backslash K'} \mathbb{1}_{e \in E_k}$. It follows that excluding e from S if $e \in E_k$ for some $k \in K \backslash K'$ and including e otherwise minimizes the cut capacity. The $e \in E$ that are included in S have only outgoing edges into K', so both its ingoing arc and its outgoing arcs do not contribute to the cut capacity.

Consequently,

$$\begin{aligned}\min_{E'\subset E}\gamma(\delta^{\text{out}}(\{s\}\cup E'\cup K')) &= \sum_{e\in\bigcup_{k\in K\setminus K'}E_k}\gamma_{se}+\sum_{k\in K'}\gamma_{kt}\\ &= \sum_{e\in\bigcup_{k\in K\setminus K'}E_k}1+\sum_{k\in K'}c_k\\ &= |\bigcup_{k\in K\setminus K'}E_k|+\sum_{k\in K'}c_k\\ &\geq \sum_{k\in K\setminus K'}c_k+\sum_{k\in K'}c_k\\ &= \sum_{k\in K}c_k,\end{aligned} \tag{95}$$

where the inequality follows from Equation (91). Since the result holds for arbitrary $K' \subset K$, $\sum_{k\in K} c_k$ is also a lower bound on Problem (94). Thus, the cut (S,T) with $S = V\setminus\{t\}$ and $T = \{t\}$ is indeed a minimum capacity cut, and the maximum integral s-t flow has total value $\sum_{k\in K} c_k$. This completes the proof. □

Now, we can use Proposition 4.14 to derive appropriate constraints limiting the amount of capacity that can be assigned to work requiring different skills. First, note that the available capacity at some time $t \in \mathcal{T}$ is determined by the number of people present and working at time t, not by the total workforce. This number can change on each $t_i \in \mathcal{T}^{grid}$, but not on the corresponding intervals τ_i. Given some $i \in [|\mathcal{T}^{grid}| - 1]$, the people present and working during τ_i are the ones that (1) are assigned to a shift which includes τ_i; (2) are not on a break during τ_i and (3) show up. Those are the people we can assign to work of skill k if they possess the right skill. Let $E_{K'} = \bigcup_{k\in K'} E_k$ for $K' \subset K$. The capacity constraints on people become:

$$\sum_{k\in K'} c_{ki} \leq \sum_{s\in S(\tau_i)}\sum_{e\in E_{K'}}\pi_{si}^{\text{working}}x_{es} \quad K'\subset K, i\in I, \tag{96}$$

where c_{ki} is the capacity assigned to work of skill $k \in K$ during time interval τ_i. From hereon, we no longer assume these capacities are integral since employees could work on different tasks during their time interval (Assumption 2.13).

Moreover, the capacity constraints on workload have to be adjusted slightly:

$$\sum_{p\in P_k} f_{pi} \leq \mu(\tau_i)c_i \quad k\in K, i\in I. \tag{97}$$

Note that the summation is now over P_k, which is the subset of the set workload packages P requiring skill $k \in K$.

All other constraints and the objective value can remain unchanged.

4.2.4 Multiple workload package, multiple skills, stochastic parameters

To generalize the problem to allow for stochastic parameters while keeping a MILP formulation, the model assumes that there is a finite number of scenarios to take into account. Let Ξ be the set of scenarios (the sample space), where $\boldsymbol{\xi} \in \Xi$ is some data vector containing one possible realization of all random data (e.g. the vector $\boldsymbol{\xi} = [\boldsymbol{r}\,\boldsymbol{\pi}^{working}]^T$ assumes that the amount of workload per workload package and the portion of people working in some time interval is stochastic data). Furthermore, let $(\Xi, \mathcal{P}(\Xi), \mathbb{P})$ be a probability space with $\mathcal{P}(\Xi)$ the powerset of $\mathbb{P} : \mathcal{P}(\Xi) \to [0,1]$ be a probability measure defined on the power set of Ξ, $\mathcal{P}(\Xi)$ (i.e. the

sigma algebra generated by the sample space Ξ). We shall use the notation $p_{\boldsymbol{xi}}$ as shorthand for $\mathbb{P}(\{\boldsymbol{\xi}\})$, so the probability of getting scenario ξ when sampling from Ξ.

The second-stage problem for one scenario $\xi \in \Xi$, given a schedule $\boldsymbol{x}$, is Problem 98. Each problem has its own set of variables (except $\boldsymbol{x}$ which is shared across all problems) and its own realization of random data. However, for the generality of the stochastic setting and brevity, we omit subscripts $\boldsymbol{\xi}$ from all of the variables and from the random data.

$$
\begin{aligned}
& Q(\boldsymbol{x}, \boldsymbol{\xi}) = \\
\min \quad & \sum_{i \in I} \left[\sum_{p \in P} (\lambda_p^l l_p) + \lambda_i^o o_i \right] \\
\text{s.t.} \quad & w_{pi} = w_{p(i-1)} + r_{pi} - f_{pi} - d_{pi} l_p && p \in P, i \in I \\
& \sum_{k \in K'} c_{ki} \leq \sum_{s \in S(\tau_i)} (\pi_{si}^{\text{working}}) \sum_{e \in E_{K'}} x_{es} && K' \subset K, i \in I \\
& \sum_{p \in P_k} f_{pi} \leq \mu(\tau_i) c_{ki} && k \in K, i \in I \\
& w_{pi^{\text{deadline}}(p)} = 0 && t_i \geq t_p^{\text{deadline}}, p \in P \\
& o_i = \mu(\tau_i) \sum_{s \in S(\tau_i)} (\pi_{si}^{\text{working}}) \sum_{e \in E} x_{es} - \sum_{p \in P_k} f_{pi} && i \in I \\
& c_{ki}, w_{pi}, f_{pi}, l_p, o_i \geq 0 && k \in K, p \in P, i \in I
\end{aligned}
\tag{98}
$$

The full, two-stage scheduling problem then becomes:

$$
\begin{aligned}
\min \quad & \text{Employee satisfaction terms} + \sum_{\boldsymbol{\xi} \in \Xi} p_{\boldsymbol{\xi}} Q(\boldsymbol{x}, \boldsymbol{\xi}) \\
\text{s.t.} \quad & x_{es} + x_{es'} \leq 1 && \forall e \in E, \forall (s, s') \in S^{overlap} \\
& \text{Scheduling constraints} \\
& x_{es} \in \{0, 1\} && \forall e \in E, \forall s \in S^T.
\end{aligned}
\tag{99}
$$

Finally, we prove that the scheduling problem has relatively complete recourse, which helps to simplify the solution process described in Chapter 5.

Proposition 4.15. The full scheduling problem with performance model (98) has relatively complete recourse.

Proof. The intuition is simple: given any schedule, it is always feasible to process zero workload. Formally, the solution $\boldsymbol{f} = \boldsymbol{0}$, $\boldsymbol{c} = \boldsymbol{0}$ admits feasible $\boldsymbol{l}$, $\boldsymbol{o}$ and $\boldsymbol{w}$. □

4.3 Uncertainty Models

The last parts of the model to be defined are the probability distributions underlying the uncertainty in workload and no-show rates. Probability models for uncertainty are not the focus of this thesis, so we will use a very simple probability model that is in practice good enough to get the desired result of a "balanced schedule", i.e. a schedule where any relative overplanning or underplanning is distributed as evenly possible over the different shifts. While this does not capture reality fully (for example, in shifts with lower demand you expect higher relative uncertainty and therefore a higher safety margin in the schedule), it is sufficient to demonstrate the effectiveness of taking uncertainty into account in the model.

4.3.1 Uncertainty in Workload

Based on expert knowledge within Picnic Technologies, we assume that the workload uncertainties in all workload packages are completely coupled as follows.

Assumption 4.16. The uncertainty in workload within a planning period can be modeled as follows: if P is the set of workload packages with t_p^{work}, $p \in P$, the workload, then $t_p^{\text{work}} = \bar{t}_p^{\text{work}} \cdot \delta^{\text{work}}$, with $\bar{t}_p^{\text{work}}$ the forecasted workload in workload packages $p \in P$ and δ^{work} a random variable with maximum likelihood at 1.

The assumption above implies that if the actual workload deviates from the forecasted workload, all workload packages have the same relative deviation from the forecast. This is often observed in practice since the main source of uncertainty within Picnic is the number of customers that will order in a given period - the division of those orders over different weekdays is quite consistent.

4.3.2 Uncertainty in no-show

In the experiments in Chapter 6, we do not consider any uncertainty in the no-show rates to keep the generation of scenarios simple. It is to be expected that taking into account uncertainty in the no-show rate will give roughly similar effects as uncertainty in workload. Consider the fact that the main purpose of scheduling is to match the people capacity and workload with each other as well as possible. If underplanning is more (less) expensive than overplanning, uncertainty in either the no-show forecast or workload will result in schedules with a positive (negative) safety margin on the forecasted people capacity.

However, we do give some suggestions here for how to model the no-show rate to facilitate future research and extensions to the implementation of the algorithm presented in this thesis.

A fundamental struggle to overcome is the dependency of the no-show rate (or the number of people that do not show up) on the number of people that are scheduled. Intuitively, the number of assigned people who are absent in a shift is an increasing function of the total assigned number of people. However, the no-show rate should be decreasing because of the law of large numbers. In any case, the stochastic MILP framework does not allow for stochastic inputs depending on decision variables. As a workaround for this, one could use an estimate of the workload to be done during a given day (or shift, or more general period) as a proxy for the number of human hours scheduled on that day. If $f_{\delta^{\text{work}}}$ is the probability density of δ^{work}, and if $\pi_s^{\text{no-show}}$ is the random variable representing the no-show rate of s, then a reasonable model would be:

$$f_{\pi_s^{\text{no-show}}}(\pi) = \int_{\mathbb{R}_+} f_{\pi_s^{\text{no-show}}}(\pi | t_s^{\text{work}} \delta) f_{\delta^{\text{work}}}(\delta) d\mu(\delta) \quad \pi \in [0,1] \tag{100}$$

where it is assumed that $f_{\pi_s^{\text{no-show}}}(\pi | t_s^{\text{work}} \delta)$ is the known probability distribution of the no-show rate given the estimated workload to be done in shift s t_s^{work} and a realization δ of the uncertainty factor δ^{work}.

Berger and Monahan (1974) suggest that the number of absent people can be modeled as a Poisson distribution. However, the Poisson distribution has support on $\mathbb{Z}_+$, which would mean that with positive probability, the number of absent scheduled people could exceed the total number of scheduled people. A more precise approximation of the real distribution would be the $\text{Bin}(n, p)$ with n the number of people scheduled (which could be approximated by the forecasted hours of workload in a shift divided by the length of a shift) and p the probability of absence. This should still be translated to a continuous distribution of the no-show rate supported on some subset of $[0, 1]$.

One could even use more advanced estimates by comparing the number of available hours (according to the contracts of employees) in a certain period to the forecasted workload to correct for the potential mismatch between pool size and workload. Furthermore, a stochastic model for the no-show rate may take any number of other relevant factors into account, such as the season (in winter, more people may be sick), the weather forecast, the day of the week, etc.

5 Methodology

The methods to solve the model proposed in Chapter 4 are explained in this chapter. Two methods are used: a naive MILP-solver approach and the L-shaped algorithm. The scope of this chapter and the remainder of this text is restricted to the workload package model as explained in Section 2.7.

5.1 Benders' Decomposition

This section explains how the L-shaped algorithm as introduced in Chapter 3.3 is applied to the scheduling problem explained in this text. Recall that the problem is formulated as a two-stage problem, where Problem (99) is the full problem, and Problem (98) is the second stage problem for a given realization of uncertain data. Because the problem has relatively complete recourse (Proposition 4.15), the second-stage problem is always feasible. Therefore a simple implementation of the L-shaped algorithm without feasibility cuts can be used.

To implement the L-shaped algorithm, we have to specify the master problem and the subproblems. The master problem is as follows:

$$\begin{aligned}
\min \quad & \text{Employee satisfaction terms} + \theta & & \\
\text{s.t.} \quad & x_{es} + x_{es'} \leq 1 & & \forall e \in E, \forall (s,s') \in S^{overlap} \\
& \text{Scheduling constraints} & & \\
& (\boldsymbol{e}^n)^T \boldsymbol{x} + \theta \geq \tilde{e}_n & & n \in N' \subset N \\
& x_{es} \in \{0,1\} & & \forall e \in E, \forall s \in S^T,
\end{aligned} \tag{101}$$

where the last inequality corresponds to optimality cuts: N is the full set of optimality cuts and N' is the subset included in the master problem at some point in the L-shaped algorithm. In general, $\boldsymbol{e}^n = \sum_{\xi \in \Xi} p_\xi (T^\xi)^T \hat{\boldsymbol{u}}^\xi$ and $\tilde{e}_n = \sum_{\xi \in \Xi} p_k (\boldsymbol{h}^\xi)^T \hat{\boldsymbol{u}}^\xi$, with T^ξ, $\boldsymbol{h}^\xi$ data of subproblem $\xi \in \Xi$, $\hat{\boldsymbol{u}}^\xi$ the optimal dual vector and p_ξ the probability of scenario $\xi \in \Xi$. The subproblem for scenario $\xi \in \Xi$ reads:

$$\begin{aligned}
\min_{\boldsymbol{y}^\xi \geq \boldsymbol{0}} \quad & \boldsymbol{q}^\xi \boldsymbol{y}^\xi \\
& W^\xi \boldsymbol{y}^\xi = \boldsymbol{h}^\xi - T^\xi \boldsymbol{x}.
\end{aligned} \tag{102}$$

For the particular problem solved in this thesis, the second-stage problem for scenario $\xi \in \Xi$ is:

$$\begin{aligned}
& Q(\boldsymbol{x}, \boldsymbol{\xi}) = \\
\min \quad & \sum_{i \in I} \left[\sum_{p \in P} (\lambda_p^l l_p) + \lambda_i^o o_i \right] & & \\
\text{s.t.} \quad & w_{pi} = w_{p(i-1)} + r_{pi}^{\boldsymbol{\xi}} - f_{pi} - d_{pi} l_p & & p \in P, i \in I \\
& \sum_{k \in K'} c_{ki} \leq \sum_{s \in S(\tau_i)} (\pi_{si}^{\text{working}}) \sum_{e \in E_{K'}} x_{es} & & K' \subset K, i \in I \\
& \sum_{p \in P_k} f_{pi} \leq \mu(\tau_i) c_{ki} & & k \in K, i \in I \\
& w_{pi^{\text{deadline}}(p)} = 0 & & t_i \geq t_p^{\text{deadline}}, p \in P \\
& o_i = \mu(\tau_i) \sum_{s \in S(\tau_i)} (\pi_{si}^{\text{working}}) \sum_{e \in E} x_{es} - \sum_{p \in P_k} f_{pi} & & i \in I \\
& c_{ki}, w_{pi}, f_{pi}, l_p, o_i \geq 0 & & k \in K, p \in P, i \in I
\end{aligned} \tag{103}$$

To make the notation a bit more readable, it is convenient to give each constraint its own dual vector: define

$$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{u}^{\text{conservation}} \\ \boldsymbol{u}^{\text{capacity}} \\ \boldsymbol{u}^{\text{processing}} \\ \boldsymbol{u}^{\text{deadline}} \\ \boldsymbol{u}^{\text{overplanning}}, \end{bmatrix} \tag{104}$$

where the five dual vectors are ordered in the same way as the constraints they correspond to. For adding a constraint $(\boldsymbol{e}^n)^T\boldsymbol{x}$ for some $n \in N$, we need to know e^n_{es} for each $e \in E$, $s \in S$. Note that only the first and fifth constraints have terms from the vector $\boldsymbol{x}$ in them. The weight of x_{es} in $\boldsymbol{e}^n$ is given by

$$\left(\sum_{\xi\in\Xi} p_\xi (T^\xi)^T \boldsymbol{u}^\xi\right)_{es} = -\sum_{\xi\in\Xi} p_\xi \sum_{i\in I(s)} \pi^{\text{working}}_{si}[u^{\xi,\text{overplanning}}_i + \sum_{K'\subset K|e\in E_{K'}} u^{\xi,\text{capacity}}_{K'i}]. \tag{105}$$

Furthermore, $\tilde{e}_n$ is given by

$$\sum_{\xi\in\Xi} p_\xi (\boldsymbol{h}^\xi)^T \boldsymbol{u}^\xi = \sum_{\xi\in\Xi} p_\xi \sum_{p\in P}\sum_{i\in I} r^\xi_{pi} u^{\xi,\text{conservation}}. \tag{106}$$

5.2 Stopping Criteria

To implement the L-shaped algorithm, two stopping criteria are required: one for the L-shaped algorithm itself and one for the MILP-solver used to solve the Master problem. For the L-shaped algorithm, we use a relative MIP-gap of 1% as the stopping criterion. For the MILP-solver, we use a dynamical MIP-gap as the stopping criterion, starting at 5% and reducing in the following manner:

$$\epsilon^{\text{MIP}}_{i+1} = \min\{\frac{1}{2}\frac{\bar{\text{UB}}_i - \bar{\text{LB}}_i}{\bar{\text{LB}}_i}, \epsilon^{\text{MIP}}_i\}(1-\gamma), \tag{107}$$

where ϵ^{MIP}_i is the stopping MIP-gap in iteration $i > 0$ (and e^{MIP}_0 as initial value), $\bar{\text{UB}}_i = \min_{j\in[i]}\{UB_j\}$ is the lowest upper bound found until iteration i (of the L-shaped algorithm, not of the MILP-solver), $\bar{\text{LB}}_i = \max_{j\in[i]}\{LB_j\}$ is the largest lower found until iteration i, and $\gamma \in [0, 1)$ is the "decay rate". Observe that the sequence is non-increasing, and decreasing if $\gamma > 0$. The decay rate helps prevent the algorithm get stuck with the same lower and upper bound in each iteration.

5.3 Solver choice

Both the naive MIP implementation of the scheduling model and the implementation of the L-shaped algorithm are done in Java 17, using Google OR-Tools 9.6 (Perron and Furnon, 2022). OR-Tools is a general interface to different MILP solvers, such as Gurobi and SCIP (Bestuzheva et al., 2021, which allows for easy comparisons of different solvers. Several other general solver interfaces exist, such as PuLP and Pyomo for Python. But PuLP and Pyomo do not interact directly with the APIs of the back-end solvers. Instead, they generate an LP file and feed that to the back-end solver (Roy et al., 2022, Hart et al., 2012). This introduces overhead and would also force a full restart of the solver in each iteration of the L-shaped algorithm. OR-Tools can interact directly with APIs of different back-end solvers, which in turn can keep a solution in memory and use it for a warm start after a modification of the problem.

As back-end solvers, we will compare SCIP/GLOP and Gurobi: a combination of widely-used open-source solvers and a widely-used commercial solver respectively. The reason we combine SCIP and GLOP is that SCIP does not allow the user to get duals from a solution. SCIP

is only an optimization framework that uses an LP-solver (such as GLOP) in the back end. Therefore, we will use GLOP to solve the Benders subproblems. In version 9.6 of OR-tools, the documentation of the Gurobi interface mentions that adding a linear constraint to a problem can be done without forcing a restart, but changing the bounds of a linear constraint will force a restart of the problem. The former has to be done to solve the master problem, and the latter must be done to solve a subproblem. But since the subproblem is only an LP (of manageable size) and not a MILP, we can assume that the additional time spent on solving the subproblems from scratch is not too bad. Moreover, the documentation mentions that the limitations are not due to Gurobi itself, but due to limitations in OR-Tools that are likely to be fixed in the future. The same limitations are not mentioned for SCIP.

We will use OR-Tools to compare the performance of SCIP and Gurobi while keeping in mind this slight difference in the advantage of SCIP.

5.4 Preventing infeasibilities

The scheduling algorithm as described so far is very vulnerable to mistakes in the input data. Mistakes in the input data could cause the entire scheduling problem to become infeasible, which is a major problem as MILP-solvers will then not return any schedule.

One of the most important things that can go wrong is the availability of employees. Consider the following example: in all countries in which Picnic is active at the time of writing, it is required to have a rest period of at least 11 hours in each period of 24 hours. In practice, this means that an employee may not be assigned to a morning shift (generally starting at 6:00) on the day directly following an assigned evening shift (ending after 21:00 in all countries). Now suppose an employee has to work three shifts in a certain week, and gives as availability: Monday evening, Tuesday morning, Tuesday evening, and Wednesday morning. Even though the employee is available for four shifts while only having to work three, it is not possible to make a feasible schedule: any schedule would violate the daily rest time rule.

Moreover, suppose an employee has a contract for 32 hours per week) and takes 16 hours of vacation in a certain week (e.g. the entire Monday and the Tuesday). Then, he or she hands in a reduced number of availabilities (which only suffice to make a schedule of at most 24 hours that week) and after that, the employee cancels the vacation plans. The employee must then be scheduled for 32 hours but is not available for enough shifts to do this.

Ideally, there should be protections in place in all involved systems to deal with such edge cases correctly. But since this can be quite hard, the scheduling algorithm should be robust against such errors in the people data.

Luckily, it is quite easy to protect the model against most issues in employee data. One can simply run the first-stage problem of the scheduling model (the one with the scheduling constraints, including all legal constraints) for each employee separately. This is equivalent to solving an instance of the scheduling problem with only one employee and no workload packages. This can be done extremely fast (i.e. milliseconds per employee). Any employees whose availabilities do not admit a feasible schedule will be flagged. This application of the scheduling algorithm will be referred to as the "feasibility checker".

The full flow of the scheduling algorithm is shown in Figure 5.1. After the input data is retrieved, the feasibility checker is applied to each employee. Employees whose availabilities do not admit a feasible schedule will be included in the algorithm, but with fixed schedules: that is, any already assigned shifts will remain in place, but no new shifts will be added. Furthermore, the scheduling constraints are not applied to this employee. Afterward, a list of those employees is returned. Their data should be fixed manually and they should be scheduled manually.

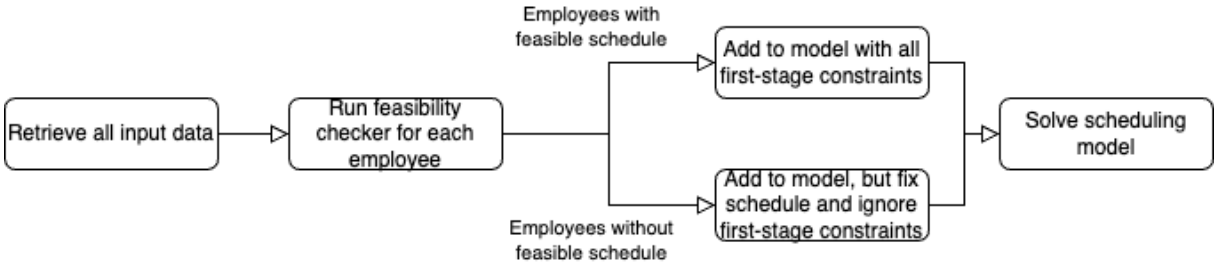


Figure 5.1: *Flow chart of the different steps in the scheduling algorithm, where special care has been taken to prevent infeasibilities.*

The steps above will not prevent all possible infeasibilities: constraints that cannot be separated by employees may still cause problems. For example, constraint O1-H (the one which prevents the presence of more than N people at the same time) could theoretically be configured to allow only ten people to work at the same time, even though there might be 500 people in the employee pool whose contracted hours need to be fulfilled. However, most scheduling constraints presented in Chapter 4 can be split up by employee, and the remainder can be configured sensibly to prevent infeasibilities.

6 Results

In this chapter, the performance of the algorithm proposed in Chapter 5 is evaluated in different test instances. The key metrics of evaluation are the run time and the optimality gap. We use real-world test cases from Picnic Technologies. Those test cases are useful for testing the practical applicability of the algorithm in the more specific setting at Picnic. Section 6.1 briefly explains how the test sets are sourced. In Section 6.2, the performance of the scheduling model and the solution approaches is evaluated.

6.1 Real-world test cases

The methodology described in this thesis is also evaluated on real-world data from Picnic Technologies. Test sets have been obtained from several fulfillment centers and from the customer success team.

The shifts, employee data, and workload packages are real. Only the uncertainty is generated arbitrarily because finding the real uncertainty distributions would require a lot of analysis that was not feasible within the scope of this thesis and is not very important for the demonstration of the methodology.

For fulfillment centers, the uncertainty factor applied to the workload was modeled as a discretized truncated normal distribution with mean 1 and standard deviation 0.1. The distribution was truncated and discretized from 0.7 to 1.3 with different numbers of scenarios in this range. For the CS dataset, the distribution was truncated between 0.2 and 1.8. The required range was determined experimentally by comparing the number of hours that can be worked by the employee pool to the required number of hours of work.

The general procedure for generating the probability distribution was as follows: let $n > 1$ be the number of scenarios and lb, ub be the minimum and maximum of the support respectively. As is conventional, let μ, σ be the mean and standard deviation of the normal distribution. Let $\Delta := \frac{\text{ub}-\text{lb}}{n-1}$. The scenarios are then

$$\delta_i^{\text{work}} = \text{lb} + \Delta i, \quad i \in \{0, 1, \dots, n-1\} \tag{108}$$

with probability

$$p_i = \frac{\phi_{\mu,\sigma}(\delta_i^{\text{work}} + \frac{1}{2}\Delta) - \phi_{\mu,\sigma}(\delta_i^{\text{work}} - \frac{1}{2}\Delta)}{\sum_{j=0}^{n-1}(\phi_{\mu,\sigma}(\delta_j^{\text{work}} + \frac{1}{2}\Delta) - \phi_{\mu,\sigma}(\delta_j^{\text{work}} - \frac{1}{2}\Delta))} \quad \{0, 1, \dots, n-1\}, \tag{109}$$

with $\phi_{\mu,\sigma}$ the cumulative distribution function of the $N(\mu, \sigma^2)$ distribution. For $n = 1$, we take $\delta_0 = 1$ with $p_i = 1$ by convention (the Dirac distribution with parameter 1).

One note on the employee data: all datasets considered in this chapter contain only employees with fixed-hour contracts. The datasets for locations allowing flexible-hour contracts (i.e. contracts that allow the company to choose the number of hours an employee works each week, within some range), were not yet available in the right format at the time of experiments.

6.2 Performance evaluation: run time

We compare the performance of two methods to solve the scheduling problem: (1) a MILP-solver; (2) the L-shaped algorithm. Furthermore, we compare two solvers: a commercial one (Gurobi) and an open-source one (SCIP / GLOP).

Since both methods can solve the problem to optimality (or as close as desired), it suffices to set a stopping criterion and compare the run times required to reach the criterion. Throughout

this section, we use a relative gap between the highest known lower and lowest known upper bounds of 1% as the stopping criterion. All experiments are capped at two hours for purposes of practicality. Only in case the two-hour limit is reached, we report the achieved optimality gap and compare solvers and methods by optimality gap. All experiments are done with an Apple M1 Pro with 10 cores (8 at 3220 Mhz and 2 at 2064 MHz) and 32 GB RAM. The parameters for the stopping criterion of the L-shaped algorithm (Equation (107)) are $\epsilon_0^{\text{MIP}} = 0.05$ and $\gamma = 0.01$.

Tables 6.1 and Table 6.2 show the run times results of a number of real-world test sets: the shifts, the workers, and the workload packages are real. Moreover, those table shows the "reported" and "true" optimality gaps. The reported optimality gap is simply the relative gap between the lower and upper bound reported by the solution method (the L-shaped algorithm or the MILP-solver). The "true" optimality gap refers to the relative gap between the best feasible solution returned by the tested solution method and the near-exact optimal solution (computed by applying the Gurobi MILP-solver with a 0.01% relative gap as a stopping criterion). The near-exact optimal solution may not be the true optimal solution, but it is so close will give a near-perfect approximation of the true optimality gap.

Table E.6 contains the real locations and planning periods corresponding to the test instances. The data is obtained from the systems of Picnic Technologies. The workload uncertainty distribution is the only artificial data: the workload distribution is modeled as the forecasted workload multiplied by a truncated discretized normal distribution with expectation 1 and a standard deviation of 0.3.

Table 6.1: *SCIP: run times and optimality gaps for a variety of test sets.*

Testset					Performance MILP-solver (SCIP)			Performance L-shaped algorithm (SCIP/GLOP)		
Name	**#Shifts**	**#Workers**	**#Workload Packages**	**#Scenarios**	**Runtime (s)**	**Reported Optimality Gap (%)**	**True Optimality Gap (%)**	**Runtime (s)**	**Reported Optimality Gap (%)**	**True Optimality Gap (%)**
1a	12	351	283	1	**3.853**	0.31	0.24	7200.000	∞	25.76
1b	12	351	283	13	**31.524**	0.37	0.37	595.142	1.00	0.52
1c	12	351	283	61	**296.918**	0.18	0.14	7200.000	∞	54.63
1d	12	351	283	601	7200.000	∞	373.55	**4934.243**	0.90	0.23
2a	12	459	266	1	**4.701**	∞	0.00	7200.000	12068.30	28.46
2b	12	459	266	13	**33.574**	0.58	0.57	232.698	0.99	0.27
2c	12	459	266	61	**227.629**	0.07	0.05	613.122	0.81	0.18
2d	12	459	266	601	7200.000	∞	304.66	**6088.361**	1.00	0.31
3a	12	328	274	1	**22.018**	0.20	0.00	7200.000	∞	446.53
3b	12	328	274	13	**40.532**	0.64	0.57	7200.000	∞	83.93
3c	12	328	274	61	**293.671**	0.06	0.01	7200.000	∞	102.84
3d	12	328	274	601	7200.000	∞	274.18	7200.000	∞	**71.30**
4a	12	227	268	1	**2.926**	0.18	0.13	7200.000	11725.60	4.89
4b	12	227	268	13	**30.084**	0.25	0.23	7200.000	10403.53	67.38
4c	12	227	268	61	**197.887**	0.04	0.01	7200.000	17723.18	41.01
4d	12	227	268	601	7200.000	∞	968.50	7200.000	21685.86	**72.51**
5a	12	227	248	1	7200.000	4.28	**0.00**	7200.000	66018.10	1910.52
5b	12	227	248	13	**17.874**	0.64	0.58	7200.000	52961.70	120.91
5c	12	227	248	61	**289.259**	0.06	0.00	7200.000	55540.66	135.80
5d	12	227	248	601	7200.000	∞	494.76	7200.000	60223.22	**155.96**
6a	12	57	220	1	**1.263**	0.01	0.00	7200.000	3344.28	3.11
6b	12	57	220	13	**11.581**	0.00	0.00	7200.000	∞	52.98
6c	12	57	220	61	**190.540**	0.49	0.46	7200.000	918.17	18.25
6d	12	57	220	601	7200.000	∞	455.28	7200.000	∞	**49.06**
7	12	447	227	13	**20.453**	0.31	0.28	105.087	1.00	0.42
8	12	316	249	13	**28.811**	0.29	0.24	57.962	0.60	0.22
9	12	222	248	13	**30.784**	0.54	0.49	97.355	1.01	0.18
10	12	217	235	13	**32.491**	0.14	0.11	60.472	0.84	0.25
11	12	46	198	13	**17.614**	0.92	0.75	7200.000	865.31	165.94
12	12	160	268	13	**14.472**	0.00	0.00	54.359	0.99	0.34
13	12	20	163	13	**6.280**	0.08	0.00	9.335	1.55	0.55
14	224	94	2688	13	7200.000	∞	∞	7200.000	1.94	**1.02**
15	224	114	2688	13	7200.000	∞	∞	7200.000	1.51	**0.80**

Table 6.2: *Gurobi: run times and optimality gaps for a variety of test sets. The test instances with 601 scenarios could not be solved with the L-shaped algorithm using Gurobi due to licensing limitations.*

Testset					Performance MILP-solver (Gurobi)			Performance L-shaped algorithm (Gurobi)		
Name	**#Shifts**	**#Workers**	**#Workload Packages**	**#Scenarios**	**Runtime (s)**	**Reported Optimality Gap (%)**	**True Optimality Gap (%)**	**Runtime (s)**	**Reported Optimality Gap (%)**	**True Optimality Gap (%)**
1a	12	351	283	1	**0,201**	0,00%	0,00%	7200,000	∞	61,28%
1b	12	351	283	13	**1,680**	0,83%	0,81%	99,038	0,97%	0,47%
1c	12	351	283	61	**10,960**	0,06%	0,00%	333,048	0,83%	0,22%
1d	12	351	283	601	**207,358**	0,07%	0,04%	NA		
2a	12	459	266	1	**0,234**	0,79%	0,79%	26,646	0,43%	0,22%
2b	12	459	266	13	**2,148**	0,36%	0,30%	122,801	0,99%	0,18%
2c	12	459	266	61	**12,989**	0,04%	0,00%	401,409	0,90%	0,24%
2d	12	459	266	601	251,890	0,03%	0,01%	NA		
3a	12	328	274	1	**0,171**	0,00%	0,00%	7200,000	963,98%	108,78%
3b	12	328	274	13	**1,527**	0,67%	0,57%	93,303	0,96%	0,33%
3c	12	328	274	61	**9,514**	0,22%	0,09%	7200,000	∞	140,42%
3d	12	328	274	601	181,274	0,08%	0,00%	NA		
4a	12	227	268	1	**0,183**	0,00%	0,00%	7200,000	7415,47%	33,32%
4b	12	227	268	13	**1,384**	0,30%	0,26%	73,633	2,93%	2,09%
4c	12	227	268	61	**8,155**	0,07%	0,01%	204,375	0,98%	0,33%
4d	12	227	268	601	142,542	0,06%	0,00%	NA		
5a	12	227	248	1	**0,153**	0,00%	0,00%	7200,000	24901,90%	660,26%
5b	12	227	248	13	**1,463**	0,74%	0,53%	98,855	0,97%	0,20%
5c	12	227	248	61	**7,776**	0,28%	0,05%	306,737	0,94%	0,15%
5d	12	227	248	601	145,614	0,19%	0,00%	NA		
6a	12	57	220	1	**0,088**	0,00%	0,00%	7200,000	∞	58,99%
6b	12	57	220	13	**0,624**	0,65%	0,52%	6118,249	0,97%	0,28%
6c	12	57	220	61	**5,256**	0,17%	0,09%	101,781	0,97%	0,16%
6d	12	57	220	601	64,511	0,10%	0,01%	NA		
7	12	447	227	13	**1,595**	0,12%	0,07%	67,824	0,92%	0,26%
8	12	316	249	13	**1,166**	0,39%	0,31%	62,712	0,95%	0,32%
9	12	222	248	13	**0,977**	0,92%	0,82%	38,974	0,89%	0,26%
10	12	217	235	13	**0,964**	0,68%	0,61%	45,081	0,99%	0,56%
11	12	46	198	13	**0,440**	0,44%	0,00%	21,040	0,72%	0,00%
12	12	160	268	13	**0,885**	0,05%	0,03%	41,838	0,99%	0,43%
13	12	20	163	13	**0,265**	0,00%	0,00%	5,410	0,74%	0,00%
14	224	94	2688	13	**35,067**	0,51%	0,50%	431,020	0,99%	0,30%
15	224	114	2688	13	**23.850**	0.16%	0.09%	65.621	0.99%	0.37%

When solving the scheduling problem with a MILP-solver, Gurobi outperforms SCIP by a lot The worst-case run-time of Gurobi over the 32 test instances was just over 4 minutes. SCIP did not manage to solve 8 of the 32 test instances to a gap of 1% within the time limit of two hours. SCIP did not find any feasible solution for test instance 15 at all. On average, Gurobi was a factor 28 faster (computed only over the 24 instances that SCIP could solve within the time limit). There was not a single instance where SCIP was faster than Gurobi.

The L-shaped algorithm was slower than the MILP-solver, except for the very large test instances solved with SCIP / GLOP When looking at Table 6.1, one can see that the L-shaped algorithm is very slow in general. In 21 of 32 cases, the algorithm did not even terminate within the 2-hour time limit. However, when looking at the test instances with 601 scenarios, the results are different: both the MILP-solver and the L-shaped algorithm did not finish within 2 hours, but the objective values produced by the L-shaped algorithm were much lower. See Figure 6.1. For test instances 14 and 15, which have a lot of binary variables due to the high number of workers and shifts, SCIP did not even find a feasible solution within two hours. This is quite remarkable, considering the fact that no cross-worker scheduling constraints were applied: each worker had its own independent set of scheduling constraints. It should therefore be quite easy to at least find a feasible solution by decomposing the first stage of the problem. The L-shaped algorithm found in two hours a solution which had an objective value of just 2% higher than the true optimal solution.

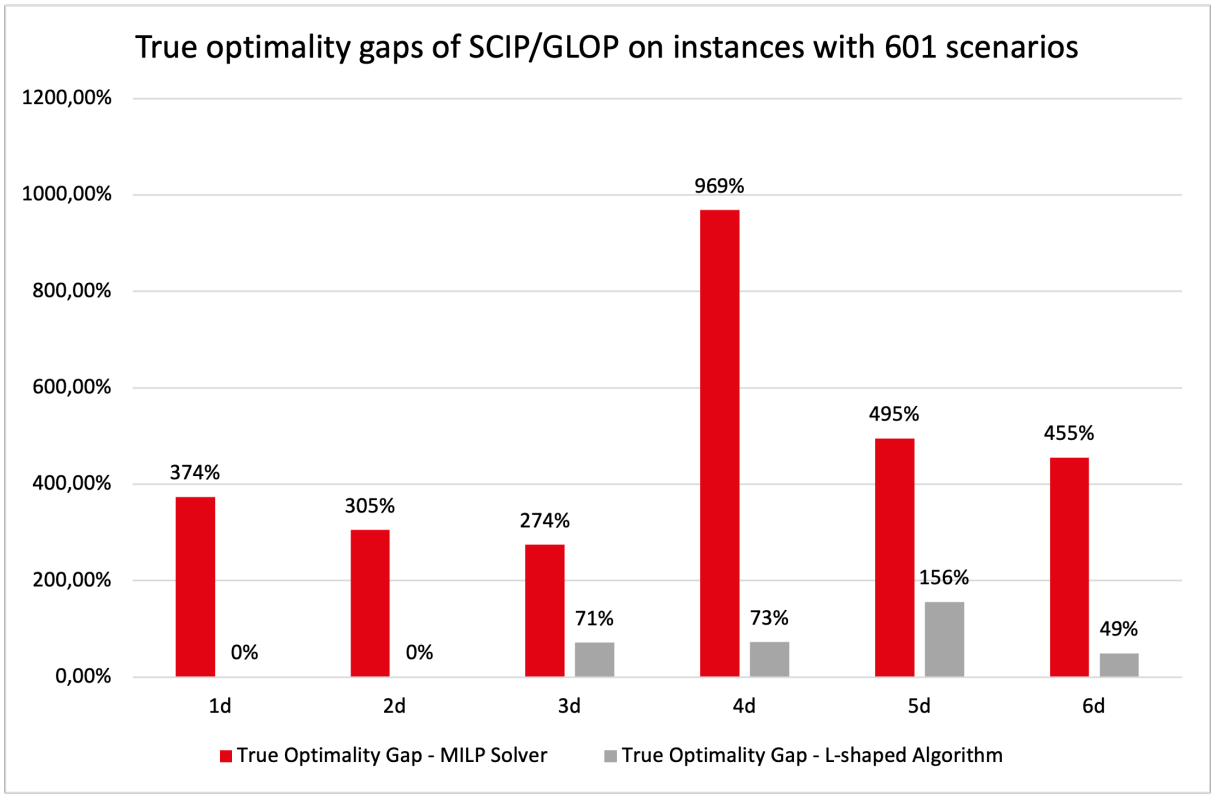
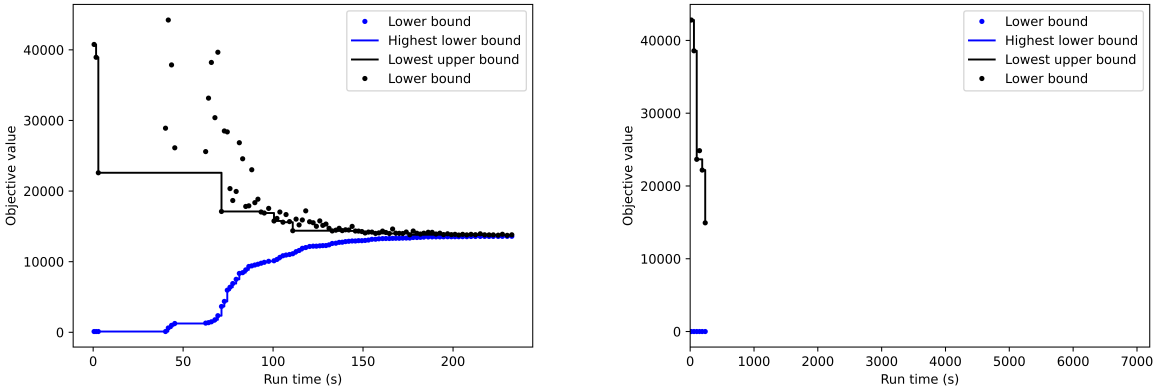


Figure 6.1: *True optimality gaps for instances 1d - 6d, obtained by applying the SCIP MILP-solver and the L-shaped-algorithm with SCIP / GLOP to the instances.*

When the L-shaped algorithm did not finish within 2 hours, this was usually because it got stuck in the master problem The L-shaped algorithm gets "stuck" on some of the datasets, without an obvious pattern in those datasets. On these datasets, the algorithm gets stuck in the Master problem phase where the MIP-solver runs very long without decreasing the optimality gap sufficiently. It seems that, adding a cut often yields a pathological MILP instance. There is, to the best of the author's knowledge, not much that can be done about that within the variant of the L-shaped algorithm proposed in this thesis. One could, however, use a different cut-generating routine or use a different variant of the algorithm (for example one where cuts are added in nodes of the branching tree).

Two examples of the progression of the solution process of the L-shaped algorithm (using SCIP/-GLOP) are shown in Figure 6.2. The first graph shows the progression of the L-shaped algorithm applied to test instance 2b. The lower and upper bounds on the objective value converge nicely to each other in a reasonable time. The second graph shows the progression for test instance 3d. The algorithm simply gets stuck in a master problem after about 4 minutes and spends the remainder of the two hours trying to solve that master problem.

Note: the objective values in Figure 6.2 are not the real objective values, but they are the real objective values multiplied by some arbitrary factor for reasons of confidentiality. This factor can be found in Table E.7.



(a) *Progression of instance 2b.*

(b) *Progression of instance 3d.*

Figure 6.2: *Upper and lower bounds on the objective value vs runtime, for two instances solved with the L-shaped algorithm using SCIP/GLOP.*

Only in case of a large number of binary variables or scenarios when using SCIP/GLOP, the L-shaped algorithm gives an advantage over the naive MILP-solver approach. The naive MILP-solver approach with Gurobi as the solver was optimal in all test instances.

6.3 Performance evaluation: the value of stochasticity

In this section, we compare the schedules of test sets generated without uncertainty in the workload taken into account (equivalently: a Dirac distribution with parameter 1) and those generated with uncertainty taken into account.

The procedure for comparison is simple:

1. generate a schedule without taking uncertainty into account;
2. generate a schedule with uncertainty taken into account;
3. solve the second-stage problem of step 2 with the schedule of step 1 used as the first-stage solution;
4. compare the objective values of steps 2 and 3.

The gap between the objective values of steps 2 and 3 is (approximately) the gain of taking uncertainty into account. It is only an approximation of the true gain because we always use a discretized approximation of the underlying continuous probability distribution.

We show in Table 6.3 the results for several different test sets, where the number of scenarios was varied per test set (while the underlying probability distribution and truncation bounds stayed exactly the same): for each test set, schedules were created with 1, 13, 61 and 601 scenarios. Then, all schedules were evaluated in the objective function of the instance with 601 scenarios, which is a very close approximation to the true expectation of the objective function. The gaps between the objective values of the schedules resulting from 1 / 13 / 61 and 601 scenarios are reported in Table 6.3. Moreover, Table 6.3 contains a column with "relative overplanning", meaning the percentual gap between the total expected working hours (the planned working hours decreased by the no-show rate) and the total workload.

Note there is one case (1c, $K = 61$) where the gap is negative, i.e., using fewer scenarios leads to better results. But this is likely due to the fact that the schedules were generated through a solver with a relative MILP-gap of 1% as the stopping criterion.

Table 6.3: *The gaps between the objective values of the schedules resulting from $K = 1$ / $K = 13$ / $K = 61$ and $K = 601$ scenarios, when all schedules are evaluated in the objective function of the test instance with 601 scenarios. The worst-case gaps are marked bold.*

Test Set					**Gap with K scenarios (compared to K = 601)**		
Name	**#Shifts**	**#Workers**	**#Workload Packages**	**Relative Overplanning (%)**	**K = 1 (%)**	**K = 13 (%)**	**K = 61 (%)**
1	12	351	283	14.2	**59.24**	0.54	-0.03
2	12	459	266	-10.3	48.88	1.18	0
3	12	328	274	2.7	7.24	**2.72**	**0.09**
4	12	227	268	13.2	53.03	0.58	0.01
5	12	227	248	0.6	1.99	1.90	0.04
6	12	57	220	30.9	0.43	0.43	0.07

It is evident from Table 6.3 that taking into account the uncertainty in the workload in the scheduling problem can make a big difference: on the problems tested, the schedules resulting from the deterministic scheduling problems ($K = 1$) yielded up to a 59% higher objective value than those resulting from the $K = 601$ test instances. But when multiple scenarios are used, even a few, the objective value (as evaluated in the objective function of the $K = 601$ instance) of the schedule improves dramatically: for $K = 13$, the worst-case gap over the different test sets was only 2.72%. For $K = 61$, the worst-case gap was a mere 0.09%. In conclusion: taking into account uncertainty, even very roughly, can have a big impact. And, at least based on this very small sample for this specific problem, it seems that increasing the precision has a diminishing return.

From Table 6.3, it is not clear though why the schedules of test sets 1, 2, and 4 are so much worse than those of 3, 5, and 6 when generated with one workload scenario. There seems to be a correlation with relative overplanning: a large (absolute value of) relative overplanning seems to lead to a higher stochasticity gap, with 6 as the exception. It is not clear though from the table if there is a causal link.

A closer inspection of the schedules gives more clarity. The particular datasets we tested have workload packages whose start and end times match up perfectly with those of the different shifts (a morning shift and an evening shift each day from Monday to Saturday). Because the workload packages and the shifts match precisely in time, it is possible to calculate the expected workload in each shift and the expected number of working hours (the planned number of people decreased by the forecasted no-show rate). We call the ratio of the expected number of working hours to the expected workload the "relative overplanning", expressed as a percentage. Figure 6.3 shows the relative overplanning per shift of test set 1, and Figure 6.4 shows the relative overplanning per shift of test set 3.

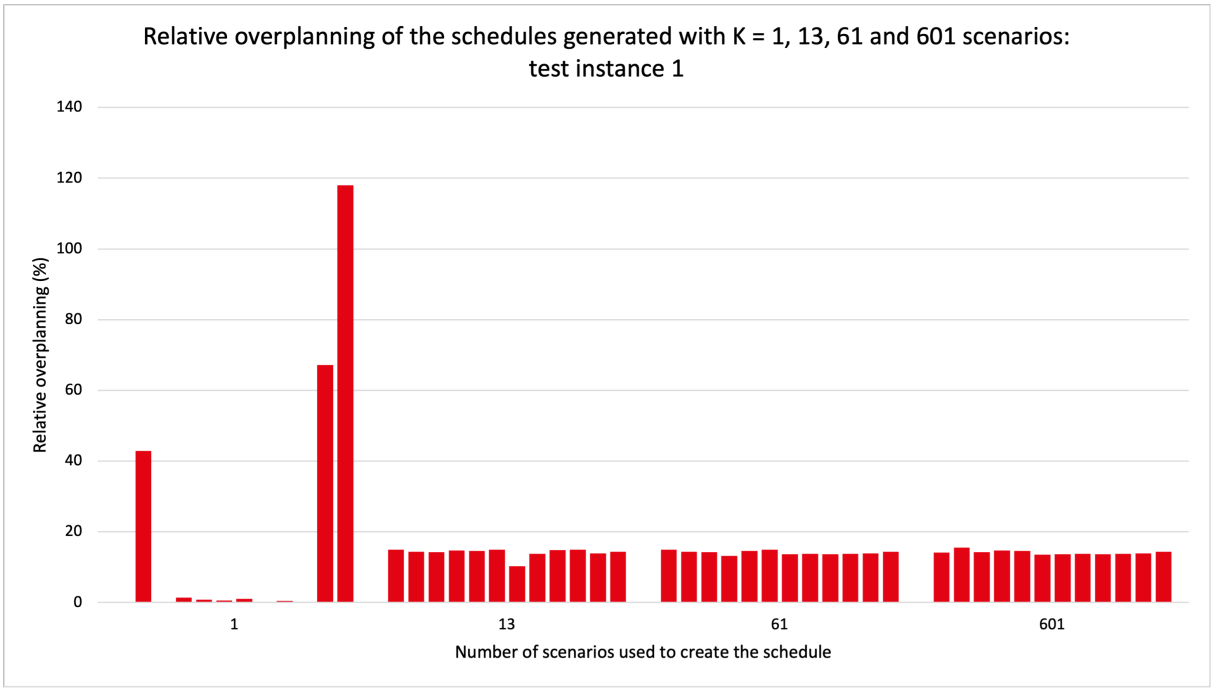


Figure 6.3: *The relative overplanning of the schedules from test set 1 generated with* $K = 1$, $K = 13$, $K = 61$, *and* $K = 601$ *scenarios respectively. Each scenario has 12 bars, each bar corresponds to the relative overplanning in one of the 12 shifts. The average overplanning is* 14.2%.

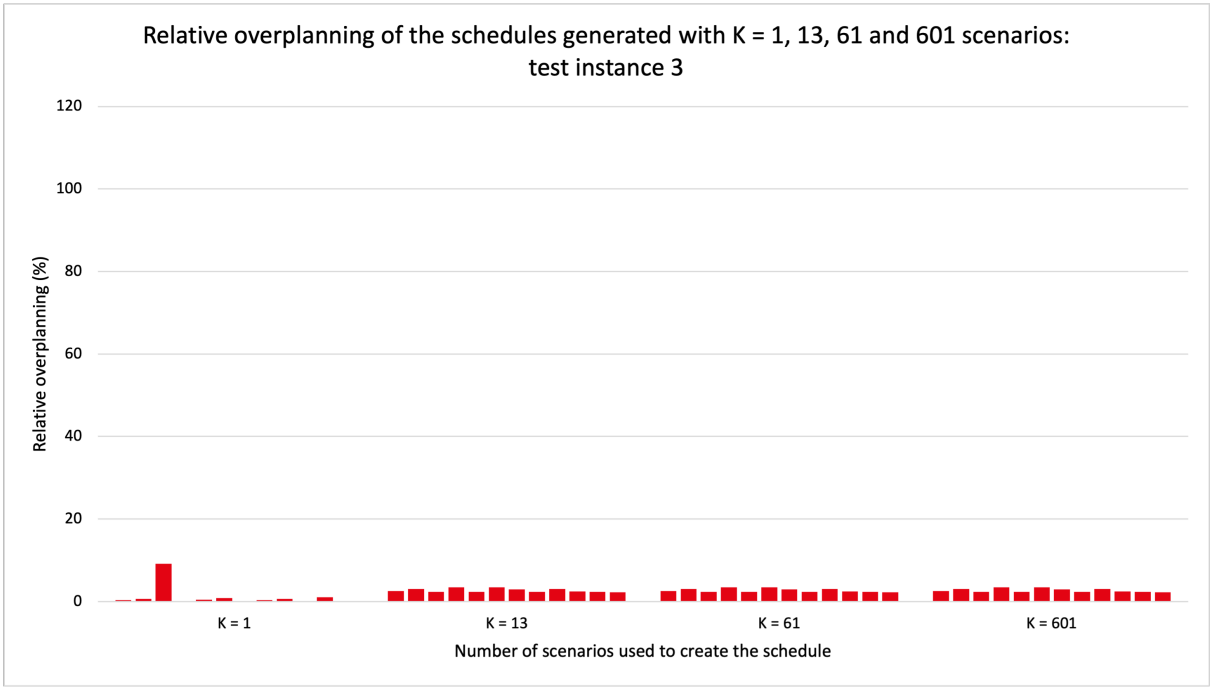


Figure 6.4: *The relative overplanning of the schedules from test set 3 generated with* $K = 1$, $K = 13$, $K = 61$, *and* $K = 601$ *scenarios respectively. Each scenario has 12 bars, each bar corresponds to the relative overplanning in one of the 12 shifts. The average overplanning is* 2.7%.

The major difference between the test cases is in the total overplanning. In test case 3, the

employee pool size matches the capacity needs very well: the total overplanning is only 1.3% (calculated using the $K = 1$ schedule). It makes sense that it is not so influential to take stochasticity into account: after all, the deterministic model still minimizes total under- and overplanning, which leads to a very balanced schedule that can meet the total workload forecast very well.

In test case 1, the total overplanning is quite high, around 14%. There are many different ways to divide this overplanning over different shifts, including very unbalanced ones. Ignoring uncertainty in workload can then result in a schedule in which overplanning is highly unbalanced. When it is taken into account that the workload can also be higher than the forecast (for example, 14% higher), the scheduling model will balance out the relative overplanning: after all, for the scenarios where the forecast is around 14% higher than expected, it makes sense to have a schedule where each shift is around 14% overplanned.

Test case 6 is an exception to the pattern. The total overplanning is about 31%, but using more scenarios barely leads to improvement. This is actually due to the truncation of the workload uncertainty distribution: the most extreme workload scenarios considered are 30% below and above the forecast. If the number of expected hours is above the highest workload scenario or below the lowest workload scenario, the schedule can no longer be balanced: even if 601 scenarios are used. From Table 6.3, it looks like all schedules are equally good. However, in reality, all schedules are equally bad. This can be seen in Figure 6.5. None of the schedules have balanced overplanning.

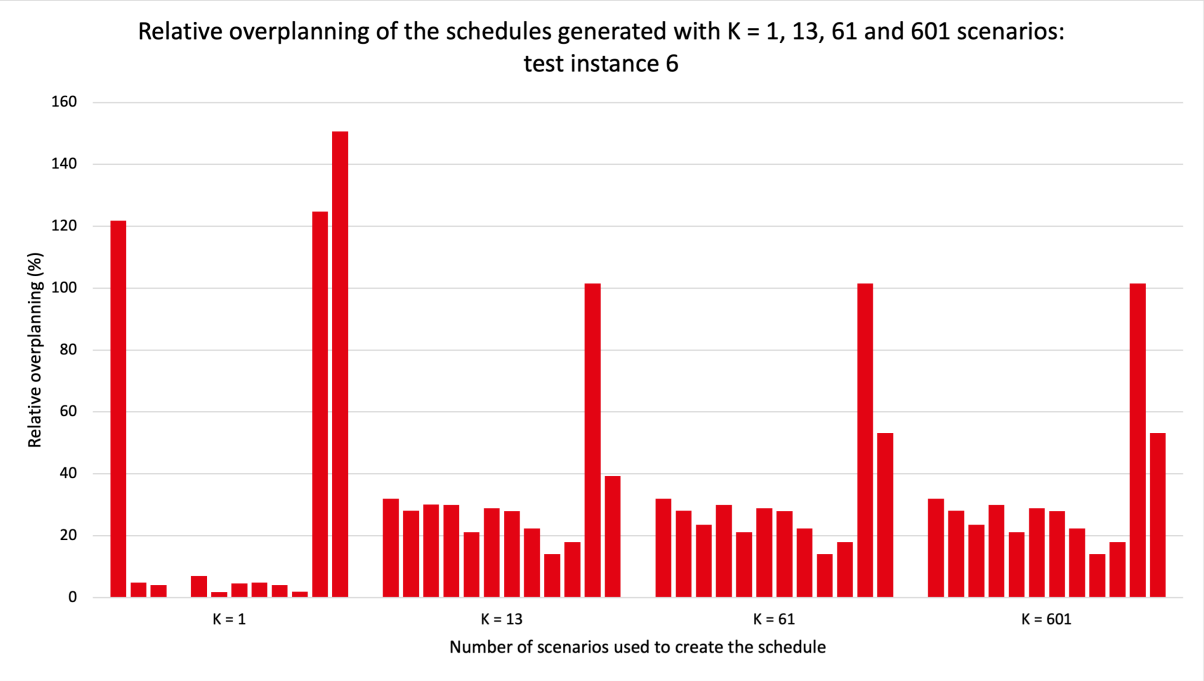


Figure 6.5: *The relative overplanning of the schedules from test set 3 generated with* $K = 1$, $K = 13$, $K = 61$, *and* $K = 601$ *scenarios respectively. Each scenario has 12 bars, each bar corresponds to the relative overplanning in one of the 12 shifts. The average overplanning is* 2.7%.

Briefly summarized: taking uncertainty into account results in a schedule with well-balanced relative overplanning (or underplanning). If the employee pool size matches the workload forecast very well, there is barely any overplanning or underplanning to balance. One should carefully consider the support of the workload uncertainty distribution: a support that is too large can

lead to many scenarios and slow convergence, while a support that is too small can lead to unbalanced schedules in case of a substantial mismatch between workload and employee pool size.

7 Conclusions and Recommendations

In this chapter, the conclusions of the research carried out during this thesis are presented. Furthermore, directions for further research are proposed, and learnings for businesses are summarized.

7.1 Conclusions

The research question was as follows: how should Picnic automate the employee scheduling process? There were three subquestions.

1. How should the scheduling problem be modeled?
2. How should the most important future uncertainties (such as the future workload) be taken into account in the model?
3. Which algorithm should be used to solve the model to near-optimality (a relative optimality gap of at most 1%) in two hours on the largest real-world Picnic instances?

How should the scheduling problem be modeled? In this thesis, we have introduced a two-stage stochastic employee scheduling model. The first stage problem contains all of the scheduling constraints, i.e., constraints restricting the shift assignments each employee can receive. This includes all legal constraints with respect to scheduling, but also employee satisfaction constraints (such as giving people the number of hours they want or letting employees indicate their availabilities and respecting them) and operational constraints (such as not having too many people in one location at the same time). The second stage problem evaluates the efficiency of the schedule. The workload is modeled as a list of workload packages, where one workload package consists of some quantity of work to be completed between some start time and end time. The amount of lost workload (workload not completed before the deadline) and the amount of overplanning (employees who have nothing to do) must be minimized. This is accomplished by modeling the problem as a flow problem: all workload flows through a graph, ending up in a "finished" or "lost" node. The flow capacities depend on the number of people working in each interval of time.

We conclude that this model worked well in the setting in which we applied it: real-world datasets, but restricted to locations with only fixed-hour contracts and a fictitious uncertainty distribution applied to the workload forecast. The model has been tested in practice: it has been brought into production and used operationally for several months for the fulfillment teams in Germany and France. Users reported about 50 percent time savings in the full planning process. The remaining time is still spent on preparing data (which can be automated in the future) and making adjustments based on information that comes in after the schedule is generated.

How should the most important future uncertainties be taken into account in the model? The experiments of Chapter 6 demonstrate that there is clear value in taking workload forecast uncertainty into account. Based on the fictitious (but reasonably realistic) distribution, gaps of up to 60% were observed between the objective values of the schedule generated without uncertainty taken into account and the schedule generated with a very close approximation of the workload distribution. The schedules generated with uncertainty taken into account lead to "balanced" schedules, meaning that any overplanning (too many people present for the expected workload) or underplanning (too few people present for the expected workload) was spread evenly over the different shifts.

According to expert opinion within Picnic, such schedules work well in practice: they are not just robust against uncertainty in workload, but also against other uncertainties such as the

stochastic no-show rate. Moreover, the experiments in Chapter 6 demonstrated that even using a very small number of scenarios can already make a huge impact. Under the fictitious probability distribution, the worst-case optimality gap over the different test sets reduced from approximately 59% to approximately 3% when increasing the number of scenarios from 1 to just 13.

However, more research should be done to find out (1) the impact of using a true empirical probability distribution to model workload uncertainty; (2) the impact of taking other uncertainties (such as the no-show rate) into account.

Which algorithm should be used to solve the model to near-optimality in two hours on the largest real-world Picnic instances? Using the L-shaped algorithm to solve the scheduling algorithm is generally slower than using an off-the-shelf MILP solver. Only when the open source solvers SCIP (MILP-solver) / GLOP (LP-solver) were applied to instances with a lot of workload scenarios, the L-shaped algorithm got a better objective value than the off-the-shelf solver in the same amount of time. The practical value of this is limited, as the optimality gap between schedules generated with a small number (but more than 1) of workload scenarios and schedules generated with a large number of workload scenarios is very limited. The cause of the slow (and in some seemingly arbitrary instances extremely slow) convergence of the L-shaped algorithm is generally caused by the master problem of the algorithm, which requires solving a MILP. In any iteration, a pathological MILP instance may arise which takes extremely long to solve. The use of the L-shaped algorithm, implemented as in this thesis, is therefore not recommended for practical application. In Chapter 6, however, we saw test cases with a lot of shifts (224) that could not be solved with an open-source MILP-solver. In case a user opts for an open-source solver, the user could try scheduling for shorter periods of time. The 224 shifts of the aforementioned test instances spanned four weeks. One can also opt to schedule for one week at a time sequentially for four times. This will likely result in a major speed gain.

7.2 Directions of future research

Better uncertainty models It would be interesting to extend the experiments in this thesis with more advanced uncertainty models, containing multiple uncertain parameters. Since the number of scenarios to consider would become very large, the scheduling algorithm should be preceded by a sampling method that can be used to approximate the distribution. The questions are then (1) what sampling method should be used; (2) what sample size is "large enough"; (3) what is the impact of the extra level of detail on the generated schedules?

Different implementations of the L-shaped algorithm As mentioned in Chapter 3, many papers have considered adding cuts in the branching tree when solving the master problem. That is, the master problem is solved using branch and bound, where there are no optimality cuts in the top node. In downstream nodes, cuts are added as required. In practice, however, the MILP-solvers use heuristics to cut off large parts of the branching tree before they are fully explored. It should be investigated whether such heuristics can be combined with branch-and-cut methods or not. If this is possible, a branch-and-cut implementation of the L-shaped algorithm may be promising.

Extensions of the model The model presented in this thesis already works in quite a general setting compared to many papers in the literature (such as the ones surveyed in Van Den Bergh et al., 2013). It can handle underplanning and overplanning, people with both fixed-hour and variable-hour contracts, shifts of arbitrary lengths, overlapping shifts, workload packages with arbitrary release and deadline times, skill requirements for different types of workload, and legislation of multiple countries. Still, more extensions could be valuable, especially around the workload models. The workload package model used in this thesis assumes (1) that each package

consists of a large number of small independent tasks; (2) that any task not done before the deadline is lost. However, in many practical settings, the workload is not lost but ends up on a backlog. Moreover, work is not always divisible among multiple people. For example, a different second-stage model (presented in Appendix C) is required to plan people for the last-mile delivery teams of Picnic: after all, a trip cannot be divided over multiple people. A general workload model that does not rely on the aforementioned restrictive assumptions would help to make the model even more widely applicable.

More practical tests The scheduling algorithm presented in this thesis should be tested in more settings, including settings with variable-hour contracts and workload packages that overlap with multiple shifts. Testing the scheduling algorithm under a large variety of settings will expose any weaknesses in the model and edge cases that are not yet properly handled.

References

Angulo, G., Ahmed, S., & Dey, S. S. (2016). Improving the integer L-shaped method. *INFORMS Journal on Computing*, *28*(3). https://doi.org/10.1287/ijoc.2016.0695

Arbeidstijdenwet. (2022).

Arbeitszeitgesetz. (2020).

Ben-Ameur, W., & Neto, J. (2006). A constraint generation algorithm for large scale linear programs using multiple-points separation. *Mathematical Programming*, *107*(3). https://doi.org/10.1007/s10107-005-0694-0

Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, *4*(1). https://doi.org/10.1007/BF01386316

Berger, P. D., & Monahan, J. P. (1974). A Planning Model to Cope with Absenteeism. *The Journal of Business*, *47*(4). https://doi.org/10.1086/295678

Bestuzheva, K., Besançon, M., Chen, W.-K., Chmiela, A., Donkiewicz, T., van Doornmalen, J., Eifler, L., Gaul, O., Gamrath, G., Gleixner, A., Gottwald, L., Graczyk, C., Halbig, K., Hoen, A., Hojny, C., van der Hulst, R., Koch, T., Lübbecke, M., Maher, S. J., ... Witzig, J. (2021). *The SCIP Optimization Suite 8.0* (tech. rep. No. 21-41). Zuse Institute Berlin. http://nbn-resolving.de/urn:nbn:de:0297-zib-85309

Birge, J. R., & Louveaux, F. (2011). Introduction to Stochastic Programming. In *Springer series in operations research and financial engineering*. https://doi.org/10.1057/palgrave.jors.2600031

Brémaud, P. (2020). *Probability Theory and Stochastic Processes*. Springer International Publishing. https://doi.org/10.1007/978-3-030-40183-2

Côté, M. C., Gendron, B., & Rousseau, L. M. (2011). Grammar-based integer programming models for multiactivity shift scheduling. *Management Science*, *57*(1). https://doi.org/10.1287/mnsc.1100.1264

Diestel, R. (2017). *Graph Theory* (Vol. 173). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-53622-3

Eitzen, G., Panton, D., & Mills, G. (2004). Multi-skilled workforce optimisation. https://doi.org/10.1023/B:ANOR.0000019096.58882.54

Ernst, A. T., Jiang, H., Krishnamoorthy, M., Owens, B., & Sier, D. (2004). An annotated bibliography of personnel scheduling and rostering. https://doi.org/10.1023/B:ANOR.0000019087.46656.e2

Grimmett, G., & Welsh, D. (2014). *Probability: an introduction* (2nd ed.). Oxford University Press.

Hart, W. E., Laird, C., Watson, J.-P., & Woodruff, D. L. (2012). Pyomo – Optimization Modeling in Python. *Advances in Modeling Agricultural Systems*, *67*.

Jiang, S., Song, Z., Weinstein, O., & Zhang, H. (2020). Faster Dynamic Matrix Inverse for Faster LPs.

Kletzander, L., & Musliu, N. (2020). Solving the general employee scheduling problem. *Computers and Operations Research*, *113*. https://doi.org/10.1016/j.cor.2019.104794

Laporte, G., & Louveaux, F. V. (1993). The integer L-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters*, *13*(3). https://doi.org/10.1016/0167-6377(93)90002-X

Nemhauser, G., & Wolsey, L. (2014). *Integer and combinatorial optimization*. https://doi.org/10.1002/9781118627372

Perron, L., & Furnon, V. (2022). OR-Tools.

Pflug, G. C., & Pichler, A. (2014). *Multistage Stochastic Optimization*. Springer International Publishing. https://doi.org/10.1007/978-3-319-08843-3

Postek, K., & Kuryatnikova, O. (2021). *Optimization under Uncertainty Course Notes*. Erasmus University Rotterdam.

Restrepo, M. I., Gendron, B., & Rousseau, L. M. (2017). A two-stage stochastic programming approach for multi-activity tour scheduling. *European Journal of Operational Research*, *262*(2). https://doi.org/10.1016/j.ejor.2017.04.055

Rijksoverheid. (2022). Wat zijn de wettelijke regels voor mijn werktijden en rusttijden?

Roy, J., Mitchell, S., & Peschiera, F. (2022). PuLP.

Spliet, R. (2021a). *Mathematical Programming Slides Set 3*. Erasmus University Rotterdam.

Spliet, R. (2021b). *Mathematical Programming Slides Set 5*. Erasmus University Rotterdam.

Tebboth, J. R. (2001). *A Computational Study of Dantzig-Wolfe Decomposition* (Doctoral dissertation). University of Buckingham. Buckingham.

Van Den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., & De Boeck, L. (2013). Personnel scheduling: A literature review. *European Journal of Operational Research*, *226*(3). https://doi.org/10.1016/j.ejor.2012.11.029

Van Tatenhove, J. (2016). *An efficient algorithm for the VRPTW with short routes* (Doctoral dissertation). Delft University of Technology. Delft.

Zou, J., Ahmed, S., & Sun, X. A. (2019). Stochastic dual dynamic integer programming. *Mathematical Programming*, *175*(1-2). https://doi.org/10.1007/s10107-018-1249-5

A Mathematical Background

In this appendix, we introduce the mathematical background required to understand the work of the thesis. An domain expert in applied mathematical optimization may know all of the results described in this Appendix, which is the reason why it is not included in the main text, but the mathematician specializing in a different sub-field may find this Appendix useful to get a more detailed understanding of the mathematics used in this thesis.

We summarize some well-known results from the (mixed integer) linear optimization theory and from polyhedral theory that are used within this thesis, so that the thesis is self-contained as much as possible. A reader unfamiliar with some of the results used in the main text may refer to this section. However, this appendix is not meant as a substitute for a course or a book - for a good introduction to MILPs, the reader can refer to Nemhauser and Wolsey (2014).

A note for transparency: a part of this appendix was written by the author during his previous master thesis project and is therefore not work. The appendix did not make it into the final version of that thesis and is instead used here.

A.1 General Results in Optimization

Definition A.1 (**Optimization Problem**)**.** An *optimization problem* is a problem of the following form:

$$\inf_{\boldsymbol{x}\in X} f(x), \tag{110}$$

where X is a set and $f : X \to F$ is a function from X to the ordered field F.

Remark A.2. A problem of the form $\sup_{x\in X} f(x)$ is also an optimization problem since

$$\sup_{x\in X} f(x) = \inf_{x\in X} -f(x). \tag{111}$$

Theorem A.3 (**Extreme Value Theorem**)**.** If X is a compact set, any continuous function $f : X \to \mathbb{R}$ attains a maximum and a minimum in X.

A.2 Mixed Integer Linear Programs

Definition A.4 (**Mixed Integer Linear Program** Tebboth (2001))**.** A *mixed integer linear program* (MILP) is an optimization problem of the following form:

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^n} \quad & \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.} \quad & A\boldsymbol{x} = \boldsymbol{b} \\ & x_i \in \mathbb{N} \quad \forall i \in I \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned} \tag{112}$$

with $\boldsymbol{c} \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, $\boldsymbol{b} \in \mathbb{R}^m$ and $I \subset \{1, \ldots, n\}$. There are two special cases: if $I = \{1, \ldots, n\}$, problem (112) is called an *integer linear program* (ILP) and if $I = \emptyset$, problem (112) is called a *linear program* (LP).

Remark A.5. Even though the form of problem (112) looks very specific, many problems can be cast in this form. Suppose the problem contains variables x_i that are allowed to be negative. Then a variable substitution $x_i = x_i^+ - x_i^-$ with $x_i^+, x_i^- \geq 0$ can be done. For constraints for the form $\boldsymbol{a_j}^T\boldsymbol{x} \leq b_j$ or $\boldsymbol{a_j}^T\boldsymbol{x} \geq b_j$, $j \in \{1, \ldots, m\}$ we can add a slack variable $s_i \geq 0$ to the problem

and modify the constraint to $\boldsymbol{a_j}^T\boldsymbol{x} + s_j = b_j$ or $\boldsymbol{a_j}^T\boldsymbol{x} - s_j = b_j$ respectively. It is however, not allowed to include strict inequalities as constraints within the linear programming framework. The reason is that it strict inequality constraints can result in ill-posed optimization problems for which the infimum solution may not be feasible. For example, $\inf x | x > 0 = 0$, but $x = 0$ is not feasible.

Definition A.6 (**Feasibility and Boundedness**, Tebboth (2001))**.** The *feasible region* of an instance of problem (112) is defined as $P = \{\boldsymbol{x} \in \mathbb{R}^n | A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}, x_i \in \mathbb{N} \forall i \in I\}$. Problem (112) is called *feasible* if $P \neq \emptyset$ and *infeasible* otherwise. A feasible instance of problem (112), is called *bounded* if there exists a $B \in \mathbb{R}$ such that $\boldsymbol{c}^T\boldsymbol{x} \geq B$ for all $\boldsymbol{x} \in P$ and *unbounded* otherwise.

Definition A.7 (**Relaxation**)**.** Suppose an instance of the general optimization problem $z = \inf_{x \in X} f_0(x)$, denoted by (P), is given. Let $z_R = \inf_{x \in X_R} f_0(x)$, denoted by (P_R). We call the problem (P_R) a *relaxation* of (P) if $X \subset X_R$. Furthermore, if two relaxations (P_{R_1}), (P_{R_2}) of (P) are given, we say that relaxation (P_{R_1}) is *stronger* than (P_{R_2}) if $X_{R_1} \subset X_{R_2}$. We call the problem (112) with $I = \emptyset$ the *LP-relaxation* of problem (112) with equivalent A, $\boldsymbol{b}$ and $\boldsymbol{c}$, but general I.

Definition A.8 (**Valid inequality**, Nemhauser and Wolsey (2014))**.** Given a mixed integer linear program P, an inequality $\boldsymbol{\pi}^T\boldsymbol{x} \leq \pi_0$ is called a *valid inequality* for P if it is satisfies by all points $\boldsymbol{x}$ in the feasible region of P.

A.3 Polyhedral Theory

Definition A.9 (**Polyhedron**, Nemhauser and Wolsey (2014))**.** A set $P \in \mathbb{R}^n$ is called a *polyhedron* if there exists an $m \in \mathbb{N}$, a matrix $A \in \mathbb{R}^n$ and a vector $\boldsymbol{b} \in \mathbb{R}^m$ such that $P = \{A\boldsymbol{x} \leq \boldsymbol{b}\}$, with $\leq$ denoting element-wise inequality.

Theorem A.10 (Tebboth (2001))**.** The feasible region of any linear program is a polyhedron.

Proof. The feasible region of a linear program can be represented by $\boldsymbol{x} \in \mathbb{R}^n | A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}$ (Definition 112). Note that $A\boldsymbol{x} = \boldsymbol{b} \iff A\boldsymbol{x} \leq \boldsymbol{b} \wedge -A\boldsymbol{x} \leq -\boldsymbol{b}$ and $\boldsymbol{x} \geq \boldsymbol{0} \iff -I\boldsymbol{x} \leq \boldsymbol{0}$. Letting $\tilde{A} = [A, -A, -I]$ and $\tilde{\boldsymbol{b}} = [\boldsymbol{b}^T, -\boldsymbol{b}^T, \boldsymbol{0}^T]^T$ gives rise to an equivalent representation of P, namely $P = \{\tilde{A}\boldsymbol{x} \leq \tilde{\boldsymbol{b}}\}$. □

Definition A.11 (**Convex set**, Nemhauser and Wolsey (2014))**.** A set P is *convex* if $\forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in P$ and $\forall \lambda \in [0, 1]$, $\lambda x_1 + (1 - \lambda)x_2 \in P$.

Theorem A.12 (Nemhauser and Wolsey (2014))**.** Any polyhedron is a convex set.

Proof. The result follows directly from the definition of a polyhedron and the linearity of matrix-vector multiplications. □

A.3.1 Extreme points and rays

Definition A.13 (**Extreme point**, Tebboth (2001))**.** Let P be a polyhedron. An element $\boldsymbol{x} \in P$ is called an *extreme point* of P if there are no $\boldsymbol{x}_1, \boldsymbol{x}_2 \in P$ such that $\boldsymbol{x}_1 \neq \boldsymbol{x}_2$ and $\boldsymbol{x} = \frac{1}{2}\boldsymbol{x}_1 + \frac{1}{2}\boldsymbol{x}_2$.

Definition A.14 (**Ray**, Tebboth (2001))**.** Given a polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^n | A\boldsymbol{x} \leq \boldsymbol{b}\} \neq \emptyset$, the vector $\boldsymbol{r} \in \mathbb{R}^n \backslash \{\boldsymbol{0}\}$ is called a *ray* of P if $A\boldsymbol{r} \leq \boldsymbol{0}$. Rays $\boldsymbol{r}$ and $\boldsymbol{s}$ are called *distinct* if there is no $\lambda \in (0, \infty)$ such that $\boldsymbol{r} = \lambda \boldsymbol{s}$.

Definition A.15 (**Extreme Ray**, Tebboth (2001))**.** Given a non-empty polyhedron P with ray $\boldsymbol{r}$, $\boldsymbol{r}$ is called an *extreme ray* if there are no two distinct rays $\boldsymbol{r}^1$, $\boldsymbol{r}^2$ of P such that $\boldsymbol{r} = \frac{1}{2}\boldsymbol{r}^1 + \frac{1}{2}\boldsymbol{r}^2$.

Proposition A.16 (Tebboth (2001))**.** A polyhedron has a finite number of extreme points and extreme rays.

Proof. See for instance Nemhauser and Wolsey (2014). □

Theorem A.17 (**Minkowski's Representation Theorem**, Tebboth (2001))**.** Suppose $P = \{\boldsymbol{x} \in \mathbb{R}^n | A\boldsymbol{x} \leq \boldsymbol{b}\}$ is a non-empty polyhedron with $rank(A) = n$, extreme point set $\{\boldsymbol{x}^l, l = 1, \ldots, L\}$ and extreme ray set $\{\tilde{\boldsymbol{x}}^l, l = 1, \ldots, \tilde{L}\}$. Then $P = P'$ with

$$P' = \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{x} = \sum_{l=1}^{L} \beta_l \boldsymbol{x}^l + \sum_{l=1}^{\tilde{L}} \mu_l \tilde{\boldsymbol{x}}^l, \sum_{l=1}^{L} \beta_l = 1, \boldsymbol{\beta}, \boldsymbol{\mu} \geq 0\}. \tag{113}$$

Proof. See for instance Tebboth (2001). □

Corollary A.18. A linear program of the form (112) with $I = \emptyset$ and $rank(A) = n$ can be rewritten as

$$\begin{aligned} \min_{\boldsymbol{\beta}, \boldsymbol{\mu}} \quad & \sum_{l=1}^{L} \beta_l \boldsymbol{c}^T \boldsymbol{x}^l + \sum_{l=1}^{\tilde{L}} \mu_l \boldsymbol{c}^T \tilde{\boldsymbol{x}}^l \\ \text{s.t.} \quad & \sum_{l=1}^{L} \beta_l A\boldsymbol{x}^l + \sum_{l=1}^{\tilde{L}} \mu_l A\tilde{\boldsymbol{x}}^l = \boldsymbol{b} \\ & \sum_{l=1}^{L} \beta_l = 1 \\ & \boldsymbol{\beta}, \boldsymbol{\mu} \geq \boldsymbol{0}. \end{aligned} \tag{114}$$

Theorem A.19 (Tebboth (2001))**.** Consider a feasible linear program of the form (112) with $I = \emptyset$ and $rank(A) = n$. If it is bounded, it has an optimal solution at an extreme point of the feasible region. If it is unbounded, it is unbounded along an extreme ray of the feasible region.

Proof. See for instance Nemhauser and Wolsey (2014). □

Corollary A.20. To find the optimal solution of a bounded linear program of the form (112) with $I = \emptyset$ and $rank(A) = n$, it suffices to consider

$$\begin{aligned} \min_{\boldsymbol{\beta},\boldsymbol{\mu}} \quad & \sum_{l=1}^{L} \beta_l \boldsymbol{c}^T \boldsymbol{x}^l \\ \text{s.t.} \quad & \sum_{l=1}^{L} \beta_l A \boldsymbol{x}^l = \boldsymbol{b} \\ & \sum_{l=1}^{L} \beta_l = 1 \\ & \boldsymbol{\beta} \geq \mathbf{0}. \end{aligned} \tag{115}$$

A.3.2 Faces

The extreme points of a polyhedron, introduced in the previous section, constitute a subset of a larger class of objects called "faces". In this section, we introduce the notion of faces and some of their properties. They are used in the main text to prove that Benders' decomposition works when adding optimality cuts based on general optimal solutions to the Benders Sub Problem, instead of optimal extreme points.

Definition A.21 (**Valid Inequality**, Nemhauser and Wolsey, 2014)**.** Given a polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$, we call the inequality $\boldsymbol{pi}^T\boldsymbol{x} \leq \pi_0$ a *valid inequality* for P if it is satisfied by all points $\boldsymbol{x} \in P$.

Definition A.22 (**Face**, Nemhauser and Wolsey, 2014)**.** If $\boldsymbol{\pi}^T\boldsymbol{x} \leq \pi_0$ is a valid inequality for P and $F = \{\boldsymbol{x} \in P \mid \boldsymbol{\pi}^T\boldsymbol{x} = \pi_0\}$, we call F a *face* of P. If $F \neq \emptyset$ and $F \neq P$, we call F a *proper face*.

Proposition A.23. If P is a polyhedron and F is a non-empty face of P, F is also a polyhedron.

Proof. See Nemhauser and Wolsey (2014), chapter I.4, proposition 3.1. □

Proposition A.24. The number of distinct faces of a polyhedron is finite.

Proof. See Nemhauser and Wolsey (2014), chapter I.4, proposition 3.1. □

Definition A.25 (**Dimension of a face**, Nemhauser and Wolsey (2014))**.** We say that a face F of polyhedron P has dimension k if and only if the maximal number of affinely independent points in F is $k + 1$

Proposition A.26. The extreme points of polyhedron P are precisely its zero-dimensional faces.

Proof. See Nemhauser and Wolsey (2014), chapter I.4, proposition 4.2. □

A.3.3 Cones

Definition A.27 (**Cone**, Nemhauser and Wolsey, 2014)**.** $C \subset \mathbb{R}^n$ is called a *cone* if $\boldsymbol{x} \in C \wedge \lambda \in \mathbb{R}_+ \implies \lambda \boldsymbol{x} \in C$.

Proposition A.28 (Nemhauser and Wolsey, 2014)**.** The set $\{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{0}\}$, with A a matrix and $\boldsymbol{x}$ a vector of appropriate dimensions, is a cone.

Proof. If $\boldsymbol{x}$ satisfies $A\boldsymbol{x} \leq 0$ and $\lambda \in \mathbb{R}_+$, $A(\lambda \boldsymbol{x}) = \lambda(A\boldsymbol{x}) \leq \lambda \boldsymbol{0} = \boldsymbol{0}$. □

Proposition A.29. The cone $A\boldsymbol{x} \leq 0$ has one extreme point, namely $\boldsymbol{x} = 0$, and a finite number of extreme rays.

Proof. The claim that $C = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{0}\}$ has a finite number of extreme rays follows directly from Proposition A.16. We proceed to prove the claim that $\boldsymbol{x} = \boldsymbol{0}$ is the unique extreme point.

To see that $\boldsymbol{0}$ is indeed an extreme point, consider $\boldsymbol{x}^1 \neq \boldsymbol{x}^2$ satisfying $\frac{1}{2}(\boldsymbol{x}^1 + \boldsymbol{x}^2) = \boldsymbol{0}$. Since $\boldsymbol{x}^1 \neq \boldsymbol{x}^2$, we can assume without loss of generality that $\boldsymbol{x}^1 \neq 0$. Then $\exists i \in [n] : x_i^1 \neq 0$. Consequently, $x_i^2 = -x_i^1 \neq 0$. It follows that either $\boldsymbol{A_i} x_i^1 > 0$ or $\boldsymbol{A_i} x_i^2 > 0$, which means $\boldsymbol{x}^1 \notin C$ or $\boldsymbol{x}^2 \notin C$. So $\boldsymbol{0}$ is an extreme point.

For any point $\boldsymbol{x} \in C \backslash \{\boldsymbol{0}\}$, $\boldsymbol{x}^1 = \frac{1}{2}\boldsymbol{x} \in C$, $\boldsymbol{x}^2 = \frac{3}{2}\boldsymbol{x} \in C$, $\boldsymbol{x}^1 \neq \boldsymbol{x}^2$ and $\frac{1}{2}(\boldsymbol{x}^1 + \boldsymbol{x}^2) = \boldsymbol{x}$. So $\boldsymbol{x}$ is not an extreme point. □

A.4 Elementary results in probability theory

All of these definitions can be found in general formal probability theory books, such as Brémaud (2020). A random process is usually modeled through the notion of a probability space. The basic concepts around probability spaces are explained here.

A.4.1 Probability spaces and distributions

Definition A.30 (**σ-algebra**)**.** A *σ-algebra* on a set X is a collection $\mathcal{A}$ of subsets of X such that

1. $\emptyset, X \in \mathcal{A}$;
2. $A \in \mathcal{A} \implies A^c \in \mathcal{A}$;
3. if $A_i \in \mathcal{A}$ for $i \in \mathcal{N}$ then $\bigcup_{i=1}^{\infty} A_i \in \mathcal{A}$ and $\bigcap_{i=1}^{\infty} A_i \in \mathcal{A}$.

Definition A.31 (**Measurable space**)**.** A pair $(X, \mathcal{A})$ is a *measurable space* if X is a non-empty set and $\mathcal{A}$ is a σ-algebra on X.

Definition A.32 (**Measure**)**.** A measure on a measurable space $(X, \mathcal{A})$ is a function $\mu : \mathcal{A} \to [0, \infty]$ such that

1. $\mu(\emptyset) = 0$;
2. if $\{A_i \in \mathcal{A}, i \in \mathbb{N}\}$ is a pair-wise countable disjoint collection of sets, then

$$\mu(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} \mu(A_i). \tag{116}$$

Definition A.33 (**Measure space**)**.** Let X be a set, $\mathcal{A}$ be a σ-algebra on X and μ a measure on $(X, \mathcal{A})$. Then $(X, \mathcal{A}, \mu)$ is a *measure space*.

Definition A.34 (**Measurable function**)**.** Given two measurable spaces $(X, \mathcal{A})$, $(Y, \mathcal{B})$, a function $f : X \to Y$ is measurable if $f^{-1}(B) = \{x \in X | f(x) \in B\} \in \mathcal{A}$ for every $B \in \mathcal{B}$.

We will now slightly change the notation to the conventions of probability theory.

Definition A.35 (**Probability measure**)**.** A *measure* $\mathbb{P}$ on a measurable space $(\Omega, \mathcal{F})$ is called a *probability measure* if $\mathbb{P}(\Omega) = 1$. If $\mathbb{P}$ is a probability measure, then $(\Omega, \mathcal{F}, \mathbb{P})$ is called a *probability space* or a *probability triple*. In this context, we call Ω a *sample space* and $\mathcal{F}$ an *event space*.

Definition A.36 (**Support**)**.** Given a measurable space $(\Omega, \mathcal{F}, \mathbb{P})$, the *support* of $\mathbb{P}$, $\text{supp}(\mathbb{P})$, is the smallest subset of Ω such that $\text{supp}(\mathbb{P}) \in \mathcal{F}$ and $\mathbb{P}(\text{supp}(\mathbb{P})) = 1$.

A.4.2 Densities

Definition A.37 (**Density**)**.** Given a measurable space $(X, \mathcal{A}, \mu)$ and a different measure ν on (X, A), we say that f is a *density* ν with respect to μ if and only if

$$\nu(A) = \int_X f \mathbb{1}_A d\mu \quad A \in \mathcal{A}. \tag{117}$$

Remark A.38. Defining the measure-theoretic integral here would go beyond the scope of this section, an intuitive understanding of the integral suffices for now.

Definition A.39 (**Absolute continuity**)**.** Given a measurable space (X, A) and two measures μ and ν on this space, we say that ν is *absolutely continuous* with respect to μ if

$$\nu(A) = 0 \quad A \in \mathcal{A}, \mu(A) = 0. \tag{118}$$

Theorem A.40 (**Radon-Nikodym theorem**)**.** Let (X, A) be a measurable space and two σ-finite (an extremely mild assumption beyond the scope of this Appendix) measures μ and ν on this space. Then there exists a density f of ν with respect to μ if and only if ν is absolutely continuous with respect to μ.

Proof. See for instance Brémaud (2020) □

Remark A.41. Most probability distributions encountered in practice have a density with respect to the Lesbesgue measure (continuous distributions, the density is then called a probability density function) or the counting measure (discrete distributions, the density is then called the probability mass function).

A.4.3 Random variables

Definition A.42 (**Random variable**)**.** Given a probability triple $(\Omega, \mathcal{F}, \mathbb{P})$ and a measureable space $(E, \mathcal{E})$, we call a measurable function $X : \Omega \to E$ a *random variable*.

Definition A.43 (**Pushforward measure**)**.** Given a measure space $(\Omega, \mathcal{F}, \mathbb{P})$, a measurable space $(E, \mathcal{E})$ and a measurable function $X : \Omega \to E$, then the *pushforward measure* on $(E, \mathcal{E})$ is defined as follows:

$$\mathbb{P}^X(B) = \mathbb{P} \circ X^{-1}(B) = \mathbb{P}(X^{-1}(B)) = \mathbb{P}(\{\omega \in \Omega | X(\omega) \in B\}) \quad B \in \mathcal{E}. \tag{119}$$

B Computing the Working Proportion of Employees

As in the main text, the let S denote the set of shifts and let E denote the set of employees. Given a shift $s \in S$ and a set of assigned employees $E_s \subset E$, we must be able to compute at any moment in time $t \in s$ which proportion of the employees is working. This is an essential step in computing how much work can be done in some interval of time.

We do not want to assign breaks to employees in advance, since this would increase the number of decision variables lead to non-linearities in the MIP formulation of the scheduling problem. Therefore, we make some simplifying assumptions to make the problem tractable.

Definition B.1 (Break slot)**.** Given a shift $s \in S$, we define a *break slot* in shift s as an interval $b \subset s$.

Definition B.2 (Break)**.** Given a shift $s \in S$, a *break* in shift s is a set of break slots all having the same measure.

The definition above is based on operational realities: each employee has to take several breaks in a shift, depending on the length of the shift. For example, in an 8-hour shift, an employee may have to take one 15-minute break, one 30-minute break and another 15-minute break. However, the canteens and food queues do not have enough capacity for all of the employees to take a break at the same time. Therefore, employees take breaks during different break slots.

The intuition described in the preceding paragraph is summarized in the following assumption.

Assumption B.3. Given a shift $s \in S$ with a list of breaks $B_s^1, \ldots . B_s^n$, for each $i \in [n]$, each assigned employee $e \in E_s$ rests during precisely one break slot in B_s^i.

Furthermore, we assume that the employees are divided equally over the different break slots in a break.

Assumption B.4. Given a shift $s \in S$ with a list of breaks $B_s^1, \ldots . B_s^n$ and a set of assigned employees $E_s \subset E$, the proportion p_b that takes rests during slot b is $p_b = \frac{1}{|B_s^i|}$. An employee cannot be assigned to two (or more) overlapping break slots.

Remark B.5. We ignore the fact that a "number of employees" is always integer and approximate the number of employees assigned to a break slot as a real number instead. This allows Assumption B.4 to work, and it is reasonable if the number of employees working during a shift is high and the break slots are not too far apart.

For Assumption B.3 (specifically, the fact that an employee must first finish one break before starting the next one), a necessary condition is that for a given shift $s \in S$:

$$p_i^{\text{done}}(t) \geq p_{i+1}^{\text{start}}(t) \quad t \in s, i = 1, \ldots, n-1, \tag{120}$$

where $p_i^{\text{done}}(t)$ is the proportion of employees that is done with break B_s^i at or before time t and $p_{i+1}^{\text{start}}(t)$ is the proportion of employees that has started break B_s^{i+1} at or before time t.

In other words,

$$p_i^{\text{done}}(t) = \sum_{b \in B_s^i : t_b^{\text{end}} \leq t} p_b \quad i = 1, \ldots, n, \tag{121}$$

and

$$p_i^{\text{start}}(t) = \sum_{b \in B_s^i : t_b^{\text{start}} \leq t} p_b \quad i = 1, \ldots, n. \tag{122}$$

Proposition B.6. If equation (120) is satisfied, then the fraction of people assigned to shift $s \in S$ working at time t satisfies

$$\pi_s^{\text{working}}(t) := 1 - \sum_{i=1}^{n} \sum_{b \in B_s^i} \mathbb{1}_b(t) p_b \geq 0. \tag{123}$$

Proof. Observe that

$$\begin{aligned}
\pi_s^{\text{working}}(t) &= 1 - \sum_{i=1}^{n} \sum_{b \in B_s^i} \mathbb{1}_b(t) p_b \\
&= 1 - \sum_{i=1}^{n} (p_i^{\text{start}}(t) - p_i^{\text{done}}(t)) \\
&= 1 - p_1^{\text{start}}(t) + p_n^{\text{done}}(t) - \sum_{i=1}^{n} (p_{i+1}^{\text{start}}(t) - p_i^{\text{done}}(t)) \\
&\geq 1 - p_1^{\text{start}}(t) - \sum_{i=1}^{n} (p_{i+1}^{\text{start}}(t) - p_i^{\text{done}}(t)) \\
&\geq 1 - p_1^{\text{start}}(t) \\
&\geq 0,
\end{aligned} \tag{124}$$

where the first and last inequalities follow from the fact that $p_i^{\text{start}}, p_i^{\text{done}} \in [0, 1]$. The second inequality follows from equation (120). □

C An alternative people scheduling model for last-mile delivery

C.1 Alternative scheduling concept: capacity targets

The main text of this thesis is focussed on scheduling people based on so-called workload packages: a workload package essentially models work as a large collection of tiny (even infinitesimal) tasks, which can be executed independently of each other within some time frame. In the setting of order fulfillment and the setting of customer success, this model is a reasonable approximation of reality: in a fulfillment center, work consists of a large number of orders to be fulfilled, a large number of items to be replenished in the storage locations etcetera. In the customer success department, the workload takes the form of small "cases", which could be phone calls, Whatsapp messages, feedback through the app, and so on. This is also modeled quite well by the workload package model: the only caveat is the workload package model assumes that workload that is not completed before the deadline is lost. But even in case of undercapacity, customer cases still need to be handled. The cases go on a backlog and will be handled at a later time. Still, if the agent pool size is roughly correct (i.e. large enough such that a good schedule that matches the workload exists), the impact of this modeling difference should be quite small.

For the last-mile delivery, the workload package model does not work. Indeed, the workload consists of long (multiple hours) trips that are indivisible. In this section, we introduce the notion of "capacity target" based scheduling to accommodate the setting of last-mile delivery. It leads to an alternative second-stage problem. The first-stage problem, consisting of a configurable set of scheduling (mostly legal) constraints, is completely reusable.

Consider a hub from which orders are delivered to customers and assume the hub has n vehicles with m (generally greater than n) trips to make. The trip routes are planned based on the available storage volume of the vehicle, the locations of the customers, and the delivery time windows requested by customers (a customer may request that the order is delivered between 13:00 and 15:00 for example). This problem is called the Vehicle Routing Problem with Time Windows: see for example Van Tatenhove (2016).

Solving the Vehicle Routing Problem is not in the scope of this thesis. Instead, we try to schedule employees to shifts in such a manner that all the scheduled trips can be executed. The assumed shift structure is as follows. Consider a set of "base shifts", which are shifts corresponding to a set of trips that are driven simultaneously. Table C.1 shows an example of a hub with eight scheduled trips in three base shifts: s_1 lasts from 9:00 to 11:00, s_2 lasts from 10:00 to 13:00 and s_3 lasts from 11:00 - 13:00.

Table C.1: *Example of a hub with eight trips and three base shifts.*

Trip	1	2	3	4	5	6	7	8
Start time	9:00	9:05	9:10	10:00	10:10	10:15	11:05	11:00
End time	10:50	10:55	11:00	12:40	12:40	12:50	12:55	13:00
Base Shift	s_1	s_1	s_1	s_2	s_2	s_2	s_3	s_3

This means that we need three people to work shift s_1, three people to work shift s_2, and two people to work shift s_3. We will refer to these numbers as the "capacity targets".

Definition C.1 (**Capacity target**)**.** The *capacity target* of a base shift is the minimal number of people required to complete all the trips associated with that base shift.

The reader may have noted that the trips in Table C.1 are rather short. This is a setting that can occur in practice, because of limited storage volume in the vehicle or because of limited range in case the operator uses electric vehicles. As a result of the short trip times, employees may legally work several base shifts per day.

Therefore, we can generate a larger list of shifts from the base shifts. Let S^{base} be the list of base shifts. Then we let the full set of shifts $S \in \mathcal{P}(S^{base})$, with $\mathcal{P}(S^{base})$ the power set of the base shifts, be generated as follows: for each element $S' \in \mathcal{P}(S^{base})$, check that the base shifts in S' are

1. non-overlapping;
2. consecutive, meaning that the time between the end time of one shift and the start time of the next shift is below some tolerance;
3. not too long when concatenated, meaning that $\max_{s' \in S'}\{t^{\text{end}}_{s'}\} - \min_{s' \in S'}\{t^{\text{start}}_{s'}\} \leq t^{\max}$, with $t^{\max}$ some specified maximum total shift length.

If and only if all of these conditions hold, the shift s resulting from the concatenation of the base shifts in S' is included in S.

Definition C.2 (**Associated base shifts**)**.** Let S^{base} be a list of base shifts and S be the full list of shifts generated using the procedure above. If $s \in S$ is the concatenation of the base shifts $S' \subset S^{base}$, we say that S' is the set of *associated base shifts* of s. Alternatively, we say that s is *generated by* S'.

Example C.3. Recall the base shifts used in the example of Table C.1: s_1 lasts from 9:00 to 11:00, s_2 lasts from 10:00 to 13:00 and s_3 lasts from 11:00 - 13:00. The generated shifts would be the ones shown in Table C.2.

Table C.2: *Example of a set of shifts generated from a set of base shifts.*

Shift	Start time	End time	Associated base shifts
s_1	9:00	11:00	s_1
s_2	10:00	13:00	s_2
s_3	11:00	13:00	s_3
$s_{1,3}$	9:00	13:00	s_1, s_3

Remark C.4. Note that we could deviate from the third requirement in the shift generation procedure described above, and let an operator decide manually per combination of base shifts whether it is too long or not. The only requirement is that the labor laws are respected. Furthermore, the second requirement can be relaxed too but letting an employee work non-consecutive shifts can negatively impact employee satisfaction.

The idea of the scheduling concept where we schedule based on shift targets is two-fold: we use the full list of shifts S for scheduling people, and we count per base shift in S^{base} how many people are working in that base shift. We try to match the capacity target in each base shift exactly, penalizing both overplanning (too many people in a base shift) and underplanning (an insufficient number of people in a base shift). For example, if, in the example of Tables C.1 and C.2, we plan two people for shift $s_{1,3}$, 1 person for shift s_1, three people for shift s_2 and nobody for shift s_3, the schedule is perfect: the two people working shift $s_{1,3}$ count towards the capacity of both s_1 and s_3.

When scheduling, we do not assume that the trips are planned to the minute, such as in Table C.1. However, we do assume that a capacity target based on some workload forecast is known for

each base shift. We may create several capacity targets per based shift for workers with different roles.

Furthermore, we make one assumption on the break times that simplifies scheduling.

Assumption C.5. Let $s \in S$ be a shift generated by base shifts $S' \subset S^{base}$. Then between each trip in s, there is a break of a specified length which is known at the time of scheduling. We assume that the base shifts are long enough and the trips are scheduled in such a way that these breaks always fit.

In contrast to when we schedule based on workload packages, knowing the start times of the breaks is not important. Only the duration of the breaks matters for respecting legal and contractual working time constraints.

The second-stage problem is given below.

$$
\begin{aligned}
& Q(\boldsymbol{x}, \boldsymbol{r}, \boldsymbol{\pi}) = \\
\min \quad & \sum_{k \in K} \sum_{s \in S^{\text{base}}} \mu(s)(\lambda^u_{ks} u_{ks} + \lambda^o_{ks} o_{ks}) \\
& \sum_{k \in K'} c_{ks} \leq \sum_{\sigma \in S(s)} (\pi^{no-show}_\sigma)^c \sum_{e \in E_{K'}} x_{e\sigma} && K' \subset K, s \in S^{\text{base}} \\
& \sum_{k \in K} c_{ks} = \sum_{\sigma \in S(s)} (\pi^{no-show}_\sigma)^c \sum_{e \in E} x_{e\sigma} && s \in S^{\text{base}} \\
& o_{ks} \geq c_{ks} - \bar{c}_{ks} && k \in K, s \in S^{\text{base}} \\
& u_{ks} \geq \bar{c}_{ks} - c_{ks} && k \in K, s \in S^{\text{base}} \\
& u_{ks}, o_{ks} \geq 0 && k \in K, s \in S^{\text{base}}
\end{aligned}
\tag{125}
$$

The capacity constraints are similar to the ones of the workload package performance model. The exception is that we now require the inequality for $K' = K$ to be tight so that in case of overcapacity, a solver cannot simply reduce $\boldsymbol{c}$ to make the overplanning 0. All people must be assigned to work of some skill.

Proposition C.6. The full scheduling problem with performance model (125) has relatively complete recourse.

Proof. Suppose a first-stage feasible schedule $\boldsymbol{x}$ is given. Let $s \in S^{base}$ be some arbitrary base shift. Let E^{assigned} be the set of employees assigned to that shift, and let $s_e \in S$ be the shift (not the base shift) assigned to employee $e \in E$ containing base shift s. We can define a mapping $\boldsymbol{f} : E^{\text{assigned}} \to [0,1]^K$ satisfying

$$
\mathbf{1}^T \boldsymbol{f}(e) = (\pi^{\text{no-show}}_{s_e})^c \quad e \in E^{\text{assigned}}, \tag{126}
$$

and

$$
f_k(e) \leq \mathbb{1}_{e \in E_k} \quad e \in E^{\text{assigned}}, k \in K. \tag{127}
$$

This function $\boldsymbol{f}$ should be interpreted as a division of each employee's expected capacity over the different skills (to be more precise: the workload packages requiring different skills). Now let

$$
c_{ks} = \sum_{e \in E^{\text{assigned}}} f_k(e) \quad k \in K. \tag{128}
$$

Observe that for any $K' \subset K$,

$$\begin{aligned}
\sum_{k \in K'} c_{ks} &= \sum_{k \in K'} \sum_{e \in E^{\text{assigned}}} f_k(e) \\
&= \sum_{k \in K'} \sum_{e \in E^{\text{assigned}} \cap E_{K'}} f_k(e) \\
&= \sum_{e \in E^{\text{assigned}} \cap E_{K'}} \sum_{k \in K'} f_k(e) \\
&\leq \sum_{e \in E^{\text{assigned}} \cap E_{K'}} \mathbf{1}^T \boldsymbol{f}(e) \\
&= \sum_{e \in E^{\text{assigned}} \cap E_{K'}} (\pi_{s_e}^{\text{no-show}})^c \\
&= \sum_{\sigma \in S(s)} (\pi_{s_e}^{\text{no-show}})^c \sum_{e \in E_{K'}} x_{e\sigma}.
\end{aligned} \tag{129}$$

The second equality follows from Equation (126). The inequality can be replaced by equality in case $K' = K$. Therefore, Equation 127 combined with the definition of $\boldsymbol{f}$ gives a valid mapping from any legal schedule $\boldsymbol{x}$ to a capacity vector $\boldsymbol{c}$ that satisfies the constraints of Problem (125). Finding feasible corresponding vectors $\boldsymbol{u}$ and $\boldsymbol{o}$ is trivial. We conclude that the full scheduling problem with the performance model given by Problem (125) has relatively complete recourse. □

D Configurations

This appendix contains the set of constraints and the parameters used in the experiments of Chapter 6.

Three different configurations were used. The configuration for Germany, Fulfillment applied to all test sets whose name starts with "DE-". The configuration for France, Fulfillment applies to all test sets whose name starts with "FR-". The configuration for Netherlands, Customer Success, applies to the test sets whose name starts with "HQ-".

D.1 Germany, Fulfillment


```
{
  "constraints": [
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT10H00M",
        "EMPLOYEE_FLAG": null,
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT24H00M",
          "WINDOW_LENGTH": "PT24H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT8H30M",
        "EMPLOYEE_FLAG": "PREGNANT",
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT24H00M",
          "WINDOW_LENGTH": "PT24H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT8H30M",
        "EMPLOYEE_FLAG": "BREASTFEEDING",
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT24H00M",
          "WINDOW_LENGTH": "PT24H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT60H00M",
        "EMPLOYEE_FLAG": null,
```

```
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT168H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT1152H00M",
        "EMPLOYEE_FLAG": null,
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT4032H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT90H00M",
        "EMPLOYEE_FLAG": "PREGNANT",
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT336H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT90H00M",
        "EMPLOYEE_FLAG": "BREASTFEEDING",
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT336H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MIN_NIGHTLY_REST_TIME",
      "params": {
        "HOURS": "PT11H00M"
      },
      "config_id": 1
    },
    {
      "type": "AVAILABILITY",
      "params": {
        "MAX_SHIFTS_OUTSIDE_AVAILABILITY": <Table E.3>,
```

```
        "OUTSIDE_AVAILABILITY_PENALTY": <Table E.3>,
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT168H00M"
        }
      },
      "config_id": 1
    }
  ],
  "cost_parameters": {
    "overplanning": <Table E.1>,
    "lost_workload": [
      {
        "priority": 1,
        "cost": <Table E.2>
      },
      {
        "priority": 2,
        "cost": <Table E.2>
      }
    ]
  },
  "general_parameters": {
    "min_bucket_value": <Table E.4>,
    "max_bucket_value": <Table E.4>
  }
}
```

D.2 France, Fulfillment


```
{
  "constraints": [
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT10H00M",
        "EMPLOYEE_FLAG": null,
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT24H00M",
          "WINDOW_LENGTH": "PT24H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT48H00M",
        "EMPLOYEE_FLAG": null,
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT168H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT552H00M",
        "EMPLOYEE_FLAG": null,
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT2016H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MIN_CONSECUTIVE_REST_TIME",
      "params": {
        "HOURS": "PT35H00M",
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT179H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MIN_NIGHTLY_REST_TIME",
      "params": {
        "HOURS": "PT11H00M"
```

```
      },
      "config_id": 1
    },
    {
      "type": "AVAILABILITY",
      "params": {
        "MAX_SHIFTS_OUTSIDE_AVAILABILITY": 0,
        "OUTSIDE_AVAILABILITY_PENALTY": 0,
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT168H00M"
        }
      },
      "config_id": 1
    }
  ],
  "cost_parameters": {
    "overplanning": <Table E.1>,
    "lost_workload": [
      {
        "priority": 1,
        "cost": <Table E.2>
      },
      {
        "priority": 2,
        "cost": <Table E.2>
      }
    ]
  },
  "general_parameters": {
    "min_bucket_value": <Table E.4>,
    "max_bucket_value": <Table E.4>
  }
}
```

D.3 Netherlands, Customer Success


```
{
  "constraints": [
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT12H00M",
        "EMPLOYEE_FLAG": null,
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT24H00M",
          "WINDOW_LENGTH": "PT24H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT10H00M",
        "EMPLOYEE_FLAG": "PREGNANT",
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT24H00M",
          "WINDOW_LENGTH": "PT24H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT9H00M",
        "EMPLOYEE_FLAG": "MINOR",
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT24H00M",
          "WINDOW_LENGTH": "PT24H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT60H00M",
        "EMPLOYEE_FLAG": null,
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT168H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
```

```
        "HOURS": "PT45H00M",
        "EMPLOYEE_FLAG": "MINOR",
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT168H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT220H00M",
        "EMPLOYEE_FLAG": null,
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT672H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT160H00M",
        "EMPLOYEE_FLAG": "MINOR",
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT672H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT200H00M",
        "EMPLOYEE_FLAG": "PREGNANT",
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT672H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT768H00M",
        "EMPLOYEE_FLAG": null,
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT2688H00M"
```

```
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_WORKING_TIME",
      "params": {
        "HOURS": "PT720H00M",
        "EMPLOYEE_FLAG": "PREGNANT",
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT2688H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MIN_CONSECUTIVE_REST_TIME_NL",
      "params": {
        "HOURS": "PT11H00M",
        "WINDOW_LENGTH": "PT24H00M"
      },
      "config_id": 1
    },
    {
      "type": "MIN_CONSECUTIVE_REST_TIME_NL",
      "params": {
        "HOURS": "PT36H00M",
        "WINDOW_LENGTH": "PT168H00M"
      },
      "config_id": 1
    },
    {
      "type": "AVAILABILITY",
      "params": {
        "MAX_SHIFTS_OUTSIDE_AVAILABILITY": 0,
        "OUTSIDE_AVAILABILITY_PENALTY": 0,
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT168H00M",
          "WINDOW_LENGTH": "PT168H00M"
        }
      },
      "config_id": 1
    },
    {
      "type": "MAX_SHIFTS",
      "params": {
        "SHIFTS": 1,
        "WINDOW_PARAMETERS": {
          "INITIAL_DATETIME": "2022-10-17T00:00:00",
          "START_DATETIME_INTERVAL": "PT24H00M",
          "WINDOW_LENGTH": "PT24H00M"
        }
      },
      "config_id": 1
    }
```

```
  ],
  "cost_parameters": {
    "overplanning": <Table E.1>,
    "lost_workload": [
      {
        "priority": 1,
        "cost": <Table E.2>
      },
      {
        "priority": 2,
        "cost": <Table E.2>
      }
    ]
  },
  "general_parameters": {
    "min_bucket_value": <Table E.4>,
    "max_bucket_value": <Table E.4>
  }
}
```

E Confidential Annex

The content of this appendix has been removed to keep company information confidential.