
THE UNIVERSITY *of* EDINBURGH

# Airport Security Workforce Planning

*Johanna Wiesflecker*


Doctor of Philosophy

Management Science and Business Economics

Business School

University of Edinburgh

2025

# Abstract

Uncontrollable passenger arrivals cause frequent situations where passenger demands exceed the available capacity. This becomes particularly apparent within the airport's bottlenecks — such as the security hall — where long queues can build up, leading to delays, which translates to unhappy passengers and airlines. Planning for and quickly adapting to changes in passenger arrivals to the security hall and staff availability is key to keeping a consistent flow of passengers and avoiding a build-up of queues.

This problem can be addressed at different stages in the roster planning timeline. At the strategic planning stage, the chief staff scheduler (CSS) and HR department can change contracts and adapt the workforce structure (e.g. via recruitment) best to suit the expected demand for the upcoming season. At the tactical planning stage, the problem is two-fold. On the one hand, a fast and flexible roster tool is needed to adapt quickly to changing work regulations and union demands. On the other hand, this is the last time the CSS can freely allocate workforce members to the schedule. Hence, they need to use this chance to build flexibilities into the roster that can help adapt to changes in demand at the subsequent operational planning stage.

A solver-independent modelling approach using Mixed Integer Programming and Constraint Programming is proposed as a fast and flexible Roster Tool to support the CSS's decisions and help with union discussions. Based on this tool, a Simheuristic is derived to allocate flexibility to the roster at the tactical planning stage. The Simheuristic combines a metaheuristic (Simulated Annealing, Genetic Algorithm and Adaptive Large Neighborhood Search (ALNS)) to adjust the flexibility metrics in the roster with Monte-Carlo Simulation of demand patterns and staff absences that need to be addressed. The proposed Simheuristic for tuning flexibility metrics is the first of its kind in rostering literature. It shows significantly better results than simply adjusting the aggregation of the demand forecast into a model week.

Furthermore, a Simheuristic consisting of ALNS to adjust the contracts and Monte-Carlo Simulation to test the feasibility of the resulting rosters is proposed to address the strategic planning problem of finding the best contractual combination for the next season. The experiments suggest that different demand patterns require different workforce structures, which implies that the airport should revisit its contracts every season to best address expected changes in passenger demands.

Finally, a case study of Edinburgh Airport's security staff scheduling problem and the development of a suitable rostering approach is presented.

# Lay Summary

This PhD Thesis presents methods to create flexible rosters for airport security staff and explores approaches to use such roster tools to support security workforce planning at airports.

Staff scheduling is a notoriously difficult problem to solve, exacerbated by larger problem sizes (i.e., higher numbers of employees). In airport security, which is unionised in the UK, schedules need to be approved by the union representatives before they are implemented. This approval process usually consists of several rounds of meetings between the chief staff scheduler (CSS) and union representatives, with subsequent schedule changes made by the CSS. As a result, schedules are based on demand forecasts made months in advance and are often impacted by disruptions caused by changes in demand patterns and staff absences that have to be repaired (at an often high cost to the airport) to ensure the smooth running of the security hall.

This thesis covers the collaboration with Edinburgh Airport to develop a flexible — adjustable to changing work rules and requirements — roster tool to support approval meetings between the CSS and union representatives. In addition, it explores further ways to introduce flexibility into rosters by finding the best levels of surplus staff — numbers of staff scheduled above the required demand — to schedule in advance. As a result, the disruptions caused by demand changes and staff absences should be easier (i.e. less costly to the airport) to repair.

Finally, this thesis explores ways to enhance long-term workforce planning decisions by using the roster tool to find the best contractual structure for a given demand forecast. Determining which contracts will be more useful than others in the long run, can help the CSS and HR departments at the airport with planning for the future, for example, by setting up long-term recruitment plans.

This thesis introduces a new variation of roster problems to address Edinburgh Airport's unique problem setting. Extensive computational experiments test the effectiveness of the proposed methods, and the results show that implementing these methods can lead to significant cost and time savings for the airport.

# Acknowledgements

First and foremost, I would like to express my deepest gratitude to my fantastic supervisory team, Maurizio Tomasella and Tom Archibald, for their unwavering guidance and support throughout my PhD journey. Our discussions over the past years have been invaluable, and I have learned so much from your expertise. It has truly been a privilege and a pleasure to work with you both.

I would also like to extend my sincere thanks to Lynne Moss at Edinburgh Airport for her invaluable insights into the complexities of airport security scheduling, which made this PhD possible. Your collaboration and feedback on the roster tool developments have been critical to this work.

My appreciation goes to Daniel Guimarans for his collaboration on the first paper of this thesis. Your thoughtful insights and contributions greatly enhanced the quality of the work.

I would like to thank Jared Cheung and Sam Song for their early contributions, particularly in conducting preliminary tests on robustness metrics and trying out different Simheuristics. Their work played an important role in shaping the content of Papers II and III, and I truly appreciate their efforts.

This PhD has been a long journey filled with both challenges and triumphs. I could not have completed it without the unwavering support of my parents, Anna and Karl Wiesflecker, and my brother, Konstantin Wiesflecker. Thank you for always standing by me, helping me push through the difficult moments, and celebrating each victory along the way.

A very special thank you goes to my fiancé, Liam Geach. You have been my rock throughout this entire journey, always offering your love, patience, and encouragement. I am especially grateful that you always reminded me of life beyond the PhD bubble, even when it felt all-consuming. For that, and so much more, I am eternally thankful. I am so excited to see what the future holds for us.

Finally, I would like to thank the PGR team at the University of Edinburgh Business School for their ongoing support and dedication to the PGR cohort. Your efforts to assist and advise us have been deeply appreciated.

(*Johanna Wiesflecker*)

# Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Johanna Wiesflecker*)

To my beloved parents.

# Table of Contents

# List of Figures

# List of Tables

# List of Acronyms

# Chapter 0

# Wayfinding

## 0.1 Preliminaries

This section outlines the motivation behind the work in this thesis and the resulting lines of research (Section 0.1.1) before presenting an overview of the structure of the remainder of this thesis (Section 0.1.2).

### 0.1.1 Motivation

Aviation is a growing sector with many people travelling for business, holidays and visiting family and friends (Department for Transport and Civil Aviation Authority, 2023, Table AVI0108). Before 2019, the UK aviation industry had seen a 30-year steady increase in passenger numbers, reaching peaks of 300 million passengers per year, until it came to an abrupt halt during the Covid-19 pandemic (UK Civil Aviation Authority, 2024, p. 15). Since then, the aviation industry has seen a fast recovery with the UK Civil Aviation Authority (CAA) reporting 78.9 million passengers in the second quarter of 2024 alone (an increase of 7% compared to the previous year) (UK Civil Aviation Authority, 2024, p.2). This puts their rolling annual total of 281 million passengers just short of the 300 million passenger peak observed in 2019 (UK Civil Aviation Authority, 2024, p.4).

Similar trends can be observed for aviation in all of Europe, with Eurocontrol expecting the number of European flights to reach 10.6 million (96% of peak levels reached in 2019) by the end of the year (Eurocontrol, 2024). Forecasts for the next few years suggest a return to previous peak levels of flights in Europe by summer 2025, with an expected further increase to a total of 12 million flights per year by 2030 (Eurocontrol, 2024).

This persistent growth in passenger numbers is causing some airports to reach their capacity limits — as evidenced by plans to expand in the future, such as Gatwick Airport's applications to add a second runway (Gatwick Airport, 2024). As a result, it is becoming more and more critical that airports use their available capacity to its full extent — even more so for capacity bottlenecks that can reduce the steady flow of passengers through the airport. Where these bottlenecks occur varies from airport to airport and often depends on the layout. Typical examples include baggage handling, passenger security, gate availability, or runway availability (TAV Technologies, 2023). At Edinburgh Airport — the collaborating partner for this thesis — the capacity bottleneck lies in the security checks, which take place in one security hall that all passengers, crew, and airport contractors working on site have to pass through.

The security officers on duty are a key component to the smooth running of the airport's security hall. They (among other things) run the security lanes with the hand luggage scanners, staff the body scanners, and perform additional checks if the scanners highlight issues. Ensuring that the security officers are happy with their work is, therefore, crucial to the airport's operation and, as a result, passenger satisfaction. What happens when people are dissatisfied with their working conditions has been repeatedly seen in the UK over the past years with walkouts and strike action happening in multiple sectors (BBC, 2022a,b, 2024b,a), which the airport clearly wants to avoid.

At Edinburgh Airport, the chief staff scheduler (CSS) is responsible for creating staff schedules. Their job is to create schedules that can cover the expected demand while also satisfying all work regulations for the security officers. This is a notoriously difficult problem to solve— even more so for large problems with high numbers of staff — and has been studied extensively by researchers in the Operational Research/Management Science (OR/MS) field (Ernst et al., 2004a; Van Den Bergh et al., 2013). The consensus in the rostering literature is that the people responsible for these scheduling decisions are often in need of a decision support tool (Ernst et al., 2004b). The CSS at Edinburgh Airport agrees with this but so far has been unable to find a suitable tool. The business has bought and tested various expensive software products over the years, never finding one flexible enough to model the airport's (perceived) unique staff scheduling problem.

As a result, the schedules for the 300+ security workforce at Edinburgh Airport are still created manually using spreadsheets. This is an extremely difficult and lengthy process for the CSS, and although it results in feasible rosters that are approved by unions, these are often far from optimal — meaning many staff preferences cannot be accommodated. In fact, a recent survey has encouraged the CSS to look at overhauling the schedules to help increase staff satisfaction.

Another factor that plays into the scheduling process at Edinburgh Airport is the fact that in the UK, airport security is unionised. This means that all schedules must be approved by union representatives before implementation. This usually happens through several rounds of discussions between the CSS and union representatives and subsequent schedule adjustments performed by the CSS. Although this process is necessary and benefits the security officers, it also adds more time and effort to an already lengthy and difficult process. As a result, schedules are created based on demand forecasts made months in advance. As time moves on, unforeseen disruptions such as demand changes and staff absences (e.g. short-term sickness) will lead to

infeasibilities that have to be fixed through (often costly) measures such as scheduling overtime — the budget for which is set based on estimated need by the CSS at the beginning of the year and should ideally not be surpassed.

To support the CSS in the tasks described above and tackle capacity constraints at Edinburgh Airport, two lines of research for this thesis can be identified:

1. **Need for a flexible roster tool** Edinburgh Airport is in need of a flexible roster tool to act as a decision support tool for tactical workforce planning. The meaning of flexibility is twofold: 1. Flexibility is needed in the schedules to help with re-rostering steps after infeasibilities have occurred 2. The tool itself needs to be flexible (i.e. it needs to be adaptable to the ever-changing rules and contract structure due changes in work regulations and preferences). So, this research aims to find ways to create a roster tool that encompasses both types of flexibility.

2. **Exploration of roster tool applications** An automated roster tool can be used for more decisions than just the tactical scheduling process. For example, the CSS is involved in budgetary decisions that determine workforce size and overtime availability. Supporting this process would be another possible application of such a roster tool. As a result, a second research line of exploring possible applications and uses of automated roster tools to support strategic decisions can be defined.

### 0.1.2 Thesis Structure

This thesis consists of five chapters (Chapter 0 - Chapter 4). The first chapter (Chapter 0) acts as an introduction to and overview of the research conducted. The following chapters (1-4) cover this thesis's theoretical and methodological contributions in the form of journal and conference papers. The first three papers (Chapter 1 - Chapter 3) are presented in the order the work was started, and Chapter 4 presents a case study of the collaboration with Edinburgh Airport that was completed in parallel.

For those readers who are not familiar with the research problem or techniques, Chapter 0 provides a self-contained introduction to the formal problem background (Section 0.2), an overview of the related literature and identified research gaps (Section 0.3), and a summary of the techniques used (Section 0.4).

Section 0.5 identifies the problem statements that could be identified from conversations with the CSS and shows how the four remaining chapters of the thesis relate to each other. In addition, it provides an overview of the contributions and learning points of each chapter (1 - 4) before presenting a summary of the contributions of the thesis overall.

The introductory chapter closes with a discussion of research limitations and possible ways to extend the work in this thesis (Section 0.6) before presenting the concluding remarks (Section 0.7).

In summary, the contents of the five chapters in this thesis can be condensed to:

- **Chapter 0** is the introduction to the research topic and presents a schematic overview of the contributions of the remaining chapters.
- **Chapter 1** presents the first paper, which covers the extension of Rotating Workforce Scheduling (RWS) problems to include teams and contracts.
- **Chapter 2** covers the second paper, which addresses ways to inject and fine-tune flexibilities (in the form of robustness metrics) in schedules.
- **Chapter 3** includes the third paper, which covers the use of simheuristics for Strategic Workforce Planning (SWP).
- **Chapter 4** contains the last paper, which is a case study about the collaboration with Edinburgh Airport to develop a flexible roster tool for their security staff scheduling problem.

## 0.2 Underlying Problems

This section introduces the problems that are covered in the thesis. It will start with an overview of the different stages of workforce planning (Section 0.2.1) and then cover staff scheduling in general (Section 0.2.2) before delving into variations of staff scheduling problems (Sections 0.2.2.1-0.2.2.3).

### 0.2.1 Workforce Planning

Workforce planning describes the decisions associated with planning for the evolution of a workforce over time. The decisions can be split into six general areas: recruitment, training, promotion, retention, attrition, and scheduling (Turan et al., 2020).

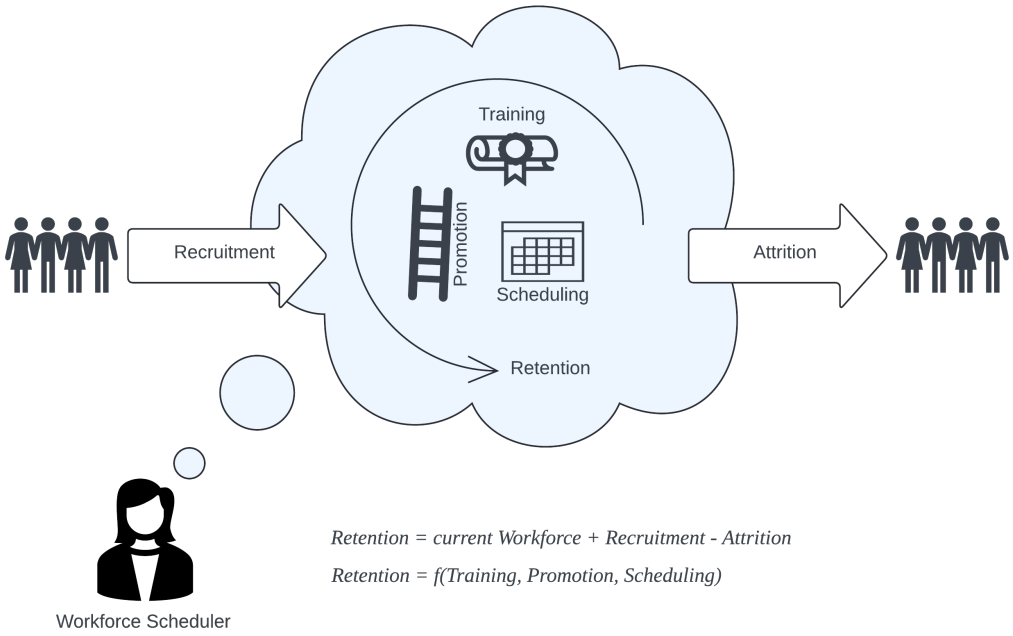



Figure 1: The six aspects of workforce planning.

Figure 1 depicts how these six aspects of workforce planning work together to model the dynamics of personnel in the workforce over time

*Recruitment* refers to the act of hiring new personnel into the workforce. While a business can directly decide how many new people they would like to hire, there are still many uncertainties that will come into play here. For example, wanting to hire a certain number of new staff does not automatically mean that many people are looking for these positions. In addition, the number of new staff that are needed can generally

be deduced from demand forecasts that can indicate the required workforce size paired with estimates of attrition. Oftentimes, neither of these measures can be predicted with 100% accuracy.

*Attrition* refers to current employees who are deciding to (or have been asked to) leave the business — for various reasons such as poor job performance, monetary constraints, being dissatisfied with their current position or finding a better job somewhere else (Madigan and Kim, 2021). High turnover is generally not ideal for a business as it means having to recruit new staff who are unfamiliar with the business and its tasks, which can slow things down and lead to inefficiencies.

*Retention* refers to the workforce that remains in the business and can be estimated as the current workforce size minus expected attrition plus recruitment, for a given reference timeframe. Generally, a company that relies on extensively trained staff — such as airport security— should endeavour to keep retention rates high, which usually correlates to high staff satisfaction. To keep employee satisfaction (and retention rates) high, the remaining three workforce dynamics — Promotion, Training and Scheduling — come into play to help and create development opportunities, favourable workforce environments, work-life balance and other factors that have been shown to increase retention rates (Kostanek and Khoreva, 2018).

*Promotion* describes the movement of employees into higher positions within the business — with the opposite movement being *demotion*. Giving clear opportunities to further career paths and positions can encourage staff to stay and work their way up in the business hierarchy instead of looking for positions elsewhere, thus keeping retention rates high (Kostanek and Khoreva, 2018). However, offering employees promotion opportunities will also give them a chance to increase their experience in the field, thus making them more suitable for similar positions in other companies. This, in turn, can have a negative impact on retention rates. Demotion, on the other hand, will have the opposite effect on retention compared to promotion. So, employees who have been demoted will have a more difficult time finding a position elsewhere, but they will be more likely to want to leave the business.

*Training* allows employees to learn more about their current role — keeping up to date with current developments in their field — or gain new skills to support staff development, which can lead to promotion opportunities within the business. However, similar to promotion itself, giving staff the opportunity to learn more skills and gain certificates will also make them suitable for more positions outside the business, thus increasing the chance of them leaving.

*Scheduling* describes the act of creating work schedules for a workforce by balancing the business needs with the contractual regulations of the workforce (Ernst et al., 2004b). Scheduling is a highly complex problem with numerous variations (see Section 0.2.2 for more details on relevant aspects of staff scheduling for this thesis).

#### 0.2.1.1 Workforce Planning Stages

Workforce planning is a recurring process that happens over a long time period and often in multiple steps. Figure 2 gives an overview of the three different planning stages (strategic, tactical and operational planning), where the timeline they occur and the typical decisions made during the stages.

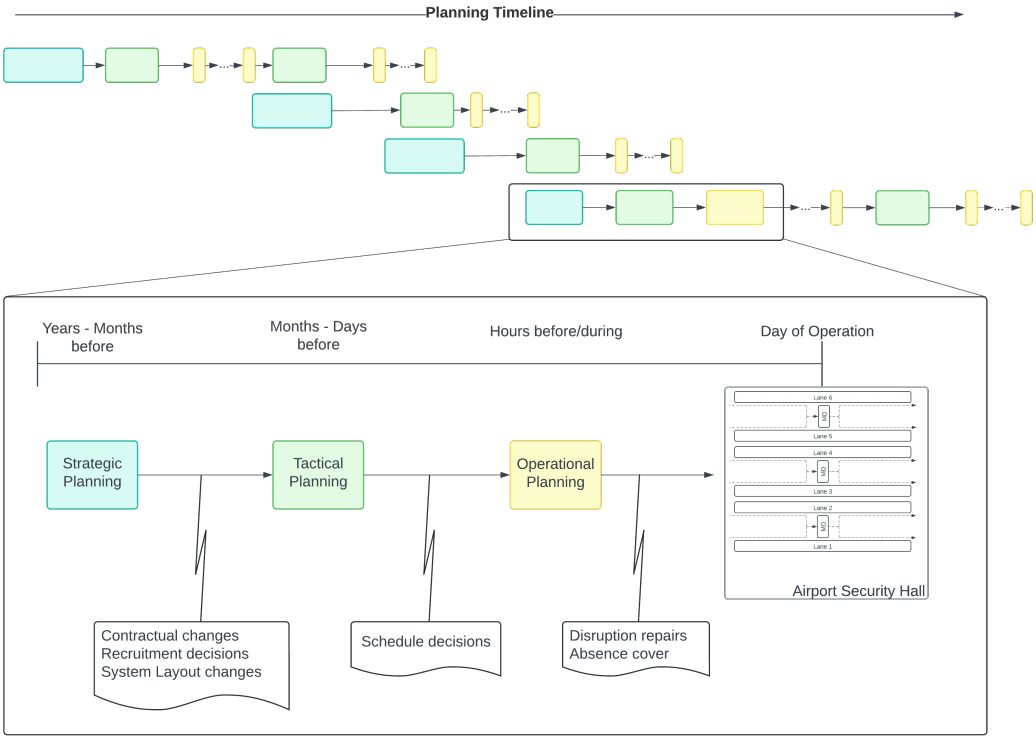



Figure 2: Cycles of Workforce Planning Stages along the Planning Timeline

Once one cycle of workforce planning is complete, it is usually time for the next cycle to start. Sometimes, the different stages of subsequent workforce planning cycles are performed simultaneously. For example, if the strategic planning stage for one time period is complete, it will be time to start the strategic planning process for the next period while tactical planning for the first takes place. In addition, tactical and operational planning is often repeated during one workforce planning cycle.

Tactical planning decisions might be reviewed a few times in a cycle, while operational planning decisions, such as disruption repairs, can occur as frequently as every day or even hourly.

*Strategic planning* is the first stage of workforce planning that takes place the longest time (several months to a year — if not several years) before the actual day of operation. During this stage, decisions will be made regarding issues that have the most significant impact on the workforce and on subsequent decisions. Often, they need to be approved by multiple stakeholders, and they might take a long time to implement. Examples include changes to the contract structure, recruitment/promotion decisions, and system layout changes — such as changes to the x-ray machines in airport security halls (Department for Transport, 2022).

*Tactical planning* follows strategic planning. At this stage, all decisions from the strategic planning stage are taken as given. For example, contracts cannot be changed during tactical planning. Instead, they make up part of the rules that the work schedules need to adhere to. The main decision to be taken at this stage of the planning process is to create work schedules — sometimes also referred to as *rosters* or *rotas* — , allocate overall tasks and plan for unforeseen changes. As time moves closer to the day of operation, this can also include small changes to the existing roster if more information becomes available — holiday and long-term sickness cover or changes to expected demand patterns.

*Operational Planning* refers to last-minute changes that can be made on the day itself. This usually refers to small changes such as reallocating tasks or asking personnel to stay on a little longer/come in earlier. If the planning during the strategic and tactical phases has gone well, there will be enough flexibility planned into the shift to cover discrepancies between expected and actual demand without changing the roster. In the worst cases, uncovered demand will build up, and tasks will not be completed because there is no staff available to deal with them — in airport security, this would lead to scenarios of passengers queuing outside the security hall and experiencing extremely long waiting times.

### 0.2.2 Staff Scheduling

Staff scheduling has been studied since at least 1954 (Dantzig, 1954) with various problem variations considered. As a result, many literature reviews exist on the topic, with the most popular and comprehensive ones written by Ernst et al. (2004b) and Van

Den Bergh et al. (2013). This section will cover the classifications of staff scheduling that are relevant to the problem addressed in this thesis.

#### 0.2.2.1 Types of Scheduling

For the purpose of this thesis, the general staff scheduling problem is split into three general problems consistent with classifications presented by Ernst et al. (2004b): days-off scheduling, shift scheduling and break assignment.

*Days-off scheduling* is the problem of assigning, for each member of the workforce and for every day in the schedule, whether it will be designated as a day off or a work day. Typical rules that apply here are the maximum or minimum number of consecutive work days/days off in the schedule, maximum number of work days per calendar week and, for industries with 7-day work weeks such as airport security, weekend off regulations (Bennett and Potts, 1968). Some models take it even further than simple "days-off" assignments by specifying at what time of the day (i.e. morning/afternoon/night — also referred to as *shift type*) shifts will be assigned. Usually, these models come with additional rules for the order of shift types in the roster. The most common is referred to as *forward rotation*, which means that two consecutive shifts must be of the same type or the second shift type must be later in the day than the first shift type. For example, with forward rotation, it would be possible to have a pattern of *"day1: morning - day2: afternoon"* but not *"day1: afternoon - day2: morning"*. In addition, it is possible to have upper and lower limits on the number of consecutive shifts of the same type — similar to days-off rules — and other specifications such as only allowing consecutive shifts of the same shift type. For the remainder of this thesis, the resulting schedule of this type of problem will be referred to as a *Rough Roster*.

*Shift Scheduling* — also referred to as *tour scheduling* — is "*...the problem of selecting, from a potentially large pool of candidates, what shifts are to be worked, together with an assignment of the number of employees to each shift, in order to meet demand*" (Ernst et al., 2004b). In the context of this thesis, this means that for every shift in the schedule (or day with an assigned shift type in the Rough Roster), the correct start and end times need to be selected so that all working regulations are satisfied and the workforce demand is covered at all times. This means that the shift start/end times match with the associated shift type that is assigned in the Rough Roster and the regulations for maximum/minimum shift duration. In addition, any member of the workforce who works that schedule will need to comply with their

contractual limitations on allowed weekly hours, meaning measures such as weekly hours and average shift duration also need to be considered. In their definition of shift scheduling, Ernst et al. (2004b) also include the assignment of meal breaks. However, for larger and more complex problems, it has been shown that a two-stage decomposition approach to Tour scheduling and break assignment is more tractable (Kiermaier et al., 2020).

*Break Assignment* describes the problem of assigning breaks (for meals or rest) to shifts in a schedule. Typical constraints for these types of problems include maximum/minimum work time between the start/end of a shift and a break, regulations for the number and duration of breaks that need to be assigned depending on shift duration, or there can be rules about work time between breaks if there are multiple breaks during one shift. Kiermaier et al. (2020) provide an overview and classification of existing break assignment problems (BAPs) that classify the problem according to three categories.

1. Number of Breaks: the number of breaks, which can be a *single break*, a set number of *multiple breaks*, or *fractionable* (multiple breaks that need to add up to a certain duration).

2. Break Duration: For single and multiple breaks, the duration can be either *fixed* or *variable*.

3. Break Assignments: breaks can be either assigned in fixed *time windows* or according to *workstretch duration* rules (i.e. upper/lower bounds on the allowed work durations without breaks in between).

Scheduling problems that include all aspects of scheduling (i.e. days-off scheduling, shift scheduling and break assignment) are often too difficult to solve as one overarching problem. Instead, they are commonly decomposed into separate problems that are solved consecutively. Figure 3 gives a simplified overview of the decomposition approach used to solve the scheduling problem considered in this thesis. The approach starts by assigning days-off, shift types and locations to the schedule (Module A1). The resulting Rotating Schedule is used as an input to the Shift Scheduling module (A2) which assigns the start and end times of all shifts. Finally, the resulting lines of work — the schedule with all assigned days-off, shift times, and tasks (if they are part of the problem) — are used as input to the Break Assignment module (A3), which adds breaks to the existing shifts.

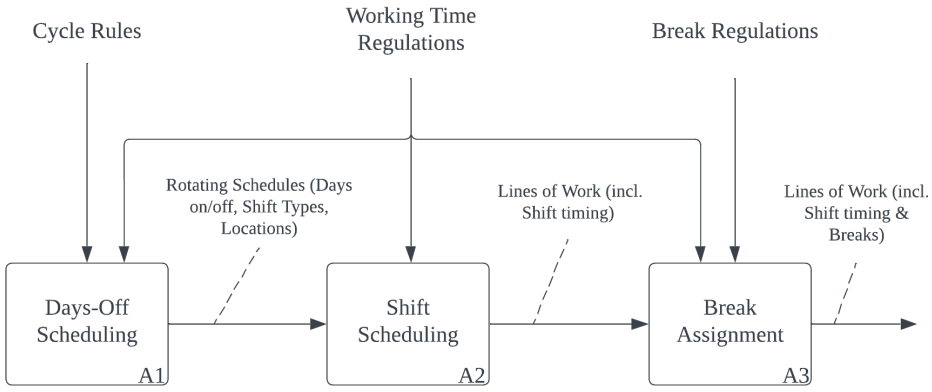



Figure 3: Scheduling Decomposition Approach.

#### 0.2.2.2 Types of Schedules

Workforce schedules are usually created to cover the entirety of the planning horizon. So, at the planning stage, each employee is assigned a schedule. There are two main types of such schedules that an employee could receive: *cyclic* and *acyclic* schedules.

*Acyclic schedules* are individualized schedules for each employee. When using acyclic schedules, each member of the workforce usually receives their own schedules for the planning horizon, which does not match or has limited overlap with the schedules of other employees.

*Cyclic schedules*, sometimes also referred to as rotating (workforce) schedules, on the other hand, are constructed in such a way that all employees work the same schedule repeatedly over the planning horizon. They are set up so that each employee (or groups of employees) start at a different line (e.g. week) in the schedule and then canonically rotates through the schedule until the end of the planning horizon. Figure 4 gives an example of a schedule with 4 lines and 4 groups of employees. Each employee starts in a different week and then cycles through the schedule. If they reach the end of line 4 they start back at the beginning of line 1.

Compared to acyclic schedules, adopting cyclic schedules comes with both advantages and disadvantages. Employees generally consider cyclic schedules to be fairer because as time goes on, everyone will work the same schedule, and there is no feeling of preferential treatment over assigning certain shift patterns to others. However, cyclic schedules are not suitable for all demand patterns. For the cyclic schedule to work, demand should either be fairly stable or follow a repeating pattern

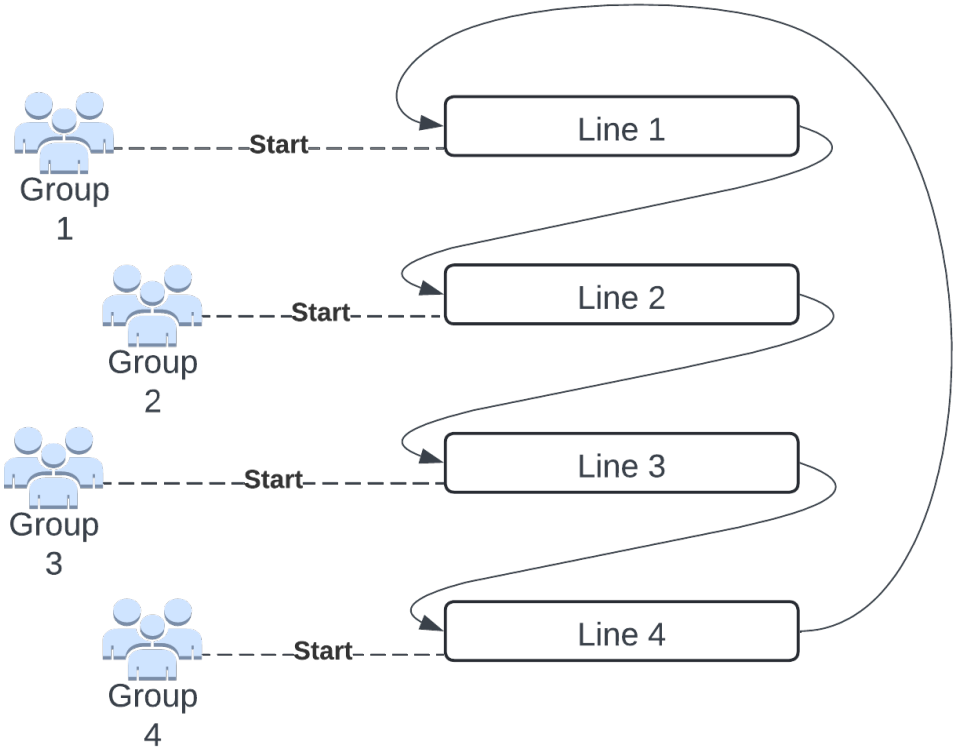



Figure 4: Cyclic schedule with 4 lines and 4 employee groups.

to which the schedule can be matched. In addition, in cyclic schedules, all rules that apply to shift assignment (such as days off rules or shift type order) must also be considered for the transition from the end of the last line to the beginning of the first line. This can make schedule creation more complicated.

#### 0.2.2.3 Workforce Features

Within a workforce, individual employees can usually be grouped together by different features — with the most common ones being *skills* and *contracts*.

*Skills* cover the different tasks a member of the workforce can perform while on shift. If not all employees have the same skills (i.e., the workforce is *heterogeneous*), then the scheduler needs to ensure that enough employees with the required skills are scheduled to be on shift to complete all tasks. Conversely, if the workforce is *homogeneous* (i.e. all employees have the same skills, such as in the airport setting considered here), the problem simplifies to ensuring that the correct number of people is on shift to cover workforce demand.

*Contracts* are the main source of the rules and regulations that schedules need to satisfy. They specify everything from weekly work hours, number of workdays per week and allowed shift patterns down to break allowance and how these breaks need to be allocated to shifts. It is often the case that different groups of employees within the workforce have different contracts. One of the most common variations is full-time vs. part-time staff. However, it is also possible to have differences in the allowed work days or shift types (e.g., night shift workers) or other contractual features (e.g., the need to protect times of the day when the employee needs to act as a carer for family).

## 0.3 Related Literature

This section will give an overview of the relevant literature for the problems discussed in this thesis, which was inspired by the collaboration with Edinburgh Airport. The problems covered can be split into three main areas of related literature. 1) Workforce Planning, 2) Rostering and 3) The use of Robustness Metrics in Rostering.

### 0.3.1 Strategic Workforce Planning (SWP)

SWP "examines the gap between staff availability ... and staff requirements ... over time, and prescribes courses of action to narrow such a gap" Doumic et al. (2017). This section aims to introduce the work on SWP in the existing literature. In recent years, there has been an evolving acknowledgement that SWP encompasses more than just the technical aspect of planning. Ultimately, workforces consist (largely) of humans and the human element must be considered during decision-making — this has also been a recurring topic during the ongoing work with Edinburgh Airport on their security staff scheduling problem. Sokol and Tarulli (2024) work towards a more coherent and overarching approach to SWP by aiming to bring together industrial and organizational psychologists and SWP professionals.

More generally, SWP is widely applicable, and as a result, literature reviews have been developed focusing on specific industries in addition to more general overarching reviews. For example, Sutton et al. (2023) provide a review of SWP in healthcare, Jnitova et al. (2017) present a literature review on workforce planning in a military setting and De Bruecker et al. (2015) discusses SWP more generally.

The existing SWP literature presents various approaches to solving stochastic SWP problems, with simulation[1] proving to be a popular approach, so much so that Jnitova et al. (2017) focus solely on simulation models in their literature review on SWP methods for military settings. As a result, this literature overview distinguishes between two main categories of SWP modelling and solution approaches: those that utilise simulation and those that do not.

For the approaches that do not use simulation, typical methods include stochastic programming and Markov chains. Jaillet et al. (2022) use stochastic programming to model the progression of their workforce through different grades. Their model also accounts for the time each person spends in their respective grade to account better for

[1] In this thesis, simulation refers to computer-based simulation, which involves the development of algorithms that represent the system to be studied (Loper and Register, 2015).

attrition rates. MacDonald and Paul (2024) also use stochastic modelling but applied specifically to a military setting with the goal of producing a goal to assess 'force readiness'. Han et al. (2024) provide an example of applying Markov chains to the SWP problem. Their approach models the skill level of individual members of the workforce as a Markov chain. The information from the collection of all individual Markov chains then feeds into an overarching Markov chain that models the workforce size at the operational level.

Zeng et al. (2019) consider a problem that could lead to an interesting application for Edinburgh Airport's roster tool. Their aim is to find the smallest possible workforce size that still makes it possible to produce feasible schedules for a given demand forecast. The authors use a branch-and-bound algorithm to solve the resulting model. Given the airport's need to balance budgets for upcoming seasons, a similar approach, including their roster tool, could prove beneficial to their strategic planning decision-making process.

An interesting work to highlight is that of Llort et al. (2019), who produce a decision-support tool for staff planning in consulting. Not only do the authors highlight the usefulness of Decision-support tools for SWP, but they also discuss the long-term effects of the staff planning decisions in question. The training for consultants can be lengthy as they are highly skilled professionals. As a result, the decision to take on and train another consultant — or security staff member — will impact the workforce size in the future rather than at the time the decision is made. Edinburgh Airport faces a similar problem, as their security staff needs to go through extensive training and security checks before they can start working. As a result, this training time needs to be considered as part of the hiring decisions.

Approaches for SWP that use simulation are not new. Parker and Marriott (1999) already used flow diagrams to represent the flow of personnel through the different job roles in a company. This allowed the authors to simulate the transitions between roles as well as attrition and recruitment to find the steady-state flow of personnel, which could then be used to make decisions about the workforce.

More recently, Willis et al. (2018) used system dynamics for strategic personnel planning in healthcare to create an overarching multi-methodology framework for SWP in England. Their framework is designed to help the public sector determine how many (medical or dental) students would need to start their courses now to cover possible workforce (i.e. fully trained doctors and dentists) demand scenarios in the future by acknowledging the 'intrinsic uncertainty and complexity of factors affecting

workforce demand and supply' rather than attempting to predict exact outcomes.

Smit et al. (2023) propose a simulation-optimization approach to SWP that is based on deep reinforcement learning. Their approach is based on a loop of optimizing pre-defined strategic planning goals (such as determining workforce size or minimizing salary costs) using a deep reinforcement level. The resulting policies are then tested using a simulation based on historical data.

Akl et al. (2022) provide a rare example of considering tactical planning decisions as part of the SWP model. The authors use a simulation-optimization model to find the required workforce size (strategic planning decision) to allow for optimal maintenance scheduling at the tactical decision level. As part of their approach, they account for 'backlog' costs due to the maintenance schedule not covering all required tasks.

## 0.3.2 Rostering

Rostering literature dates back to Dantzig (1954), who considered a toll booth scheduling problem. Since then, extensive research has been conducted on various scheduling problems and several literature reviews provide overviews on the existing research on the topic (Baker, 1976; Ernst et al., 2004a,b; Alfares, 2004; Van Den Bergh et al., 2013). The aim of this section is not to replicate what has already been done by these reviews. Instead, it provides an introduction to rostering literature and an overview of recent and important articles that are relevant to the problems considered in this thesis.

### 0.3.2.1 Cyclic Rostering

Cyclic rostering dates back to Bennett and Potts (1968), and presents a decomposition approach for creating a rotating roster for a transit system. Their method first allocates days off and subsequently allocates shifts to the schedule. Other early approaches to cyclic rostering include work on the so-called (k,m) cyclic staffing problems, in which personnel are scheduled to work for k consecutive shifts and then assigned off for m-k shifts. Solution approaches include formulating the problem as a set of network flow problems and Linear Programming (LP) -based approaches (Bartholdi et al., 1980; Bartholdi, 1981). Bechtold (1981) propose several heuristic algorithms for cyclic rostering problems with more advanced days-off regulations (e.g. one day off per cycle or two consecutive days off). The heuristic algorithms are specifically designed to address each of these constraints while covering arbitrary demand patterns.

Heuristic and metaheuristic approaches were also popular for tackling cyclic rostering problems in the early 2000s with Mörz and Musliu (2004) proposing a Genetic Algorithm (GA) at Industrial plants — which have to be operational 24h for seven days a week. A few years later, Purnomo and Bard (2007) presented a branch and bound algorithm to solve a cyclic nurse scheduling problem.

Musliu et al. (2018), and the subsequent extension by Kletzander et al. (2019) propose the first modelling approach that can solve all standard benchmark instances available at the time of publication as well as a set of new benchmark instances that the authors added. Their approach uses a single high-level and solver-independent modelling language to express the model. This makes it possible to quickly switch between different (Constraint Programming (CP) and Mixed Integer Programming (MIP)) solvers to test their efficiency on the problem. They compare two new formulations of the cyclic rostering problem based on modelling rules on the length of 'work' and 'off' blocks in a roster. In this setting, 'blocks' describe a set of consecutive days in the roster that are all assigned either a shift in general, the same shift type, or a day off. The first employs global constraints from CP to express certain constraints in the model. Using CP is not a new idea in cyclic rostering. In fact, Laporte and Pesant (2004) already successfully applied CP to cyclic rostering problems. However, pairing CP with the solver-independent modelling language, as shown by Musliu et al. (2018) and the resulting versatility, provides a novel twist on the approach. Expanding on the use of CP constraints, the second model presented by the authors encompasses the days-off rules in a single deterministic finite-state automaton that is expressed through the use of the so-called *regular* constraint Pesant (2004).

Moving away from work/off block formulations, Becker et al. (2019) propose two formulations based on work stints for their cyclic rostering problem for emergency services. Stints refer to patterns of shifts and days off that would be allocated to the roster, and the problem consists of selecting the right combinations of those patterns to satisfy the given demand. Becker et al. (2019) solve their problem with a MIP solver — thus improving the previously used manual approach for the emergency service provider they worked with. Continuing the stint-based rostering work Becker (2020) proposes a decomposition approach, which focuses on creating a set of feasible stints before allocating the stints to a schedule.

The current state-of-the-art approach to solving cyclic rostering problems was presented by Becker et al. (2022). The authors propose a graph-based formulation for

their stints-based cyclic rostering problem, which is solved via a branch and cut algorithm which significantly outperforms the previous state-of-the-art approach (Musliu et al., 2018) for all known cyclic rostering benchmark instances at the time of publication.

Kletzander and Musliu (2022) apply hyper-heuristics to cyclic rostering problems in industry. The authors show that while their approach does not improve the state-of-the-art at the time, there is a lot of value in having an approach that provides the flexibility to quickly adapt the model to changing work regulations — something that is important when schedules are subject to lengthy negotiation such as in the context of Edinburgh Airport.

Guo and Bard (2024) provide a recent example of cyclic rostering paired with tour scheduling and break assignment, which is solved via a decomposition approach — as this thesis does with Edinburgh Airport's scheduling problem. The authors addressed an Air Traffic controller scheduling problem through a 2-phase approach (consisting of a greedy algorithm and mixed-integer programming) by first assigning shifts to controllers before allocating breaks. They tested their approach on realistic instances provided by the Federal Aviation Administration and on instances with up to 52 employees, which are considered large by the authors but compared to airport security rostering would still be seen as small.

#### 0.3.2.2 Cyclic Rostering with Teams, Contracts and Skills

The underlying idea of cyclic rosters is that the entire workforce works the same schedule in the long run, thus increasing the feeling of fairness among colleagues. Combining this with a workforce that is split into different contracts or with different skills can, at first glance, seem counter-intuitive. However, there are also benefits from using cyclic rosters for a heterogeneous workforce. This section covers recent literature that addresses the use of contracts, teams (or groups) and preferences in combination with cyclic rosters.

Laporte (1999) is one of the few papers that use different rosters for full-time and part-time (different contracts) staff who need to work together to cover the workforce demand. The idea is that employees with the same contract (i.e., full-time or part-time) will work the same schedule, and those on different contracts will work different schedules.

In addition, Laporte (1999) gives a detailed account of how to produce cyclic rosters (for relatively small cycles of 4-5 weeks) by hand while touching on critical aspects of cyclic rostering that he encountered in his own work as an academic consultant on the topic of rostering. In the article, he states:

> "Over the years, I have observed that several of the problem's constraints must be bent to produce manageable schedules, sometimes in a major structural way, but it is difficult to tell beforehand which particular constraints should be relaxed in a given situation, which is why most algorithms typically fail to produce acceptable schedules."

This notion of rosters created through mathematical approaches being too rigid has been a key issue while working with Edinburgh Airport. Chapter 4 will present this journey in more detail. Still, a key takeaway (in line with the given quote) is that no matter how good a solution approach is, in the end, rostering/scheduling is working with people. There will always be a human factor affecting the final solution — meaning the rigid constraints established during rostering will ultimately need to be relaxed once real people are assigned to the schedules.

Rocha et al. (2013) also consider a cyclic rostering problem with full-time and part-time staff. However, their approach focuses on first allocating as many full-time employees as possible. Part-time employees are only used to make up the difference between required and scheduled demand where necessary. As a result, part-time staff do not actually follow a cyclic roster.

Instead of contracts Rocha et al. (2014) addressed the problem of creating cyclic schedules for teams of otherwise homogeneous employees. In their problem, the team size is fixed in advance, so the teams become the rostering unit, which makes the problem no different to scheduling single individuals. Kiermaier et al. (2016) also consider a cyclic rostering problem for groups of employees. However, their discussion around team size is more extensive. They highlight how the team size implicitly affects the length of the roster cycle. Given a set workforce size, the team size determines how many teams of employees there are to be rostered — a higher team size equals a smaller number of groups and vice versa. In a cyclic roster, each employee (or group of employees) starts at a different line of the roster. Hence, the number of available teams directly determines the number of lines in the cycle. So, if, for example, a shorter cycle is preferred, larger teams are required to make the schedule possible.

Maenhout and Vanhoucke (2009) address cyclic rostering problems for nurses with differing work regulations and preferences. To solve this problem, the authors first produce two basic cyclic schedules which address the two main work regulations (i.e.

one schedule with night shifts and one without). For each individual nurse, the basic cyclic schedule that fits their work regulations is then adapted to suit any additional preferences. The resulting schedules differ from nurse to nurse so they are not truly cyclic schedules in the end.

### 0.3.3 Robust Rostering

In the typical planning timeline categories of strategic, tactical and operational planning, Rostering typically happens within the tactical planning stage that takes place weeks if not months before the day of operation (Figure 2). This means that everyone who works the roster knows weeks in advance when their shifts are scheduled to happen, which helps with planning and organising their life outside of work — so this is seen as beneficial to the workforce. However, problems start to arise when uncertainty comes into play. Van Den Bergh et al. (2013) differentiate between three main types of uncertainty in rostering:

- *Uncertainty of demand* which refers to the levels of workload
- *Uncertainty of arrival* which refers to the arrival pattern of the workload
- *Uncertainty of capacity* which refers to the availability of the workforce

In airport security rostering, uncertainty of demand and arrival manifests in the arrival of passengers to the security hall. While airport operators know when flights are scheduled to take-off, they do not have the information about how fully booked each flight is, nor do they know when passengers plan to arrive for their flights. Uncertainty of capacity, on the other hand, would address any unforeseen changes to the available workforce on the day. While the airport operator knows in advance how many members of staff are scheduled to work on a particular day, they cannot know in advance if anyone will be off sick or otherwise unable to work their scheduled shift. As a result, there is a clear need to plan ahead for these eventualities and ensure that the roster is sufficiently robust to unforeseen disruptions.

In the rostering literature, there are several approaches to mitigate the effect of disruptions due to uncertainty, such as *stochastic modelling*, *(dynamic) rescheduling* approaches and *building robustness directly into the roster*.

Out of the recent rostering literature reviews, Van Den Bergh et al. (2013) provides the most detailed discussion on the topic. They found that the majority of papers they reviewed reduced the inherently stochastic problems to deterministic rostering

problems, while less than half included some level of stochastic demand, arrival or capacity in their approach (Alfares, 2007; Atlason et al., 2004; Dietz, 2011; Liao et al., 2009). A more recent example of a stochastic modelling approach to cyclic staff scheduling was presented by Kiermaier et al. (2016) who presented a two-stage stochastic model for a staff scheduling problem in the service industry. However, these approaches are always limited by the available information about the uncertainty that is modelled as part of the approach. Ultimately, there will always be some eventualities that cannot be accounted for.

One way to rectify any infeasibilities that arise due to unforeseen disruptions is through rescheduling. Rescheduling problems have been studied extensively in different rostering environments and several literature reviews are available on the topic (Herroelen and Leus, 2005; Ouelhadj and Petrovic, 2009; Larsen and Pranzo, 2019)

More recent work on dynamic rescheduling problems is presented by Maenhout and Vanhoucke (2023), who recognise that disruptions occur progressively throughout the planning horizon and present strategies for identifying the most suitable rescheduling decision based on the scenario faced by the scheduler.

Regardless of how well it is executed, rescheduling will almost always negatively affect the workforce. No one likes to have their shifts changed at the last minute or be called into work on a day off. To make it easier to adjust schedules for disruptions, we need to introduce flexibilities into the roster. Van Den Bergh et al. (2013) refer to this as "...adding capacity buffers to make personnel rosters more robust".

Rostering literature has been addressing this problem in different ways over the past years. In general, methods for introducing flexibility or robustness to rosters are either focused on 1. the shift times in a roster or 2. the staff scheduled to work.

Lusby et al. (2012) fall into the former category by presenting a way to introduce flexibility into ground crew rosters by allowing shift start and end times to overlap to introduce flexibility that leads to more robust rosters. Along a similar line of thought, Ingels and Maenhout (2018) acknowledge that one — costly — way to address disruptions is though allocating overtime. Their approach looks at balancing initial staffing costs and overtime budget through pre-emptively scheduling overtime in the roster and thus making the roster more robust to unforeseen disruptions. Their results show that for small workforce sizes, it is best to preemptively schedule more overtime and reduce planned understaffing, whereas larger workforce sizes benefit from a mixture of pre-emptively and reactively scheduled overtime.

Ingels and Maenhout (2015) fall into the second category of adding flexibility to rosters by considering different strategies for pre-emptively scheduling reserve shifts. They test the different strategies through simulated disruptions and rescheduling models. As expected, they find that higher levels of reserve shifts lead to better performance under uncertainty but also higher staff costs. Also focusing on staff as a way to introduce robustness, Ingels and Maenhout (2017) introduce a measure called employee substitutability which measures how easily a shift can be substituted with another. The aim is to have an employee substitutability value that is a high as possible to make the schedulers job during the reactive re-rostering phase easier. Another approach to improving a roster's robustness is presented by Ingels and Maenhout (2019), who use simulation-optimization techniques to determine ideal levels of buffer shifts for their rostering problem and compare these to more traditional approaches of pre-defined buffer levels.

Wickert et al. (2021) are the first to call for and introduce formal robustness metrics for rostering in the form of surplus shifts and measuring expected re-rostering cost. Until now, there has been no further development on this front, so there is still a need for a more comprehensive list of robustness metrics for rostering as well as techniques to find the best metric levels and combinations thereof to enforce in schedules.

### 0.3.4 Identifying the Research Gaps

After reviewing the existing literature, the following research gaps can be identified:

RG1 With the exception of Akl et al. (2022) there is a lack of approaches that incorporate tactical planning decisions (i.e. scheduling) directly into their SWP models. Given how crucial the creation of feasible rosters is to the smooth running of the operation, it should clearly be considered as part of the wider workforce planning decisions (such as SWP).

RG2 There are no solution approaches for cyclic rostering problems with workforce structures (multiple contracts with teams of different sizes) as faced by Edinburgh Airport. As the previous section shows, some articles cover aspects of the overarching problem but no work combines all problem features.

RG3 Research on cyclic rostering problems mostly focuses on improving the state-of-the-art solution approach by beating runtimes. However, there is little work on creating flexible approaches that can be easily adapted as the rostering requirements change. Kletzander and Musliu (2022) touch on this issue in their work, but they are the exception within the wider literature on cyclic rostering.

RG4 The problem sizes considered in cyclic rostering literature are far below those faced by schedulers in airport security. Most standard benchmark instances for cyclic rostering problems include workforce sizes of $\leq 70$ (Becker et al., 2022) and Guo and Bard (2024) refer to 52 employees as a large workforce. However, the security workforce at Edinburgh Airport (and other airports) currently consists of 300+ employees, with the plan to increase in the future. There is a clear lack of research for such problem sizes in existing cyclic rostering literature

RG5 As the work of Wickert et al. (2021) has shown, there is merit in formally defining robustness metrics for rostering. As a result, there is clearly a need to continue research into more comprehensive and formally defined robustness metrics to inject flexibility into staff schedules

RG6 As more robustness metrics are defined, strategies for fine-tuning metric levels and combinations thereof are needed. Especially, for newly defined metrics, it will be necessary to have methods that help schedulers decide what levels of these metrics are suitable to enforce in their rosters

The work in Chapters 1 - 4 will add to the existing literature by addressing these research gaps using the techniques outlined in Section 0.4. Section 0.5 summarises the contributions made in this thesis and directly links them to the above-listed research gaps.

## 0.4 Underpinning Techniques

This section gives an introduction to the main techniques used in this thesis. Figure 5 shows where the work in this thesis applies these techniques in the planning stages of the problem.

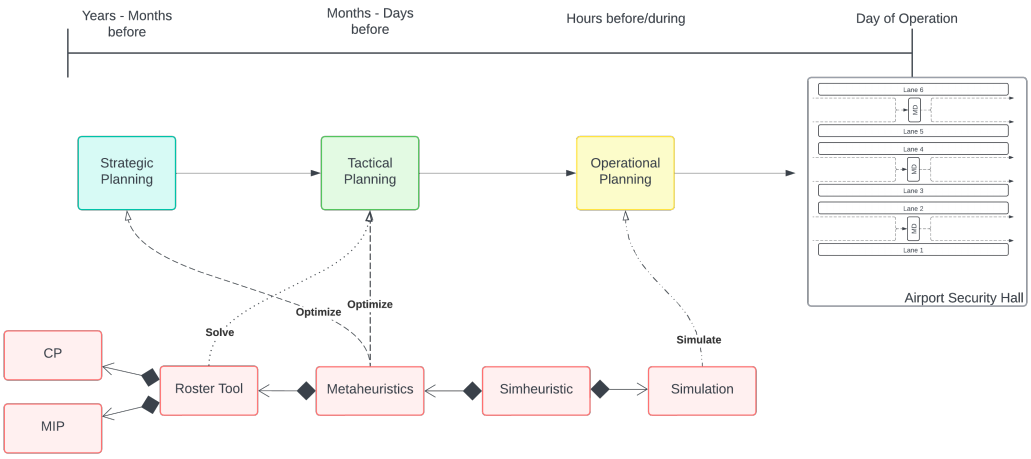



Figure 5: Planning Stages with Underpinning Techniques

As the chapters unfold, MIP and CP — which are introduced in Sections 0.4.1 and 0.4.2 — will be used to tackle the tactical planning problem of creating cyclic rosters for airport security personnel (Chapter 1).

This is followed by the use of Simheuristics to introduce flexibility into the rosters at the Tactical Planning stage (Chapter 2) before tackling the strategic planning problem of selecting the best contractual combination for the upcoming season (Chapter 3). In both approaches the Simheuristic simulates the operational decisions of repairing the roster after disruptions. Section 0.4.3 provides an introduction to Simulation-Optimization and Simheuristics.

### 0.4.1 Mixed Integer Programming (MIP)

Mathematical programming (e.g. LP, integer programming (IP) or MIP) constitutes the most widely used solution method for rostering problems in the literature Van Den Bergh et al. (2013). Bixby (2012) provides an extensive summary of the developments on LP and MIP and associated solution methods since the 1950s.

Early approaches to modelling the cyclic (or rotating workforce) rostering problem use IP with circulant matrices to express the scheduling constraints Bartholdi et al. (1980); Bartholdi (1981).

Glover and McMillan (1986) provide a general rotating workforce problem formulation that was later used and extended (with additional rules) as part of multiple papers Musliu et al. (2002); Mörz and Musliu (2004); Erkinger and Musliu (2017); Musliu et al. (2018); Kletzander et al. (2019).

**MIP-Formulation: Cyclic Rostering**

Musliu et al. (2018) provide an example of a cyclic rostering problem formulation using the following parameters:

$n$: Number of Employees

$\mathbf{A}$: Set of $m$ shifts. $\mathbf{A}^+$ refers to the set of all shifts, including days off ($O$)

$w$: schedule length

R: $m \times w$ Requirements matrix — each element $R_{i,j}$ represents the workforce demand for shift type $i \in \mathbf{A}$ for day $j$.

$F_2$: $= (sh_1, sh_2)$ Forbidden shift combination of length 2. This means that a shift of type $sh_1 \in \mathbf{A}$ can never be followed by a shift of type $sh_2 \in \mathbf{A}$.

$F_3$: $= (sh_1, sh_2)$ Forbidden shift combination of length 3. This means that a shift of type $sh_1 \in \mathbf{A}$ can never be followed by a day off, which is then followed by a shift of type $sh_2 \in A$ (i.e. you can never see the pattern '$sh_1 - O - sh_2$' in the schedule).

$l_s, u_s$: minimum and maximum permitted length of consecutive shifts of type $s \in \mathbf{A}$.

$l_w, u_w$: minimum and maximum permitted length of consecutive work days in the schedule.

$S$: $n \times w$ matrix representing the workforce schedule. Each element $S_{i,j} \in \mathbf{A}^+$ represents the shift type (or day off) that employee $i$ works on day $j$.

To help make the expression of constraints easier, the authors represent the schedule as a list $T$ of length $n \times w$ (with index $TT = \{0 \dots n \times w - 1\}$) where each element maps onto the schedule $S$ as follows: $T_k = S_{k \div w+1, k \mod w+1}$. In addition, they define the function $t(x) = x \mod (n \times w)$ to help with indexing.

The RWS problem can be defined as follows:

$$\sum_{k=0}^{u_w}(T_{t(j+k)} = O) > 0, \qquad j \in TT \tag{1}$$

$$\sum_{k=0}^{l_w}(T_{t(j+k)} = O) = 0, \qquad j \in TT, T_j = O \wedge T_{t(j+1)} \neq O \tag{2}$$

$$\sum_{k=0}^{u_O}(T_{t(j+k)} \neq O) > 0, \qquad j \in TT \tag{3}$$

$$\sum_{k=1}^{l_O}(T_{t(j+k)} \neq O) = 0, \qquad j \in TT, T_j \neq O \wedge T_{t(j+1)} = O \tag{4}$$

$$\sum_{k=0}^{u_{sh}}(T_{t(j+k)} \neq sh) > 0, \qquad j \in TT, sh \in \mathbf{A} \tag{5}$$

$$\sum_{k=1}^{l_{sh}}(T_{t(j+k)} \neq sh) = 0, \qquad j \in TT, sh \in \mathbf{A}, T_j \neq sh \wedge T_{t(j+1)} = sh \tag{6}$$

$$T_j = sh_1 \rightarrow T_{t(j+1)} \neq sh_2, \qquad j \in TT, (sh_1, sh_2) \in F_2 \tag{7}$$

$$T_j = sh_1 \wedge T_{t(j+1)} = O \rightarrow T_{t(j+2)} \neq sh_2, \qquad j \in TT, (sh_1, sh_2) \in F_3 \tag{8}$$

$$\sum_{i=1}^{n}(S_{i,j} = sh) = R_{sh,j}, \qquad j \in 1 \ldots w, sh \in \mathbf{A} \tag{9}$$

Constraints (1) and (2) ensure that work blocks (stretches of consecutive work days in the schedule) stay between the defined minimum and maximum number of consecutive work days. Constraints (3)-(4) limit the length of off blocks to the predefined maximum and minimum number of consecutive days off. Constraints (5)-(6) restrict the length of shift blocks to lie between the given maximum and minimum length. Finally, constraints (7) and (8) exclude forbidden shift combinations of length 2 and 3 while constraint (9) ensures that the workforce requirements are satisfied.

The problem of determining whether a feasible cyclic roster exists is NP-complete Lau (1996). Nevertheless, many approaches exist to solve these MIP problems successfully Bixby (2012) albeit for small problem instances that do not necessarily reflect the problem sizes faced by airport security schedulers (see Chapter 1). Commercial solvers such as Gurobi (Gurobi, 2024) and IBM ILOG CPLEX (IBM, 2024) have been developed to tackle these problems as efficiently as possible, and particularly Gurobi has been used to successfully solve the standard benchmark cases for cyclic rostering Musliu et al. (2018).

### 0.4.2 Constraint Programming (CP)

Historically, CP has not been as popular as MIP for solving rostering problems (Van Den Bergh et al., 2013). However, it has been successfully applied to RWS problems in the past (Musliu et al., 2002; Laporte and Pesant, 2004; Triska and Musliu, 2011). In fact, Musliu et al. (2018) used CP in their solution approach, which proved to be the state-of-the-art solution approach to RWS at the time.

Similar to MIP, several commercially available CP solvers can be used to solve CP problems (e.g. Gecode Schulte et al. (2022) or Chuffed Chuffed (2022)). However, one problem that all these solvers have in common is that they all use different modelling languages, making it difficult to compare and benchmark these solvers against each other. Nethercote et al. (2007) propose a solution by presenting a solver-independent modelling language (MiniZinc) that makes it possible to model the problem once and then plug different solvers in to test their performance. All example constraints are presented in the notation used by Minizinc, and all models in Chapters 1 - 4 are implemented and tested using MiniZinc.

CP works by searching the problem's solution space using constraint propagation to exclude solutions because one or more constraints explicitly forbid them Bessiere (2006). To help the search of CP solvers, it is possible to use so-called *global constraints* as part of the problem formulation. van Hoeve and Katriel (2006) provide an introduction to global constraints in CP. The use of these constraints is two-fold. On the one hand, they provide a convenient shorthand for expressing common constraints as one expression that would otherwise need to be expressed via multiple simple constraints. One example would be the requirement that all values $x_1 \dots x_n$ are pairwise distinct, which can be simply expressed via the global constraint $\texttt{all\_different}(x_1,\dots,x_n)$. On the other hand, the use of global constraints can speed up the CP solver significantly as it can help it to recognise patterns in the problem that it might otherwise not be able to exploit.

For cyclic rostering, a couple of global constraints have proven particularly useful. Musliu et al. (2018) show that the global cardinality constraint $\texttt{gcc\_low\_up}([S_{i,j}|i \in 1\dots n, \mathbf{A}, [R_{sh,j}|sh \in \mathbf{A}], [R_{sh,j}|sh \in \mathbf{A}])$ can be used to replace constraint 9 in the cyclic rostering formulation presented in Section 0.4.1. This constraint ensures that the number of occurrences of each shift type on each day equals the corresponding workforce requirement.

The other global constraint that is used by Musliu et al. (2018) is the `regular` constraint (Pesant, 2004), which is used to model an automaton that represents all possible shift combinations that could occur in the schedule and the transitions between them.

Figure 6 shows an example automaton that encodes the following work rules:

1. There are two possible shift types (D = Day, A = Afternoon)
2. There have to be between 2 and 3 consecutive day shifts
3. There have to be between 1 and 3 consecutive afternoon shifts
4. Work blocks (i.e. consecutive days of work in the schedule) need to be between 3 and 4 days
5. There have to be two consecutive days off
6. Day shifts cannot directly follow afternoon shifts

Each state represents a possible shift pattern, and the arrows represent the possible choices for the next shift and the state to which the choice will lead.

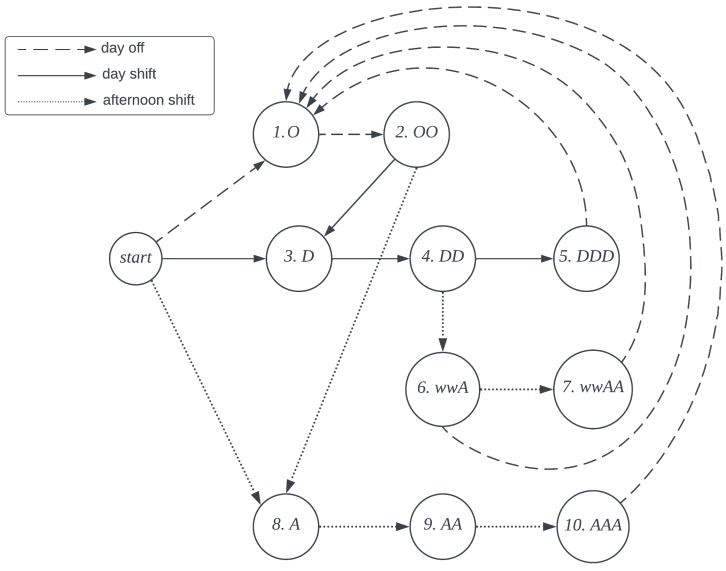



Figure 6: Example Automaton from Chapter 1.

This means that constraints (1) - (8) of the MIP formulation in Section 0.4.1 can be replaced by a single automaton.

### 0.4.3 Simheuristics

For a long time, simulation and optimization were considered separate, if not opposing techniques (Figueira and Almada-Lobo, 2014). However, with the improvements in modern computing, their respective research communities started to (separately) develop hybrid approaches to enhance their own methods — the simulation community focused on optimizing their simulation models and the optimization community aimed to enhance their analytical models via simulation techniques (Juan et al., 2015). Together, they form the field of Simulation-Optimization, which has been since coined the "Holy Grail of operations research/management science (OR/MS)" Cheng et al. (2002). Figueira and Almada-Lobo (2014) reviewed the existing approaches in the field and presented a taxonomy of Simulation-Optimization approaches by classifying them into three areas:

1. *Solution Evaluation Approaches*, which consists of creating a simulation model that represents the system and using it to evaluate the performance of different solutions (e.g. from an analytical model). Within this category Figueira and Almada-Lobo (2014) distinguish further between *Evaluation Function* approaches that use simulation to test solutions at certain iterations and feed information back to the search engine and *Surrogate Model Construction* where simulation is used to build a surrogate model that is either used to guide the search engine or the subject of the search itself.

2. *Solution Generation Approaches*, which consists of formulating analytical models and simulating the solutions. The aim of these approaches is not to evaluate the performance of the solution — as with Solution Evaluation approaches — but simply to generate some values (i.e. solution generation).

3. *Analytical Model Enhancement Approaches*, where simulation is used to enhance an existing analytical model.

In addition to the three categories of Simulation-Optimization approaches, Figueira and Almada-Lobo (2014) also identify four main ways in which simulation and optimization can interact with each other:

1. *Optimization with Simulation-based iterations*: optimization-driven approach that calls a simulation model at different iterations (e.g. to test solutions)

2. *Alternate Simulation-Optimization*: an approach that alternates between optimizing and simulating

3. *Sequential Simulation-Optimization*: a simulation followed by an optimization approach

4. *Simulation with Optimization-based iterations*: simulation-driven approach that uses optimization at different iterations.

Figure 7 shows how the Simulation-Optimization approaches and the different types of interaction between simulation and optimization overlap in the literature.

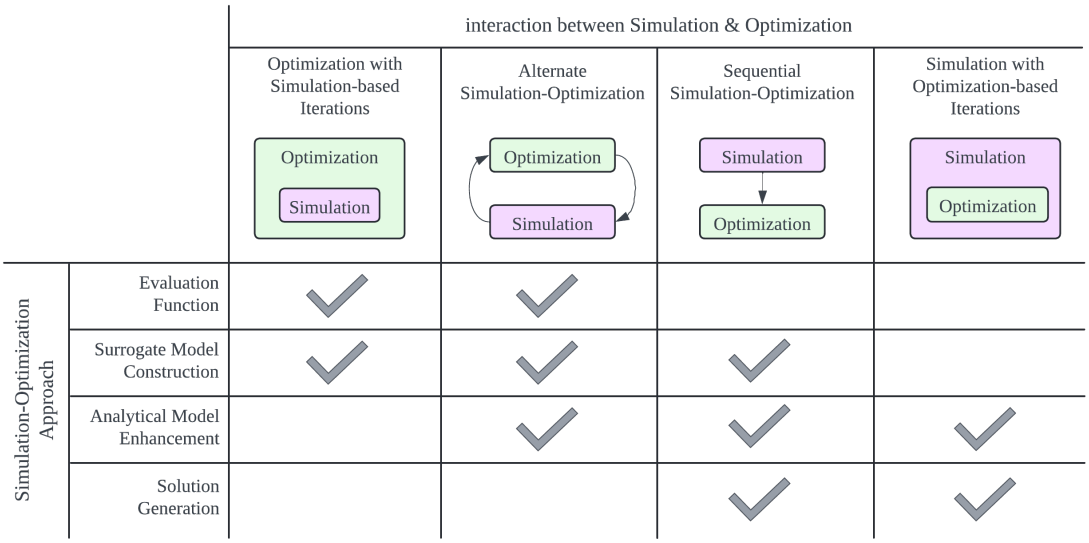



Figure 7: Overlap of Simulation-Optimization approaches and hierarchy (simplified from (Figueira and Almada-Lobo, 2014, Fig. 2., p. 127))

Juan et al. (2015) also looked at the hierarchy between simulation and optimization and distinguished more generally between *simulation-based optimization* — which is driven by the optimization approach and simulation acts as the supplementary element — and *optimization-based simulation* — where the main focus lies on simulating the system and optimization is used occasionally to improve/optimize certain values in the model.

Simheuristics are one of many Simulation-Optimization approaches developed over the years and refer to the hybridisation of simulation with metaheuristics Juan et al. (2023). They typically fall into the category of *simulation-based optimization* where they are either used in the form of *solution evaluation approaches* or *analytical model enhancement* (Juan et al., 2015). As a result, simheuristics can fall into any of

the four main categories of interaction between simulation and optimization (see Figure 7), where the metaheuristic addresses the optimization part of the approach. *Metaheuristics* are "high-level frameworks that combine basic heuristics to efficiently and effectively explore the search space" and can be used to address continuous and combinatorial optimization problems (Figueira and Almada-Lobo, 2014). They can be separated into three categories based on the way they explore the solution space (Juan et al., 2023):

- *Local search based*: these methods build on exploring solutions that are near the 'current' solution with additional features to help escape local optima. (e.g. Adaptive Large Neighborhood Search (ALNS) (Ropke and Pisinger, 2006), Simulated Annealing (SA) (Kirkpatrick et al., 1983) etc...)
- *Adaptive memory programming*: metaheuristics in this category work based on a memory of already visited solutions to make strategic decisions about finding the next possible solution (e.g. Tabu Search Glover (1986), Ant Colony Optimisation Dorigo et al. (1996), etc...)
- *Evolutionary methods*: these approaches worked based on natural phenomena that can be observed in population genetics (e.g. GA (Holland, 1992), memetic algorithm (Neri and Cotta, 2012), etc...)

Similar to the choice of metaheuristics, several simulation approaches exist that can be used to create simheuristics.

*Monte-Carlo Simulation* is associated with random experiments and consists of repeated random sampling of available data with subsequent statistical analysis to obtain results (Raychaudhuri, 2008). The underlying approach is simple, easy to grasp, and requires relatively low computational power. Over the years, Monte-Carlo simulation has been successfully used to model problems in different areas of application, such as engineering and finance.

*System Dynamics* was initially developed to model management problems in the 1950s (Forrester, 2007). It is based on the assumptions that a system consists of multiple components that together perform a function, that components can influence each other and that entities can move between components (Whitaker, 2015b). In that same book chapter, Whitaker (2015b) summarise a range of applications that System Dynamics models have been used for in the past to determine and analyse possible outcomes such as the relationship between supply and demand, epidemics or the effect of politics.

*Discrete-Event Simulation* (Goldsman and Goldsman, 2015) is used to study systems where changes occur at discrete time intervals. Discrete-Event simulations are driven by a *future events list*, which contains all the events that will happen during the simulation and their timestamps. Examples of such events include the arrival of a new entity to the system, the completion of a manufacturing process, and the end time of the simulation. These types of models are most suitable to model queuing systems (e.g., airport check-in counters or bank service counters where people arrive, wait in line, get served and leave again), manufacturing processes (where products go through multiple processes while being built and assembled) or inventory systems (where products are ordered from customers at random times and inventory needs to be restocked).

*Agent-Based Simulation* (Whitaker, 2015a) models autonomous agents (either individual entities or groups of entities) that can, for example, carry out tasks, retain memory, monitor or influence their environment in the system and interact with other agents in the system. Contrary to discrete-event simulations, where entities are idle unless the next event triggers them to move around the system, entities in Agent-based systems are always active. These type of models can be used to model complex and extensive systems and test possible outcomes to changes in the environment. Examples include the interaction between neighbours in a neighbourhood or the modelling of bacteria in a playground.

## 0.5 Thesis: Genesis, Praxis and Synthesis

The work in this thesis is inspired by Edinburgh Airport's security workforce planning problem. Throughout this PhD, I have had continuous meetings with the airport's CSS about their scheduling process, problems, needs and vision for a working rostering tool. These discussions highlighted four main problem statements:

PS1 **Roster Tool** Edinburgh Airport requires a fast and flexible roster tool that can be used to solve the scheduling problem at hand and is adaptable (flexible) to the ever-changing working requirements set by legislation and union representatives. The current method employed by the CSS consists primarily of manual spread-sheet-based manipulation of existing rosters. While the CSS can create feasible schedules in this manner, they are far from optimal. This is highlighted by high levels of staff dissatisfaction with rosters. The airport has made several attempts at automating the process through commercially available software, but to date, no suitable solution has been found.

PS2 **Flexible Rosters** In the UK, airport security labour is unionised (i.e. union representatives must approve all schedules at a tactical planning level before implementation). The approval process can be lengthy, so schedules are created based on demand forecasts made months before the actual day of operation. As a result, unforeseen disruptions will cause infeasibilities in the roster that have to be repaired through costly re-rostering steps. To mitigate these costs, the CSS wants to produce *flexible* rosters — with surplus staff and possibilities to extend shifts without breaking contractual agreements — to make re-rostering steps easier. However, at the moment, formal techniques for measuring, injecting and fine-tuning these flexibilities are missing.

PS3 **Strategic Planning Decision Support** Each year, the airport has to set a budget that determines how many security officers can be hired and limits how much paid overtime can be scheduled. To support these decisions, the CSS has to estimate the workforce size they will require, which is currently based on their years of experience in the role. However, being able to create rosters with different workforce structures can be a valuable source of information to support these decisions in the future.

PS4 **Fostering Uptake** Developing a roster tool to support scheduling decisions is only useful for the airport if the tool is also integrated into the workflows of the workforce planning process. To help facilitate this, the airport's scheduling environment needs to be studied in detail to ensure that the tool solves the problem at hand and that the tool and its outputs are in a format that is usable by the airport. This includes the creation of a user guide detailing all possible settings and inputs of the roster tool. The likeliness of the airport taking on such a tool is increased by the presence of a member of staff at the airport with the relevant training in OR/MS techniques to translate and implement new work requirements as they arise.

The work in this thesis addresses the first three problem statements in the form of three papers (Chapters 1 - 3) — one per statement. The fourth statement is addressed by a case study describing the development of a roster tool and accompanying handbook for Edinburgh Airport's security workforce scheduling problem (Chapter 4). Figure 8 gives an overview of how the work in the four chapters fits together and where the four chapters support the decisions in the workforce planning timeline.

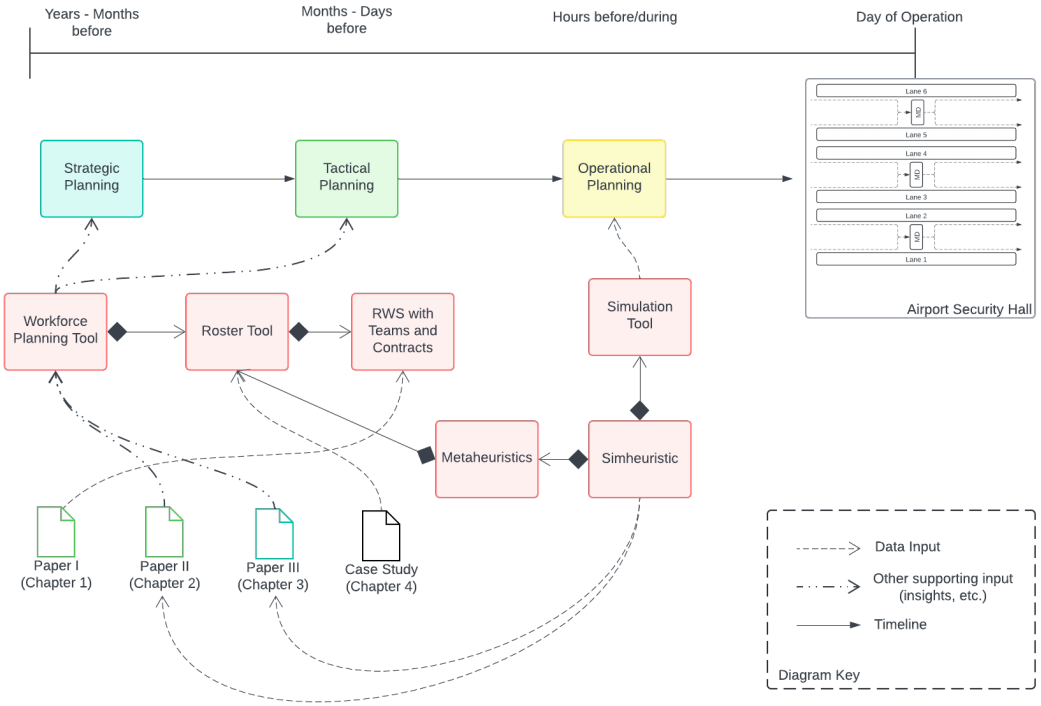



Figure 8: Workforce Planning Stages with Chapter Positioning.

Sections 0.5.1 - 0.5.4 will introduce each of the three papers and the work with Edinburgh Airport, cover the contributions made in each chapter and explain how these address the research gaps (RG1 - RG6) identified in Section 0.3.4.Section 0.5.5 provides a summary of all the contributions made in this thesis is provided.

### 0.5.1 Paper I: Rapid Prototyping of Rotating Workforce Schedules for Applications in Airport Security[2]

The first paper originated while developing the first version of Edinburgh Airport's roster tool. During the development process, it quickly became apparent that the literature does not cover cyclic rostering problems with a workforce consisting of multiple contracts and team sizes (RG2) and hence, the paper was put together to formally present this problem variation.

Paper I follows an atypical structure for the OR/MS community by presenting two model formulations and associated experiments. This was necessary to tie the paper to the existing literature on RWS and the state-of-the-art solution approach at the time (Musliu et al., 2018) — and arguably still the best approach in ever-changing environments such as airport security — while also covering the more advanced modelling constraints encountered by staff schedulers in airport security.

The main contributions of this paper can be summarised as follows:

CI.1 **RWS with multiple contracts and team sizes**: This paper extends the existing RWS problem to include multiple contracts and team sizes by presenting two new problem formulations and thus addressing RG2. The first formulation extends the existing formulation of RWS as presented by Musliu et al. (2018) to include the notion of multiple contracts and team sizes, and the second formulation includes additional constraints that are based on the problem constraints that the CSS at Edinburgh Airport needs to adhere to but have not been covered by Musliu et al. (2018) or similar works in the rostering literature (e.g. enforcing two consecutive days off every two weeks and ensuring a weekend off every three weeks).

[2]This paper is a joint work with Maurizio Tomasella, Daniel Guimarans and Thomas W. Archibald. It was submitted to Expert Systems with Applications and underwent three rounds of review before it was rejected. The version in this thesis is the final product after the edits requested by the reviewers.

CI.2 **New Problem Instances**: Adding teams and contracts to the RWS formulation used to solve the existing benchmark instances (Musliu et al., 2019) meant that the instances alone could not be used to test the new formulation. So, a new set of benchmark instances (Wiesflecker, 2022) — some of which are based on the original set of instances, by combining multiple instances into one — was created to run the experiments, thus addressing RG4.

CI.3 **Implementation and Development Guidance**: The paper provides extensive guidance to practitioners interested in using solver-independent-modelling to model and solve their scheduling problem. For example, a UML diagram detailing the different components of the solution approach is provided, and the results analysis takes the practical implications — such as the time taken to model and implement the approach — of each model into account.

CI.4 **Testing on Realistic Problem Sizes**: Problem sizes in the rostering literature are generally small, with the current set of benchmark instances (Musliu et al., 2019) mostly consisting of workforce sizes below 64. The instances used in this paper include cases with over 200 (up to 666) employees which is much more realistic in terms of airport security workforce scheduling and addresses RG4. This means that the results obtained during the experiments can be directly translated to implications for the airport without the need to consider the impact of having a larger workforce.

The study led to the following major learning points:

LPI.1 **Solver-Independent Modelling** The benefits of solver-independent modelling from an empirical research perspective are evident as it takes away the complexity of comparing different solvers (with potentially different modelling languages). Yet, from a practitioner's perspective, the benefits might not be as obvious, and in many cases, solver-independent modelling will not be necessary. However, in the ever-changing scheduling environment of airport security (or similar environments), the problem structure may change drastically between rounds of discussions, thus requiring an approach that can be quickly adapted without the need for an OR/MS expert to re-write the problem to fit a more suitable solver. A solver-independent approach can eliminate these steps and allow the CSS to not only change constraints at will but also switch out solvers at the mere press of a button.

LPI.2 **Implementation Considerations** From a practical perspective, the fastest runtime does not necessarily translate to the best approach — as was shown by the automata-based models in the experiments. In both sets of experiments in the paper, automata-based models consistently performed well — with very few exceptions — with regard to runtime and number of solved instances. This would usually lead to the conclusion that an automata-based model would be preferential to use. However, these models can quickly become an issue in environments with many changing constraints. For the instances modelled on the security officer scheduling problem at Edinburgh Airport, the number of states in one automaton varied between 257 and 559. Implementing and changing such automata regularly can become extremely tedious and time-consuming — let alone complicated if the user is not familiar with the creation of automata. As a result, automata-based models are unsuitable for problem applications such as airport security workforce scheduling.

LPI.3 **Gurobi the All-Rounder** Having the flexibility to switch between models and solvers is great when prototyping schedules and with the time available to try out different combinations. However, this is not always possible. In some instances the CSS might want to just have one set-up where they add/remove constraints and then press go. In those instances, Gurobi has proven to be the best option. It is a fast and reliable solver and consistently performed well in the experiments.

My CRediT author statement for this paper:
Conceptualization, Methodology, Software, Validation, Formal Analysis, Investigation, Data Curation, Writing – Original Draft, Writing - Review & Editing, Visualization, Project Administration

### 0.5.2 Paper II: Fine-tuning Flexibility in Workforce Schedules[3]

With the first version of the roster tool complete, the research focus turned towards possible uses of the tool in the workforce planning timeline. A key issue that kept arising for the CSS was the long timeframe of several months between schedule creation and schedule usage. This meant that demand changes and unforeseen staff absences would lead to disruptions that had to be fixed before the day of operation to

[3]This paper is a joint work with Maurizio Tomasella and Thomas W. Archibald with acknowledgements to Sam Song and Jared Cheung for preliminary testing of the methods. At the time of writing the aim is to submit the paper to the Journal of Scheduling.

ensure a smooth experience for passengers. This issue led to the idea of creating schedules with fine-tuned flexibilities — surplus (buffer and reserve) staff — to make repairing infeasibilities caused by disruptions easier.

Paper II presents a generalisation of existing robustness metrics (Wickert et al., 2021) — to include contracts and team sizes — and proposes the use of demand aggregation rules as a formal robustness metric. Three Simheuristics to fine-tune combinations of robustness metrics in staff schedules are also presented and tested on data based on Edinburgh Airport's security staff scheduling problem.

The contributions of Paper II can be summarised as:

CII.1 **Metric Extension**: Existing *surplus shift* (*buffer* and *reserve*) metric formulations (Wickert et al., 2021) cannot be applied directly to problems with workforces consisting of multiple contracts and team sizes. This paper extends the existing formulations to suit such problems — in line with the extension of RWS to include multiple contracts and team sizes in Paper I. This extension is essential to make the existing flexibility metrics more widely applicable to rostering problems and addresses the previously identified lack of formal robustness metric definitions in the rostering literature (RG5).

CII.2 **Demand Aggregation as Robustness Metric**: With the introduction of formal flexibility metrics in rostering literature, existing measures — such as the aggregation rules to create model weeks for cyclic rostering problems — should also be added to the collection of 'formal robustness metrics' (RG5). The experiments in this paper show that by (reasonably) reducing the demand coverage of the roster, more workforce is made available to freely allocate, for example, in the form of *surplus shifts*.

CII.3 **Simheursitics for Parameter Tuning**: With the advent of formal flexibility metrics in rostering comes a need to fine-tune and balance the different parameters to suit the particular needs of the user and problem setting (RG6). This paper presents three Simheuristics for robustness parameter tuning.

CII.4 **Testing on Realistic Problem Sizes**: The three Simheuristics in this paper are tested on a problem modelled closely on Edinburgh Airport's security workforce scheduling problem (RG4). This includes realistic seasonal demand forecasts and associated workforce structures. This helps draw conclusions that are directly applicable to the CSS and their everyday decisions.

The learning points that can be drawn from the work in this paper are:

LPII.1 **Effective Tool** The paper has shown that Simheuristics based on Monte-Carlo Simulation are an effective tool to find the best metric combinations by systematically selecting different combinations of robustness metrics and testing them against the expected demand forecast. In addition, Simheuristics — such as the ones used in the paper — are easy to understand from a practitioner perspective — something that is possibly more important than the effectiveness of the method in order to ensure that it will be implemented by the user!

LPII.2 **Metric Impact** The experiments in the paper have shown the positive impact the use of good (i.e. fine-tuned to the problem) metric combinations can have on the rescheduling effort of the CSS. The results of pairing weekly surplus shift levels with demand aggregation show an average reduction of 20% in average weekly re-rostering cost compared to just tuning demand aggregation. For Edinburgh Airport, this suggests substantial cost savings over the year.

My CRediT author statement for this paper:
Conceptualization, Methodology, Software, Validation, Formal Analysis, Investigation, Data Curation, Writing – Original Draft, Writing - Review & Editing, Visualization, Project Administration

### 0.5.3 Paper III: Simheuristics for Strategic Workforce Planning at a Busy Airport[4]

The main goal of the project with Edinburgh Airport was to produce a roster tool that would support scheduling decisions at a tactical planning level. The first two papers address this goal and cover methods for adding flexiblity to rosters. However, during discussions with the CSS, it also became apparent that there was a need for strategic decision support during yearly budget meetings. The roster tool alone can be used to support these meetings by quickly trying out different workforce structures to test for feasibility. While this is a great way to integrate work initially created for tactical decision-making into strategic decision support, this paper aims to take

[4]This paper is a joint work with Maurizio Tomasella and Thomas W. Archibald. It is the third paper that originated from the work in this PhD and has been accepted for presentation at the 2024 Winter Simulation Conference. Wiesflecker et al. (2024)

strategic decision support one step further. It presents a simheuristic that mimics allowed changes to the workforce structure (contract features such as weekly hours or team sizes) and tests whether the resulting workforce will make it possible to produce feasible rosters. Changing the structure of a workforce takes a long time because existing contracts often cannot be changed, or if they can be changed, the possible changes are minimal. So, being able to determine the best structure for the future would allow the CSS and HR department to plan ahead and adjust their hiring policies accordingly.

The contributions of this paper are summarised by the following points:

CIII.1 **Strategic Workforce Planning with Scheduling Decisions**: Existing literature on Workforce Planning problems mainly focuses on the flow of personnel through the workforce structure. However, there is little work that includes testing the workforce structure through scheduling to see if a newly proposed structure will allow feasible schedules with enough flexibility to run smooth daily operations (RG1). This paper addresses this gap by presenting a simheuristic approach that manipulates the contractual structure of the workforce and tests it for feasibility at the tactical planning stage (i.e. tests if a feasible roster can be created).

CIII.2 **Problem Insights for Practitioners** The results highlight the need for additional testing of workforce structures at the planning stages, as there were no clear patterns to be identified that could be used to support decisions about future workforce structure. This is an important finding for schedulers and provides further evidence of the need for a fast and flexible roster tool to help with scheduling decisions and support SWP at the airport.

CIII.3 **Testing on Realistic Problem Sizes**: The experiments in this paper use a workforce structure and demand forecasts modelled closely on Edinburgh Airport's security scheduling problem. Using realistic problem sizes and features can help draw conclusions that are directly applicable to airport schedulers and their everyday decisions without needing to consider how much the differences in problem sizes or demand patterns might influence the expected results.

The major learning points from this work can be summarised as follows:

LPIII.1 **Scheduling decisions at strategic planning stage** The study provides evidence to support the inclusion of tactical planning decisions such as rostering at the SWP stage. Being able to quickly test rosters for suggested workforce structures can be valuable to support budgetary decisions such as hiring additional staff — as the CSS at Edinburgh Airport wants to do with the roster tool. Beyond the question of budgetary constraints, the CSS should be able to confidently enter the next season with a workforce that can handle the expected demand.

LPIII.2 **Simheuristics for strategic planning** Simheuristics once again prove to be an effective tool for solving the problem at hand. Their versatility allows for close replication of the allowed changes to the contractual structure before testing the feasibility of roster creation and re-rostering after disruptions.

LPIII.3 **Contract features & demand variability** The experimental results suggest that there is no direct correlation between the level of certain contract features in the best workforce structure and the demand level. This suggests that as the demand from passenger arrivals increases in the security hall, using the same workforce structure that works might not be the best approach for the CSS. Instead, a simheuristic approach — such as the one presented in the paper— should be used to determine the best workforce structure for the future. This would allow the HR department to work on slowly restructuring contracts over time.

My CRediT author statement for this paper:
Conceptualization, Methodology, Software, Validation, Formal Analysis, Investigation, Data Curation, Writing – Original Draft, Writing - Review & Editing, Visualization, Project Administration

### 0.5.4 Case Study: Security Staff Scheduling at Edinburgh Airport

The inspiration for the work outlined in this thesis came from Edinburgh Airport's security staff scheduling problem. By the time this project started, the airport had already attempted (and failed) to implement different commercial solvers and invested in one consulting project to develop a VBA-based purpose-built solution. Ultimately, all approaches had been deemed not flexible enough to integrate into the decision-making process at the airport.

The final chapter in this thesis covers the collaboration with the airport to develop a new tool based on MIP/CP. It provides an overview of the problem background and the constraints that need to be satisfied, shows the mathematical formulations of the modules in the decomposition approach, discusses the implementation and summarises the feedback provided by the CSS.

The contributions of this case study are:

CIV.1 **Airport Rostering Insights**: The case study provides an extensive overview of the rostering process at Edinburgh Airport, detailing the problem stakeholders and their roles, procedures, timelines and the decisions that have to be made at each stage. Additional insights are provided on how external factors — such as COVID or the incoming hand luggage regulations (Department for Transport, 2022) — are impacting the airport and its workforce.

CIV.2 **Roster Tool**: Overall, this thesis covers the development of a roster tool and possible applications of such a tool to solve different strategic and tactical workforce planning problems. As part of this, the case study addresses the development and improvement of the working roster tool prototype to suit the requirements and expectations of the CSS. This includes insights into the development process — such as feedback from the CSS after testing the prototype and the resulting changes and features that were implemented — the possible operation of the tool and the reporting of outputs.

CIV.3 **Implementation and Development Guidance**: The chapter provides an overview of both the decomposition approach used to model the problem and the model formulations, as well as guidance for implementation in the form of a UML diagram. All of this can be used by practitioners as a roadmap for the implementation of their own roster tool.

The learning points from this collaboration with Edinburgh Airport are:

LPIV.1 **Need for a Roster Tool** At the beginning of this project, the airport desperately wanted an automated roster tool to support scheduling decisions, and they tried several times to find suitable solutions to their problem. Now, at the end of the project, this requirement remains unchanged. They still want a roster tool, but with the development of the tool and the increased understanding of its usefulness, the list of possible applications of such a tool has only increased — such as supporting strategic budgetary decisions.

LPIV.2 **Issue of Flexibility** The issue of flexibility remains one of the biggest obstacles to overcome in the development of a usable roster tool for Edinburgh Airport. The scheduling environment and the work regulations change so much that the tool essentially needs to be fast and flexible (i.e. *fully customisable* for each contract) while also allowing for partial roster inputs and providing a selection of different rosters that all satisfy the same constraints.

LPIV.3 **Changing Expectations** As the project progressed, the requirements and milestones for the roster tool started to change. From an implementation perspective, this caused difficulties and required restructuring of the models. However, it also showed that the CSS was getting more comfortable with the idea of using such a tool and started to expand possible uses and ideas for it. In the end, this collaboration highlighted the importance of flexibility in scope for a successful "live" research project and having the CSS on board and being excited to use the tool was a positive development of the project.

### 0.5.5 Thesis Contribution Summary

To conclude, the contributions made in this thesis can be summarised into four main statements:

T1 **RWS with Teams and Contracts** This thesis provides an extension of the well-known RWS problem by introducing contracts and multiple team sizes. Introducing this extension adds a formal representation of these types of rostering problems to the wider scheduling literature. In combination with the newly defined set of benchmark instances (Wiesflecker, 2022) and analysis of the suitability of existing techniques, this provides a baseline for researchers to continue working on these problems and support airport schedulers and practitioners facing similar problems in the future.

T2 **Flexible Rostering** The research conducted shows that existing methods for aggregating demand into model weeks for cyclic rosters should be considered as a robustness metric for rostering. In combination with other robustness metrics, tuning demand aggregation has been shown to increase roster flexibility. In line with contribution T1, this thesis also generalises existing robustness metrics to make them applicable to a broader range of rostering problems. To solve the issue of balancing the benefits of multiple metrics — a problem that will become

more and more prevalent as the list of formal robustness metrics in rostering increases — this thesis also presents and tests three Simheuristics for robustness parameter tuning.

T3 **Strategic Workforce Planning** The thesis proposes a Simheuristic approach to fine-tune workforce structures for upcoming demand forecasts by systematically changing contract features and testing to see if the resulting workforce can produce feasible rosters. On top of adding to the scarce literature on SWP with consideration of scheduling decisions, this study also showed the benefit of using a fast and flexible roster tool for strategic decision-making — a possible application for the roster tool at Edinburgh Airport.

T4 **Roster Tool and Practical Insights** This thesis presents the development of a fast and flexible roster tool for Edinburgh Airport's security staff scheduling problem. This includes detailed development and implementation guidance to help practitioners who want to create a similar tool for their own scheduling problems. In addition, all experiments were conducted using data based on the real problem size and demand forecasts encountered at Edinburgh Airport. This allows the CSS and other scheduling practitioners to directly apply the conclusions and learning points to their own problems without needing to consider how much changes in problem size (i.e. workforce size) will impact the results.

## 0.6 Limitations and Future Works

Throughout this thesis, great care was put into the design of the experiments to ensure meaningful conclusions could be drawn. However, there are still limitations that could not be avoided and can be addressed by further research in the area.

LF1 **Airport security focus** The research in this thesis originated from a collaboration with Edinburgh Airport. As a result, all the insights and data provided are based on airport security workforce planning/scheduling problems. This means that the conclusions can only be directly applied to airport security rostering settings. However, similar scheduling problems can be found in many industries (e.g. train driver or bus driver scheduling). An interesting line of research would be to investigate how the work in this thesis can be extended to other settings. In fact, during my final year working on this thesis, I met with a local bus company to see if there were similarities between their scheduling problems and those at Edinburgh Airport. The conversation showed significant overlap in the issues faced by the CSS at Edinburgh Airport and the scheduler responsible for the bus driver's schedules — for example, the need for a flexible roster tool, constantly changing work regulations in a unionised setting, and the variety of available contracts. Exploring the applications of the roster tool developed in this thesis to the bus driver scheduling problem provides a possible direction for future research.

LF2 **Data availability** Throughout the collaboration, the CSS has been very forthcoming with insights about the scheduling process, work regulations and the security hall operations. In addition, it was possible to obtain a seasonal demand forecast and example contract structure and one roster — although anonymized for security reasons and data protection. The lack of additional data, such as multiple demand forecasts or rosters, made it difficult to test the roster tool against different scenarios and replicate experiments. In particular, for Paper II, which covered Simheuristics for parameter-tuning to introduce flexibility to rosters, it would have been great to obtain additional data on real demand vs. the corresponding seasonal forecast to test the effect of the metrics further. If security restrictions permit the release of this data in the future, additional tests on the actual demand would provide a great extension.

LF3 **Roster tool implementation** Although extensive work was put into the development of a suitably flexible roster tool — including a working prototype that the CSS could try out — the final version of the tool did not meet the shifting expectations of the CSS. Final discussions about the prototype uncovered that the original assumption of contract-specific constraints vs. general constraints — that can be changed and switched on/off but apply to all contracts equally — did not apply anymore. Instead, all general constraints should also be changeable on a contract by contract basis. This implies a major overhaul of the implementation of the roster tool from its current state, which was not possible to complete during this thesis. Given the development of a roster tool for Edinburgh Airport was a key goal of the collaboration, it would be great to continue the development of the tool until a suitable product was created.

LF4 **Metric focus on surplus shifts** The flexibility metrics used in this thesis focus on demand aggregation and the addition of surplus shifts. However, this only addresses one half of the possible actions the CSS can perform to address infeasibilities in the roster — namely calling in additional staff to cover absences. The other actions available to the CSS would extend shifts to cover changes in demand patterns. During the research for this thesis, an additional metric to ensure the roster includes certain levels of slack in available work hours — so shifts can be extended without violating overtime constraints — was developed but proved to add insignificant benefits to the schedules. Refining this metric is, therefore, another possibility for future lines of research.

LF5 **Simheuristic components** The simheuristics used in this thesis consist of a combination of Monte Carlo Simulation and one of ALNS, SA or GA. In the setting of this research and with the available data, this proved to be the best approach to use. However, as shown in Section 0.4, there are many possible choices of metaheuristics and simulation approaches that can be used to create a simheuristic. Exploring additional options, such as building a discrete event simulation model of the security hall, could provide additional insights into the problem.

## 0.7 Conclusion

At the beginning of this work, two lines of research were identified, and the ensuing research addressed them in several ways:

**1. Need for a flexible roster tool**

Edinburgh Aiport's need for a flexible roster tool was distilled into two main problem statements (**PS1 - Roster tool** and **PS2 - Flexible rosters**). The former recognises the fact that Edinburgh Airport needs a roster tool that can be used to support discussions with unions (and the resulting frequent changes in work requirements and preferences), whereas the second one covers the issue of disruptions that occur between the creation of the schedule and the day of operation. In addition, it was acknowledged that producing a working roster tool alone is insufficient to ensure that it will be used, leading to an additional problem statement **PS4 - Fostering Uptake**.

The development of a roster tool (**PS1**) for Edinburgh airport is covered in the case study presented in Chapter 4 with features of the approach covered in more detail in Paper I (Chapter 1). Chapter 4 also covers the requirements and considerations that are needed to ensure that a roster tool can and will be used by Edinburgh Airport after it has been developed (**PS4**).

The creation of flexible rosters (**PS2)** builds on a working roster tool. Paper II (Chapter 2) introduces the use of Simheuristics to fine-tune the required levels of robustness metrics, which are added to the existing roster tool formulation as additional constraints.

**2. Exploration of roster tool applications**

While the main aim of this collaboration with Edinburgh Airport was the development of a flexible roster tool itself, during the course of the research, possible applications of the tool were also considered. This led to problem statement **PS3 - Strategic Planning Decision Support**. The CSS already suggested that the roster tool would be useful during budget meetings to support their budgetary requests for the upcoming year. Paper III (Chapter 3) took this strategic decision support one step further by exploring Simheuristic approaches to finding the best contractual set-up for a given demand forecast. The idea behind this work was to support the development of long-term recruitment plans at the airport — which would otherwise be supported by a separate piece of software.

### 0.7.1 Final Statements

The final goal of creating a flexible roster tool for Edinburgh Airport's security staff scheduling problem was not completed due to changes in requirements and expectations for the tool. However, the resulting research provides rich insights for scheduling practitioners who strive to implement their own roster tool or are looking to enhance existing procedures through added flexibility.

Since the final meeting with Edinburgh Airport, there has been an initial conversation about continuing the work beyond this PhD, but no final agreement has been reached at the time of writing. Discussions are ongoing. If successful, the airport should consider also implementing the additional flexibility features and strategic decision support methods, as the experimental results suggest that they will lead to significant time and cost savings.

# Chapter 1

# Paper I: Rapid Prototyping of Rotating Workforce Schedules for Applications in Airport Security


## Abstract

Recently we paired with an airport operator in the UK to create a semi-automated rostering tool for its security staff. The tool produces cyclic rosters involving staff working under several contracts, split into teams the size of which are contract dependent. In this paper we extend an existing solution approach to suit this setting, a very typical one in airports. Our rosters encode the days-off patterns as well as the shift types (morning, afternoon, night) on work days for the different teams of workers. The innovations include the introduction of contracts, team sizes and more realistic constraints on days-off patterns in the schedule. We present several solver independent models with extensions and compare them using both constraint programming and mixed integer programming solvers. Our results verify the effectiveness of grouping a workforce into multiple teams (under various contracts) and we demonstrate the advantages of using a solver independent solution approach. Through our experiments as well as modelling guidelines provided, we aim to show airport operators how they can feasibly tackle security rostering, often their single most expensive cost centre with regard to human resources.

## 1.1 Introduction

In staff scheduling (Van Den Bergh et al., 2013), rostering problems aim to allocate employees to shifts whilst considering both hard and soft constraints, with the former to be met at all times, and violations of some combination of the latter to be minimised (Ernst et al., 2004b; Komarudin et al., 2013). The great variety of practical applications results in numerous problem-specific arrangements of constraints and objectives. Still, a decade ago Van Den Bergh et al. (2013) pointed out that "many characteristics of the personnel scheduling problem are often neglected. This puts a limit on the applicability chances of the solution method, since in real-life problems, these characteristics do appear". With the development of technology, we are now in a better position to address these changes in constraints and objectives fast and efficiently, as we demonstrate in this paper.

Rotating workforce schedules, also known as cyclic rosters (Rocha et al., 2013), are adopted in staff scheduling to cover demand whilst aiming for fair allocation of work throughout the planning horizon. The workforce is divided into groups that follow canonically the exact same pattern of days on/off work, and shifts. This problem frequently surfaces in the service industry. Cyclic rostering is most suitable for situations with repeating demand patterns, such as train services or flight schedules running according to regular weekly timetables.

In this paper we will take a closer look at RWS in airport security, by discussing, modelling and solving a new extension to the problem that we found was much needed by some of our industrial partners. Our approach to modelling and solving, a novel extension of one of the most recent state-of-the-art approaches in RWS (Musliu et al., 2018), was tested twice: (i) first, in instances comparable in size to those used as benchmarks in the related literature; and (ii) in a real case study of an international airport in the UK, consistently ranked top-ten nationally in terms of number of passengers per year. The second set of tests, in particular, yielded insights that the industrial partner found especially useful. This paper aims to share these insights with the broader research community as well as with interested practitioners.

### 1.1.1 Research Context

Since 2015, an action research programme has been active at the University of Edinburgh, focusing on collaborations with UK-based airport operators. Among the many problem areas tackled on the programme, workforce planning and staff

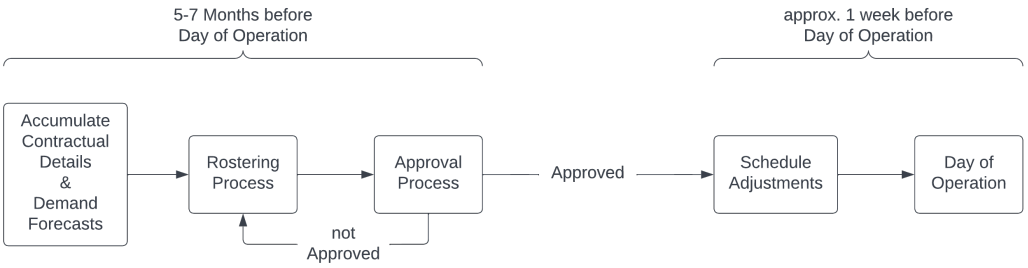



Figure 1.1: Image of the rostering process for a scheduler.

rostering have been (perhaps rather unsurprisingly) one of the most recurring. Staff scheduling in airport security, the biggest cost centre — human resource-wise — for many airport operators in the UK and other countries, has required the most attention. In the course of the research programme, we have engaged with various airport operators, and developed solutions for six different airports, ranging from small facilities acting as lifeline services for remote communities, e.g. based on islands, to major international airports. Staffing-wise, we have worked on security rosters ranging from about ten up to 400 employees. Irrespective of scale, we found a few common features, which we now discuss.

Figure 1.1 provides a simplified, end-to-end view of the considered staff rostering process. In the UK (and most probably elsewhere) airports are heavily unionised workplaces where staff schedules for an entire operating season (say the airport's summer schedule) have to be approved (usually 5-7) months in advance. The negotiation process itself has reportedly toughened and become longer over the years, with more and more rounds of discussions taking place, at each of which one intermediate version of the schedule is examined in detail, and often substantial changes to the underpinning constraints to be met by the next version are introduced. This loop only concludes when both parties (the airport operator and the unions involved) sign a formal agreement which effectively 'freezes' the days on/off work and the shifts worked by each employee for the forthcoming season. These can be modified only in the short-term, when, about a week before any one day of operation, demand forecasts are more reliable (flight schedules undergo partial variations all the time) and workforce composition up to date (e.g. some employees may have left the company in the meantime, or will be unavailable for the day due to sick leave). Even then, fine-tuning of the schedule would be allowed only under strict rules.

The typical staff scheduler we have met features profound expertise and knowledge of working regulations in the sector, long-established experience in negotiating with unions, and is as such considered a key asset in their own organisation. They also have no formal education in operations research/business analytics/management science/computer science but are normally extremely proficient at manually using spreadsheet-based software tools. The typical airport operator organisation we found has relatively small-sized analytics teams, the members of which would be able to only partially support the scheduler's job, always stretched amongst the many analytics projects ongoing at any one time. In the cases of the two bigger airports we worked with, off-the-shelf roster optimisation software had been bought, only to be met with frustration on the side of the schedulers, once deployed and used. In attempting to understand the reasons behind this, we carried out a series of interviews. The major point for criticism we found was the inability of the involved software products to allow the scheduler to remove, add or simply change constraints quickly and frequently enough to allow for the dynamic negotiations introduced above to take place. In both cases, the expensive software moved from an off-the-shelf (purchase) status to one of 'on-the-shelf, permanently parked', never to be used again! This was no surprise, as the long-established academic discussion on this sort of organisational failure has demonstrated over the years (Ackoff, 1967; Van Den Bergh et al., 2013). In both cases, the decision was made to revert to the use of spreadsheet-based tools, with 100% manual development of rosters. The common issue all the interviewed schedulers confirmed was that however, with the current length of the negotiation process as well as the number of loops to be run in any one negotiation cycle, a fully manual approach was deemed no longer tenable. In one case, the scheduler shared with us that creating a new roster from scratch, with the current workforce size, would take about 2 weeks at each iteration/loop, even with her 20+ years of experience in the role and manual dexterity with spreadsheet-based software! Semi-automating roster creation was felt to be badly needed by all involved schedulers, seen as a key enabler for both parties to devote a higher portion of the overall process to more interactive, meaningful negotiations, for the benefit of everyone, workers included. A clear need was highlighted for an approach that made it possible for airport staff schedulers to rapidly prototype a new roster at each iteration (ideally within seconds), for every new proposal to be ready for discussions in quasi real-time (ideally on the same day), thus enabling a more dynamic negotiation process. How to facilitate this transition was

then made the core of our investigations across all airport operators. The underpinning literature and arguments for our choice of developing an approach based on Musliu et al. (2018) will be discussed in Section 1.2. Before we reach the literature review, the next stop will be to introduce the main novelties we found in airport security rostering that created in the first place the need to extend previous formulations.

### 1.1.2 New Problem Features

Our version of RWS features a workforce subdivided into a number of different contracts (e.g., $c_1$ and $c_2$ in Table 1.1). Each contract comes with its own set of specific constraints (such as rules about allowed shift types, number of work days per week, forbidden shift combinations, etc.). Demand for different shift types — say morning, afternoon or night (M, A and N in Table 1.1, respectively) — requires given staffing levels over time, which in turn depend on the operations that are forecast to take place throughout the week (bottom of Table 1.1). To meet the demand exactly (say for the Morning shift on Mondays), it is possible to pick and mix from different contracts. This requires a fully flexible, equally skilled (homogeneous) workforce, which is normally the case in airport security.

Secondly, the scheduling object in airport security RWS is normally seen as teams rather than individual employees. This is also not an absolute novelty (we discuss this later in Section 1.2), having been already studied in various rostering applications, such as airline crews, healthcare nurses, manufacturing facilities, and other settings. Understanding the impact of this particular problem feature on the modelling and solving of airport security problems was however key for us to choose the right approach to develop further. In their review of the literature on personnel scheduling, Van Den Bergh et al. (2013) found a paucity of works attempting to study the effect of given hard/soft constraints on problem complexity. One of the aims of this paper is to discuss to some degree how the joint inclusion of teams and contracts impacts both modelling and solving of airport security rostering problems. To begin with, we argue that team sizes are not merely a variable to be optimally set by whatever approach would be eventually chosen. As consideration of teams in other settings confirms, team sizes are often tied into the specific process, steps thereof, and system layout, with each and every operation being unique in this sense. This often leaves the scheduler with a small set of possible team sizes to choose from. Paired with the requirement for relatively short cycles this leaves the scheduler with very few choices

that can be immediately resolved by hand and does not require the additional complexity of setting team sizes as decision variables in the rostering problem. The possibility or, at times, desirability to have teams of different sizes employed under the many contract types being considered was another common feature of the real case studies we tackled. Clearly, considering groupings of employees instead of individuals comes with benefits in terms of reduced problem complexity. It also comes with additional intricacies once it is decided that a fully cyclic roster needs to be developed, as we shall see later. Finally, even the number of roster lines for each contract type (more simply, the number of rows in Table 1.1) is not necessarily left to be decided by whatever algorithm shall be adopted. Number of contracts, number and size of teams by contract, and number of roster lines by contract were all seen, in our engagement pieces, as fundamental problem parameters for the scheduler to experiment with, and we found no industrial partner accepting the idea that any of these might be a decision variable to be freely set by an algorithm. Laporte (1999) made a strong case in this same direction, a long time ago and from a more academic perspective.

Table 1.1: Example Schedule and Demand

| Schedule | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| contract | team size | line | Mon | Tue | Wed | Thu | Fri | Sat | Sun |
| $c_1$ | 2 | 1 | M | M | M | A | A | / | / |
| | | 2 | / | / | / | N | N | N | N |
| | | 3 | N | / | / | M | M | A | A |
| | | 4 | A | N | N | N | / | / | / |
| $c_2$ | 3 | 1 | A | A | N | / | / | / | M |
| | | 2 | M | M | A | A | N | N | / |
| | | 3 | / | / | M | M | A | A | N |
| | | 4 | / | / | / | / | M | M | A |

| Memand | | | | | | | |
|---|---|---|---|---|---|---|---|
| shift type | Mon | Tue | Wed | Thu | Fri | Sat | Sun |
| Morning (M) | 5 | 5 | 5 | 5 | 5 | 3 | 3 |
| Afternoon (A) | 5 | 3 | 3 | 5 | 5 | 5 | 5 |
| Night (N) | 2 | 2 | 5 | 4 | 5 | 5 | 5 |

### 1.1.3 Two Key Messages

In the pages to follow, we have two key arguments to make:

1. RWS with multiple teams and contracts, an important problem in airport security and perhaps elsewhere, currently lacks academic attention in a way that supports meeting the key requirements just suggested in Subsections 1.1.1 and 1.1.2 through our interactions with multiple airport schedulers;

2. Adopting a solver independent, CP-based approach — such as in Musliu et al. (2018) — to enable rapid prototyping of cyclic rosters comes with both major benefits to the human expert (the scheduler) as well as with specific modelling challenges, both of which we will discuss in detail whilst proposing a feasible approach to tackle the latter.

### 1.1.4 Paper Contributions

As the rest of this paper unfolds, the following contributions will surface:

a An extension of RWS problems to include multiple teams and contracts. We also share the set of artificial instances we developed and investigated numerically to show the benefits of our extension.

b Detailed guidelines to support airport schedulers with needs similar to those highlighted in Subsections 1.1.1 and 1.1.2, in developing their own implementation. Our guidelines are summarised in the paper mainly through the discussion of Figure 1.2.

c A case study of the extended problem in a real-life airport security setting, the biggest of our six case studies in terms of workforce size.

### 1.1.5 Paper Structure

Having made the case in this section for how relevant the extended problem (RWS with multiple teams and contracts) is in the airport industry (argument 1 in the above list), Section 1.2 will argue the reasons behind our methodological choices (argument 2), whilst discussing the (surprisingly) very few relevant works we found in the available literature (also argument 1).

We then provide the extended problem formulation in Section 1.3 (contribution a in the above list), discuss the many modelling and solving techniques involved in Section 1.4 (argument 2 and contribution b), and evaluate their effectiveness through a first set of computational experiments in Section 1.5, working on artificial instances (argument 2 and contribution a).

After that, we present our airport case study in Section 1.6 (argument 2 and contribution b), which we study in detail through further experimentation in Section 1.7 (argument 2 and contribution c), prior to concluding the paper with a detailed discussion of our findings and the related implications to airport schedulers in Section 1.8 (contribution c).

## 1.2 Related Works

The study of cyclic rosters dates back to Bennett and Potts (1968), with the first survey appearing a decade later (Baker, 1976). Other notable early contributions are Bartholdi et al. (1980) and Bechtold (1981). Van Den Bergh et al. (2013) provide a popular, very comprehensive and more recent review featuring several works in RWS (e.g. Laporte and Pesant, 2004; Mörz and Musliu, 2004; Purnomo and Bard, 2007) while Levner et al. (2010) present a discussion of the complexity of cyclic scheduling problems.

Laporte (1999) gave a detailed account of the benefits of creating cyclic rosters literally by hand (pen and paper or manual use of spreadsheet tools). Coincidentally, it is also one of very few papers discussing the benefits of separate rosters for different contract types (full-time *vs* part-time), an idea we also embraced and that we will defend towards the end of this section. More centrally, Laporte explains the simple but rigid arithmetic principles underpinning the design of cyclic rosters, recalling examples of his own work of over 25 years as an academic consultant on this topic. In the opening section, the author states:

> "Over the years, I have observed that several of the problem's constraints must be bent to produce manageable schedules, sometimes in a major structural way, but it is difficult to tell beforehand which particular constraints should be relaxed in a given situation, which is why most algorithms typically fail to produce acceptable schedules."

Laporte argues that whilst cyclic rostering problems can be formulated as integer linear programs, the rigidity of such an approach contrasts sharply with the artistic skills needed to produce the 'manageable schedules' that problem owners often need.

In light of the context presented in the previous section, and particularly of the high frequency of changes and major restructuring of many of the rules that the mentioned negotiations require, we could not agree more!

However, the skilled and experienced airport schedulers we have worked with have shown us, time and again, that the cardinality and complexity of the rule set typical of their problem instances could not be handled by a fully manual approach alone. Even if the mentioned two weeks needed to create a roster by hand could be shortened to a couple of days as mentioned by Laporte in the same paper, the sheer number of stages of negotiations required to reach an agreement on a seasonal roster would still call strongly for some form of *effective*, *efficient* and *semi*-automated decision support to the expert scheduler. This just made our search for suitable candidate approaches even more demanding, with the overall requirement being one that emphasises effectiveness of the algorithm over efficiency. In other words, airport schedulers seem to favour (within reason) high acceptability of the rosters produced by the algorithms against very low computational times. They also put flexibility of the modelling approach in allowing for fast, manual changes in the rules/constraints — applied directly by the expert scheduler — at the very top of their requirements list. Based on our projects so far, 'short' runtimes would see a preference for anything up to a few seconds, with anything up to a handful of minutes still considered feasible, provided highly acceptable solutions are produced. High acceptability — often also referred to as "palatability" — of the produced rosters rests on highly subjective, case-specific judgements that can oftentimes only be spotted (for example, when visually displaying a roster) by a highly experienced scheduler. Even assuming that any two otherwise equivalent rosters meet all constraints, the two may display very different patterns in terms of days on/off work and/or shift sequences within recurring stints (i.e. the combination of a work stretch of a few consecutive work shifts with the immediately subsequent rest period). These patterns will be quickly caught by the eye of the experienced scheduler, who will almost instantaneously understand if the roster produced in front of them can be proposed for the next round of discussions. In some cases, one roster may be perceived as more likely to encounter the favours of the unions, while in others it may be more advantageous to the employer. What schedulers often want, based on our experience, is to have many alternative options (all meeting the involved constraints) produced each time the algorithm solves, so they can visually inspect alternatives and choose which one should be put forward for the next round of discussions. Alternatively, schedulers

would like to be able to reject the proposed single solution given by a single run of the algorithm, and have the algorithm produce a new, different roster, and to do so repeatedly and as quickly as possible. This kind of user requirement is difficult to model in absence of clear metrics. A similar requirement is discussed in the literature on "visual attractiveness" of solutions to routing problems, as in Rossit et al. (2019), where such metrics are proposed and tested. We are not aware of any similar articles in relation to cyclic rostering problems. This is one of the reasons why we made the choice to not model any objective functions in this paper. Furthermore, any new rule in this type of problem (e.g. minimizing the occurrence of single days off in the schedule), is generally first tested as either a hard or soft constraint to the problem. Otherwise, the constant adding of objectives that might be considered would lead to a multi-objective optimization problem which is counterproductive at the prototyping stage due to the considerable increase in model complexity.

With the above considerations made, in Table 1.2 we group the most relevant works we found from our harvesting of the cyclic rostering literature. A checkmark in the table means the corresponding paper addresses the related problem feature (namely — multiple contracts, teams, team sizes).

| Paper | Multiple Contracts | Teams | Team sizes |
|---|---|---|---|
| Maenhout and Vanhoucke (2009) | ✓ | | |
| Rocha et al. (2013) | FT/PT | ✓ | |
| Rocha et al. (2014) | | ✓ | |
| Kiermaier et al. (2016) | | ✓ | |
| Musliu et al. (2018) | | | |
| Kletzander et al. (2019) | | | |
| Becker (2020) | | | |
| Kletzander and Musliu (2022) | | | |
| Becker et al. (2022) | | | |
| This Paper | ✓ | ✓ | ✓ |

Table 1.2: Summary table of relevant cyclic rostering literature and the problem components they address. FT and PT refers to full-time and part-time respectively.

Becker et al. (2022) propose an approach to RWS which represents, runtime-wise, the current state-of-the-art. The authors use a branch-and-cut approach to solve a cyclic rostering problem for a set of single employees who all work under the same contract.

The contractually possible combinations of shifts are modelled as stints which are then combined to create a cyclic roster. From an end user's perspective, any change to work rules results in having to define a new set of allowed stints that can be combined into a schedule. While there are no doubts about the efficiency and effectiveness of the underpinning branch-and-cut algorithm enabled by the authors' graph based compact formulation, especially the latter requires expert knowledge to adjust the formulation for changes in constraints. That means that our end user would not be able to easily adopt this approach into their daily work routines. In addition, the authors confirm, in their closing remarks, the difficulty of using their approach to model constraints that span multiple planning periods (say multiple weeks), something that normally surfaces in airport security, as we will see while discussing our case study in the later pages.

The immediately previous state-of-the-art approach was due to Musliu et al. (2018), together with its subsequent extension provided in Kletzander et al. (2019). The approach is solver-independent, in the usual sense that problem modelling and problem solving are kept separate. More crucially, the authors' approach rests on three building blocks: (1) a single high-level modelling language; (2) interfaces to multiple solvers — both CP and MIP solvers; and (3) control mechanisms — including solver-specific — over the search for solutions. Musliu et al. (2018) present the first complete method capable of solving all the standard benchmarks at the time of publication, with many new benchmark instances also proposed in the same article. They discuss two solver-independent models and show that the best of the two outperforms the then state-of-the-art. One model formulates each rule for the schedule as a separate constraint in the CP model, thus allowing for easy switching of rules between consecutive rostering runs. The other model embeds as much as possible into a single global constraint (a constraint that captures a relation between a non-fixed number of variables — a cornerstone in CP) named `regular` (Pesant, 2004), by means of a deterministic finite-state automaton. The pairing of both models with both CP and MIP solvers is thoroughly tested in the paper. The relative ease with which one can explore the efficiency/effectiveness trade-off when pairing different models with different solving technologies suggested to us that this approach may prove of particular value to the human schedulers in the airport security settings we were investigating. The rest of this paper is largely a demonstration that this was indeed the case and a discussion of the learning points from having chosen this route. Table 3 in Becker et al. (2022) shows the scale by which the current state-of-the-art outperforms Musliu et al. (2018) and other works in the area, for a range of instances

with varying workforce size. Still, the runtimes guaranteed by Musliu et al. (2018) fall within the acceptable range mentioned above for airport security problems we worked on. Lastly, the largest instances considered by both Becker et al. (2022) and Musliu et al. (2018) relate to workforce sizes that are far below those of the case studies we have investigated (about five times smaller, compared to the biggest real-world instance we tackled). Therefore, beyond the obvious need to first have to extend either approach in a way for it to deal with multiple contracts, teams and team sizes, an additional underlying research question was around whether either approach would prove feasible for the workforce sizes of our problems.

Becker (2020) and Kletzander and Musliu (2022) also present recent solution approaches to RWS without considering any of the additional features relevant to our problem. More crucially, neither does significantly better than Musliu et al. (2018), in our opinion, in terms of providing fast and easy adjustability of the rules/constraints from one iteration to the next.

Many dichotomies pervade the rostering literature, including (a) homogeneous *vs* heterogeneous workforce (primarily in terms of skills and tasks (Steenweg et al., 2021), but also of contract types) and (b) individual workers *vs* teams/groups/crews thereof as the rostering entity. Van Den Bergh et al. (2013) testify that neither multiple contracts nor teams/groups/crews, at the time of its publication, seemed to be popular (with about 80 and 8 papers covering each, respectively). Teams and groups feature though in more recent literature, particularly healthcare (Olya et al., 2022) and manufacturing applications (Kiermaier et al., 2016; Rocha et al., 2014).

Steenweg et al. (2021) consider a cyclic rostering problem with a heterogeneous workforce (distinguished by different skill levels and inter-dependencies thereof). However, their main focus lies on planning for the uncertainty of workforce availability through the use of stochastic simulation and a reactive workforce allocation model — making their method not applicable to our work.

Guo and Bard (2024) address a 2-week cyclic rostering problem for a homogenous workforce of air traffic controllers. They solve their scheduling problem through a two-stage approach using a greedy algorithm and mixed-integer programming. In addition, to assigning shift types (morning/afternoon/night) they also consider shift start/end times and break assignment — which is out of scope for our work.

Kiermaier et al. (2016) use identically sized groups of homogeneous workers to reduce the size of their cyclic rostering problem. They mention that the group size and thus the number of lines in a roster could be set as decision variables in the problem

but that doing so would make the problem intractable. They opted instead for setting the group size and line number as parameters in their stochastic MIP formulation and running several different scenarios. While incorporating more relevant problem aspects than Musliu et al. (2018)(see Table 1.2) they work on a two-stage stochastic rostering problem which is far beyond the scope of this paper. Their problem formulation and solution approach is therefore not applicable to our context.

Rocha et al. (2014) consider a cyclic rostering problem in the glass industry which entails the scheduling of five teams of homogeneous workers who work under the same contract. Teams can be seen as the rostering unit in this problem making the example equivalent to a cyclic rostering problem for five individuals. The authors propose a construction heuristic to solve the given problem, which is otherwise quite different from our problem as seen in airport security.

Workforce scheduling under multiple contracts is a prevalent problem in nurse rostering, where contracts are often divided into full-time and part-time (Martin et al., 2013; Maenhout and Vanhoucke, 2009). Pairing cyclic rosters with multiple contracts however can cause problems due to different working requirements as imposed by different contracts. For this reason, workers under different contracts normally do not follow the exact same pattern. Schedules have to be differentiated instead, as the already cited pragmatic approach found in (Laporte, 1999) demonstrated a long time ago.

Maenhout and Vanhoucke (2009) outline a process for creating cyclic schedules for nurses with different preferences and work requirements. Their process starts with generating two cyclic schedules: a basic schedule with early, late and night shifts, and a supplementary schedule with only early and late shifts. These schedules are then adjusted for each nurse to fit their needs and requirements thus leading to different schedules for all nurses, which makes this approach not applicable to our problem.

Rocha et al. (2013) study a cyclic rostering problem for a workforce consisting of both full-time and part-time workers. The aim of the model is, however, to first schedule as many full-time workers as possible. Part-time workers are only used to cover the demand that cannot be covered by scheduled full-time workers. In this way part-time staff do not truly follow cyclic rosters. Despite the mapping shown in Table 1.2, the approach from the paper is therefore unsuitable for our problem.

As the latter articles demonstrate, it is possible to combine cyclic rosters with multiple contracts. However, unless a different schedule for each contract is created, individual adjustments to the schedule of each worker might be required, which largely

defeats the overall idea of a cyclic working pattern. The following sections will outline our approach to tackling this problem by creating separate cyclic rosters for each contract.

## 1.3 RWS Extension to Multiple Contracts and Team Sizes

We now provide a formulation for the general cyclic rostering problem with multiple contracts and team sizes, for a fully homogeneous workforce where everyone can perform all tasks.

Solving the problem produces a cyclic roster featuring a number of lines (Table 1.1) that is exactly equal to the number of teams in the workforce. Each contract has its own number of lines (teams), as indicated by higher level workforce planning decisions. Each line dictates the sequence of shift types/days off to be followed by the team which starts its cycle from that very line (shift types are different non-overlapping portions of the day — as in Table 1.1).

The resulting weekly patterns that correspond to any one line ensure the demand for workforce is met, for every day and shift type in the week, whilst adhering to the contractual requirements of the workforce. Staff working under different contracts follow slightly different versions of the same underlying working regulation. Therefore, not all teams can follow the exact same schedule throughout the planning horizon. To rectify this issue we introduce a separate cyclic roster for each contract. So, each team rotates through all the lines of the related contract. Once a team reaches the last line, they move back to the first line of the same contract for the following week.

For this approach to work, all teams of the same contract need to have the same team size. Otherwise, the number of scheduled employees for each day and shift type will vary from week to week which will lead to unsatisfied demand. However, team sizes can differ by contract. In some cases, they have to differ because the different areas of the process where staff are employed may be more suitable to accommodate teams of certain sizes at any one time (shift type).

Varying team sizes between contracts comes with an additional advantage. Namely, more flexibility for satisfying demand patterns with a homogeneous workforce, whilst keeping the problem size small to allow for shorter runtimes.

Introducing groups to reduce the problem size is a well-known method in the literature (for example, Kiermaier et al., 2016) and therefore not a key finding of this paper. However, paired with the retained flexibility to cover demand due to varying team sizes between contracts, it is a very useful feature for scheduling practitioners facing large rostering problems.

Theoretically, in most cases, one small team size, equal for all contracts, could be adopted, thus maximising said flexibility. Clearly, this will increase the problem complexity. However, more pressing issues that disable this approach exist. When considering a large workforce, having lots of small teams will lead to extremely long cycles. For example, 180 employees split into teams of 3 will give a 60-week roster which is over one year long. At this point, the benefits of a cyclic roster, namely a more fair schedule that is equal for all employees, are lost. In addition, most planning periods are shorter than 1 year. However, increasing the team size of all contracts to say 12, which produces a 15-week roster, will make the workforce too inflexible to schedule efficiently. For example, if demand is in multiples of 6 workers to open a new security lane (the unit by which airport security capacity scales up and down during any one day of operation), then half of the time too many employees will be scheduled, which is unacceptable from a cost standpoint. Or, should the requirement to open lanes be different in different parts of the airport (5 in Terminal A and 6 in Terminal B) then no single team size other than 1 would work. Therefore, we allow for team sizes to be different between contracts.

To formulate our problem we need the following parameters and sets:

- $nc$: number of contracts
- $\mathscr{C} = \{1, \ldots, nc\}$: set of all contracts
- $nt$: number of teams (lines)
- $\mathscr{T} = \{1, \ldots, nt\}$: set of all teams — teams of the same contract have consecutive indices
- $g$: line length — most often 7 days (one working week, which is normally the 'planning period' in rostering parlance)
- $ft_c$ and $lt_c$: (indices of the) first and last team on contract $c \in \mathscr{C}$
- $ts_c$: team size for contract $c \in \mathscr{C}$

- $\mathscr{A}_c$: set of permissible shift types for contract $c \in \mathscr{C}$ — often: morning (M), afternoon (A) and night (N). The additional option of a 'day off'(annotated by $O$) also exists. $\mathscr{A}_c^+$ denotes the set of all permissible shift types for contract $c$ including days off. We also consider $\bigcup_{c=1}^{nc} \mathscr{A}_c = \mathscr{A}$ and $\bigcup_{c=1}^{nc} \mathscr{A}_c^+ = \mathscr{A}^+$.

- $b\_up_{sh,c}$ and $b\_lo_{sh,c}$: maximum and minimum 'block' lengths for shift type $sh \in \mathscr{A}_c^+$ under contract $c \in \mathscr{C}$ — a block being a set of consecutive days of the same shift type in the schedule

- $w\_up_c$ and $w\_lo_c$: maximum and minimum lengths of 'work blocks' for contract $c \in \mathscr{C}$, where a work block refers to a set of consecutive working days, i.e. any run of consecutive shift types without a day off.

- $R$: requirement matrix — each element $R_{sh,j}$ encodes the workforce demand for shift type $sh \in \mathscr{A}$ on day $j$ where $1 \leq j \leq g$. $R0$ also includes the workforce requirement for days off, which is calculated by subtracting the demands for all shift types of the day from the total size of the workforce.

- $\mathscr{F}_c$: set of pairs $(sh_1, sh_2)$ of forbidden shift type combinations for contract $c \in \mathscr{C}$, meaning that $sh_1$ in any one day of the planning horizon cannot be followed by $sh_2$ the next day (or, equivalently, $sh_2$ cannot be preceded by $sh_1$ in the previous day) — a common example of this is "forward rotation", where the shift type for the next day has to be either the same or a later shift type than the one allocated to the day immediately before, or a day off.

- $\mathscr{F}3_c$: set of pairs $(sh_1, sh_2)$ of forbidden shift type combinations of length 3 for contract $c \in \mathscr{C}$, where $sh_1$ cannot follow $sh_2$ with a days-off in between (i.e. a sequence of $sh_1, off, sh_2$).

The decision variable for the problem is $S$, an $nt \times g$ matrix where each element $S_{i,j} \in \mathscr{A}^+$, $1 \leq i \leq nt$, $1 \leq j \leq g$ encodes the shift type assigned to line $i$ on day $j$.

To model the rotation through the lines of teams of the same contract we introduce two functions:

$$u_c(i) = \begin{cases} lt_c - (ft_c - i - 1) \bmod (lt_c - ft_c + 1), & i < ft_c \\ (i - lt_c - 1) \bmod (lt_c - ft_c + 1) + ft_c, & ft_c \leq i \end{cases}$$

$$v_c(i,j) = \begin{cases} (i,j), & 1 \leq j \leq g, \forall i \in \mathscr{T} \\ (u_c(i + \lfloor j - 1/g \rfloor)),\ (j-1) \bmod g + 1), & g < j, \forall i \in \mathscr{T} \end{cases}$$

Both definitions make use of modular arithmetic. The length of a line is normally $g = 7$ (the seven days of the calendar week). With that being said, function $u_c(i)$ maps the rotation of teams through subsequent weeks of the schedule, ensuring that the schedule for the days of the immediately subsequent week is the one found in the very next row (line) of the roster of the corresponding contract. Function $v_c(i, j)$ instead makes sure that, for any team $i$, after following the indications of row (line) $i$ for the first week, the shift/day off for the first day of the following week is the first on the row (line) immediately next according to the roster of the corresponding contract, and so on.

Armed with the above concepts/notation, the following constraint satisfaction problem can be formulated:

$$\sum_{k=0}^{w\_up_c} (S_{v_c(i,j+k)} = O) > 0 \qquad j \in 1 \dots g, \quad c \in \mathscr{C}, i \in ft_c \dots lt_c \tag{1.1}$$

$$\sum_{k=1}^{w\_lo_c} (S_{v_c(i,j+k)} = O) = 0 \qquad j \in 1 \dots g, \quad c \in \mathscr{C}, i \in ft_c \dots lt_c, \quad S_{i,j} = O \wedge S_{v_c(i,j+1)} \neq O \tag{1.2}$$

$$\sum_{k=0}^{b\_up_{sh,c}} (S_{v_c(i,j+k)} \neq sh) > 0 \qquad j \in 1 \dots g, sh \in \mathscr{A}_c^+, \quad c \in \mathscr{C}, i \in ft_c \dots lt_c, \tag{1.3}$$

$$\sum_{k=1}^{b\_lo_{sh,c}} (S_{v_c(i,j+k)} \neq sh) = 0 \qquad j \in 1 \dots g, sh \in \mathscr{A}_c^+, \quad c \in \mathscr{C}, i \in ft_c \dots lt_c, \quad S_{i,j} \neq sh \wedge S_{v_c(i,j+1)} = sh \tag{1.4}$$

$$S_{i,j} \in \mathscr{A}_c^+ \qquad j \in 1 \dots g, \quad c \in \mathscr{C}, i \in ft_c \dots lt_c \tag{1.5}$$

$$S_{i,j} = sh_1 \rightarrow S_{v_c(i,j+1)} \neq sh_2 \qquad j \in 1 \dots g, (sh_1, sh_2) \in \mathscr{F}_c, \quad c \in \mathscr{C}, i \in ft_c \dots lt_c \tag{1.6}$$

$$S_{i,j} = sh_1 \wedge S_{v_c(i,j+1)} = O \rightarrow S_{v_c(i,j+2)} \neq sh_2 \qquad j \in 1 \dots g, (sh_1, sh_2) \in \mathscr{F}3_c, \quad c \in \mathscr{C}, i \in ft_c \dots lt_c \tag{1.7}$$

Constraint (1.1) ensures that there is no block of consecutive work days in the schedule which is longer than the contracted maximum, $w\_up_c$, and (1.2) ensures that there is no block of consecutive work days which is shorter than the contracted minimum, $w\_lo_c$. Similarly, constraints (1.3) and (1.4) ensure that the number of consecutive days of the same shift type is within the contracted bounds, $b\_up_{sh,c}$ and $b\_lo_{sh,c}$, for each contract $c \in \mathscr{C}$ and shift type $sh \in \mathscr{A}_c^+$. Constraint (1.5) restricts the assignment of shift types, $sh$, to be from the pool of allowed shift types for each contract, $\mathscr{A}_c^+$. Constraints (1.6) and (1.7) rule out forbidden shift combinations for each contract, where the arrows, $\rightarrow$, are used to express an if-statement.

To ensure the demand of workers is satisfied for every day and every shift type, the following constraint is also needed:

$$\sum_{c=1}^{nc} \left( ts_c \sum_{i=ft_c}^{lt_c} (S_{i,j} = sh) \right) = R_{sh,j} \qquad j \in 1 \ldots g, sh \in \mathscr{A} \qquad (1.8)$$

This ensures that the number of assigned workers satisfies the demand of workers (not just teams). Given that teams of different team sizes might be combined to satisfy demand at certain locations it is important that schedulers keep the workforce demands in mind when setting team sizes for contracts before the scheduling process. Otherwise, the problem becomes infeasible.

## 1.4 Modelling and Solving Approach

CP technology has long provided a viable approach to tame managerial problems at various levels, whether strategic, tactical (such as RWS) or operational. Wallace (2020) makes a good case for adopting CP to develop 'intelligent' decision support systems whilst guaranteeing the level of interactivity, effectiveness and efficiency that we argued in our opening section. In this same direction, the book presents many successful examples of modelling and solving various combinatorial problems through a solver independent approach, grounded on a high-level modelling language, interfaces to multiple solvers, and control mechanisms over the search for solutions. The approach due to Musliu et al. (2018) belongs to the same school of thought, therefore it seemed sensible to attempt an adaptation of it that can tackle the extended general RWS defined in Section 1.3.

The present section describes all the main features that this adaptation entails, including considerations at both the conceptual and implementation levels. The

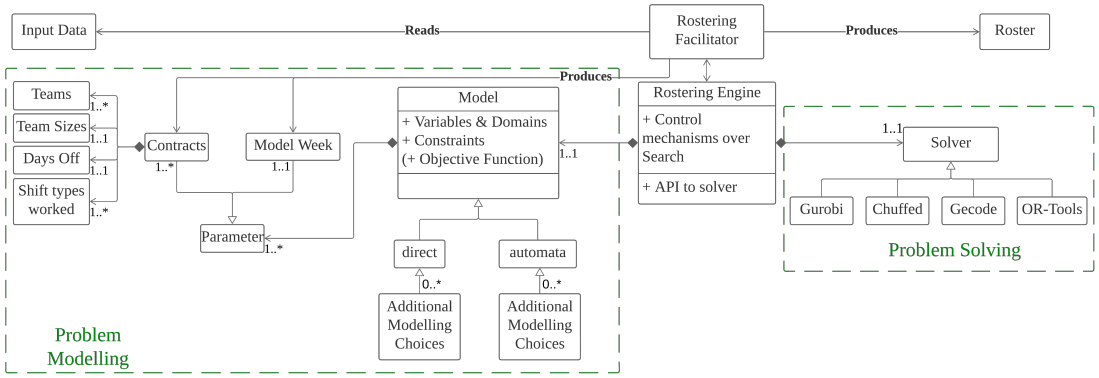



Figure 1.2: Class diagram of the problem implementation.

intention is to provide guidance to whoever may decide to venture onto a similar pathway, especially when prototyping rotating schedules.

The next section will then show some empirical evidence from our own implementation, and offer an early discussion of the pros and cons of such a solver independent, CP-based approach to solve our extended RWS problem. If anything, this will give us the opportunity to then move on to discussing our experience of applying this same approach to our real-world case of RWS in airport security. This will require, to begin with, that additional instance specific logic and constraints be accommodated within the same problem formulation.

### 1.4.1 Modelling and Solving Architecture

Adopting a UML class diagram style notation, Figure 1.2 provides a conceptual overview of our modelling and solving approach to the problem just formulated, showing how the components of any implementation of this approach might be suitably organised.

The core element of the architecture in Figure 1.2 is what we call the Rostering Engine. The way this class is organised reflects the solver independent nature of the approach, showing the clear separation between Model and Solver, its two main component classes. The Rostering Engine provides the means for the rostering professional using our system to pair exactly one model with exactly one solver, each chosen from separate collections, and to do so at virtually any new run of the system. This is achieved via appropriate Application Programming Interfaces (APIs) to the available solvers. Figure 1.2 names the solvers we used in our own implementation — more details on each to appear in the next sections, but the choice is wider and

different collections should be considered for different problems. The Rostering Engine also provides for more specific options to fine-tune the search parameters and thus apply additional controls to the search strategy. For instance, in a CP-based approach to modelling and solving, search 'annotations' can be used within the Rostering Engine, to control the variable order and the value order to be followed by the chosen solver, effectively changing the search tree for the problem, with no other modifications required. In our implementation, the Rostering Engine is implemented in MiniZinc, one of the many features of which is that it allows the separation of data (Input Data class) from the model of the problem (Model class).

Input data consist of problem and instance specific details such as: what contracts are available, the size and 'profile' of the workforce employed under each contract (number of teams, team size, shift types allowed, etc.), any crucial elements of the layout and logic of the systems/processes in which the teams work (e.g. an airport's security hall, or a factory's warehouse), and demand forecasts. It is then the Rostering Facilitator element that reads all the available data and prepares it in a format that is suitable for consumption by the solver. This new version of the input data becomes part of the Model element, which contains, just like any model to be used in problem solving (Ackoff, 1978): decision variables, parameters, constraints and (perhaps, though not in our problem investigated in this paper) an objective function to be 'optimized' (in whatever sense of the word). Using a language closer to our CP-based view of problem modelling and solving (Wallace, 2020) our Model class is organised around: variables, constraints and parameters. Variables represent the decisions that need to be made in the model and are usually accompanied by a set of possible values - their domains. Constraints impose rules on the decision variables, ruling out incompatible choices, and parameters define each problem instance to be investigated.

The Rostering Facilitator class acts as a 'wrapper' layer (implemented in Python, in our case) to connect the input data with both the engine/model and the roster that is produced (Roster class) as an output of the search process. On transforming the raw input data into its usable format as specified within the Model class, the Rostering Facilitator effectively creates the problem instance. Its other main tasks are to set the Solver and Model choices in the Rostering Engine and to create readable outputs of the problem solution (an object of the Roster class).

One last component of the Model class is worth discussing — the Model Week element. In the case study we discuss later, as well as in many other rostering applications, a 'template week' is artificially created to represent the whole planning

horizon. This is normally an effective way to smooth out demand variability over time, thus reducing the cyclic rostering problem to one that assumes the same weekly demand to present itself repeatedly over the entire horizon. While this facilitates problem solving, it relies on appropriate choices to be made by the rostering professional in defining what a model week looks like. For instance, the demand on Monday morning between 6am and 7am may be defined as the maximum value of demand taken, within the same time interval, across all Mondays of the planning horizon. Other alternatives clearly exist (average, mode, etc.), with the final choice being partially an arbitrary one.

Figure 1.2 also shows that different versions (subclasses) of models may exist (as in our implementation), effectively leading to a host of model options that can be tested, each of which being paired with one or more solvers. As we shall later discuss, the model variant may be a significant factor, in itself or as paired to a particular solver, when solving a problem such as our extended version of RWS. In the following Subsections 1.4.2 - 1.4.5, we go on to detail all of the specific options we tested in tackling our extended RWS. To stay in line with our aim to demonstrate the extension of the most suitable solution approach to RWS (Musliu et al., 2018), as concluded in Section 1.2), the following Section will present a selection of the most suitable modelling choices to our problem that were also presented by Musliu et al. (2018).

### 1.4.2 Direct Model

One of the simplest modelling choices is to create a direct implementation of the mathematical formulation, where all constraints are outlined in the modelling language — MiniZinc included — (almost) exactly as they are expressed in the mathematical model. However, while these models can be understood faster by someone who is not familiar with the intricacies of the programming language used, they may not lead to the fastest results, either in a specific instance or more generally for a type of problem.

### 1.4.3 Automata-based Models

An automaton is a device that is capable of representing a ‘language’, according to well-defined rules (Cassandras and Lafortune, 2008). As a modelling formalism, it is generally convenient whenever the concepts of a system’s ‘state’, the ‘transitions’ between any states, and the ‘events’ driving such transitions may be applicable. In rostering problems, states may refer to all possible shift combinations that can occur

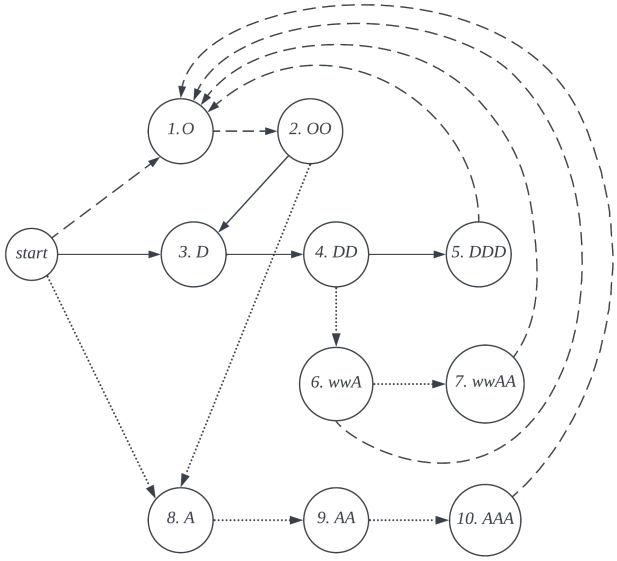



Figure 1.3: Example of an automaton encoding work requirements.

given the workforce rules. Hence, the rules surrounding shift types and days off may be encoded in an automaton.

Figure 1.3 gives an example automaton for a rostering problem with two shift types (day = $D$, afternoon = $A$), days off ($O$) and forward rotation enforced. We also use $w$ to represent a day where any type of work shift can be assigned. For the purpose of this example, different contracts are not considered. The minimum and maximum lengths of work and shift blocks are as follows: $b\_lo_D = 2$, $b\_up_D = 3$, $b\_lo_A = 2$, $b\_up_A = 3$, $b\_lo_O = 2$, $b\_up_O = 2$, $w\_lo = 3$, $w\_up = 4$. Solid lines represent the decision to have a day shift next, dashed lines represent the decision to have a day off next and dotted lines represent the decision to have an afternoon shift next. Dummy state '*start*' has the purpose of showing that any of $O$, $D$ or $A$ can be used as a starting point in the diagram. Note that in this example the states *wwA* and *wwAA* could also be called *DDA* and *DDAA*. However, in other instances, there could be additional shift types that can also be followed by an afternoon shift. In that case, *wwA* and *wwAA* can be used to encode the transfer from any shift type to afternoon shifts. When dealing with multiple contracts, where shift type rules tend to be contract specific, a separate automaton is created for each contract. Having a single 'catch-all' automaton for all contracts is unnecessary, due to the nature of automata which would lead to mutually exclusive sets of states for each contract in the "master" automaton. Most importantly, having separate automata offers a better overview of the modelling approach. It also scales better over time, within the same application, as the pool of contract options

being considered evolves.

### 1.4.4 Global Cardinality Constraints

In CP, a vast collection of global constraints exist. Given their effectiveness, when starting from a direct model a viable option to be put for testing is to replace one or more constraints in the model with one or more global constraints, and take note of the effects such a change may induce in model runtime and quality (acceptability) of the obtained solutions. In the first part of our work, we considered one such global constraint, namely `global_cardinality()`. The `global_cardinality`$(x,a,c)$ constraint ensures that in the input array, $x$, the number of occurrences of $a$ are exactly equal to $c$ (Stuckey et al., 2022, 4.2.1.1. Counting constraints). This constraint has an extension called
`global_cardinality_low_up`$(x,a,l,u)$ which ensures that in the input array $x$ the number of occurrences of $a$ are between $l$ and $u$ (Stuckey et al., 2022, 4.2.1.1. Counting constraints). Both constraints can also be used on sets of occurrences $a$. In that case the limits (either $c$ or $l$ and $u$) need to have entries for each element of $a$.

Global cardinality constraints are ideal for modelling demand satisfaction constraints. For example, constraint (1.8) can be rewritten as

$$\texttt{global\_cardinality}([S_{i,j}|c \in \mathscr{C}, i \in ft_c \dots lt_c, r \in 1 \dots ts_c], \mathscr{A}^+, [R_{sh,j}|sh \in \mathscr{A}^+]) \quad j \in 1 \dots g \tag{1.9}$$

The expression "$r \in 1 \dots ts_c$" is used to repeat each term of $S_{i,j}$ for every member of team $i$, to model team size. This is necessary since the demand is given in single workers rather than teams.

### 1.4.5 Redundant Constraints

More modelling options exist that can help to either try and improve runtimes and/or the quality of solutions. Redundant constraints are but one such option. They are employed to tell the computer program something that is obvious from the given constraints. One such example is adding a constraint with demand for days off. This is implicitly defined by the demand for all working shifts since the remaining workforce is automatically assigned a day off. However, writing this constraint explicitly can

improve runtimes in the model. This would be done by replacing constraint (1.8) with

$$\sum_{c=1}^{nc} \left( ts_c \sum_{i=ft_c}^{lt_c} (S_{i,j} = sh) \right) = R0_{sh,j} \qquad j \in 1 \dots g, sh \in \mathscr{A}^+ \tag{1.10}$$

The demand for days off is evaluated by removing the demand for all other shift types from the overall workforce.

In the following section, we shall analyse the results of experiments in which we pair a host of CP and MIP solvers with all our model versions. These represent various combinations of the modelling choices just outlined — see Table 1.3 for the full mapping.

| model | direct | automata | global | redundant |
|---|---|---|---|---|
| direct | ✓ | | | |
| direct_g | ✓ | | ✓ | |
| direct_gr | ✓ | | ✓ | ✓ |
| automata | ✓ | ✓ | | |
| automata_g | | ✓ | ✓ | |
| automata_gr | | ✓ | ✓ | ✓ |

Table 1.3: Overview of modelling choices by model version for the experiments in Section 1.5. Note that each of the columns refers to one of the modelling choices outlined in Sections 1.4.2 - 1.4.5 each.

Note that in table 1.3 there is a checkmark in the direct and automata columns for the automata model. This is because not all constraints (i.e., demand satisfaction) can be expressed in the automata. They have to be coded as direct or global constraints instead.

## 1.5 Early Testing: Benchmark Instances

To evaluate the performance of our models when paired with different solvers, we adapted the 50 benchmark instances used by Musliu et al. (2018) to fit the extended RWS problem. This way we produced 24 new instances (available via GitHub Wiesflecker (2022)) for our analyses. Subsection 1.5.1 discusses our instance creation process. We then present the experimental analysis of the different solver/model combinations in Subsection 1.5.2.

All Experiments were carried out on an iMac running macOS Monterey Version 12.3.1 with a 3.1 GHz 6-Core Intel i5 ($10^{th}$ generation) processor and 8GB memory. The models were implemented using MiniZinc Version 2.4.3, the MiniZinc Python interface and Python 3.8. We imposed a runtime limit of 30 minutes on all instances and used MiniZinc's built-in solvers 'Chuffed' (Version 0.10.4) (Chuffed, 2022) and Gecode (Version 6.3.0)(Schulte et al., 2022) as CP solvers. We also tested the CP solver from the OR-tools suite (Google Developers, 2022), which has won four gold medals in the 2021 MiniZinc Challenge (Stuckey et al., 2014; MiniZinc Challenge 2021 results , 2021), a global CP competition. In terms of MIP technology, we only tested a single solver. The reason behind this choice was that our aim in this paper was to see if anything (algorithm runtimes, patterns in the produced rosters, etc.) changes (and if yes, what exactly) when investigating the same underlying model for the extended problem with CP solvers vs MIP solvers. In this sense, knowing if a MIP solver offers more with respect to another was less crucial to our study. More specifically, we settled for Gurobi — version 9.0.3, (Gurobi, 2024) — for two different reasons. First, it appeared a natural choice, given that Musliu et al. (2018) made the same choice. Secondly, one of the two biggest airports in our projects already adopt Gurobi to tackle other optimisation problems, and the other one (the same considered in the case study of Sections 1.6 and 1.7) is presently considering using it in the future.

### 1.5.1 Experimental Set-up & Instance creation

Our first set of experiments in this paper was designed to somewhat mimic those in Musliu et al. (2018), the only difference being the introduction of teams and contracts to the RWS problem — the two experimental settings are thus not directly comparable to one another. In our tests we aimed to: (1) test whether and to what extent the solver independent approach in Musliu et al. (2018) would succeed in solving the extended RWS problem; and (2) investigate what model/solver combination performs best when multiple contracts and team sizes are explicitly examined.

An analysis of the 50 benchmark cases (Musliu et al., 2019) which were studied in Musliu et al. (2018) and also considered by Becker et al. (2022) found that the majority rely on the same set of forbidden shift combinations of length 2 and 3. Some instances only consider combinations of length 2, others do both, no instance only includes combinations of length 3. These two sets of forbidden combinations are then paired with different demand patterns and limits on work and shift blocks. In our work, we

recreated the same pattern, based on the same set of forbidden shift combinations, in the same proportions. We then randomly sampled workforce size, work requirements and demand patterns from the 50 benchmark cases and combined and/or multiplied them to create the 24 instances considered in this section and are available through the GitHub repository Wiesflecker (2022). Each instance has 2 contracts with different work requirements, and the team sizes vary between 1 and 10, with the workforce size varying between 8 and 65 (average of 37). These instances were then solved using all of the models outlined in Table 1.3 and the different solvers given at the beginning of this Section.

### 1.5.2 Results Analysis

Table 1.4 gives an overview of our results. Column '#tot' gives the total number of instances run and '#tot avg. rt.' gives the total average runtime across all instances. Similarly, '#sat' and '#sat avg. rt.' give the number and average runtime across all solved instances. Bold entries highlight the shortest average runtime for all solved instances and the highest number of solved instances within the runtime limit. For this set of experiments we found that all instances were solvable, but for some instances certain solver/model combinations failed to find a solution within the given time limit of 30 minutes.

Gurobi and Chuffed were the only solvers that managed to solve all instances, although none of the two did it across all model variants. In line with Musliu et al.'s conclusions (Musliu et al., 2018), Gurobi performs best when paired with automata-based models, even with the extended problem. Chuffed outperforms Gurobi with regards to runtimes of satisfied instances (significantly so when paired with automata-based models). It also produces the fastest runtimes overall, but solves fewer instances when compared to Gurobi. Musliu et al.'s conclusions (Musliu et al., 2018) that Chuffed performs best when paired with direct constraints is also confirmed with the extended problem. Surprisingly, the CP solver from the OR-tools suite struggled to solve all instances (with over a third of them stopped by the runtime limit) and showed some of the slowest runtimes overall for solved instances. Gecode also failed to solve, on average, just under half of all instances within the set time limit, across all models. However, when just considering the satisfied instances, the runtimes are second only to the results produced by Chuffed.

When looking at the results from the different modelling options, we can see that,

| solver | model | #tot | #tot avg. rt. | #sat | #sat avg. rt. |
|---|---|---|---|---|---|
| Gurobi | automata | 24 | 41.00s | **24** | **41.00s** |
| | automata_g | 24 | 106.71s | **24** | 106.71s |
| | automata_gr | 24 | 68.82s | **24** | 68.82s |
| | direct | 24 | 136.98s | 23 | 64.65s |
| | direct_g | 24 | 287.59s | 21 | 71.47s |
| | direct_gr | 24 | 153.11s | 23 | 81.49s |
| Chuffed | automata | 24 | 161.70s | 22 | 12.73s |
| | automata_g | 24 | 247.98s | 21 | 26.21s |
| | automata_gr | 24 | 96.98s | 23 | 22.92s |
| | direct | 24 | 49.52s | **24** | 49.52s |
| | direct_g | 24 | 523.83s | 18 | 98.28s |
| | direct_gr | 24 | 296.43s | 21 | 81.56s |
| OR-tools | automata | 24 | 803.47s | 15 | 205.36s |
| | automata_g | 24 | 946.68s | 14 | 336.94s |
| | automata_gr | 24 | 803.48s | 15 | 205.38s |
| | direct | 24 | 777.54s | 14 | 46.97s |
| | direct_g | 24 | 797.80s | 14 | 81.68s |
| | direct_gr | 24 | 777.52s | 14 | 46.92s |
| Gecode | automata | 24 | 714.15s | 15 | 62.43s |
| | automata_g | 24 | 703.01s | 15 | 44.64s |
| | automata_gr | 24 | 875.35s | 13 | 92.55s |
| | direct | 24 | 899.91s | 13 | 137.96s |
| | direct_g | 24 | 985.85s | 12 | 171.30s |
| | direct_gr | 24 | 969.32s | 12 | 138.26s |

Table 1.4: Average runtime for all instances and satisfied instances, for all model/solver combinations.

with the exception of the solver from the OR-tools suite, automata-based models tend to produce faster runtimes and a higher number of solved instances. Furthermore, in all model/solver combinations, apart from Gecode paired with automata-based models, adding the redundant constraint of having demand for days off decreases the total average runtime of all instances. In addition, adding the redundant constraint either increases the number of solved instances within the time limit or decreases the average runtime of solved instances. For Chuffed and OR-tools both effects on satisfied instances can be seen. This clearly shows the positive impact such modelling choices can have on the results.

Overall, the best model/solver combination is Chuffed paired with the direct model, and automata-based models perform better across multiple solvers. One issue with automata-based models is the lesser user-friendliness for non-expert users (a similar argument was raised on the graph based compact formulation that is so crucial in Becker et al. (2022)), at least in comparison to the direct model versions. However, works such as Musliu et al. (2018) demonstrate that in some cases automata-based models may clearly represent the best option, which in our view further strengthens the importance of a solver independent approach, one that lets the modeller/user investigate multiple model/solver options as we have just seen.

On a slightly different but still related note, as demonstrated by Musliu et al. (2018), one often does not know beforehand whether CP or, say, MIP technology would perform better when faced with a problem that is substantially new to the decision maker and/or the modeller. We saw this happening many times while working with our partner airports, when within the space of just a few weeks, depending on how discussions with the unions were developing, major changes to the rules of the game had to be investigated numerically. In all such cases, having the possibility to test which of the two technologies has more to offer may eventually lead, in cases where one clearly outperforms the other, towards an implementation that better exploits the functionality of the winning approach and the specific software and algorithms ultimately adopted. While this is happening, as we write, at one of the airports we are still working with, it should be acknowledged that the solver specific approach may threaten, to some or perhaps a major extent, the usability of the implemented tool directly by the scheduler. This may not be such a good idea, in the dynamic context of repeated negotiations outlined in the opening section, unless there was an analytics team who can support the scheduler throughout (which appears costly, resource-wise!), as well as a well-oiled, efficient process of interaction between scheduler and analyst.

Having schedulers who are also well versed in decision analytics seems a better option, but based on our albeit partial experience, this appears rarely to be the case.

Finally, the instances investigated in this section have very simple shift block and order rules that are encoded in the automata. These rules are not necessarily realistic for real-life applications, as we came to see in some of our airport security applications. This makes our second round of tests on the extended version of the problem — one that tackles a real-world airport instance — much needed. And this represents the next stop on our journey.

## 1.6 Case Study

The problem extension of RWS with teams and contracts is inspired by a case study we conducted with a UK-based airport operator. As part of this work, we also found that the standard benchmark examples for RWS could be considered simplistic compared to the true extent of rules and regulations faced by the CSS at our partner airport. This section aims to provide an overview of the more extensive set of rules and regulations that need to be considered when creating schedules for airport security officers. We do this by first providing a problem description (Section 1.6.1) and then showing the problem formulation of the RWS with teams and contracts in that setting (Subsection 1.6.2).

### 1.6.1 Problem Description

As part of this case study, we worked closely with a UK-based airport operator on their security staff rostering problem.

Firstly, airport passenger security does not appear to offer the most straightforward setting for adoption of cyclic rosters, for a reason we now explain. In fact, two types of demand co-exist in airport security. *Shift-based demand* (Ernst et al., 2004b, Section 2.1), the same type addressed in RWS, manifests itself at a minority of physical locations within airports, such as security gateways dedicated to airport employees and contractors, or airline crews. At such locations, demand volumes are relatively low and in general easier to predict more accurately. *Flexible demand* (Ernst et al., 2004b, Section 2.1) instead affects passenger security halls, the more critical (and visible!) components of airport security. In airport security halls, flexible demand is affected by both short- and long-term fluctuations in demand

patterns, involving relatively high volumes of passengers, whose arrival processes are essentially uncontrollable—passengers reach security halls either from check-in halls or directly from their access route of choice (airport train/tram/bus station or car park, etc.). Across the five to seven months of a typical airport security roster, long-term fluctuations are driven by changes in the flight timetable. These can be dealt with by creating suitable 'model weeks' that represent the variety seen in the timetable and can be used as reference from which to develop suitable rosters. However, short-term fluctuations on any day of operation can vary significantly, over time intervals of 15 minutes or less, being driven by the largely uncontrollable passenger flows and variable behaviour when passing through security (the latter can also have a tremendous impact on process efficiency, and is extremely difficult to control). At this level, airport operators do not just schedule shift types for their staff, but 'proper' shifts with specific start and end times (and they go even beyond, by allocating breaks for each shift, etc.). Demand can still be predicted to a certain extent, but the model ought to be more detailed (that is broken down into 15-minute intervals) than the one outlined in the opening sections. Therefore, the problem setting at our partner airport goes beyond the allocation of days-off and shift types, in terms of time granularity (15-minute time bins), complexity, and even types of demand addressed (both flexible and shift-based, related to multiple security locations across the same airport). Still, the way we tackled this more extensive problem entailed decomposing the rostering process into smaller sub-problems (a common approach in the rostering literature, see (Ernst et al., 2004b)), one of which specifically focuses on days-off and shift types and as such, corresponds to our extended RWS problem discussed so far (with a different set of constraints, as we shall soon see). The other steps comprise the setting of shift start and end times and break assignment. Both are out of scope for the present paper, and will be addressed separately.

On a different note, the scale of our airport security rostering problem is larger than both the one we have seen dealt with in the literature (Section 1.2) and the one investigated in our artificial instances so far. At our partner airport, before Covid-19 broke out, the security workforce consisted of about 400 workers, out of a total of just under 750 for the airport overall. The specific instance studied in this section relates to more recent times (the airport industry is renowned to have been hit badly by the Covid-19 pandemic!), and addresses 222 workers, spread over six contracts, with different work and days-off regulations by contract. The workforce ought to cover up to ten different 'posts' (locations) across the airport, with different mixes

of shift-based and flexible demand (either one of the two, or both). The majority of posts are located in the main security hall of the airport, where all passengers have to undergo security checks (hand luggage scans and metal detectors/body scans) before reaching their allocated gates. The security hall consists of multiple lanes, each equipped with luggage scanners. Each pair of contiguous lanes shares a body scanner portal in between, thus creating a 'cell'. Demand forecasts of passenger arrival processes, available from other functions within the same organisation, are translated into an equivalent number of lanes to be kept open at different times of the day to cope with the passenger influx. In addition, the number of workers required to open a new lane at any time depends on whether the other lane in the cell is already open or not (fewer workers are needed to open a cell that already has one of its two lanes open).

An important requirement at the time when we ran the instance discussed in the next section was that each team would cycle through their schedule at least three to five times during the planning horizon (season). While covering all locations would suggest using lots of small teams to foster higher levels of flexibility in covering the predicted demand at the requisite 15-minute granularity, this would quickly lead to too many lines in the roster, which in turn would make it impossible to meet this particular requirement. This problem was solved by having varying team sizes by contract (larger team sizes for contracts with a high number of employees and smaller team sizes for contracts with fewer employees), thus keeping the flexibility of covering demand at all locations whilst maintaining the cycle for larger contracts to a reasonable length.

Another crucial aspect to security rostering at our partner airport is finding, in the medium- to long-term, the best workforce split given the available contracts —a more strategic type of decision. This ultimately facilitates the rostering process, as much as it helps to keep staffing costs under control. Having a rostering tool that solves within minutes (or, better, seconds) as opposed to weeks would be useful for quickly testing the effect of any such decision on the resulting roster. This related decision making is also currently supported only by spreadsheets, and is one area of application where intelligent decision support systems (otherwise more akin to tactical level decision support) have demonstrated to be particularly useful (Wallace, 2020). The reality is that airports (including our partner) would prefer to have as many staff in their workforce as possible who are signed on low-hour contracts (say 20 hours/week vs 40 hours/week), and small teams (where this can be decided at all), to gain more flexibility in their scheduling. The opinion of the unions is, in general, a different one, hence (once more) the importance of being able to support the dynamic negotiation

process as argued so far in this paper.

To conclude on this concise presentation of our case study, our partner airport's security rostering problem is indeed very similar to the RWS problem of Section 1.3, the most notable differences being the addition of the following constraints:

- Each team must have two consecutive days off at least once every two weeks;
- Each team must have a weekend off at least every 3 weeks;
- There is an upper limit of teams of a certain size to be on shift each day due to location constraints at a later stage of the decomposed problem;
- There is an upper limit on how many days each team can work per line.

In addition, not all of the constraints from the general formulation in Section 1.3 apply here. Namely, there is no lower limit on the length of work blocks, nor an upper (a lower) limit on the length of shift blocks (constraints (1.2), (1.3), and (1.4) in the general problem formulation). The next subsection provides a detailed formulation for the rostering problem of our partner airport.

### 1.6.2 Problem Formulation

To formulate the RWS problem at our partner airport, the following additional notation is required:

- $cr_i$: contract of team $i \in \mathscr{T}$
- $d\_up_c$: maximum number of working days per line for teams under contract $c \in \mathscr{C}$
- $w\_off_i$: binary indicator encoding whether team $i \in \mathscr{T}$ has been allocated a weekend off or not.
- $q, r$: integers used for defining the weekends off requirement — often, one must ensure that at least $q$ out of any $r$ weekends are set as off work, where $q \leq r$.
- $v\_off_i$: binary indicator encoding whether team $i \in \mathscr{T}$ has been allocated two consecutive days off or not.
- $R6$: Requirement matrix for teams sized six. This matrix encodes the maximum number of teams of size six that can be rostered on any day for any shift type.

- $x, y$: integers used for defining the consecutive days off requirement. A typical form is to ensure that at least every $x$ out of $y$ weeks are set to have two consecutive days off, where $x \leq y$.

Using this notation paired with the one outlined in Section 1.3 the airport's *rough* roster problem can be formulated as follows:

$$\sum_{j=1}^{g} (S_{i,j} \neq O) \leq d\_up_{cr_i} \qquad \forall i \in \mathscr{T} \tag{1.11}$$

$$\sum_{k=0}^{w\_up_c} (S_{v_c(i,j+k)} = O) > 0 \qquad j \in 1 \dots g,$$
$$c \in \mathscr{C}, i \in ft_c \dots lt_c, \tag{1.12}$$

$$S_{i,j} \in \mathscr{A}_c^+ \qquad j \in 1 \dots g,$$
$$c \in \mathscr{C}, i \in ft_c \dots lt_c, \tag{1.13}$$

$$S_{i,g} = O \wedge S_{i,g-1} = O \leftrightarrow w\_off_i = 1 \qquad i \in \mathscr{T} \tag{1.14}$$

$$\sum_{k=1}^{r} (w\_off_{u_c(i+k)} = 1) \geq q \qquad i \in ft_c \dots lt_c, c \in \mathscr{C} \tag{1.15}$$

$$\exists (j \in \{1 \dots g-1\}) \big( (S_{i,j} = O) \wedge (S_{i,j+1} = O) \big) \leftrightarrow v\_off_i = 1 \qquad i \in \mathscr{T} \tag{1.16}$$

$$\sum_{k=1}^{y} (v\_off_{u_c(i+k)} = 1) \geq x \qquad i \in ft_c \dots lt_c, c \in \mathscr{C} \tag{1.17}$$

$$S_{i,j} = sh_1 \rightarrow S_{v_c(i,j+1)} \neq sh_2 \qquad j \in 1 \dots g, (sh_1, sh_2) \in \mathscr{F}_c,$$
$$c \in \mathscr{C}, i \in ft_c \dots lt_c \tag{1.18}$$

$$\sum_{c=1}^{nc} \big( ts_c \sum_{i=ft_c}^{lt_c} (S_{i,j} = sh) \big) = R_{sh,j} \qquad j \in 1 \dots g, sh \in \mathscr{A} \tag{1.19}$$

$$\sum_{c=1}^{nc} \big( (ts_c = 6) \sum_{i=ft_c}^{lt_c} (S_{i,j} = sh) \big) \leq R6_{sh,j} \qquad j \in 1 \dots g, sh \in \mathscr{A} \tag{1.20}$$

Constraints (1.11) ensure that no team, $i$, works more than their contracted number of days per line, $d\_up_{cr_i}$. Similarly, constraint (1.12) ensures that each block of consecutive work days over a contract's, $c \in \mathscr{C}$, cycle is below the contracted limit for consecutive days of work, $w\_up_c$. Constraints (1.13) restrict the assignment of shift types, $sh$, to be just out of the pool of shift types, $\mathscr{A}_c^+$, allowed for each contract, $c \in \mathscr{C}$. Constraints (1.14) and (1.15) model the required pattern of having $q$ weekends off out of every $r$ consecutive weeks. Similarly, constraints (1.16) and (1.17) model the requirement of having $x$ out of every $y$ weeks with at least two consecutive days

off. Constraints (1.18) rule out forbidden shift combinations for each contract. Finally, constraints (1.19) ensure the workforce demand is satisfied by the schedule, while constraints (1.20) limit the number of teams of size 6 to ensure that location-specific demands at a later stage can be satisfied.

### 1.6.3 Modelling Choices

The model faced by our partner airport is clearly guided by more complex constraints than the simple RWS with multiple contracts and team sizes outlined in Section 1.3. However, this also gives more opportunity to use additional modelling choices to represent the problem. The following Sections 1.6.3.1 - 1.6.3.3 will outline the modelling choices that we used in addition to the ones outlined in Section 1.4 while the last Section 1.6.3.4 will summarise all the models considered in the experimental analysis in Section 1.7 with the modelling choices implemented in each of them.

#### 1.6.3.1 Global Constraints: Sliding Sum

In addition to the global cardinality constraint already used by Musliu et al. (2018), we also opted to use the sliding sum constraint, another option within MiniZinc's global constraints selection, to model some of the constraints of our problem. The `sliding_sum`(l,u,s,a) constraint ensures that the sum of every $s$ consecutive terms in the array $a$ lies between $l$ and $u$ (Stuckey et al., 2022, 4.2.1.12. Other declarations). This global constraint is ideal for modelling rules for days off such as weekends off or maximum consecutive days worked.

To model the constraint that at least $q$ out of $r$ weekends need to be assigned off (constraint (1.15)) we can write

$$\texttt{sliding\_sum}(q, r, r, [w\_off_i | i \in ft_c \dots lt_c] ++ [w\_off_i | i \in ft_c \dots ft_c + r - 1]) \qquad c \in \mathscr{C} \tag{1.21}$$

Note that "++" is the operator for concatenating arrays in MiniZinc. This is necessary here, to ensure that the weekends off constraint is satisfied for every $r$ consecutive lines in the schedule.

To model the maximum consecutive days worked constraint we first need to define a new binary matrix $O$ which has entries $O_{i,j}$. For every day off in the schedule ($S$) the matrix $O$ encodes a zero ($O_{i,j} = 0$), and $O_{i,j} = 1$ for every other day in $S$. We can then rewrite the maximum consecutive days worked constraint (1.1) as follows:

$$\texttt{sliding\_sum}(0, w\_up_c, w\_up_c + 1, \\ [O_{i,j} | i \in ft_c \dots lt_c, j \in 1 \dots g] ++ \\ [O_{ft_c,j} | j \in 1 \dots w\_up_c]) \qquad c \in \mathscr{C} \tag{1.22}$$

Similar to the weekends off constraint, we repeat some terms at the end to ensure that the constraint is satisfied throughout the entire cycle. Note here that the first $w\_up_c$ entries of $O_{i,j} = 0$, for each contract are repeated. This notation only works if $w\_up_c \leq g$. In most scheduling settings it is not the case that employees are allowed to work a whole line (or week) without a day off. Therefore, we keep this simplified version of the constraint.

#### 1.6.3.2 Symmetry Breaking

In addition to the above-stated modelling choices, there is also the option of adding a symmetry breaking constraint to the model. Due to the nature of cyclic rosters, there will always be multiple equivalent solutions to the problem. An example would be using a different line as the starting week of the schedule. The workforce would still work the exact same schedule but have a different starting point. Setting the weekend of the first line of the schedule for each contract to be a weekend off, removes all equivalent solutions that would start with a different line in the roster. This method, therefore, reduces the solution space of the problem without impacting the quality of the solution produced. This method to improve solver performance is well known in cyclic rostering and was also used by Musliu et al. (2018).

#### 1.6.3.3 Reduced complexity

Symmetry Breaking can be taken a step further by hard-coding some constraints and thus reducing problem complexity. An example of this would be to set weekends off in the model so that constraint (1.15) is satisfied before the solver is applied to the model. This method might not always work as it could remove all feasible solutions from a problem. But for larger instances, it can reduce the runtimes significantly, which might be more desirable than finding a particular solution in some cases.

#### 1.6.3.4 Model Overview

For the purpose of our experiments in the following section, we consider five different model variants. Table 1.5 shows all the models and the modelling choices used. The columns can be read as follows:

- direct: all constraints (1.11) - (1.20) are represented in direct mathematical formulations. This does not take into account the direct constraints used to channel weekends off or consecutive days off (constraints (1.14) and (1.16)).
- automata: all work rule related constraints ((1.11) ,(1.12), (1.13), (1.15), (1.17), and (1.18)) are encoded in a separate automaton for each contract
- gcc: constraints (1.19) and (1.20) are modelled using `global_cardinality()`
- sum off: constraints (1.15) and (1.17) are modelled using `sliding_sum()`
- sum work: constraint (1.11) is modelled using `sliding_sum()`
- sym: symmetry breaking is enforced by setting the first weekend of every contract as a weekend off. This is coded as a direct constraint but does not count toward the “direct" column
- reduced: constraints (1.15) and (1.17) are replaced by hardcoded versions that automatically set certain weeks as the required weeks to have a weekend off or two consecutive days off. Again, this does not count towards the entry in the “direct" column.

| model | direct | automata | gcc | sum off | sum work | sym | reduced |
|---|---|---|---|---|---|---|---|
| direct | ✓ | | | | | ✓ | |
| direct_g | ✓ | | ✓ | ✓ | ✓ | ✓ | |
| direct_gd | ✓ | | ✓ | | ✓ | ✓ | ✓ |
| autom_g | | ✓ | ✓ | | | ✓ | |
| autom_gd | | ✓ | ✓ | | | ✓ | ✓ |

Table 1.5: Models and their respective modelling choices.

## 1.7 Case Study - Model Analysis and Impact

### 1.7.1 Instance Creation

To further investigate the efficiency of the proposed model/solver combinations, we created a set of 30 synthetic instances based on the case study data we worked on originally. The synthetic instances come in 3 sizes (10 instances per size):

1. size 1: identical problem size to the case study example (48 teams with a total workforce of 222 staff members). The demand patterns are a combination of constant shift-based demand and randomised flexible demand (for up to six lanes) for each day and shift type.

2. size 2: the original problem size is doubled (96 teams, 444 staff, up to 12 lanes).

3. size 3: the original problem size is tripled (144 teams, 666 staff, up to 18 lanes).

The synthetic instances of size 1 are based on the workforce required to staff the security hall of an airport accommodating about 12 mil. passengers per year (after Covid-19). This means that the instances of size 3 would be representative of an airport with a security hall three times as big. This is roughly equivalent to the number of passengers seen by the top 15 busiest airports in Europe before the Covid-19 Pandemic (The Port Authority of New York and New Jersey, 2019, p. 30). Larger airports are usually split into multiple terminals, with security often scheduled separately for each terminal. Heathrow airport in the UK, 7th busiest airport in the world, saw approximately 80mil. passengers in 2019 (The Port Authority of New York and New Jersey, 2019, p. 30), but spread over 5 terminals (on average, 16mil. per terminal). These numbers suggest that roughly speaking and assuming the same security layout as in our partner airport, the instance sizes considered in this paper are applicable even to some of the largest airports in the world.

To solve these instances with all of the defined models (Table 1.5) we also had to create the corresponding automata for all contracts. Given the additional rules on weekends off and consecutive days off, these automata now have to model work patterns over weeks rather than days. As a result, the number of states encoded in automata ranges from 257 to 559 in this set of instances. Not only does this method result in large automata but the process of translating all work rules into states and their respective transitions can be a lengthy process. This is the opposite of what practitioners need for their schedule prototyping phase where lots of rules need to

be implemented and tested quickly. Still, an underlying research question for us was around the extent to which the automata-based approach that had worked so well originally in Musliu et al. (2018) and also performed reasonably for the smaller instances of the extended problem, would still do well when faced with real-world sized instances.

### 1.7.2 Experimental Analysis

For these experiments, we used the same experimental setup as outlined in Section 1.5, again with a runtime limit of 30 minutes imposed on all instances.

Figure 1.4 shows the results of our experiments.

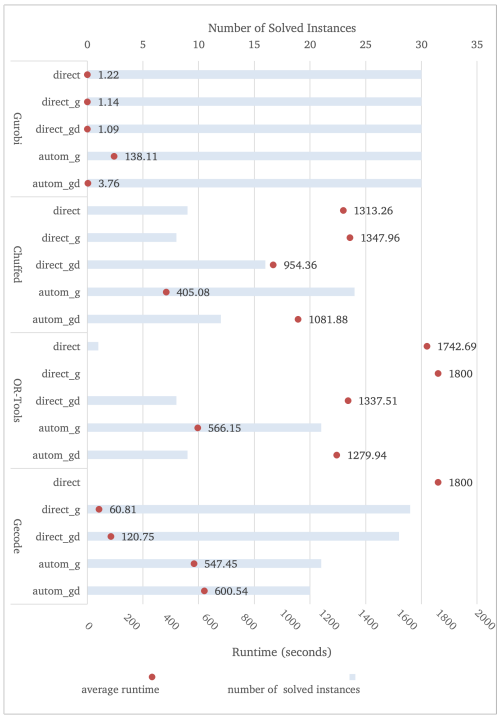
We can see that Gurobi is the only solver that successfully solves all instances for all models. The solver from the OR-Tools suite yielded somewhat opposite performance, with the least number of solved instances and slowest runtimes.

Overall, the best model/solver combination is Gurobi paired with the direct_g and direct_gd models, with direct_gd running faster on average than direct_g. The direct_gd model also includes constraints to assign weekends off and consecutive days off in fixed weeks. While this clearly gives faster results, there might be cases where this approach leads to infeasibility, which is not the case with direct_g. In addition, from a practitioner's perspective, the average time difference of 0.05s would not be noticeable during the roster prototyping process. We can therefore conclude that direct_g would be the better choice between the two.

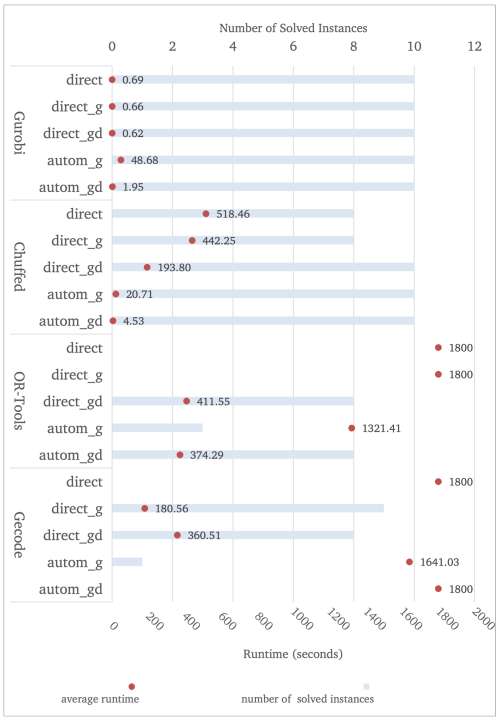
In Section 1.5, the conclusion was that Gurobi/automata-based yielded the best results. The opposite happens here. This goes on to show that, on top of the difficult process of creating and changing automata-based models in real-life scheduling problems, the resulting model is no longer as competitive against the more direct approach of expressing each work rule as a separate constraint. Two likely reasons: (1) the significant increase in the number of states when moving from the benchmark examples to real-life instances; and (2) the real-world problem has fewer upper and lower bounds on block lengths of shifts, which might have reduced the effectiveness of the automata-based approach.

Looking at the problem size specific results we have a number of findings.

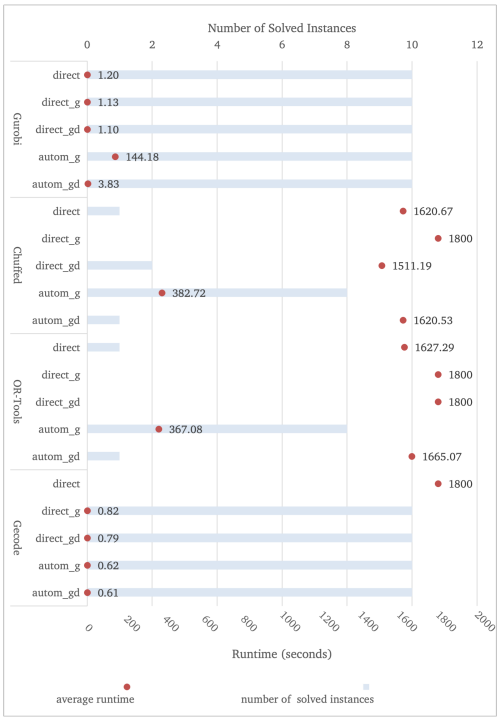
Firstly, three combinations fail to produce meaningful results across all instances, namely OR-Tools paired with the direct and direct_g model, and Gecode paired with the direct model.



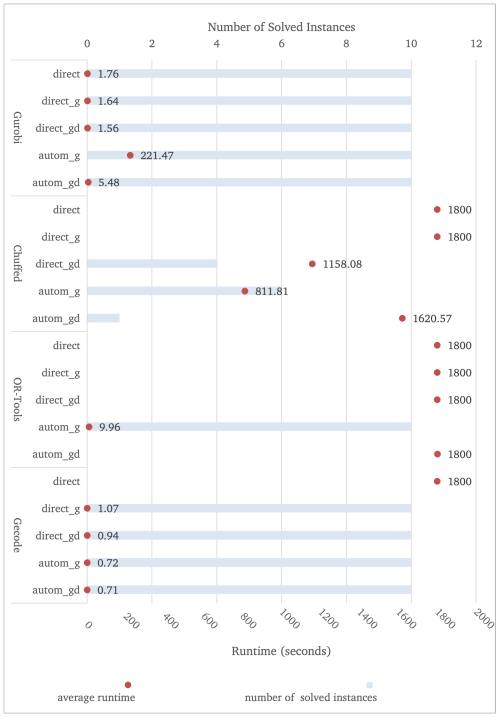

(a) all instances




(b) size 1 instances




(c) size 2 instances




(d) size 3 instances

Figure 1.4: Average runtime (in seconds) depicted by red dots and number of solved instances (blue columns) for all synthetic instances and split by instance size.

Secondly, whenever a combination solves problem instances, we can observe an overall trend for all models paired with Gurobi, Chuffed and OR-Tools where either the number of solved instances decreases or the runtime increases as the problem size increases, which is not surprising. The only exceptional cases are Chuffed paired with the direct_gd model, where size 3 experiments have more solved instances than size 2 experiments and OR-Tools paired with autom_g.

A third interesting finding relates to this second exception, for which the number of solved instances increases with the problem size! Further analysis of just the satisfied instances for this model/solver combination gives the following average runtimes: 203.94s, 8.75s and 9.96s for instances of size 1, 2 and 3 respectively. Once again, we can see that the runtime increases as the problem size goes from 2 to 3, which is in line with normal expectations. However, the solver struggles with the instances of size 1. But this is the exact same pattern found for Gecode paired with all models (apart from the already mentioned direct model). One explanation for this could be the fact that we are dealing with a relatively small sample of 10 instances per problem size. While the workforce increases by 222 employees from one size to the next, and while physical size of the process/system (i.e. the number of lanes) also increases proportionally, the demand patterns (that is the number of lanes required at any one time of the same day) are simply randomly generated. With such a small sample size of 10, there are higher chances of variations in aspects of the problem such as average workload (ratio between demand and the number of employees per instance). To better understand what happened in our experiments, we now point at Table 1.6, to highlight that, while the average number of shifts increases with the problem size, the average workload per employee in fact decreases as the problem size increases. Paired with the fact that in this case study there are no rules on maximum days off or minimum days worked, this would explain why CP solvers perform better for larger instances. In fact, with a higher employee to shift ratio (i.e. a lower workload), it becomes easier to satisfy all work rules, something that CP solvers specialise in.

Altogether, the points just highlighted further support the idea that the methodology developed and tested in this paper for the extended RWS problem in airport security is not just viable, but potentially produces very rich information for the scheduler to consider, with a number of practical implications. Which leads us to the concluding remarks in the next (and final) section.

| instance size | avg. number of shifts | avg. workload |
|---|---|---|
| 1 | 672.6 | 3.03 |
| 2 | 1077.6 | 2.43 |
| 3 | 1315.2 | 1.97 |

Table 1.6: Average number of available shifts and workload for each instance size.

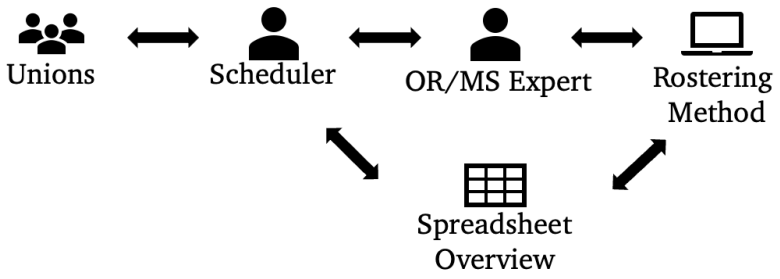



Figure 1.5: Rostering Method in practice.

### 1.7.3 Early Impact

The method and results shown in this paper are only one step of a more extensive Rostering Tool we are currently developing and implementing with our partner airport. Once implemented, this rostering tool will address the entire rostering problem of our partner airport which includes rough rostering — as shown in this paper —, tour scheduling, and break assignment for a large multi-contract workforce. Driven by the need for flexibility, each of those problem parts will be implemented with an array of constraints and objective functions to work with.

In our experience, the person responsible for the schedule has a vast knowledge of all the required rules and constraints and is very proficient at creating schedules by hand with the help of spreadsheets. Clearly, this approach is not scalable indefinitely, especially in light of the increased frequency of negotiations with the unions; hence the need for a more automated approach.

Figure 1.5 gives an overview of how our approach will be used in practice. The scheduler — who is not formally trained in OR/MS/computer science techniques — will continue to have discussions with the unions while being able to use the Rostering Tool through a Spreadsheet-based interface. This interface will allow the scheduler to do the following without needing to make any changes to the underlying rostering model:

- Change contractual details (i.e., add/remove contracts, change team number and size, weekly work hours/days, select allowed shift types, etc.);
- Change predefined problem parameters (i.e., number of days off per week, weekend off rules, rules about shift duration, etc.);
- Import the newest demand forecasts;
- Update break requirements;
- Select/deselect constraints and objective for all steps of the rostering process.

In addition, the scheduler will receive support from a member of the airport's Analytics team who is familiar with the programming languages of the Rostering Tool. This "OR/MS Expert" will be able to add new constraints to the Rostering Model — and link them to the Spreadsheet-based interface — as they arise from discussions with the unions or changes to the system layout. From discussions with our partner airport, we would expect these more substantial changes to happen at most 1-2 times per year. Also, as time goes on, this approach will lead to a more and more substantial catalogue of possible constraints and objective functions for the separate steps. Such a catalogue will be representative of the working time regulations in the country where the airport is located, as well as any other specific features of the particular context in which the airport is immersed, its organisational culture, its strategy, etc. The many (costly!) commercial rostering products bought and tested by our airport partner over the years were found to be unusable in their organisational context, mainly because of the impossibility of dealing extensively and efficiently enough with such an ever-changing and growing catalogue. With this need now fulfilled, the airport analysts and scheduler are currently starting to use our Rostering Tool more regularly in negotiating with the unions, and have started to think of a possible deployment of the same approach to staff scheduling in other areas of their organisation.

## 1.8 Discussion and concluding remarks

In this paper, we have investigated a new extension to RWS problems, introducing contracts and teams with multiple sizes. This was inspired by a number of real case studies originating in airport security processes.

Our own approach to modelling and solving retained all the main features of one of the most suitable approaches for RWS problems as identified in our Literature Review in Section 1.2 — namely a solver independent, CP-based approach, in turn, grounded on a high-level modelling language, interfaces to multiple solvers, and control mechanisms over the search for solutions (Musliu et al., 2018). We believe that our experimental results, particularly those from Section 1.7, provide support for the use of such a solver independent approach in a prototyping setting — certainly in airport security rostering.

On the one hand, without the existence of a clear 'winner' (either model or solver or a combination of both), there hardly seems any reason for schedulers in our roster prototyping setting to settle on a single overall direction (say CP vs mathematical programming), nor on a specific solver or model. This appears to us defensible at least as long as solving RWS problems from the end user's viewpoint is closer to 'satisficing' than it is to 'optimising', in the meaning of these terms as originally discussed by Herbert A. Simon (see (Simon, 2019) for a recent reissue of one of his seminal works) or the already cited Russell L. Ackoff (Ackoff, 1978).

On the other hand, in the same setting, even if a clear winner existed, the extremely dynamic nature of the airport/unions negotiations would normally make it quite likely that soon (i.e. in a few negotiation steps from now), the problem to be solved will be so different from the current one that a different winner may surface. This would make the rigid choice of any one particular direction, model and solver even less defensible, in our view.

Overall, our work supports the argument that a solver independent, CP-based approach for rostering with multiple teams and contracts can be very convenient to airport schedulers, for a number of reasons. Firstly, runtime performance from across all our experiments seems very reasonable, compatible with the timescales of the negotiation process in which the problem needs solving, and in any case orders of magnitude quicker than the current, highly manual, spreadsheet-based approach.

Secondly, the approach does allow for fast changes in the model/solver combination put under test. This in turn enables the production of rich pictures such as Table 1.4 and Figure 1.4, so that the existence of an eventual winning combination may be rigorously tested and, where a winner does not exist, choice of one combination as the 'currently preferred' configuration can be made with rigour and based on evidence. At the airport of our case study discussed in Section 1.6, no clear winning model/solver combination existed. Still, one could argue for the adoption of

Gurobi vs the three CP solvers tested and, perhaps (but only in light of additional implementation and experimentation), a mathematical programming approach and/or a different MIP solver.

Similarly, when multiple model/solver combinations show acceptable runtimes, the host of solutions produced will provide the scheduler with a portfolio of candidate rosters to be put forward for further negotiation. In the ideal case where the end-user of this intelligent decision support system for rostering coincides with the expert scheduler, the possibility to quickly and visually examine the many roster options available in the portfolio will ensure that any roster being discussed only includes scheduling patterns that are highly acceptable to both parties involved. It should also be noticed that many standard solvers already have the capability to produce 'symmetric' solutions that are equivalent to one another in the sense that they all meet the same set of constraints for the same instance being investigated. The rich pictures of Table 1.4 and Figure 1.4 therefore further amplify the portfolio of outputs available to the scheduler. In our own projects in this area, we noticed that this was the key feature that would help to foster acceptability of the approach. When looking at the outputs produced by our implementation of the approach, the expert scheduler of our case study would start comparing what was being displayed versus what would be produced by the fully manual approach in use at the time. The most striking advantages of our approach would then be perceived as: (1) the quick lead time to produce solutions (seconds of algorithm runtime vs days of manual work with spreadsheets); and (2) the prompt availability of multiple solutions (not as easy with the fully manual approach). While the former was simply 'expected', the latter seemed to impress the end-user more strongly, for the reasons discussed so far.

Inspired by our results, the airport of this case study has started to move away from solely spreadsheet-based and manual workforce planning. They are now testing the use of our approach (as outlined in Section 1.7.3), with the support of spreadsheets as both a front-end for data input as well as to collect and display the generated roster options to be put forward for discussion with the union representatives.

In conclusion, this paper focused on introducing a new RWS problem variant with multiple teams and contracts and exploring solver independent modelling as a suitable approach for practitioners. The experiments we conducted raised some interesting problem related aspects that could be explored as part of further research on RWS problems. In particular, further investigation into the effects of constraints and workload on model performance could be of interest to the wider research community.

# Acknowledgements

We would like to express our thanks to the CSS at our partner airport for the problem insights and data provided that allowed us to conduct this research.

# Chapter 2

# Paper II: Fine-tuning Flexibility in Workforce Schedules


## Abstract

Staffing is one of the most expensive cost centres at many airports. Being able to use the available workforce to its full potential without incurring unnecessarily high additional costs due to unforeseeable disruptions (such as staff absences and changes in demand patterns) can be crucial in reducing staffing costs. One way to do this is by building flexibility into rosters for schedulers to adapt more easily to existing shifts when such disruptions occur. But flexibility comes at a (often high) cost. We provide a new formulation for measuring surplus shifts (one such flexibility) for workforces with multiple contracts and team sizes. We also propose three Simheuristics, each combining Monte-Carlo Simulation with (alternatively) Adaptive Large Neighborhood Search, Simulated Annealing, and Genetic Algorithms, to fine-tune combinations of flexibility metrics for staff scheduling. Our experiments suggest that implementing a combination of flexibility metrics (such as demand aggregation levels and measures of surplus shifts) into the rostering process can drastically reduce average weekly re-rostering costs for airport security staff (the application domain that stimulated our research).

## 2.1 Introduction

In tactical workforce planning, the creation of staff schedules to cover variable, largely unpredictable demand is a particularly tricky task. We argue this may be largely due to the inevitable lack of control the employer has over the operating environment. Regardless of how well-planned these schedules are, there will always be some level of unforeseeable disruption to the smooth operation of the business. As a result, schedulers have to make last-minute adjustments to the roster— such as calling in staff to cover absences or unexpectedly high levels of demand at the *wrong* times of day— often to the dismay of the people directly affected by the changes (which often includes the customer). In the worst cases, nothing can be done to rectify the situation, which means the workforce scheduled to work at that time has to pick up the slack and/or deal with disgruntled customers. The ability to plan for some levels of disruption and thus avoid last-minute roster adjustments (especially the more costly options) would impact positively on the smooth running of the operation and on both employee and customer satisfaction.

When talking about uncertainty in scheduling, Van Den Bergh et al. (2013) present three main categories of *uncertainty*:

- *of demand* — unpredictable *workload*;
- *of arrival* — unpredictable *arrival pattern* of workload;
- *of capacity* — unpredictable deviation between planned and actual *processing capabilities*.

We argue that a different lens on uncertainty — one that looks at the level of impact the unforeseeable disruption has on actual operations — might be just as useful:

- *Major* events such as prolonged power outages, the majority of the workforce suddenly falling sick (COVID-19), major weather events, etc.— these cause significant disruption to the system, if not complete failure.
- *Day-to-day* events such as short-notice staff absenteeism (illness, car accident on the way to work, . . . ), minor discrepancies between predicted and actual demand patterns, etc.— these may cause occasional but temporary halts to the operation, and will generally impact the quality of service delivery and, more generally, operational performance.

In this paper, we focus on the latter. As we shall see, the 'day-to-day' qualifier should not be seen as an indication of 'lesser' costs (versus, say, 'major' events). In sectors such as commercial aviation (where this research originated), this point is already widely recognised (CAA, 2017). We present and investigate a novel way to account for day-to-day uncertainties preemptively and, as a result, introduce *focused* levels of flexibility into a roster, thereby keeping *re-rostering* costs under control. In the later sections of the paper, we provide evidence to argue that the benefits of adopting our approach may be significant, especially in settings where staff schedules must be agreed upon weeks, if not months, ahead of the day of operation.

The literature on rostering is extensive, encompassing a wide range of approaches, methodologies, and techniques. A wealth of authoritative, comprehensive reviews exists, such as Ernst et al. (2004a) and Brucker et al. (2011). Incorporating uncertainty remains much less prevalent when compared to deterministic approaches (Van Den Bergh et al., 2013). The remainder of this paper will cover a Simheuristic approach for fine-tuning a new, extended version of existing metrics of *schedule robustness* (Wickert et al., 2021), to better address the levels of day-to-day uncertainty the business faces.

The two main arguments of this paper are:

A1 Disruptions caused by day-to-day uncertainty often require costly re-adjustments to previously agreed staff rosters.

A2 Simheuristics provide a viable approach to fine-tune the levels of schedule robustness that are most suitable to the problem context on hand.

From a methodological standpoint, this paper produces the following three contributions:

C1 We extend *surplus shift* metrics (*buffer* and *reserve*), originally proposed in Wickert et al. (2021), to suit workforce settings with *multiple teams* and *multiple contracts*.

C2 We demonstrate the efficacy of Simheuristic approaches for parameter tuning in cyclic staff rostering to set ideal levels of robustness the staff schedule should possess while keeping re-rostering costs meaningfully under control.

C3 We discuss in detail the merits as well as drawbacks of three possible choices of Simheuristic for parameter tuning, each one combining Monte-Carlo Simulation with a different metaheuristic (ALNS, SA, GA).

We also outline the specific practical nuances of the problem of interest that make it relevant to scheduling scholars (Section 2.2). Our discussion is based on direct experience with the European-based airport operator who originated this research and has supported us from the outset. Whilst the organisation chose to remain anonymous, we discuss the implications our results have on its staff scheduling problem in the concluding Section 2.6. The algorithms are presented in detail in Section 2.4 — together with the schedule robustness metrics we employ —, and investigated empirically in Section 2.5. Before all of that, our discussion begins in Section 2.3 by positioning our methodological approach within the state of the art in the area of robust staff scheduling.

## 2.2 Problem Statement

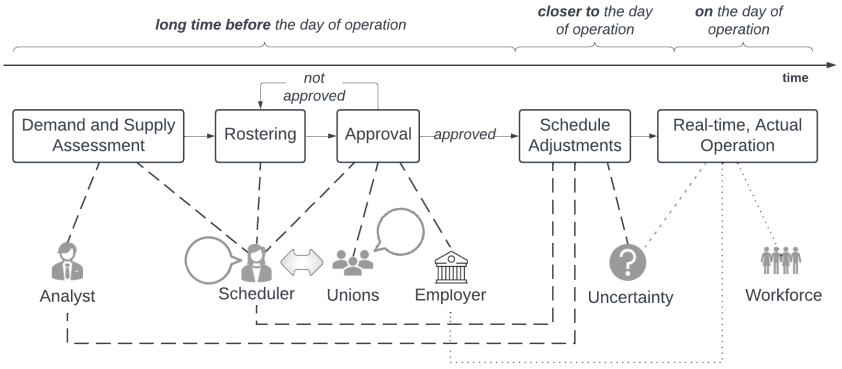



Figure 2.1: Scheduling process with problem stakeholders

In this section, we state the general staff scheduling problem addressed in this paper. We exemplify each element of our problem statement using the situation faced by the airport that motivated our interest in this area, and for which the method discussed in this paper constitutes one of three main components of a solution approach now under testing at the partner organisation.

Let us consider (Figure 2.1) an operation taking place in some well-identified setting. We identify two organisations, one that directly employs the personnel running the operation and another organisation with accountability for the operation and its performance (service level) but no direct control of the primary resources involved in running the operation, nor its real-time, actual running. There are instances where these two organisations are the same but they do not have to be. In the airport context, the

operation is passenger security checks which may be run by the airport operator itself (as in the case of our partner organisation) or subcontracted to a third-party service provider. In both cases, the airport operator is accountable for the operation and its performance.

The idea of fine-tuning flexibilities arises from rostering problems in unionised settings (e.g. airport security). The rostering process in these settings can become extremely drawn out due to the many stakeholders (scheduler, union representatives, business owners) and their many and often differing opinions on what a good roster looks like. As a result, the scheduler often has to leave approval meetings with lists of changes to the roster (a lengthy and difficult process in itself) to complete before the next approval meeting with all stakeholders.

Due to the long time taken for everyone to agree, the final roster will ideally be used for several months to a year. This poses the next problem: because of the length of time the roster will be used for and concerns about fairness between employees, the ideal solution will be a cyclic roster where everyone works the same schedule over the whole planning horizon but starting at the beginning of a different week in the schedule and cycling through the weeks (i.e. starting back at week one once they have reached the end of the schedule). While this is seen as a fairer way to produce rosters, it also poses additional challenges in an environment with uncertain demand patterns. This has to do with the fact that in a cyclic roster, any day of a week in the schedule will have the same number of people for each shift as the same day in the previous week (or the following week). Ultimately, this adds rigidity to a roster that should ideally be flexible to unforeseen changes in the future.

The information available during the rostering process is based on estimates made at the start (demand estimates for the season, expected levels of staff absenteeism). Regardless of how good these forecasts are, the realisations of demand and staff absences will deviate from the estimates made months beforehand (day-to-day uncertainty). Ideally, the scheduler will consider these possible deviations when creating the roster by creating longer shifts and adding more people onto shifts than necessary (surplus shifts). In the case of our partner organisation, so far, this has happened based on the experience of the scheduler, who would assign additional people to each shift based on what has historically worked well.

Wickert et al. (2021) are the first to introduce a formal metric to help quantify the level of surplus personnel (buffer and reserve shifts) during a particular time in the rostering period. However, this approach still requires the scheduler's expertise

to inject the correct level of surplus shifts. Building on this approach, we propose a Simheuristic approach to fine-tune any suitable flexibility metric (or a combination of them) to the data that is available for the planning horizon without relying on the CSS's expertise.

## 2.3 Related Works

Due to the lack of formal robustness metrics in the rostering literature (Wickert et al., 2021), methods for selecting the right robustness metric level for a roster are non-existent in the literature — beyond selecting levels based on experience, past data or to cover forecasts (Wickert et al., 2021; Lusby et al., 2012; Maenhout and Vanhoucke, 2023). The closest existing method in rostering literature to this problem is parameter tuning to enhance the performance of an algorithm that creates a workforce schedule — adding adaptive layers to the algorithm (Algethami and Landa-Silva, 2017; Pereira and Madureira, 2013). However, with more than one metric to consider — as we can have in this work — the situation becomes even more complicated because the different metrics have to be weighed against each other to optimize the final outcome.

Ultimately, we want to select the optimal combination of robustness metrics to minimize the re-rostering cost caused by everyday uncertainty. At the time of rostering, the available data for estimating the effect of everyday uncertainty are demand forecasts and an expected staff absence rate. One way to approach this problem is to classify it as a Simulation-Optimization problem — where we aim to optimize the parameter selection and use Simulation to evaluate the re-rostering cost under everyday uncertainty. Within the taxonomy of Simulation-Optimization problems (Figueira and Almada-Lobo, 2014), our problem falls into the category of "Optimization with Simulation-based iterations". Recent work (Panadero et al., 2024) has shown that simheuristics — the combination of simulation and metaheuristics — are preferable to other approaches (e.g., stochastic programming) when solving large Simulation-Optimization problems such as the one considered in this paper. We refer to Juan et al. (2023) for an overview of simheuristics and their use.

In this paper, we will integrate ALNS (Ropke and Pisinger, 2006), GA (Holland, 1992) and SA (Kirkpatrick et al., 1983) into our Simulation-Optimization approach to fine-tune robustness parameters. All three are popular metaheuristics in the rostering literature (Petropoulos et al., 2023) and have been widely applied to solve a variety of

scheduling problems without the use of Simulation. Examples include the use of GA and SA to solve multi-skilled project scheduling problems (Barghi and Sikari, 2022) and home healthcare network scheduling problems (Goodarzian et al., 2021), SA to solve part-time and mixed-skill worker scheduling problems (Akbari et al., 2013) and no-wait flow-shop scheduling problems (Babor et al., 2021) or ALNS to solve vehicle routing problems (Pisinger and Ropke, 2007).

In most cases in the rostering literature, simulation is integrated into metaheuristic approaches to enhance their effectiveness for solving problems under uncertainty. Once simulation is integrated into the approach, different patterns of combining simulation and metaheuristics arise.

One of the most common ways to combine simulation and metaheuristics is the use of metaheuristics to produce the schedules, which are then tested for feasibility via a simulation (either at each iteration or to select the best solution at the end). The metaheuristic and simulation technique used, as well as the application, vary from paper to paper. Examples include Kuo (2014) who use SA to create new schedules for physicians — the resulting schedules are tested for viability via simulations after each iteration. Zhang et al. (2020) who combine ALNS and Fuzzy simulation to solve vehicle routing problems. ALNS combined with discrete-event simulation has been used to solve both electric vehicle routing (Keskin et al., 2021) and supply vessel routing (Kisialiou et al., 2018, 2019). Schwemmer et al. (2023) use a combination of GA and discrete-event simulation to solve a workforce rostering problem.

Other approaches to combining metaheuristics and simulation include the use of simulation to test the performance of the algorithm (van Twist et al., 2021; Zhang et al., 2018; Hajad et al., 2019), the use of metaheuristics to enhance the simulation model while the simulation is running (e.g. Morgenstern et al. (2020) use metaheuristics to solve an agent positioning problem as part of their simulation model), the use of simulation to create scenarios that the metaheuristic needs to produce schedules for (Nasri et al., 2020) or the use of metaheuristics to select optimal input parameters for the simulation model in order to improve performance measures of the simulated systemHaddock and Mittenthal (1992).

Our approach falls between the last of these combinations, the use of metaheursitics to select parameters for the simulation, and the most common approach of using simulation to test the schedule at each iteration of the metaheuristic At each iteration of our method, the metaheuristic (GA, SA, or ALNS) will select a combination of robustness metrics that will act as input parameters for the existing MIP/CP rostering

technology. So, while the metaheuristic does not create the roster itself, it also does not select the input parameters for the simulation model. In our approach, Monte-Carlo Simulation will be used to calculate the expected re-rostering cost of the resulting roster.

## 2.4 Method

Our Simheuristic approach centres around two main ideas

1. The definition of metrics suitable to the problem objectives
2. The Simheuristic used to fine-tune the metric levels

In addition, we assume that an automatic solution approach (e.g. CP or MIP based) exists that allows for additional constraints to be added.

### 2.4.1 Automatic Solution Approach - Rostering

The Simheuristics and flexibility metrics in this paper are built on the assumption that an automatic solution approach already exists for the problem in question. In the case of our application of rostering in airport security, we are dealing with a cyclic rostering problem that needs to be solved 5-7 months before the day of operations. An overview of the method used for our automatic solution approach is outlined in Figure 2.2. This approach is specifically tailored to the problem faced by our partner airport and is provided for completion of our approach.

It starts with using Passenger arrival data and passenger throughputs of the security hall to create estimates of expected workloads in the form of seasonal demand forecasts (Module A1). To produce cyclic rosters, one model week that represents the typical weekly demand pattern for the season is required. This model week is created in Module A2 with the use of pre-defined aggregation rules (I.e. maximum/average demand for each period of the day). The actual scheduling steps happen in Modules A3-A6, which represents a decomposition approach to tackle the extensive and challenging rostering problem of our partner airport. Each step feeds directly into the next one as the different stages are solved consecutively. First, the approach fixes the rotating schedule (module A3) of assigning shift types (morning, afternoon, night) and days off, taking into account all contractual requirements (such as rules on maximum number of work days per week or weekend off regulations)

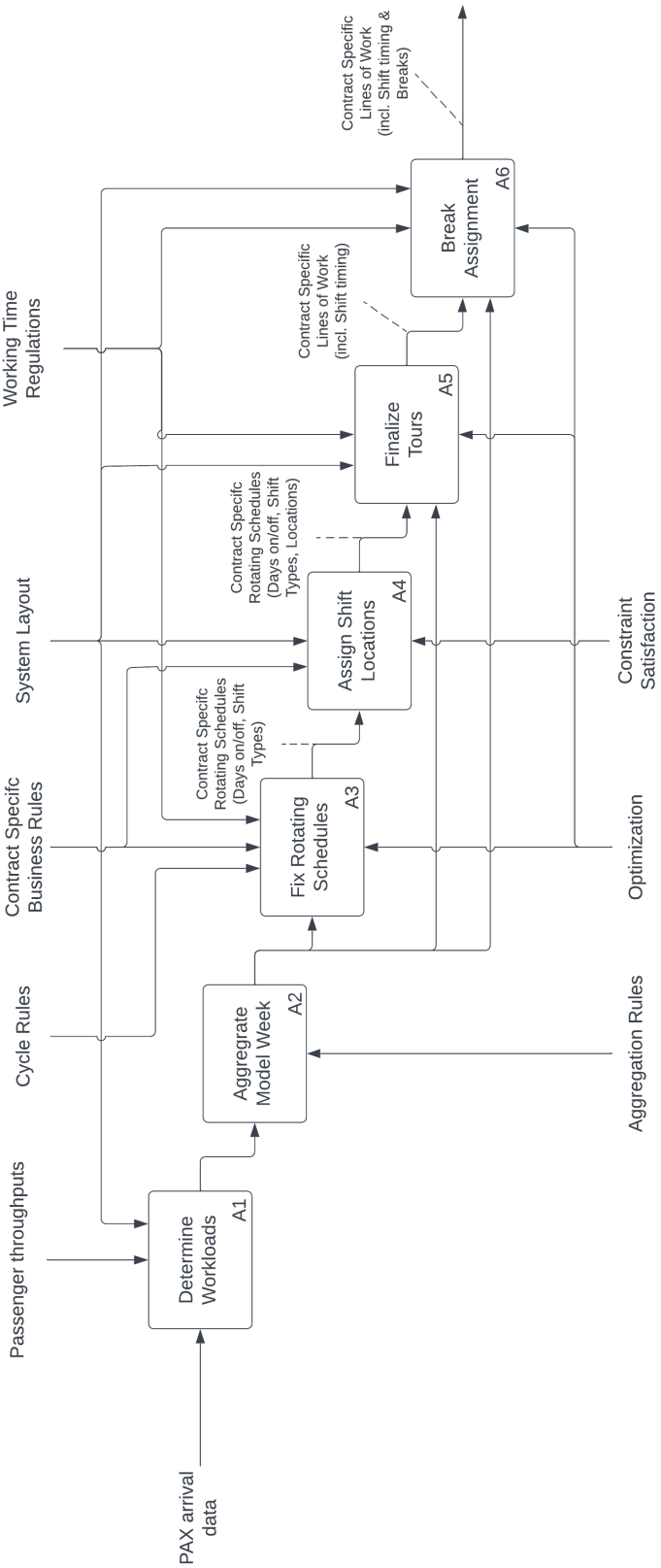



Figure 2.2: IDEF0 diagram of our Automatic Solution Approach.

which also need to hold whenever an employee switches from the last day in the schedule to the first one (cycle rules). The resulting roster is used as a baseline to assign shift locations in Module (A4). Here, the system layout needs to be considered to ensure that all posts are covered while keeping the spread of location assignments as even as possible over the different contracts. In Module A5, the start and end times of shifts are specified (i.e. tours are finalized) while ensuring that no working time regulations are broken. Finally, in Module A6, breaks are allocated to shifts in line with the rules and regulations set by contracts.

### 2.4.2 Metric Definition

We consider two metrics in this paper, the first to deal with demand variations while building the model week for the cyclic roster and the second to measure the level of surplus shifts in the roster.

#### Aggregation Level ($m$)

When creating a model week, we must combine the demand of all weeks in the planning horizon into one single week that best represents the demand forecast. We call the measure used to do this the *aggregation level* ($m$). To build a model week out of a planning horizon of $n$ weeks, we would look at each day (i.e. Monday to Sunday) and possible timeslot (e.g. 15, 30, 60-minute intervals) in the model week separately and then select the $\frac{m}{n}^{\text{th}}$ quantile (of all the demand values that occur in the planning horizon for that day and timeslot). This way, we can exclude the highest peaks in the demand forecast, which may free up parts of the workforce that would be underutilised for the majority of the planning horizon.

#### Surplus Shifts ($\rho$)

Wickert et al. (2021) provide a formal definition of measuring the level of surplus shifts in a roster. Surplus shifts consist of two types of shifts:

1. *Buffer shifts* describe surplus staff who are working alongside staff on regular shifts and can take over if one of the regular staff members does not turn up

2. *Reserve shifts* refer to employees who are considered on-call and only have to come to work if they are specifically asked to do so. Otherwise, they have a day off (which might still be interrupted during the day itself).

However, the metric proposed by Wickert et al. (2021) is based on the assumption that the workforce is split into singular people. We, therefore, extend their formulation to include contracts with teams of different sizes, which allows for a broader usage of the metric.

| Parameter | Definition |
|---|---|
| $nt$ | number of teams (lines) |
| $\mathscr{T}$ | $= \{1, \ldots, nt\}$ set of all teams — teams of the same contract have consecutive indices |
| $cr_i$ | contract of team $i \in \mathscr{T}$ |
| $g$ | line length — most often 7 days (one working week, which is normally the 'planning period' in rostering parlance) |
| $ts_c$ | team size for contract $c \in \mathscr{C}$ |
| $\mathscr{A}_c$ | set of permissible shift types for contract $c \in \mathscr{C}$ — often: morning (M), afternoon (A) and night (N). An additional option for a 'day off'(annotated by $O$) also exists. $\mathscr{A}_c^+$ denotes the set of all permissible shift types for contract $c$ including days off. We also consider $\bigcup_{c=1}^{nc} \mathscr{A}_c = \mathscr{A}$ and $\bigcup_{c=1}^{nc} \mathscr{A}_c^+ = \mathscr{A}^+$. |
| $R$ | requirement matrix — each element $R_{sh,j}$ encodes the workforce demand for shift type $sh \in \mathscr{A}$ on day $j$ where $1 \leq j \leq g$. $R0$ also includes the workforce requirement for days off, which is calculated by subtracting the demands for all shift types of the day from the total size of the workforce. |

| Decision Variable | Definition |
|---|---|
| $S_{i,j}$ | $\in \mathscr{A}^+$, $1 \leq i \leq nt$, $1 \leq j \leq g$ encodes the shift type assigned to line $i$ on day $j$ |

Table 2.1: Parameters and decision variables required for the metric definition of surplus shifts.

Using the problem parameters and decision variables listed in Table 2.1 we can define the ratio of buffer shifts ($buffer_j$) and reserve shifts ($reserve_j$) to regular shifts for any particular day ($j$) in the model week.

$$buffer_j = \sum_{sh \in \mathscr{A}} \sum_{i \in \mathscr{T}} \frac{\left((S_{i,j} = sh) \times ts_{cr_i}\right) - R_{sh,j}}{R_{sh,j}} \tag{2.1}$$

$$reserve_j = \sum_{j=1}^{g} \sum_{i \in \mathscr{T}} \frac{(S_{i,j} = res) \times ts_{cr_i}}{\sum_{sh \in \mathscr{A}} R_{sh,j}} \tag{2.2}$$

The total ratio of surplus shifts to regular shifts is then expressed by the sum of the two.

$$surplus_j = buffer_j + reserve_j \tag{2.3}$$

To ensure the correct level of surplus shifts is enforced in the roster, we add a constraint setting the surplus shift ratio equal to a robustness level ($\rho_j$) while allowing for a small margin of flexibility ($\delta$). This way, the set level of flexibility is ensured while leaving some room to create a feasible roster.

$$\rho_j - \delta \leq surplus_j \leq \rho_j + \delta \tag{2.4}$$

The definition shown by equation (2.4) allows for a different level of flexibility for each day in the model week. If a more general definition is required, we can also define the flexibility levels for the entire week (*surplus*).

$$surplus = \frac{\sum_{j \in \mathscr{D}} surplus_j}{g} \tag{2.5}$$

Which is again set to equal a set robustness level for the week ($\rho$), allowing for a small margin of flexibility ($\delta$).

$$\rho - \delta \leq surplus \leq \rho + \delta \tag{2.6}$$

The aim of the Simheuristic method presented in the following Section (2.4.3) will be to find a suitable combination of those metrics (m,$\rho$) for the rostering problem faced by the scheduler.

### 2.4.3 Simheuristics for Parameter Tuning

Having defined the flexibility metrics for our problem, we can focus on the method for fine-tuning them. All Simheuristics presented in this paper follow the same general structure (Figure 2.3). The first step of simplifying the stochastic rostering problem by formulating and producing an automatic solution approach for the deterministic problem and defining suitable flexibility metrics are summarised in sections 2.4.1 - 2.4.2. Once these steps are completed, the methods follow the same steps of initializing

a random metric combination $(m, \rho)$ and using the automatic solution approach to generate a roster $r$. This roster is tested in an initial shorter Simulation (see section 2.4.3.1 before the approach enters the Simheursitic where a metaheuristic (such as ALNS, SA or GA) will be used to generate a new metric combination $(m, \rho)$. This metric combination is then used to produce a new roster, which is again tested using the shorter simulation. Based on the Simulation results (in our case, the average weekly cost of repairing the roster, i.e. the re-rostering cost), the incumbent metric combination is updated.

The re-rostering cost here only encapsulates the costs directly associated with changing the roster as a result of disruptions (such as staff absences or changes to demand forecasts) — where each action that the scheduler can take to repair the roster will ultimately lead to the accrual of overtime cost (either because a shift takes longer or an officer needs to work a whole new shift). The assumption is that the workforce is salaried with hiring decisions made at a strategic planning level — which is out of scope of this problem. So, all costs incurred due to wages are considered sunk costs at this stage and will not be factored into the decision-making process. Instead, the CSS's aim is to use the available workforce as effectively as possible to minimise overtime costs. The underlying idea here is that the CSS can reallocate the workforce that is freed up by reducing the aggregation level ($m$) to more flexible surplus shifts ($\rho$) and can thus reduce the cost associated with overtime due to calling in additional staff or extending shifts.

These steps are repeated until the stopping condition (e.g. number of iterations, time limit, etc.) has been reached. All visited solutions are then ranked according to their re-roster cost. The best solutions are then tested again with a more extensive Simulation, the results of which are used to re-rank the solutions and select the best metric combinations for the problem.

#### 2.4.3.1 Simulation to Evaluate Re-Rostering Cost

The aim of the Simulation is to replicate the actions the CSS has to take to repair the roster once a new (more up-to-date) demand roster is available for the week and staff absences are better known and to calculate the associated repair cost.

For each of the $R_n$ replications, the method runs through $W_n$ weeks by randomly sampling one weekly demand pattern from the available forecast $\omega_w$ and assigning absences to the workforce $a_w$. The re-roster model inputs these and the existing roster, producing the new repaired roster $r_{new,w}$ and the associated repair cost $c_w$. In the end,

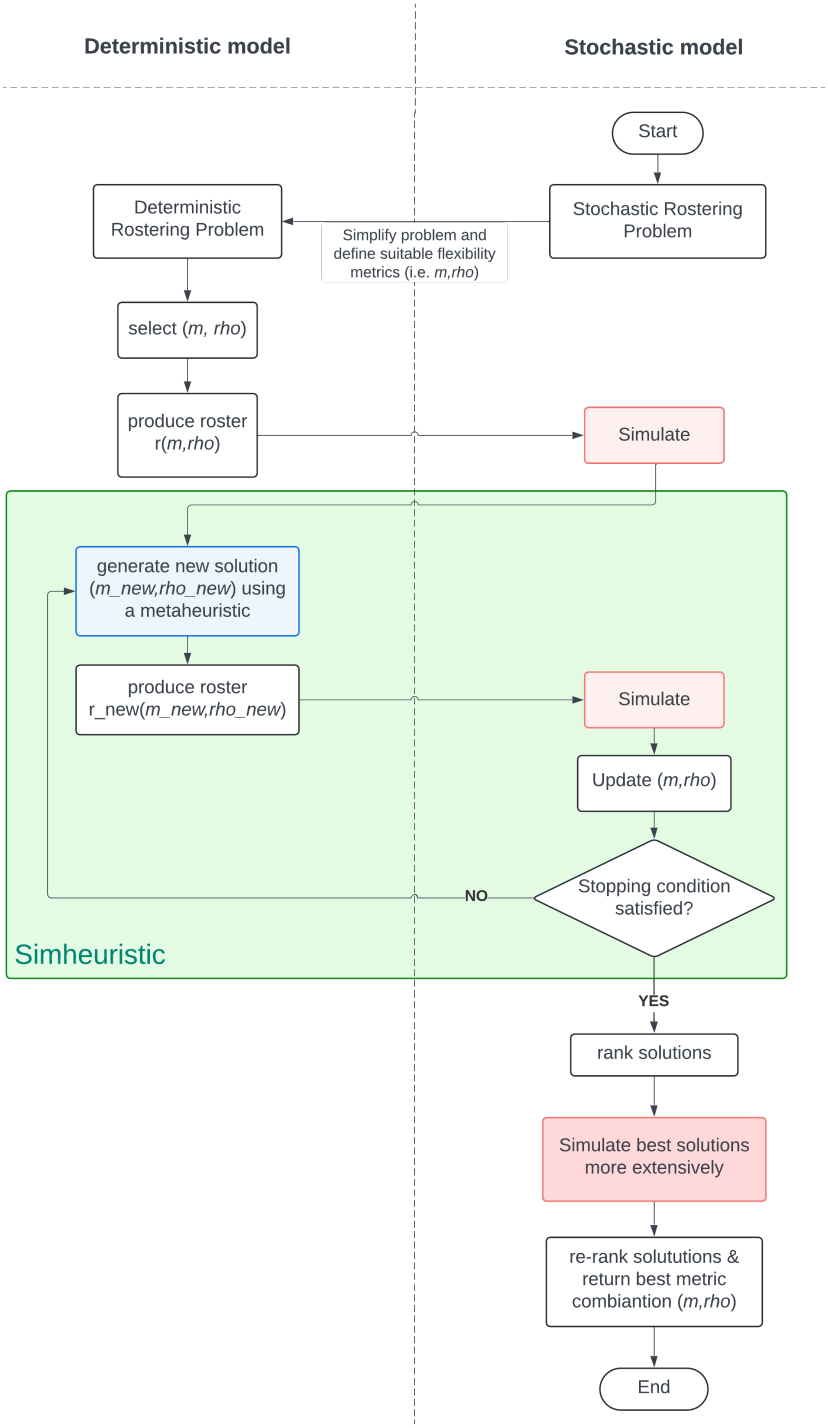



Figure 2.3: Simheuristic scheme for fine-tuning flexibility metrics in rostering.

**Algorithm 1** Simulation to Evaluate Re-Rostering Cost

**Input:** $r, W_n, R_n$
**for** $i \in 1 \dots R_n$ **do**
  **for** $w \in 1 \dots W_n$ **do**
    $a_w \leftarrow$ randomly assign staff absences
    $\omega_w \leftarrow$ random week from forecast
    $(r_{new,w}, c_w) \leftarrow ReRoster(r, a_w, \omega_w)$
  **end for**
  $c_i \leftarrow \frac{\sum_{w=1}^{W_n} c_w}{W_n}$
**end for**
$c_{avg} \leftarrow \frac{\sum_{i=1}^{R_n} c_i}{R_n}$
**Return:** $c_{avg}$

the Simulations return the average re-roster cost ($c_{avg}$) over all replications $R_n$. The metaheuristic later uses the average cost to decide whether the current roster performs better under uncertainty than the previous one.

In Figure 2.3, we refer to two simulations: an initial shorter simulation used during the main Simheuristic approach and a more extensive simulation at the end. Both simulations follow Algorithm 1. The difference is the number of weeks that are sampled. For the shorter simulation, we randomly sample $W_n$ weeks from the demand forecast during each replication. In contrast, for the longer, more extensive simulation, the algorithm runs through all weeks of the demand forecast.

#### 2.4.3.2 Metaheuristic for Parameter Selection

In this paper, we use three metaheuristics (GA, SA and ALNS) to explore different metric combinations (surplus shifts and aggregation level) for our staff scheduling problem.

##### Genetic Algorithm (GA)

The GA in our approach is based on the basic GA introduced by Holland (1992). In the context of fine-tuning flexibility metrics, the genes in GA represent the combinations of metric levels (aggregation level ($m$) and surplus staff ($\rho$)). So starting with a set of genes ($\mathscr{G}$) we consider three options for creating new genes:

1. Mutate the aggregation level $m$ by randomly selecting a new level from the range

of allowed values

2. Mutate the level of surplus staff $\rho$ by randomly selecting a new level from the range of allowed values

3. Create two children by using cross-over on two existing genes in $\mathscr{G}$.

**Algorithm 2** GA for parameter tuning

**for** $i \in 1 \dots N$ **do**
  $g_i \leftarrow (m_i, \rho_i)$ ▷ generate random metric combination
  $r_i \leftarrow Roster(g_i)$
  $c_i \leftarrow Simulation(r_i, W_n, R_n)$
**end for**
$\mathscr{G} \leftarrow \{g_1 \dots g_N\}$
**while** Stopping condition not satisfied **do**
  $\mathscr{G}_{new} \leftarrow New\_Genes(\mathscr{G}, M)$
  **for** $j \in 1 \dots M$ **do**
    $g_j \leftarrow \mathscr{G}_{new,j}$
    $r_j \leftarrow Roster(g_j)$
    $c_j \leftarrow Simulation(r_j, W_n, R_n)$
  **end for**
  $\mathscr{G}_{all} \leftarrow \mathscr{G} \cup \mathscr{G}_{new}$
  $\mathscr{G} \leftarrow \min(\mathscr{G}_{all}, N)$ ▷ keep 10 best genes
**end while**

The method starts with a set of $N$ randomly generated genes ($g_i$), each of which has an associated roster ($r_i$) and associated average re-roster cost ($c_i$). Then, the algorithm loops through the following steps until the stopping condition (i.e. maximum number of iterations, runtime limited, etc.) has been satisfied. Each iteration starts with the generation of a set of new genes ($\mathscr{G}_{new}$) by calling Algorithm 3. Then, for each new gene ($g_j$ the associated roster ($r_{ij}$) and average re-roster cost ($c_j$) are evaluated. Then, the new genes are combined with the existing genes, and the genes with the $N$ lowest re-roster cost are kept for the next iteration.

Algorithm 3 shows the steps to creating a new set of genes $\mathscr{G}_{new}$, where the allowed actions are repeated to create new genes until the desired set size $M$ of new genes has been reached.

**Algorithm 3** New_Genes

**Input:** $\mathscr{G}, M$
$\mathscr{G}_{new} \leftarrow \{\}$
**while** length$(\mathscr{G}_{new}) < M$ **do**
  $o \leftarrow$ *random*(mutate m, mutate $\rho$, create child)
  **if** $o$ = mutate m **then**
    $g_{old} \leftarrow (m_{old}, \rho_{old}) \leftarrow$ random$(\mathscr{G})$
    $g_{new} \leftarrow ($random$(m), \rho_{old})$
    $\mathscr{G}_{new} \leftarrow \mathscr{G}_{new} \cup \{g_{new}\}$
  **else if** $o$ = mutate $\rho$ **then**
    $g_{old} \leftarrow (m_{old}, \rho_{old}) \leftarrow$ random$(\mathscr{G})$
    $g_{new} \leftarrow (m_{old},$ random$(\rho))$
    $\mathscr{G}_{new} \leftarrow \mathscr{G}_{new} \cup \{g_{new}\}$
  **else if** $o$ = create child **then**
    $g_{old,1} \leftarrow (m_{old,1}, \rho_{old,1}) \leftarrow$ random$(\mathscr{G})$
    $g_{old,2} \leftarrow (m_{old,2}, \rho_{old}, 2) \leftarrow$ random$(\mathscr{G})$
    $g_{new,1} \leftarrow (m_{old,1}, \rho_{old,2})$
    $g_{new,2} \leftarrow (m_{old,2}, \rho_{old,1})$
    $\mathscr{G}_{new} \leftarrow \mathscr{G}_{new} \cup \{g_{new,1}, g_{new,2}\}$
  **end if**
  $\mathscr{G}_{new} \leftarrow$ remove all previously visited genes
**end while**
**Return:** $\mathscr{G}_{new}$

### Simulated Annealing (SA)

The SA metaheuristic is modelled on the one provided by Kirkpatrick et al. (1983). The approach starts by randomly selecting a metric combination from the range of allowed demand aggregation and surplus shift values. The associated roster and average re-rostering cost for this starting combination are also evaluated, and the starting temperature is calculated. Then, the method loops through the following steps until the stopping condition (i.e. maximum number of iterations, runtime limited, etc.) has been reached. First, a neighbourhood of the current solution is created by calling Algorithm 5. We define a neighbourhood of the current solution ($m$,$\rho$) as all demand aggregation and surplus shift combinations that are within a range of *s_size* from the current solution if only one metric is different or within a range of *s_half*

each from the current solution if both metrics are different. Next, one of those neighbours is selected randomly, and the associated roster and average re-roster cost are evaluated. If the new solution produces a lower re-roster cost than the current solution, it is automatically accepted as the new incumbent. It will still be accepted with probability “prob" if it is not better. Finally, the current temperature ($T$) is adjusted by the cooling factor ($\alpha$). Throughout the approach, we save all the visited solutions with associated costs in a list to return at the end for further investigation.

**Algorithm 4** SA for parameter tuning

$0 < \alpha < 1$ ▷ cooling factor
$g_{inc} \leftarrow (m_{inc}, \rho_{inc})$ ▷ generate random metric combination
$r_{inc} \leftarrow Roster(g_{inc})$
$c_{inc} \leftarrow Simulation(r_{inc}, W_n, R_n)$
$T \leftarrow \frac{-0.1 c_{inc}}{\log(0.5)}$
**while** stopping condition not satisfied **do**
  $\mathscr{S} \leftarrow Create\_Neighbours$
  $g_{new} \leftarrow (m_{new}, \rho_{new}) \leftarrow random(\mathscr{S})$
  $r_{new} \leftarrow Roster(g_{new})$
  $c_{new} \leftarrow Simulation(r_{new}, W_n, R_n)$
  **if** $c_{new} \leq c_{inc}$ **then**
    $(g_{inc}, m_{inc}, \rho_{inc}) \leftarrow (g_{new}, m_{new}, \rho_{new})$
    $c_{inc} \leftarrow c_{new}$
  **else**
    prob $\leftarrow \exp^{-\frac{c_{new} - c_{inc}}{T}}$
    **if** random$(0,1) <$ prob **then**
      $g_{inc} \leftarrow g_{new}$
      $(m_{inc}, \rho_{inc}) \leftarrow (m_{new}, \rho_{new})$
      $c_{inc} \leftarrow c_{new}$
    **end if**
  **end if**
  $T \leftarrow \alpha T$
**end while**

Algorithm 5 shows the pseudo-code for obtaining the neighbourhood of a metric combination ($m$,$\rho$). It starts with an empty set of neighbours and then adds all the neighbours of $(m, \rho)$ by first varying $m$ from $m - s\_size$ to $m + s\_size$ and selecting all

combinations with $\rho$ that are not equal to the initial solution or have an $m$ outside the allowed range. The same is repeated by varying $\rho$. Finally, we vary both $m$ and *rho* by $\pm s\_half$ at the same time and add all combinations that are within allowed ranges to the set of neighbours. Each time a neighbour is added, we use the union of existing neighbors and the new combination to avoid replications of the same solution.

**Algorithm 5** Create_Neighbours

**Input:** $m, \rho, s\_size$
$\mathscr{S} \leftarrow \{\}$ ▷ set of neighbours
$s\_half \leftarrow floor(\frac{s\_size}{2})$
**for** $s \in m - s\_size \ldots m + s\_size)$ **do**
  **if** $s \neq m$ & $m_{\min} \leq s \leq m_{\max}$ **then**
    $\mathscr{S} \leftarrow \mathscr{S} \cup \{(s, \rho)\}$
  **end if**
**end for**
**for** $r \in \rho - s\_size \ldots \rho + s\_size$ **do**
  **if** $r \neq \rho$ & $\rho_{\min} \leq r \leq \rho_{\max}$ **then**
    $\mathscr{S} \leftarrow \mathscr{S} \cup \{(m, r)\}$
  **end if**
**end for**
**for** $s \in m - s\_half \ldots m + s\_half)$ **do**
  **if** $s \neq m$ & $m_{\min} \leq s \leq m_{\max}$ **then**
    **for** $r \in \rho - s\_half \ldots \rho + s\_half$ **do**
      **if** $r \neq \rho$ & $\rho_{\min} \leq r \leq \rho_{\max}$ **then**
        $\mathscr{S} \leftarrow \mathscr{S} \cup \{(s, r)\}$
      **end if**
    **end for**
  **end if**
**end for**
**Return:** $\mathscr{S}$

### Adaptive Large Neighborhood Search (ALNS)

Our ALNS for parameter tuning is based on the initial design of ALNS by Ropke and Pisinger (2006). At each iteration of the algorithm, we use a set of operators — that are

designed to suit the particular problem we are dealing with— to create a new solution to compare to the old one.

Before the ALNS algorithm can start to run, a set of parameters need to be initialized (Algorithm 6). These are inherent to the ALNS procedure as presented by Ropke and Pisinger (2006) and can be adjusted to suit the problem setting. This is followed by the generation of a random metric combination $(m_{inc}, \rho_{inc})$, which are used to create an initial roster $(r_{inc})$ and initial re-rostering cost $(c_{inc})$. At the start these initial values are also automatically set to be the best solutions found $((m_{best}, \rho_{best}, r_{best}, c_{best})$. Finally, the starting Temperature $(T)$ is defined.

**Algorithm 6** ALNS initialization

| | |
|---|---|
| $\sigma_a, \sigma_b, \sigma_c$ | ▷ initialize costs for updating weights |
| $\alpha$ | ▷ cooling factor |
| $\beta$ | ▷ operator weight update ratio |
| $o$ | ▷ intialize number of operators |
| $\theta \leftarrow \theta_1 \dots \theta_o$ | ▷ operator usage tracker |
| $\pi \leftarrow \pi_1 \dots \pi_o$ | ▷ operator effectiveness tracker |
| $\omega \leftarrow \omega_1 \dots \omega_o$ | ▷ operator weights |
| $\gamma \leftarrow \gamma_1 \dots \gamma_o$ | ▷ operator probabilities |
| $(m_{inc}, \rho_{inc})$ | ▷ random metric combination |
| $r_{inc} \leftarrow Roster(g_{inc})$ | |
| $c_{inc} \leftarrow Simulation(r_{inc}, W_n, R_n)$ | |
| $(m_{best}, \rho_{best}) \leftarrow (m_{inc}, \rho_{inc})$ | |
| $r_{best}, c_{best} \leftarrow r_{inc}, c_{inc}$ | |
| $T \leftarrow \frac{-0.1 c_{inc}}{\log(0.5)}$ | |

Using the values that were initialized in Algorithm 6 the ALNS procedure works as follows (see Algorithm 7).

For a pre-defined (problem-specific) number of segments, we run through a "segment length" a number of iterations where first Algorithm 8 is called to produce a new solution $(m_{new}, \rho_{new})$. The associated operator tracker $(\theta_o)$ is updated and the roster $(r_{new})$ and re-roster cost $(c_{new})$ are evaluated. Then the method first tests if the new solution produces a lower cost than the current incumbent, in which case both the incumbent $(m_{inc}, \rho_{inc})$ and the best solution $(m_{best}, \rho_{best})$ are updated to be equal to the new solution and the operator effectiveness tracker $(pi_{\text{op}})$ for the operator (op) is increased by $\sigma_a$. If, instead, the new solution is only better than the incumbent

**Algorithm 7** ALNS for parameter tuning

**while** number of desired segments not reached **do**
  **for** $s \in 1 \ldots$ segment length **do**
    $m_{new}, \rho_{new}, \text{op} \leftarrow New\_Solution($
      $m_{inc}, \rho_{inc}, s\_size, \gamma, \theta)$
    $\theta_{\text{op}} \leftarrow \theta_{\text{op}} + 1$
    $r_{new} \leftarrow Roster(m_{new}, \rho_{new})$
    $c_{new} \leftarrow Simulation(r_{new}, W_n, R_n)$
    **if** $c_{new} \leq c_{best}$ **then**
      $(m_{best}, \rho_{best}) \leftarrow (m_{new}, \rho_{new})$
      $r_{best}, c_{best} \leftarrow r_{new}, c_{new}$
      $(m_{inc}, \rho_{inc}) \leftarrow (m_{new}, \rho_{new})$
      $r_{inc}, c_{inc} \leftarrow r_{new}, c_{new}$
      $\pi_{\text{op}} \leftarrow \pi_{\text{op}} + \sigma_a$
    **else if** $c_{new} \leq c_{inc}$ **then**
      $(m_{inc}, \rho_{inc}) \leftarrow (m_{new}, \rho_{new})$
      $r_{inc}, c_{inc} \leftarrow r_{new}, c_{new}$
      $\pi_{\text{op}} \leftarrow \pi_{\text{op}} + \sigma_b$
    **else**
      $\text{prob} \leftarrow \exp^{-\frac{c_{new} - c_{inc}}{T}}$
      **if** $\text{random}(0,1) < \text{prob}$ **then**
        $(m_{inc}, \rho_{inc}) \leftarrow (m_{new}, \rho_{new})$
        $r_{inc}, c_{inc} \leftarrow r_{new}, c_{new}$
        $\pi_{\text{op}} \leftarrow \pi_{\text{op}} + \sigma_c$
      **end if**
    **end if**
    $T \leftarrow T \times \alpha$
  **end for**
  **for** $i \in 1 \ldots o$ **do**
    **if** $\theta_i > 0$ **then**
      $\omega_i = \omega_i(1-\beta) + \beta \frac{\pi_i}{\theta_i}$
    **end if**
    $\pi_i, \theta_i \leftarrow 0$
  **end for**
  $\gamma_i \leftarrow \frac{\omega_i}{\sum_{j=1}^{o} \omega_j} \qquad \forall i \in 1 \ldots o$
  $T \leftarrow \frac{-0.1 c_{best}}{\log(0.5)}$
**end while**

$(m_{inc}, \rho_{inc})$, then only the incumbent solution is updated to be the best, and $(pi_{\text{op}})$ is increased by $\sigma_b$. Finally, if the new solution is not better than the best or the incumbent solution, it will still be accepted with probability $\exp^{-\frac{c_{new}-c_{inc}}{T}}$. In that case $(pi_{\text{op}})$ is increased by $\sigma_c$. At the end of each iteration, the Temperature is updated by the cooling factor $(\alpha)$. After each segment, the operator weights $(\omega)$ and the operator probabilities $(\gamma)$ are updated, and the operator usage and effectiveness trackers are set to zero. Finally, the temperature is reset based on the current best solution before the next segment starts again. While running, the algorithm stores all newly created solutions and associated costs in a visited solutions list so that the results can be further analysed, for example, by running more extensive simulations on the top solutions.

Algorithm 8 shows the method for creating new solutions as part of our SimALNS procedure. There are nine different operators that either increase, decrease or randomly select $m$ and $\rho$ or a combination of both. The operator is randomly selected based on weights $\gamma$ that are adjusted based on the operator's effectiveness in finding a better solution. After producing a new solution, the method returns both the solution and the operator to the main method (Algorithm 7).

## 2.5 Computational Experiments

As part of the experiments, we use data inspired by a real airport security rostering problem. The first subsection (Section 2.5.1) gives a closer overview of the problem details that are included in our models. All experiments were carried out on an iMac running macOS Sonoma Version 14.0 with a 3.1 GHz 6-Core Intel i5 ($10^{\text{th}}$ generation) processor and 8GB memory. The algorithms were implemented using Python 3.8, MiniZinc version 2.7.6 (Nethercote et al., 2007) and Gurobi version 9.1.2 (Gurobi, 2024).

### 2.5.1 Rostering in Airport Security

The airport's rostering process starts with the accumulation of all contractual details of the workforce, which has been hired at this point and cannot be changed for the upcoming season — hiring additional staff is not an option in this problem. The workforce in our example is considered homogeneous with regards to tasks that have to be performed on shift and consists of multiple teams that are grouped into different

**Algorithm 8** New_Solution

**Input:** $m, \rho, s\_size, \gamma$,

op $\leftarrow$ random$\big((m_-, m_+, m_{rand}, \rho_-, \rho_+, \rho_{rand},$
$(m_-, \rho_+), (m_+, \rho_-), (m_{rand}, \rho_{rand}))$
$, \text{weights} = \gamma\big)$

**if** op = $m_-$ **then**
  $m_{new} \leftarrow \max\{m_{\min}, m - s\_size\}$
  $\rho_{new} \leftarrow \rho$
**else if** op = $m_+$ **then**
  $m_{new} \leftarrow \min\{m_{\max}, m + s\_size\}$
  $\rho_{new} \leftarrow \rho$
**else if** op = $m_{rand}$ **then**
  $m_{new} \leftarrow \text{random}(m_{\min}, m_{\max})$
  $\rho_{new} \leftarrow \rho$
**else if** op = $\rho_-$ **then**
  $\rho_{new} \leftarrow \max\{\rho_{\min}, \rho - s\_size\}$
  $n_{new} \leftarrow m$
**else if** op = $\rho_+$ **then**
  $\rho_{new} \leftarrow \min\{\rho_{\max}, \rho + s\_size\}$
  $n_{new} \leftarrow m$
**else if** op = $\rho_{rand}$ **then**
  $\rho_{new} \leftarrow \text{random}(\rho_{\min}, \rho_{\max})$
  $n_{new} \leftarrow m$
**else if** op = $(m_-, \rho_+)$ **then**
  $m_{new} \leftarrow \max\{m_{\min}, m - s\_size\}$
  $\rho_{new} \leftarrow \min\{\rho_{\max}, \rho + s\_size\}$
**else if** op = $(m_+, \rho_-)$ **then**
  $m_{new} \leftarrow \min\{m_{\max}, m + s\_size\}$
  $\rho_{new} \leftarrow \max\{\rho_{\min}, \rho - s\_size\}$
**else if** op = $(m_{rand}, \rho_{rand})$ **then**
  $m_{new} \leftarrow \text{random}(m_{\min}, m_{\max})$
  $\rho_{new} \leftarrow \text{random}(\rho_{\min}, \rho_{\max})$
**end if**

**Return:** $(m_{new}, \rho_{new})$, op

contracts. These contracts differ in the allowed weekly work hours and permissible shift types (morning, afternoon, night) in the 24-hour operation. The teams of each contract work a cyclic roster and there are additional restrictions on average weekly work hours over the cycle. It is important to note that the entire workforce has contracts with fixed weekly hours that are paid regardless of whether the full hours were worked or not. This means that wages are considered a *sunk cost* in this setting and one of the main goals of the scheduler is to use the available work hours as efficiently as possible to minimise overtime pay.

In addition to the contractual details, a demand forecast for the upcoming 5-7 month season — the period in which the schedule will be used — needs to be obtained. This information is then passed on to the scheduler, who produces a roster, which is put through the approval process. During this process, the scheduler, the unions, and the airport operator discuss the roster and potential changes. The outcome of this meeting is either “approval" of the roster or “non-approval". The latter means the scheduler has to go back to the drawing board and work all the suggestions from the airport operator and unions into the schedule before it is sent through another round of approval. All in all, this process can take several weeks, if not months, to complete.

Given this long process, the approved schedule is based on forecasting information acquired months before the day of operation. As time moves closer to the day of operation, everyday uncertainty will cause small schedule disruptions that must be rectified — if possible — to allow for smooth operation. The scheduler has 3 types of actions (with increasing associated cost) they can take to address the disruptions:

1. Extend shifts
2. Change the shift location/post
3. Call in additional staff that was scheduled to have a day off

In addition, there is a cut-off (7-10 days) before the day of operation until which the scheduler can make final adjustments without automatically incurring overtime pay to all schedule adjustments (even if the additional work hours are within the contracted weekly limit) or needing to seek the affected staff members’ permission to make the changes. We assume the re-rostering step will occur just before this cut-off point for our experiments.

For the remainder of this section, we will first discuss the Simulation Model settings, then we will analyse the effect of varying the aggregation level ($m$) alone

(Section 2.5.3) before comparing the performance of the different Simheuristics (Section 2.5.4) and analysing the impact of surplus shifts ($\rho$) on a weekly (Section 2.5.5) and daily (Section 2.5.6) level.

### 2.5.2 Simulation Model

For the purpose of these experiments, we define two simulations: a *short simulation* and a *long simulation*. They both follow the method outlined in Algorithm 1 with the only difference being the number of weeks sampled during the procedure. Sampling a week of demand patterns and absences and running the re-roster model can take several minutes, which makes running through the entire demand forecast of 29 weeks (as is done in the *long simuation*) infeasible for each iteration of the Simheuristics — especially when the simulation is replicated several times. We, therefore, use a *short simulation* where only ten weeks are sampled instead and only run the *long simulation* for the ten best results found by each of the Simheuristics.

For both the long and short simulations, we opted for five replications of the simulation, which provides a relative error of less than 0.01 with confidence level $\alpha = 0.99$ (Law, 2015, Chapter 9).

### 2.5.3 Demand Aggregation Impact

One of the metrics we consider in this paper is the demand aggregation level ($m$). To see how varying this metric alone can affect the average weekly re-rostering cost for the roster, we varied the aggregation level from 51% to 100% and ran five replications of the simulation each.

Figure 2.4 gives an overview of the average weekly re-rostering cost and the three components (overtime cost, absences, calling in people who are “off") that contribute to the cost. Looking at the average weekly re-rostering cost image, we can see a clear pattern of the cost gradually decreasing as the aggregation level increases. This makes sense in the context because accounting for higher variation in the demand patterns results in the scheduling of more and longer shifts to cover the demand fluctuations. So when demand patterns with higher fluctuations occur, the roster is already set up to deal with them. Therefore, there are fewer costs associated with covering higher-than-expected demand.

Now, looking at the cost contributors, we can first of all see that the average weekly overtime cost does not significantly change as the aggregation level increases. This

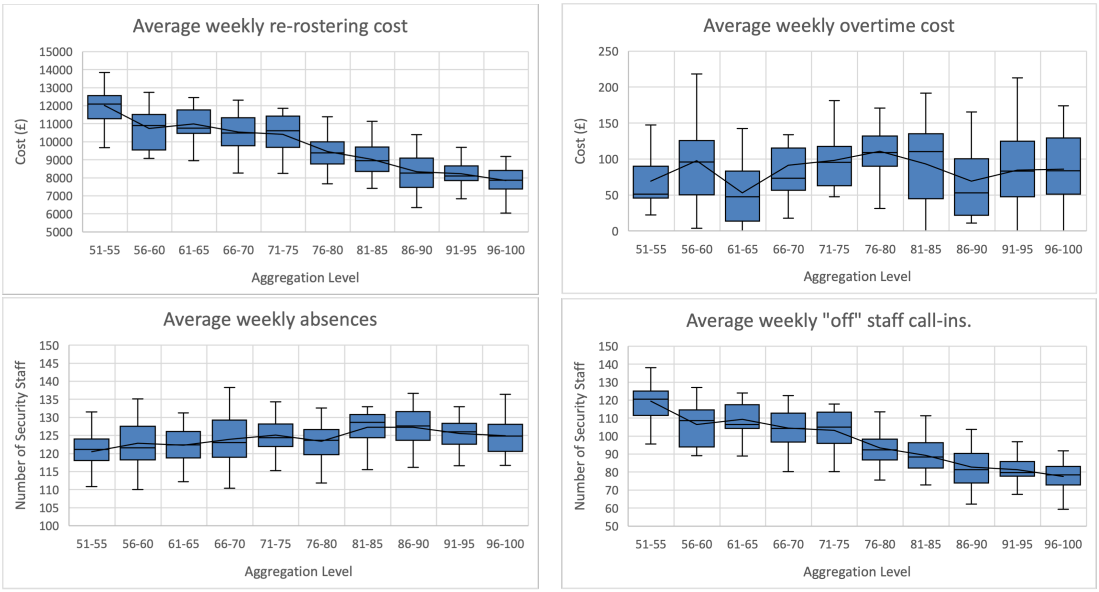



Figure 2.4: Average weekly measures for different aggregation levels ($m$).

is likely because even though more shifts are scheduled initially, they might not be distributed evenly to allow for extension (without overtime) at the re-rostering point. This suggests that regardless of how much possible variation is accounted for, there will always be the need to extend shifts to adapt to the new shift patterns. This is important for schedulers to consider when initially setting up the schedules, so they can build some slack into the roster to ensure that staff members do not enter overtime.

The average weekly number of absences increases slightly as the aggregation level increases. Given that a higher aggregation level coincides with more scheduled shifts (and thus more staff members working) this result is unsurprising. Similarly, the average number of weekly staff call-ins from their days off decreases for higher values of $m$. As expected, accounting for more demand variation in the initial roster leads to fewer situations where covering demand requires calling in additional staff. Even though this measure decreases significantly, we wouldn't expect to see zero here, as there can always be instances where critical members of staff fall ill and need to be replaced for the day.

In summary, the re-rostering cost at this stage is largely driven by the cost of calling in staff who are scheduled to be on their day off. This suggests that the introduction of surplus shifts should help further decrease the average weekly re-rostering cost.

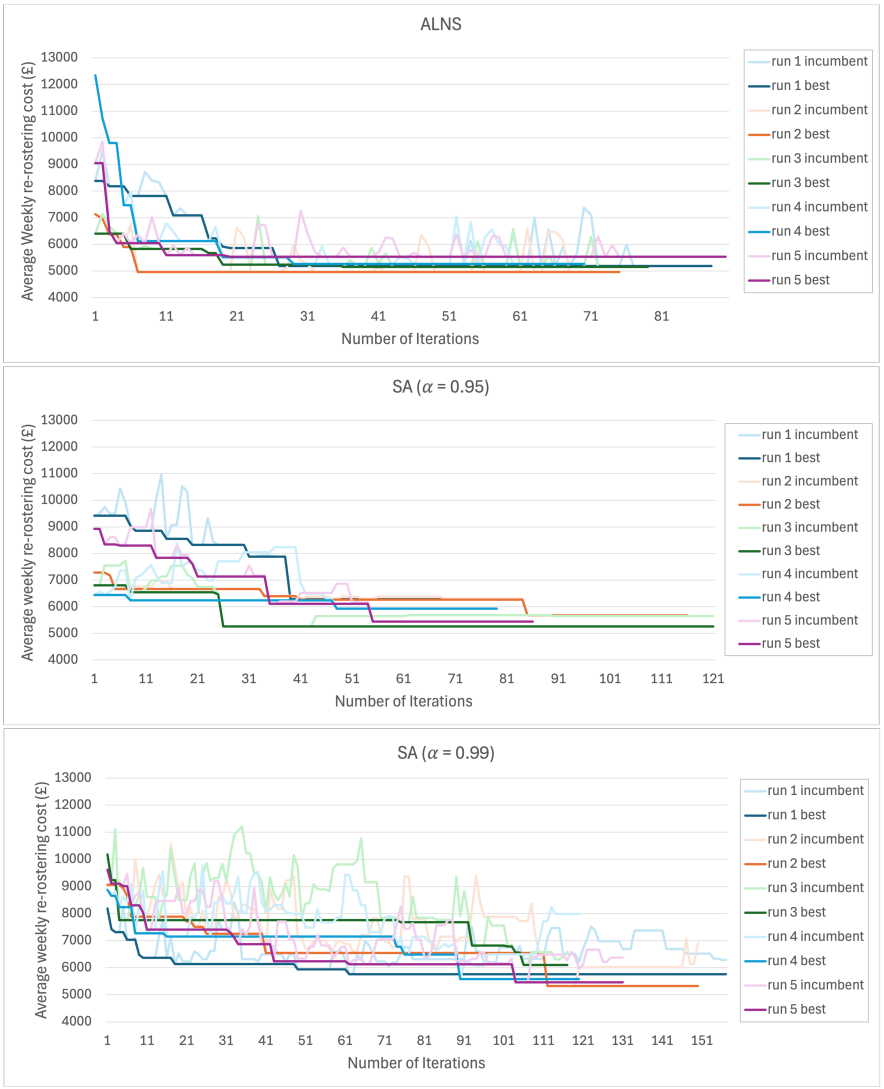



Figure 2.5: Best and incumbent cost progression for 5 runs of ALNS and SA.

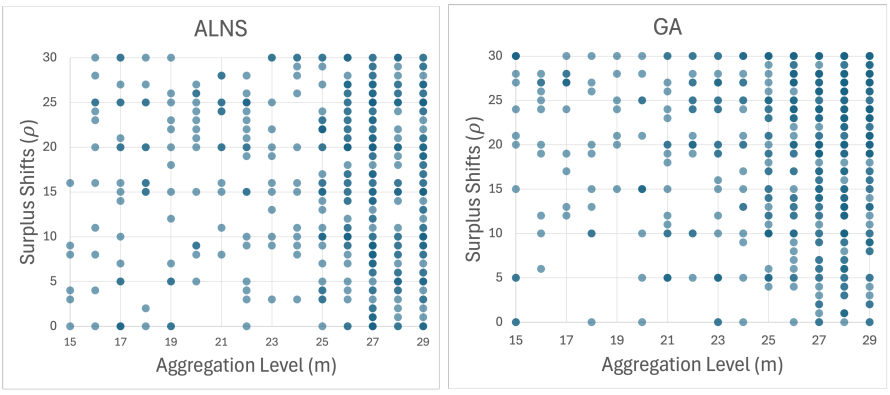



Figure 2.6: Overall spread of visited metric combinations for five runs of ALNS and GA

### 2.5.4 Comparing Simheuristics

Using these parameter settings, we ran all three Simheuristics five times to obtain a list of each run's ten best metric combinations (i.e., metric combinations with the lowest associated cost). Figure 2.5 gives an overview of how the best and incumbent solutions change as the algorithms run through the iterations.

With ALNS, we can see that the best solution quickly decreases for the first 30 iterations and stays at the same level for the remainder of the runtime. At the same time, the spikes in the incumbent solution indicate that the algorithm keeps trying to escape the potentially local minimum by accepting suboptimal solutions.

For SA with a cooling factor of $\alpha = 0.95$, we can also see a gradual decrease of the best solutions for each run, with the best solutions stabilizing after 60 iterations for most runs. However, the incumbent values do not spike at all after the best solutions have stabilised, suggesting that the cooling factor is, in fact, too low here. Increasing the cooling factor to $\alpha = 0.99$ rectifies this issue, as can be seen from the image for SA with $\alpha = 0.99$. Here, the incumbent solutions keep spiking to try and escape local minima, and the best solution keeps decreasing as the iterations go on.

Now, looking at the spread of the visited metric combinations over the entire runtime of the Simheuristics, we can see that for both ALNS and GA, the spread is heavily skewed towards the top left corner of the graph. This corner represents the combinations of high value of surplus shifts ($\rho$) and high levels of demand aggregation ($m$). This suggests that both algorithms quickly converge to those values and then explore the surrounding combinations to try and improve the best solution. Considering the implications to average weekly re-rostering cost, this area should be associated with the lowest cost values, as we simultaneously add additional staff members to the roster and account for the highest variations in demand.

For SA the pattern is different. This becomes even more apparent when separating the visited solutions by runs (See Figure 2.7). Looking first at the results for SA with ($\alpha = 0.95$), it is clear that the algorithm does not explore the entire solution space and stays around the starting point instead. This behaviour is more strongly exhibited for surplus shift values as well, which could be due to the larger range of possible values (0-30 for surplus shifts compared to 15-29 for aggregation level). Increasing the cooling factor ($\alpha$) to 0.99 improves this behaviour somewhat. However, the algorithm still does not explore the full solutions space (in particular, high values of $\rho$, which in theory should be associated with lower re-rostering costs.

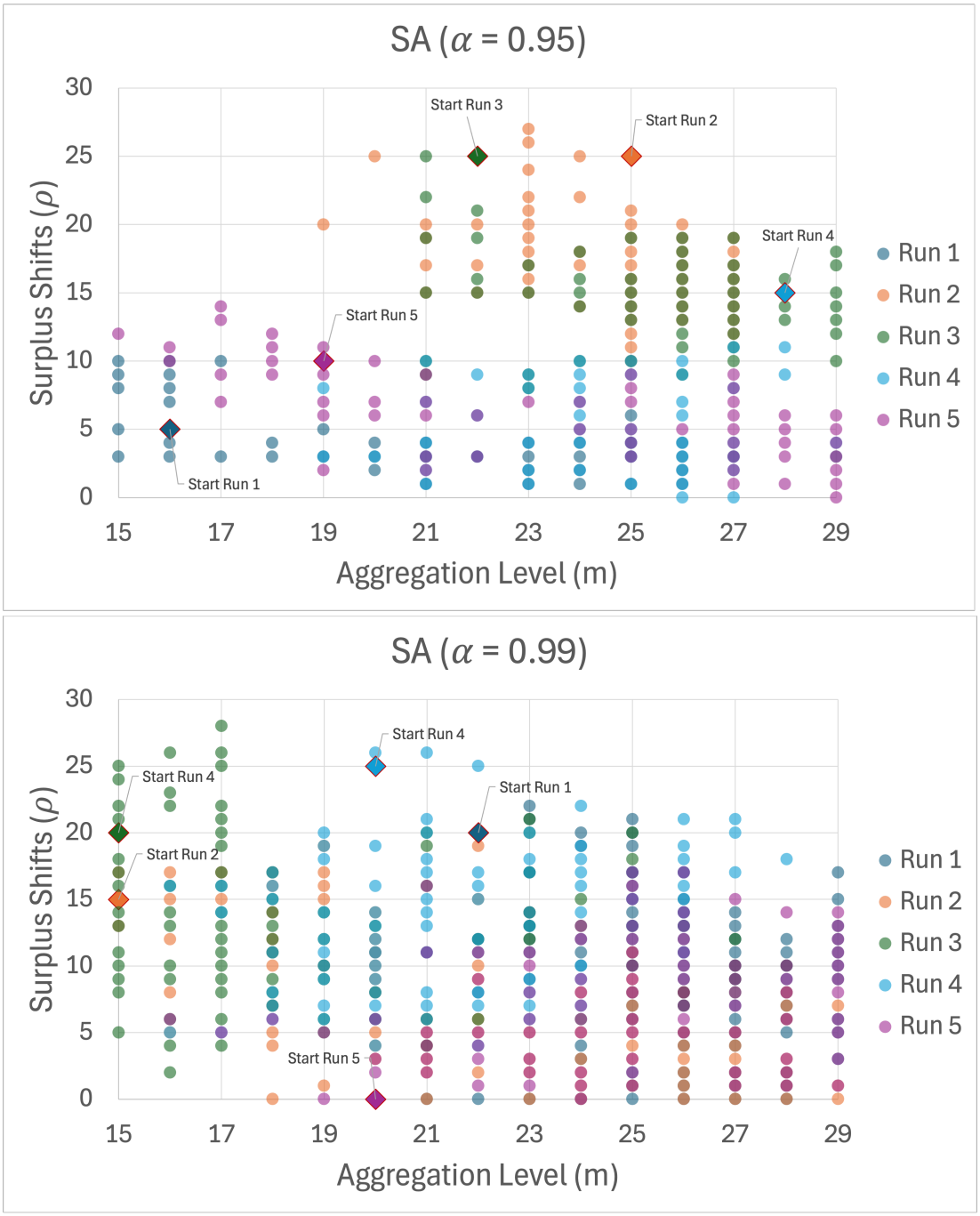



Figure 2.7: Spread of visited metric combinations by run for SA with $\alpha = 0.95$ and $\alpha = 0.99$.

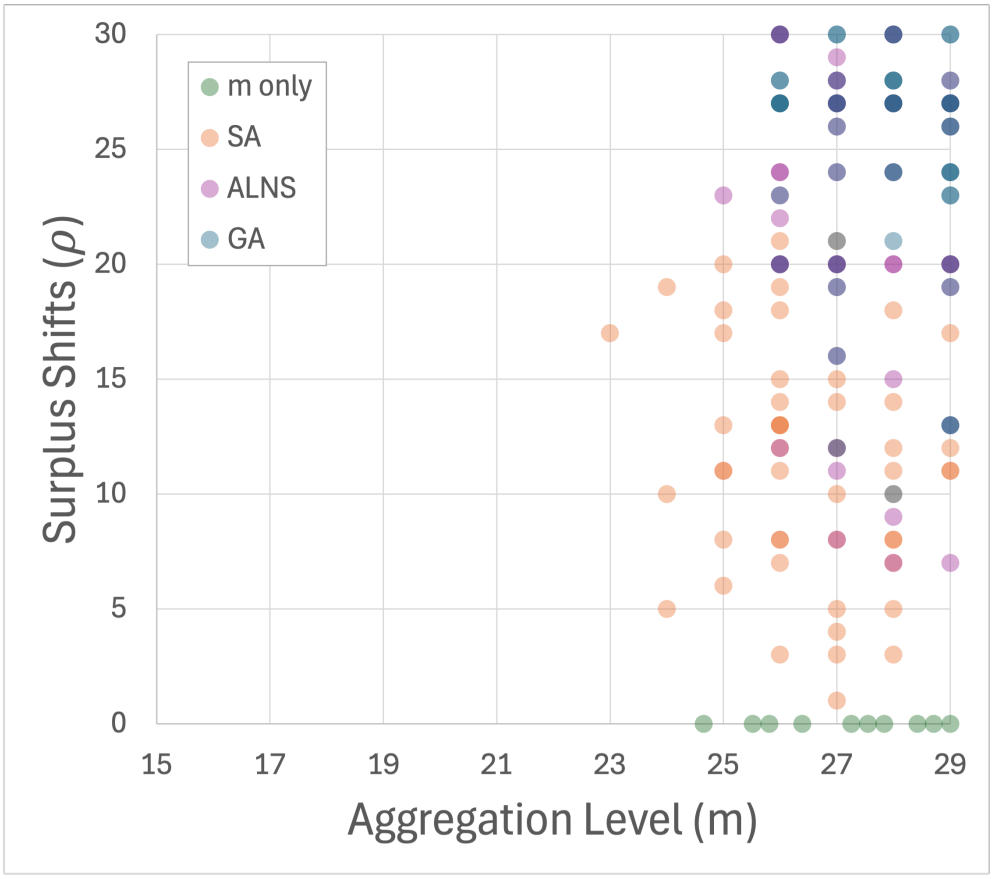



Figure 2.8: Ten best metric combinations for five runs of each Simheuristic and applying $m$ only.

### 2.5.5 Weekly Surplus Shifts

To analyse the impact of introducing a weekly surplus shift metric ($\rho$) we will compare the ten best solutions of the five runs of ALNS, GA and SA with the best solutions of just using the demand aggregation level ($m$).

Figure 2.8 shows the best metric combinations for all four approaches. It clearly shows that the best-performing demand aggregation levels can all be found at the high end of the range of allowed values — with all values being 26 and higher. For the surplus shift level, we can observe a much wider spread over all possible values. In particular, the results from SA are much lower $\rho$ values compared to ALNS and GA, which are mostly grouped in the top right corner of the diagram (i.e. high $m$ and $\rho$ values).

Now, looking at the average weekly re-roster costs over the best solutions for all four approaches (Table 2.2), we can see that out of the three Simheuristic approaches, SA has the highest associated weekly re-rostering cost. In addition, GA, which has

| Approach | weekly re-roster cost |
|---|---|
| $m$ only | £7923.29 |
| SA | £6754.94 |
| ALNS | £6133.84 |
| GA | £6100.96 |

Table 2.2: Average of best average weekly re-roster cost from the best solutions of all four approaches.

the highest $\rho$ values in the set of best metric combinations, also has the lowest weekly re-roster cost. This supports our initial assumption that the lowest weekly re-rostering costs can be obtained by combining high $\rho$ and $m$ values. Finally, the table also shows that just using the demand aggregation level $m$ produces the highest weekly re-rostering out of all four approaches. To see if these differences are significant, we performed t-tests on the sets of weekly re-rostering costs (Table 2.3).

| Sample 1 | Sample 2 | p-value |
|---|---|---|
| $m$ | ALNS | 5.33e−13 |
| $m$ | GA | 3.19e−12 |
| $m$ | SA | 4.59e−10 |
| ALNS | GA | 0.68 |
| ALNS | SA | 5.92e−10 |
| GA | SA | 1.23e−11 |

Table 2.3: p-values for t-tests on best weekly re-rostering costs of ALNS, GA, SA and $m$ only

The t-tests show that all three Simheuristics produce significantly better results than enforcing the demand aggregation level ($m$) alone. In addition, we can see that both ALNS and GA produce significantly better results than SA. However, even though GA seems to produce metric combinations with a lower average weekly re-rostering cost, they are not significantly better than the results produced by ALNS. In summary, we can say that enforcing levels of surplus shifts is always better than not using them, and both GA and ALNS perform the best at exploring the solution space and finding the metric combinations with the lowest associated re-rostering cost.

### 2.5.6 Daily Surplus Shifts

| | $m$ | $\rho$ | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Rank | | Mon | Tue | Wed | Thur | Fri | Sat | Sun |
| 1 | 26 | 11 | 5 | 5 | 10 | 15 | 20 | 25 |
| 2 | 25 | 12 | 8 | 26 | 14 | 17 | 22 | 6 |
| 3 | 25 | 12 | 8 | 26 | 13 | 10 | 22 | 15 |
| 4 | 26 | 11 | 5 | 6 | 10 | 15 | 20 | 25 |
| 5 | 25 | 12 | 8 | 26 | 12 | 17 | 22 | 6 |
| 6 | 24 | 12 | 8 | 25 | 12 | 16 | 22 | 21 |
| 7 | 24 | 12 | 8 | 26 | 13 | 23 | 22 | 10 |
| 8 | 25 | 12 | 8 | 26 | 13 | 17 | 22 | 6 |
| 9 | 24 | 11 | 8 | 26 | 13 | 17 | 22 | 21 |
| 10 | 24 | 12 | 8 | 26 | 6 | 17 | 22 | 21 |
| Average | | 11.7 | 7.4 | 21.8 | 11.6 | 16.4 | 21.6 | 15.6 |
| Rank | | 6 | 7 | 1 | 5 | 3 | 2 | 4 |

Table 2.4: The ten best metric combinations for weekly surplus shift levels with the daily average $\rho$ level and the associated rank.

To analyse the impact of daily surplus shift measures compared to weekly surplus shift measures, we adapted the ALNS based Simheuristic to allow for daily surplus shift measures. For this experiment, both GA and ALNS were viable adaptation options. However, ALNS only runs one simulation per iteration (rather than one for each new child created in GA), which makes it more responsive to stopping conditions and, therefore, easier to work with in this setting. For the daily surplus shift levels, the ALNS operators were adapted to apply to single weekdays only. In addition, we also added operators that would increase/decrease all surplus shift levels for the week. All other aspects of the Simheursitic remain unchanged.

Looking at the ten best metric combinations for daily surplus shift levels, we can see a striking difference between certain days of the week (Table 2.4). For example, Tuesday is consistently allocated a low level of surplus shifts, whereas Wednesday and Saturday are allocated high levels of surplus shifts for most results. This suggests that there might be differences in the demand variations for the different weekdays.

To see if this is the case, we need to look closer at the lane demand in the forecast. Figure 2.9 shows the week-by-week progression of the total daily lane demand over all

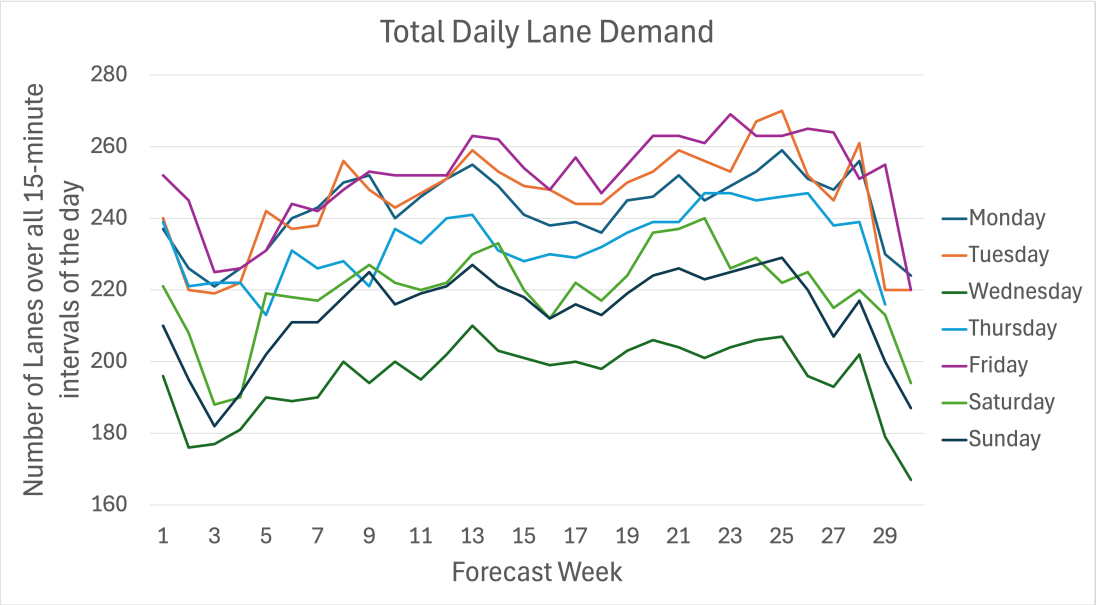



Figure 2.9: Week-by-week progression of total daily lane demand over all 15-minute intervals by weekday.

15-minute intervals of the day. It shows a clear difference in the total daily lane demand for the different days of the week. Interestingly, the Spearman's rank correlation coefficient between the total daily lane demand and the daily surplus shift metrics suggests an inverse relationship between lane demand and assigned surplus shift level for the day. This means a scheduler is better off assigning surplus shifts to days with lower average demand. A possible reason for this could be that days with high overall demand can better absorb additional demand spikes since the maximum number of lines is already open, so in the worst case, the already existing shifts just need to be extended. Demand spikes on low-demand days, on the other hand, could mean that additional lanes need to be opened, which requires additional levels of staff that won't be scheduled to work unless there are additional levels of surplus shifts in the roster.

Finally, comparing the average weekly re-rostering costs for daily and weekly surplus shift levels ($\rho$), we can see from Table 2.5 that using the daily metric could lead to a more than £1,400 reduction in weekly re-rostering cost. Performing a t-test on these results suggests that the difference between the two metrics is significant with a p-value of 3.78e$-$08. Extrapolating this number to a whole year suggests savings for the airport of around £72,800.

| Rank | daily | weekly |
|---|---|---|
| 1 | £3,934.40 | £5,613.33 |
| 2 | £4,008.69 | £5,540.98 |
| 3 | £4,011.72 | £5,613.73 |
| 4 | £4,112.84 | £5,466.21 |
| 5 | £4,134.87 | £5,079.76 |
| 6 | £4,220.13 | £5,373.95 |
| 7 | £4,232.24 | £5,966.09 |
| 8 | £4,271.26 | £5,361.55 |
| 9 | £4,364.58 | £6,387.54 |
| 10 | £4,565.47 | £5,537.53 |
| average | £4,185.62 | £5,594.07 |

Table 2.5: Average weekly re-rostering cost for daily and weekly surplus shift levels.

## 2.6 Conclusion

In this paper, we present an extension to measuring surplus shifts for workforce structures consisting of teams and contracts and present three Simheuristics for fine-tuning flexibility metrics in scheduling. The experimental results suggest that using these methods can lead to significant re-rostering cost reduction for scheduling practitioners. However, from discussions with the CSS at our partner airport we know that there are other factors that might slow down the introduction of flexibility metrics in practice. Firstly, the airport is always struggling to hire more security officers. This is partly due to the realities of shift work and how they impact people's home life and partly due to the long training and screening process needed before a new hire can join the security team. Nevertheless, our work provides structural evidence to avoid planning for highest demand peaks and use the freed up workforce to add flexibilities instead. The results also support hiring additional staff to bolster the workforce in the long run and, in turn, save re-rostering costs down the line — something that is important when considering hiring decisions in the futures. This means that taking a focused flexibility approach gives practitioners, as a by-product to the approach, a way to update capacity over time.

Ultimately, this work was conducted on airport security scheduling data, it would be great to investigate the impact of a focused flexibility approach in other settings which might also lead to the formulation of new robustness metrics.

# Declarations

The authors have no financial or proprietary interests in any material discussed in this article.

# Acknowledgements

We thank Jared Cheung and Sam Song for the preliminary testing of the robustness metrics and simheuristics used in this paper. Additionally, we would like to express our thanks to the CSS at our partner airport for the problem insights and data provided that allowed us to conduct this research.

# Chapter 3

# Paper III: Simheuristics for Strategic Workforce Planning at a Busy Airport[1]


## Abstract

Airport demand frequently exceeds capacity. An airport's capacity bottleneck is often located in its runway system. Other times, it is elsewhere, including the terminal facilities. At one of the busiest UK airports, the subject of this study, the bottleneck is located in the passenger security hall, where staff are employed directly by the airport operator. This airport plans to restructure the current workforce's overall size and types of contracts. Our work proposes a Simheuristic approach that utilizes current workforce data and demand forecasts for the upcoming season to adjust the contractual configurations systematically. We employ simulation to test the expected costs (vs flexibility needs) of the likely rosters that each contractual configuration will allow. Using the simulation results, the algorithm aims to identify the optimal contractual configuration to minimize costs while ensuring the adaptability required to address unforeseen changes in the flight schedule as each season is underway.

## 3.1 Motivation

Capacity planning — including but not limited to workforce planning (Lusby et al., 2012) and slot declaration for future seasons (Zografos et al., 2017) — is constantly at the top of the agenda of most airport operators. Its centrality to airport planning and operation has been extensively reported and studied (Jacquillat and Odoni, 2015). In many cases, airport demand exceeds capacity (Jacquillat and Odoni, 2018). Oftentimes, the airport's runway system is the bottleneck to achieving higher throughput. Other times, including at one of the Top-10 UK's busiest airports — the subject of our study — it occurs in their security hall (in yet different cases, the bottleneck might be the check-in hall, etc.). This particular airport is extremely keen to improve staffing issues and satisfaction with the used rosters. Furthermore, work demands are changing, with people wanting more flexibility to suit their lives outside of work — the literature has consistently reported on this trend for a while, see Laporte (1999). In addition, the UK's aviation sector is facing extensive security updates with upcoming changes to hand luggage regulations (Department for Transport, 2022) — thus requiring new ways to run passenger security at airports. As a result, our partner airport operator is looking to overhaul their current rosters to suit better the needs of its employees (especially in the security hall) whilst accommodating growth and keeping staffing costs at its security hall under control (as this is the single biggest cost center at the airport).

The current method adopted for creating staff rosters and deciding suitable workforce sizes largely depends on the experience and expertise of the chief staff scheduler (CSS). They have been in this position for years and have an intrinsically profound knowledge of all relevant working regulations and the process for producing schedules. However, they are human and lack the computational power of modern modeling software (they use spreadsheets for the task). On a related note, as the CSS moves on to the next stage of their career at the same airport, someone else will have to take over their role, creating a very clear case for the need for effective knowledge management.

Currently, the strategic planning decisions for upcoming seasons start with budget decisions. The workforce size is set based on the available budget. Here, the CSS needs to argue whether or not the given budget is sufficient to employ a workforce large enough to cover all upcoming demand, staff holidays, unforeseen absences, unforeseen demand fluctuations, staff training, and much more. All in all, it is an enormous task

to perform by hand and spreadsheet!

About 25 years ago Laporte (1999) observed that algorithmic approaches to the construction of work schedules may be "often too constraining and not sufficiently flexible", especially with regard to accommodating workers' preferences. The CSS at the partner airport agrees with this view, adding that none of the costly software products the airport has tried over the years has ever provided the requisite flexibility to adapt some of the rules (which is virtually always required) in a way to make the schedule manageable and accepted by the unions who drive the negotiation of each and every roster. Laporte is also completely right that at the basis of staff rotas (*rotating workforce schedules*, such as those employed at the airport), there exists a fairly basic arithmetic. But the complexity of the current working regulations, the frequency with which these are modified, the variety of available contracts, and the size of the workforce (all major issues at the airport) make it impossible for anyone having to carry out workforce planning and scheduling tasks fully by hand to attempt any form of optimization. Additionally, unions require that the finalizations of the roster (consisting of frequent cycles of negotiations between the CSS, airport operator and unions) for the upcoming season take place as early as possible into the start of the current season. But the time required to produce a single complete roster manually is so high that it has become virtually impossible to accommodate the pace that the mandated timeline and frequent negotiation cycles require. Therefore, shortcuts are needed almost everywhere along the process. Currently, the CSS relies heavily on past schedules, the existing workforce size (and contractual setup), and their experience, to estimate whether or not the budget will suffice for the upcoming season. In summary, the estimates depend on basic full-time equivalent (FTE) calculations and highly educated guesses to fill the gaps.

What is currently not possible is the strategic testing of different workforce sizes and contractual configurations to see which combination of workforce size per available contract will best cope with the uncertainty of the upcoming season. Contracts available to the airport's security staff differ in their work regulations (maximum weekly work hours, allowed shift types, etc.) and team structure (size of teams that follow the exact same roster for the entire season). To facilitate the rostering process (e.g., to devise a lower number of roster lines, thereby ensuring repeatability of the same cycle more times during the same season), the contracts with a larger workforce are split into multiple smaller contracts with the same work regulations and team size. The question we help answer in this paper is: "Given the

demand forecast, what is the best contractual configuration to head into the next season with?"

Based on Simheuristics (Juan et al., 2015), we recently developed a software tool to support the above-described decision-making process at our partner airport. At the time of writing, this is being field-tested directly by the CSS for future adoption. Our Simheuristic takes a starting workforce (e.g., the current contractual configuration) and the demand forecast for the upcoming season. It then adjusts the contracts to find the lowest FTE combination across contracts, one that best passes a series of robustness tests — these being carried out via simulation. Once the best FTE and accompanying contractual setup have been devised, the CSS and HR department will still need to recruit the required staff before the plan can be fully implemented. Between advertising, recruitment, and onboarding/training, a cycle of several months awaits. Our proposed tool is, therefore, a decision support system that helps the CSS test their workforce against the upcoming season, but it does not take over the entire job of strategic workforce planning. The latter will remain with HR and the airport's senior management team.

In the remainder, we will start with an analysis of the related literature on both strategic workforce planning and Simheuristics (Section 3.2). We will then present our Simheuristic approach (Section 3.3), delving into our approaches to both heuristic search and the way we help the algorithm move towards better solutions, by integrating simulation. We then analyze (Section 3.4) results from an extensive experimental campaign and discuss the benefits and costs of adopting our approach before we wrap up the major learning points that we believe might apply outwith our partner airport and security hall (Section 3.5).

## 3.2 Related Literature

### 3.2.1 Strategic Workforce Planning

Strategic workforce planning (SWP) studies workforce dynamics, including attrition, promotion, and retention, paired with recruitment decisions to cover the expected workforce demand for the upcoming seasons. In other words, it "examines the gap between staff availability ... and staff requirements ... over time, and prescribes courses of action to narrow such a gap" Doumic et al. (2017). For a general introduction to workforce planning, we refer the reader to De Bruecker et al. (2015).

Oftentimes, studies of the phenomenology of workforce dynamics and decision-making quickly lead to some form of stochastic modelling. Given the above-mentioned aspects (staff availability, to cite but one), this is hardly surprising. The SWP literature broadly differentiates between approaches that integrate simulation (such as the current paper) and those that do not. Examples of the latter subset include, for instance, works grounded in stochastic programming, such as Jaillet et al. (2022).

Still, deterministic approaches remain popular. For example, Zeng et al. (2019) work on a similar problem to the one experienced by our partner airport. The authors' aim is to minimize their workforce mix while providing enough demand coverage vs. a given demand forecast. The formulation of their version of the problem, though, is centred around *hierarchical skills* such that staff with higher level skills are permitted to cover demand for staff with lower levels. However, in airport security halls, staff employed have roughly homogeneous skills.

As another such example, Llort et al. (2019) looks at SWP for management consulting firms. Similar to our partner airport's problem, workforce decisions made now will have long-term consequences, as "consultants are highly qualified workers who need very long learning periods to achieve enough expertise". The paper proposes and validates a decision support tool for strategic staff planning, built upon mixed-inter linear programming, tackling decisions on attrition, recruitment, and promotion.

Turning our focus to approaches that indeed include simulation, the reader should look at Jnitova et al. (2017) for an extensive overview of simulation models for workforce planning in military settings.

Partially aligned with the problem studied in Llort et al. (2019) — but two decades earlier, Parker and Marriott (1999) model the workforce through a personnel flow diagram that encapsulates the modeling of transitions to different positions within the company (pay grade, etc.), attrition rates, and recruitment. Using a simulator, the authors then derive the steady-state flow of personnel, on the basis of which a range of SWP decisions can be supported (minimizing cost, maximizing personnel efficiency, etc.).

The incorporation of recruitment, attrition, promotion, retention, and training — or a selection thereof — in SWP studies is, by our very definition of SWP, a central, recurring aspect of most papers. However, it is generally hard to find works adding workforce scheduling to the mix. In particular, there appears to be a lack of papers

looking at ways to ensure that staff rosters will remain feasible throughout the time horizon of interest within a context where the demographics of the workforce evolve as a result of the decisions being made over time. A recent exception is represented by Akl et al. (2022) and their simulation-optimization framework to optimize a maintenance schedule and decide the requisite workforce and skills to accomplish that schedule whilst accounting for backlog costs.

We propose a simple yet effective Monte Carlo simulation paired with a metaheuristic search scheme to test if the proposed workforce structure and contract configuration can handle the expected demand patterns whilst allowing for absences due to sickness or holidays. Decisions about how this structure can be attained (attrition, recruitment, promotion, retention, and training) can then be made jointly by the CSS and the HR team directly on the basis of the information generated as the output of our model.

### 3.2.2 Simheuristics

With reference to the taxonomy of Simulation-Optimization problems in Figueira and Almada-Lobo (2014), our problem falls into the category of "Optimization with Simulation-based iterations". Suitable methodologies to address such problems include the joint adoption of metaheuristics with simulation. For an overview of *Simheuristics* and their use, see Juan et al. (2023). We now look at how Simheuristics have been successfully applied to various scheduling problems.

Calvet et al. (2016) use SimILS to solve a Distributed Flow-shop Scheduling Problem. The iterated local search (ILS) strategically searches the solution space, and Monte Carlo simulation is used to evaluate the solution at each iteration. Somewhat similarly, Gök et al. (2020) use Large Neighborhood Search (LNS) paired with a Petri-Net-based discrete-event simulation to schedule airport turnaround staff. Building upon that, Gök et al. (2023) use two nested Simheuristics to produce robust schedules for the same problem. The 'outer loop' targets a resource-constrained project scheduling problem (RCPSP) with an enhanced *SimLoop* algorithm (Guimarans et al., 2015). At each iteration of this algorithm, the 'inner loop' solves a travelling salesman problem with time windows (TSPTW) through an enhanced version of *SimLNS* (Guimarans et al., 2015). In both loops, the metaheuristic is used to produce new solutions to the problem, while simulation is used to evaluate the solution's robustness.

Kızıloğlu and Sakallı (2023) propose two Simheuristics to simultaneously solve Flight scheduling, Fleet assignment, and Aircraft Routing problems. The metaheuristics involved are Simulated Annealing and Cuckoo Search, respectively. Solution evaluation happens through Monte Carlo simulation, which produces an expected objective function that is fed back to the metaheuristic part of the method for further iterations. Ultimately, the final solution is extensively tested through longer Monte Carlo Simulations.

Dehghanimohammadabadi et al. (2023) use a multi-objective metaheuristic, based on particle swarm optimization and paired with a discrete event simulation, to optimize patient appointment scheduling in a healthcare setting. The metaheuristic creates the patient schedule, consisting of patient arrival time and type. The simulation then runs through one full working day to test the schedule and measure patient dynamics in the system. The information is fed back to the metaheuristic so that the schedule may be improved. This follows the same ideas from Gök et al. (2020) and Gök et al. (2023). Building upon this evidence of prior success from this sort of approach, we adapted it to our SWP airport problem, by combining Adaptive Large Neighborhood Search (ALNS) (Ropke and Pisinger, 2006) and Monte Carlo Simulation.

While not referred to as 'SimALNS', this combination has been used widely in recent years to improve the effectiveness of ALNS for solving problems under uncertainty. The simulation approach varies in the literature from fuzzy simulation (Zhou et al., 2020), over discrete event simulation (Kisialiou et al., 2018) , to Monte Carlo Simulation (Nasri et al., 2020). Occasionally (e.g., Nasri et al. (2020)), simulation is used in the scheme in a slightly different way, to create realizations for which ALNS is then used to create solutions.

As the next section will show, in this paper, ALNS is used to alter the contractual setup for the rostering problem (the input values to the rostering algorithm rather than the roster itself), and simulation is used to test whether the resulting rosters are feasible under uncertainty.

## 3.3 Method

Our method for finding a good contractual combination for the upcoming season builds on existing scheduling technology at the airport. The assumption is, that there exists a tool that can produce a feasible roster for a given workforce based on the input demand forecast of one model week that satisfies all working regulations set by unions and

contracts (i.e. a deterministic solution to the rostering problem). For the purpose of this paper the exact details and rules represented in the rostering problem are irrelevant as long as the contractual features that are considered in the ALNS procedure are also used as input parameters in the rostering tool (e.g. weekly hours for each contract). We refer to (Ernst et al., 2004b; Van Den Bergh et al., 2013) for detailed reviews on scheduling literature. The approach aims to provide a way to find acceptable solutions for all problem stakeholders (CSS, unions, employees) rather than achieving optimality.

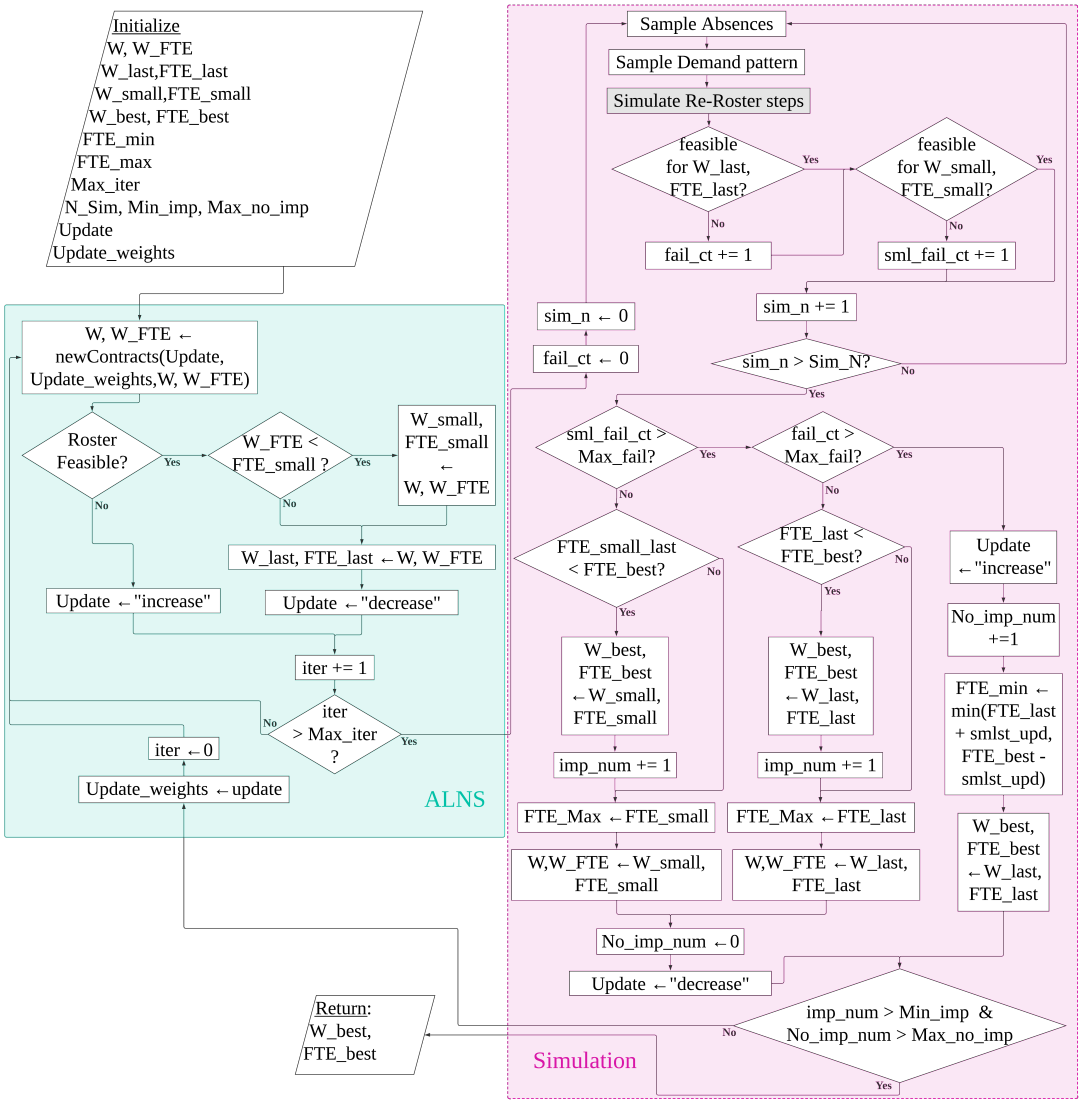



Figure 3.1: SimALNS for Strategic Workforce Planning.

### 3.3.1 Adaptive Large Neighborhood Search (ALNS)

In our approach (Figure 3.1), ALNS aims to find the workforce with the lowest FTE that enables feasible rosters that also pass all robustness tests performed via simulation.

Starting from an initial feasible workforce (*W*) with associated FTE value (*W_FTE*), each ALNS iteration first updates the current workforce by either increasing or decreasing the associated FTE value through adding or removing (depending on the current setting of *Update_action* — see below) teams, workhours and contracts. The upper and lower limits of *FTE_max* and *FTE_min* are not strict bounds on the solution space but rather guidelines to help lead solutions back to the feasible solutions space. So, once an infeasible solution has been reached, the operators will increase/decrease the contracts until they are back within the limits, before exploring further solutions.

Next, we apply the airport's existing roster tool to deterministically test whether a feasible roster (i.e. one that meets the demand forecasts) can be produced for the current workforce. If there is no feasible roster, *Update_action* is set to increase — based on the assumption that higher FTE values more likely lead to feasible rosters than lower FTE values might do. A maximum allowed computational budget (*Max_iter*) ensures the algorithm eventually moves on to the simulation stage.

If the roster is feasible, the associated value could be the lowest in the current ALNS run. In that case, the associated variables (*W_small* and *FTE_small*) are updated before updating the variables associated with the last feasible roster (*W_last* and *FTE_last*). Otherwise, only the latter step is performed. Finally, the *Update_action* is set to 'decrease' to try and improve the current solution further.

Once ALNS has run for *Max_iter* iterations, the lowest (*W_small* and *FTE_small*) and last feasible (*W_last* and *FTE_last*) solutions are sent to the Simulation to test if they are also feasible once the uncertainty inherent with the problem is accounted for.

### 3.3.2 Monte Carlo Simulation

At the simulation stage, we aim to replicate the Re-Rostering steps that take place throughout the season to account for and repair infeasibilities in the roster. Regardless of how good demand forecasts are, they can never account for everything that could happen throughout a season. The most common discrepancies happen due to staff absences and unforeseen changes to the demand patterns. To account for this, the simulation runs through multiple rounds of sampling demand patterns and staff

absences and then checks if the roster can be repaired without breaking any working regulations. Each time the re-rostering steps fail, the method keeps track of this through a fail counter. This step is repeated for both contract setups that are fed to the Simulation from the ALNS run.

Once the simulations are complete, we first check if the lowest solution (*W_small*) satisfies the pass condition of the simulation. One round of simulation is considered as 'passed' if it is possible to repair the roster via re-rostering — after sampling absences and the new demand pattern — without violating any work regulations. A solution passes the simulation rounds if the re-rostering steps fail no more than the maximum number of fails (*Max_fail*). If it did pass, we check if the solution is better than the previous best solution *W_best* and update *W_best* accordingly, before setting the FTE upper limit *FTE_max* to be equal to *FTE_small*. When comparing two feasible contractual combinations that both passed the simulation we only consider the FTE value where a lower FTE is better (i.e. less expensive). This is due to two main reasons. Firsly, the FTE value is independent of the simulation result — it simply measures the available work hours per week. Secondly, if a contractual combination passes the simulation we can assume that it will be able to handle the majority of demand patterns and unforeseen absences that might occur during the season. How good the pass was does not matter at this decision stage as the real demand fluctuations and absences will be different anyways and a lower FTE value is of higher importance here. The upper limit (*FTE_max*) helps ensure the ALNS operators keep bringing *W* back to feasible solutions throughout the iterations. Finally, the current solution (and starting point for the next round of ALNS) *W* is set to equal *W_small* and the *Update_action* is set to 'decrease' as ALNS will then start from a feasible solution to try and improve. If (*W_small*) does not pass the simulation tests, the same steps are repeated with the last feasible solution *W_last*. Should both *W_small* and *W_last* fail the simulation tests, the current solution *W* is set to equal *W_last*, but the *Update_action* will be 'increase' to try and find a feasible solution. In addition, the lower bound for the operators *FTE_min* will be updated as well. The SimALNS procedure will eventually stop once a minimum number of updates (*Min_imp*) of the best solution have been found and a maximum number of no improvement steps (*Max_no_imp*) have been reached.

## 3.4 Experiments

Our Simheuristic approach was tested with data provided by the airport encompassing a demand pattern forecast — consisting of staffing requirements for every 15-minute interval of the day— for a 29-week season, paired with a matching contract set-up. The goal of this study was to investigate contractual combinations that can handle the most common disruptions (demand pattern changes throughout the season and staff absences). More detailed stress testing could be performed by adding matching demand patterns to the input forecast, which will then be sampled during the Monte-Carlo simulation. We consider 15% unforeseen overall absences (based on the CSS's estimate) and 28 days paid leave per FTE employee per year (based on the UK statutory paid annual leave entitlement (UK Government, 2024)). With regard to absences, our assumptions are as follows:

- Each individual has a 5% chance of being absent on any particular day
- This is increased to 15% if a team member was absent on the previous day
- Absence duration lies between 1-5 days, with decreasing probability ($\frac{5}{15}, \frac{4}{15}, \frac{3}{15}, \frac{2}{15}, \frac{1}{15}$ respectively)

The modeling of absences is based on rough estimates by the CSS because it was not possible to obtain the exact absence data due to security reasons. The absence patterns were tested to ensure an average absence percentage of 15% over the season. Similarly to the demand patterns, the absence simulation can be adapted to account for a higher or lower absence rate depending on the scenario that is being investigated. The allocation of holidays in our model works as follows:

- Each full-time employee is assigned 16 days off over the 29-week season ($16 \approx \frac{29 \text{ weeks}}{52 \text{ weeks}} \times 28$ days). All part-time employees (i.e. those working 'less than 40-hour' weeks) are assigned a number of days off that is proportional to their work hours (e.g. a 30-hour contract employee would get $16 \times \frac{30}{40} = 12$ days off for the season)
- Holidays are 1-5 days in length with equal probability and based on the availability of days off.
- There is a maximum number of employees that can be off on the same day. We set the maximum to be $2 \times \frac{\text{total number of holidays required (days)}}{\text{total number of days in the season}}$.

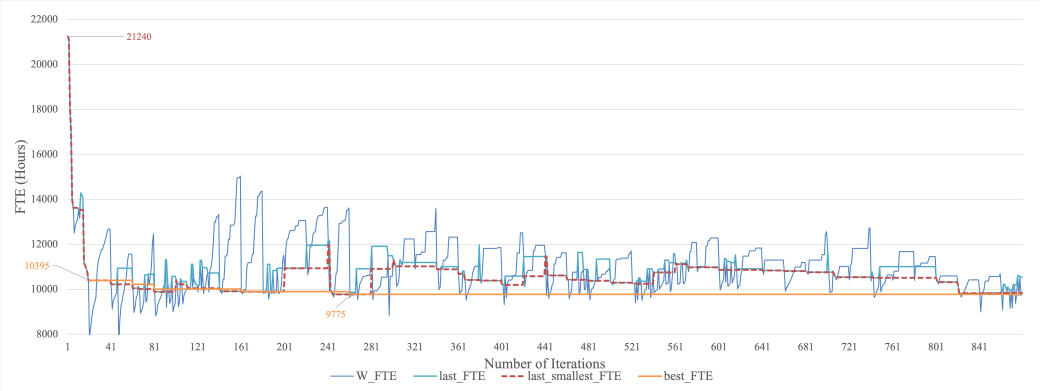
The SimALNS parameters were set based on trial and error to suit this particular problem setting. However, more detailed methods for metaheuristic parameter tuning have been used successfully in the past (Yadav and Tanksale, 2023).

- operator weight updates: +2 for improving the current solution, +4 for improving the best solution
- decay parameter: 0.8
- stopping conditions: minimum number of improvements (3) and maximum number of iterations without improvement (100) are both satisfied. These values were based on the outputs obtained after running the SimALNS approach for a time limit of 3 days.
- number of simulation replications: 10 — This value was chosen to balance the runtime of the re-roster model that needs to be executed during each simulation replication and the need for replications to ensure that significant results can be obtained. Here, 10 replications cover roughly 34% of the demand forecast. In addition, we set three of the 10 replications to cover the three highest demand weeks of the forecast. Paired with a required pass rate of 80%, this ensures that each roster that passes the simulation will be able to handle most weeks during the season.
- The ALNS operators are defined to mimic allowed actions to the contractual set-up of a workforce. The main actions include adding/removing a contract or team and increasing/decreasing weekly hours by 5 hours. The actions are applied to a random contract or to the contract with the highest/lowest number of employees or weekly hours.

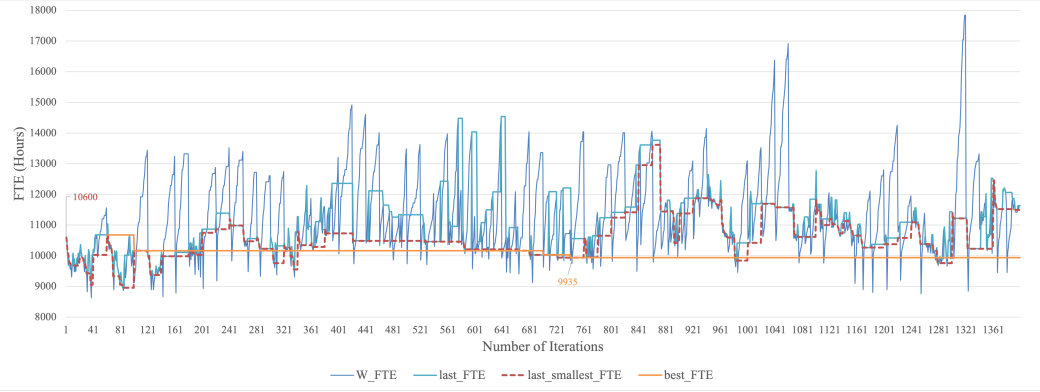
All experiments were run on a personal laptop (Apple M1 Pro) running macOS Sonoma 14.4.1. The SimALNS implementation was coded using Python 3.8 and calling Minizinc Version 2.8.3 (Nethercote et al., 2007) with Gurobi 9.5.1 (Gurobi, 2024) for the rostering and re-rostering actions.

Figure 3.2 shows the progression of FTE values for all visited solutions (*W_FTE*), all feasible solutions (*last_FTE*), the lowest feasible solution for each metaheuristic run (*last_smallest_FTE*), and the overall best solution (*best_FTE*) for a 12-hour run of our SimALNS method. For the graph in Figure 3.2 (a), we randomly created a feasible starting contractual combination with an FTE value double that of the airport's

current workforce. The image shows a steep decrease (FTE of 21240 hours down to 10395 hours) of the smallest feasible solution (*last_smallest_FTE*) during the first set of 20 ALNS iterations. This solution is shown to be feasible during the first set of simulations after 20 ALNS iterations (as shown by the start of the *best_FTE* line in orange). The spikes of the blue *W_FTE* and *last_FTE* lines show that as the iterations continue, the approach tests various solutions around the current best solution to avoid getting stuck in a local minimum. The best solution is continuously improved until it reaches an FTE value of 9775 (equivalent to $\approx 245$ full-time employees) after 260 iterations. In summary, the metaheuristic first quickly tests a range of solutions and improves the smallest feasible solution with regard to creating an initial roster. Then, the computationally more expensive simulation only tests the best solutions as found by the metaheuristic to see if they also stay feasible under uncertain demand patterns, staff absences, and holidays. Figure 3.2 (b) shows the progression of FTE values while




(a) Start with a contractual combination with double the airport's FTE value




(b) Start with the airport's contractual combination

Figure 3.2: FTE change over a 12-hour run of SimALNS

applying our SimALNS approach to the current contractual combination at the airport. Starting from an initial FTE of 10600 hours, we can see that the airport's current set-up is already relatively close to the best possible FTE value. Therefore, we can also not see a drastic decrease in FTE values at the beginning (as seen in Figure 3.2 (a)). We do, however, see an initial decrease in the best solution between iterations 40 and 120 and a further decrease between interactions 680 and 760. Between these improvements, the graph clearly shows how the method tries to escape the potential local minimum by consistently moving away from the current best FTE and then decreasing again (spikes in blue lines of *W_FTE* and *las_FTE*).

We also tested our Simheuristic by running five random starts for four different scenarios and different random seeds. For the scenarios, we started with a basic run (**basic** demand scenario), where the demand patterns match that of our partner airport, and we model absences and holidays as described at the beginning of the section. We then investigated a scenario with the same demand pattern but without the holiday allocation (**no holiday**), to see how much of the final FTE hours can be associated with holiday allocation. Finally, we tested our approach with two demand forecasts that differ in the number of security lanes needed throughout the season ($\pm$10%, scenarios **10% more** demand and **10% less** demand). For each of the runs, a random combination of contracts was created that satisfied a minimum FTE value (1.6 times the FTE value of the current workforce structure for the 10% more demand case and 1.3 for all other cases) and led to a feasible roster. The results of these runs can be found in Table 3.1.

| **demand pattern** | FTE values ranked | | | | | Average | % change |
|---|---|---|---|---|---|---|---|
| | $1^{st}$ | $2^{nd}$ | $3^{rd}$ | $4^{th}$ | $5^{th}$ | | |
| **basic** | **9355** | 10230 | 10320 | 10365 | 10405 | 10135 (10330) | N/A |
| **no holiday** | 9835 | 9870 | 9945 | 10185 | 10235 | 10014 | -1.19% (-3.06%) |
| **10% less** | 9505 | 9665 | 9935 | 9945 | 10080 | 9826 | - 3.05% (-4.88%) |
| **10% more** | 10110 | 10255 | 10460 | 10650 | 11085 | 10512 | + 3.72% (+1.76%) |

Table 3.1: Ranked best FTE values (hours) for five random starts in various scenarios. Statistics with the basic outlier omitted are given in brackets.

One very noticeable result is that of the best result for the **basic** demand pattern, which produces a significantly lower FTE result than any other run-scenario

| **basic demand** | $1^{st}$ | $2^{nd}$ | $3^{rd}$ | $4^{th}$ | $5^{th}$ |
|---|---|---|---|---|---|
| FTE | 9355 | 10230 | 10320 | 10365 | 10405 |
| headcount - size 3 teams | 45 | 81 | 45 | 57 | 99 |
| headcount - size 5 teams | 115 | 320 | 195 | 150 | 150 |
| headcount - size 8 teams | **184** | 0 | 48 | 104 | 88 |
| headcount - 40h | 60 | 0 | 48 | 88 | 0 |
| headcount - 35h | 55 | 10 | 240 | 31 | 158 |
| headcount - 30h | 45 | 81 | 0 | 192 | 80 |
| headcount - 25h | 0 | 250 | 0 | 0 | 99 |
| headcount - 20h | **184** | 60 | 0 | 0 | 0 |
| total headcount | 344 | 401 | 288 | 311 | 337 |

Table 3.2: Headcounts for different contract features and all runs with the **basic** demand pattern, where the grey column corresponds to the outlier run.

combination (we use 'scenario' and 'demand pattern' interchangeably). To see what might be causing this drastic difference in FTE value we take a look at the headcounts for various contract features — Table 3.2— of the results associated with the **basic** demand pattern. We can see from the table that the majority of the workforce (184 out of 344) employees are assigned to a 20-hour contract with a team size of 8, which does not happen in any other contractual combinations. In addition, we performed additional Simulations (as outlined in Figure 3.1) to test the performance of this solution further and find a very low rejection rate of $\approx 7\%$. Given, that no other results get close to the FTE value of this particular solution (even though the Simheuristic approach consistently trials contractual combinations in this FTE range, see Figure 3.2), we also present in brackets the results with the data of that run omitted (Table 3.1).

Looking more generally at the percentage change in FTE for the different demand patterns in Table 3.1, we find that not modeling holidays leads to an average 3.06% reduction in FTE (hours), suggesting that only just over 316 additional FTE hours ($\approx$ 8 people) are needed per week to cover the holiday requirements of the current workforce. Furthermore, we find that a 10% decrease in demand from the current scenario leads to a significantly higher change in FTE compared to the scenario with 10% additional demand (4.88% decrease vs. 1.76% increase). This can be linked to the set-up of the security hall at our partner airport where all security lanes are paired

into so-called 'cells' — two adjacent lanes form a cell — by sharing the costly body scanner between the two. As a result, opening a new cell requires more staff than manning the second lane in an already open cell (or, vice versa, closing a lane in a cell reduces the workforce by less than closing down a cell altogether). Paired with the fact that demand is measured in the number of lanes that are required to cope with the flow of passengers, it is plausible that the discrepancy between demand changes and required FTE hours is mostly due to the cell layout of the security hall.

| **categories** | contracts | | teams | | employees | |
|---|---|---|---|---|---|---|
| | **Start** | **Best** | **Start** | **Best** | **Start** | **Best** |
| count | / | 0.32 | 0.50 | 0.43 | 0.61 | 0.37 |
| team size 3 | 0.19 | 0.42 | 0.46 | 0.42 | 0.46 | 0.42 |
| team size 5 | 0.40 | 0.13 | 0.00 | 0.19 | 0.00 | 0.19 |
| team size 8 | 0.17 | -0.07 | 0.19 | -0.09 | 0.19 | -0.09 |
| 40h | / | 0.47 | 0.50 | -0.13 | 0.61 | -0.09 |
| 35h | / | -0.03 | / | 0.08 | / | 0.05 |
| 30h | / | -0.05 | / | 0.02 | / | 0.03 |
| 25h | / | 0.11 | / | 0.14 | / | 0.13 |
| 20h | / | -0.12 | / | 0.01 | / | -0.13 |

Table 3.3: Correlation coefficients for best FTE Hour variation and the number of contracts, teams, and employees in various categories (team size, weekly hours)for the starting and best solution.

To better understand the relationship between the solution's contractual features (e.g., the number of employees, teams, and contracts with varying team sizes and weekly hours), we examine the correlation coefficients between these features and the final FTE values. Table 3.3 displays the correlation coefficients for the various contractual features (number of contracts, teams, and employees for different team sizes and weekly hours) between the starting contractual combination and the final results (i.e. best FTE). A backslash indicates that the contractual features did not appear (20h-35h variations) or had all equal values (number of contracts) in the starting solution, making it impossible to calculate a correlation coefficient in these cases. Overall, the lack of correlation coefficients in Table 3.3 above magnitude 0.68 indicates that there is no strong correlation between contractual combinations (in the starting or final solutions) and final FTE values. From this, we can draw two main

conclusions. Firstly, the starting contractual combination does not affect the final result — a highly important conclusion for our approach as we do not want the final result to depend on the manual input decided by the CSS. Secondly, the final contractual combination also does not appear to affect the final FTE value, meaning that a contractual combination that performs well in one season might not do so in the following season. Connecting this back to the current approach at our partner airport, it suggests that their approach of rolling old schedules and contracts over to the next season (as a starting combination) may not be the best approach to finding a good contractual combination for the upcoming season. From an HR perspective, the approach makes sense, as it is significantly easier to hire staff for existing contracts that have been approved by union representatives and are tried and tested with the existing workforce. Both are important factors in keeping retention rates high. However, our approach suggests that changes might be beneficial to the performance and cost efficiency of the final workforce to adapt the contracts from season to season — within the possibilities of changing existing contracts, for example, by changing team size rather than weekly hours of contracts.

## 3.5 Conclusion

Coming back to our partner airport, we can draw conclusions about their work and suggest future changes.

Our results indicate a lack of correlation between contractual combinations and low FTE values. This means that a workforce that works for one season might not be the best choice for the next season. In reality, it will not be possible to revamp the entire contractual set-up of a workforce from season to season, but the CSS should at least try to validate the acceptability of some of the best solutions (perhaps those closer to the current roster) by the workforce and their representative unions.

One interesting result is that of the first-ranked run for the basic demand pattern. It gives the lowest FTE value by a large margin, and the contractual combination is the only one with a high number of 20-hour contracts. From conversations with the airport CSS, we know that they prefer to have lots of employees on these low-hour contracts because they provide more flexibility during the scheduling process, thus making their job much easier. However, high numbers of these contracts are rarely approved by the union representatives — making these contractual combinations infeasible in real life.

# Chapter 4

# Case Study: Security Staff Scheduling at Edinburgh Airport


## Abstract

For the purpose of this thesis, a collaboration with Edinburgh Airport was established to address their security officer scheduling problem. Creating work schedules for airport security officers is a lengthy and often complicated problem involving several stakeholders. This case study summarises the work regulations that must be considered when creating rosters in this setting and provides a mathematical formulation of the problem. A roster tool implementation with an accompanying user handbook is presented — a prototype of which was provided to Edinburgh Airport for testing. The CSS's feedback on the prototype was that while it is the best solution they have seen so far, it still lacks the flexibility needed to create security officer schedules that are usable by the airport.

## 4.1 Introduction

Edinburgh Airport is the 6th busiest airport in the UK (UK Civil Aviation Authority, 2024, p. 17) and the busiest airport in Scotland (WorldData.info, 2024). In 2023, it was used by 14.4m passengers (an increase of 28% from the previous year) travelling to 152 worldwide destinations using 35 different airlines (Edinburgh Airport, 2024a). The airport predicts that passenger numbers for 2024 will increase even further to reach record levels beyond the previous highest numbers in 2019 (Edinburgh Airport, 2024b). To make this expansion possible, the airport needs to ensure the smooth flow of passengers through the building and, in particular, bottlenecks such as the security hall that all passengers need to pass through.

Edinburgh Airport's security hall is currently operated by a workforce of roughly 330 security officers whose schedules are created and adjusted by one CSS. The CSS has been in this role for years and has extensive knowledge of all rules and regulations (including typical preferences by the security staff) that need to be considered during the scheduling process. Their current approach to creating schedules is spread-sheet-based manual manipulation of existing or old rosters to suit the demand forecast. Occasionally, when a completely new schedule is required, they will work on creating a schedule from scratch, which can take several days and is therefore avoided as much as possible.

At present, the CSS can still manage the difficult task of creating schedules for the security workforce. However, with the expected continuous increase in passenger numbers and changes to hand luggage regulations, which require new baggage scanners (Department for Transport, 2022) and thus additional requirements for security staff, this task will soon become even more challenging. Further complexity is added due to the need to negotiate with unions representing the security workforce to find a mutually agreeable schedule. The resulting approval process for the rosters can become lengthy, with several rounds of discussions and subsequent requirements to change the schedule.

As a result, Edinburgh Airport has attempted to switch to commercial roster solutions in the past, which have proven to be too rigid for this challenging and ever-changing set of requirements. One attempt consisted of a consulting project with the aim of building a 'made-to-measure' decision-support tool that mimics the steps the CSS runs through to create a new roster. Ultimately, this approach, which relied on metaheuristics, was again too rigid to be used in the scheduling process, leaving

the airport still wanting an automated solution to support scheduling decisions but unable to find a flexible enough approach.

After the completion of that project, the aviation industry was hit heavily by the impacts of COVID restrictions put in place in 2020, with Edinburgh Airport needing to let go of large numbers of their staff (BBC, 2020). However, with that crisis also came opportunities. Having a much-reduced skeleton staff gave the CSS the opportunity to try out different schedules and approaches to staffing the security hall. During the recovery from COVID, new staff were hired on more flexible contracts, which helped to make the CSS's job easier. However, now, with passenger numbers back up to pre-COVID figures and the security team back at full staffing levels, old issues have started to arise again. Recent feedback has shown widespread dissatisfaction with the current schedules and the lack of flexibility to adjust schedules to the requirements posed by security officers' home lives. As a result, the CSS is planning for a review of the current scheduling process and regulations to help improve staff satisfaction — a process that could be eased by a flexible roster tool to support scheduling decisions.

What happens when these concerns from staff are not addressed has been seen and felt in the UK several times over the past years through strike action and walkouts — examples include bin strikes (BBC, 2022a), rail strikes (BBC, 2022b) and University strikes (BBC, 2024b). At the time of writing this thesis, border force staff at Heathrow airport have walked out in a dispute over their rosters (BBC, 2024a). This represents the last resort and is far from Edinburgh Airport's current situation, but it is still a good reminder of what could happen if staff concerns are not heard and effectively addressed.

### 4.1.1 Statement of Purpose

Overall, the work in this thesis is inspired by the security officer rostering problem at Edinburgh Airport. An essential aim of the collaboration with the airport was to create a roster tool that the CSS could use to support their everyday decision-making — the exact details of which could not be added to the papers in Chapters 1 - 3 for data protection reasons. This chapter, therefore, summarises in more detail the work conducted as part of this collaboration. This includes the scheduling problem at the airport, the development of a suitable roster tool and the challenges encountered.

Therefore, the purpose and objectives of this chapter can be summarised as:

O1 Provide insights into the rostering process at a UK-based airport. This includes the steps involved, a typical timeline, and the challenges faced by the CSS (Section 4.2).

O2 Outline the mathematical formulation of the security officer rostering problem at Edinburgh Airport (Section 4.3).

O3 Give guidance on the implementation of such a roster tool for other practitioners and researchers who would like to create a similar tool for their own rostering problem (Section 4.4).

O4 Summarise the feedback received from the CSS and outline additional challenges that need to be overcome before this tool can be used for the rostering process at the airport 4.5).

## 4.2 The Rostering Process & Rules

In order to be able to model the security officer scheduling problem at Edinburgh Airport and subsequently develop a roster tool to create suitable schedules, we first need to understand the problem background and rules that govern the process. This Section aims to capture this by first summarising the problem background and process (Subsection 4.2.1) before presenting the problem specifics (Subsection 4.2.2) that lead to explicit constraints that will be captured in the mathematical formulation in Section 4.3.

### 4.2.1 Problem Background

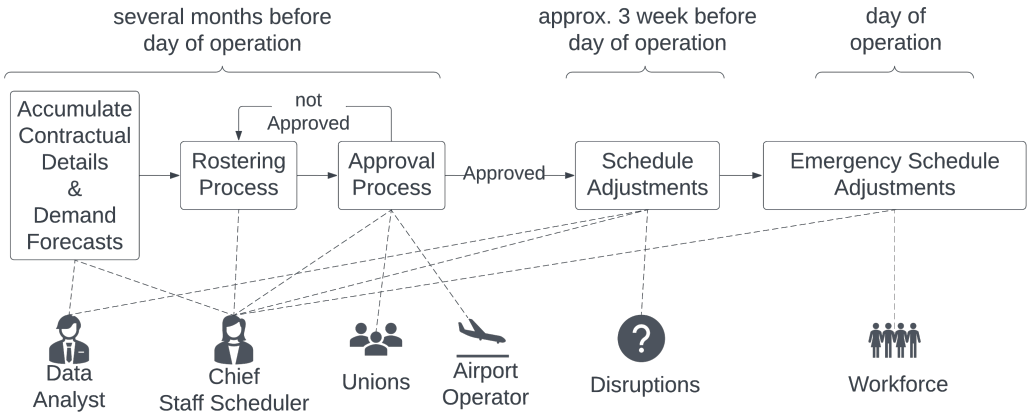



Figure 4.1: Security Staff Scheduling Problem Stakeholders.

The scheduling process at Edinburgh Airport (see Figure 4.1) is driven by the CSS. They start by accumulating contractual details and demand forecasts from the data analysts. These are used to create the first roster for the upcoming season. Each roster needs to be approved by the unions and the airport operator. In most cases, the approval process of the new schedule takes a few rounds where the roster is rejected with comments for adjustments each time. The CSS then has to go away and adjust the schedules before presenting them again for approval. This can be a lengthy process that takes several months to complete, so the CSS has to start it months before the schedule is implemented and used. Once a roster has been approved, it is used for at least one season of 5-7 months, but if the demand forecasts allow, even longer (or with minor tweaks).

Given the long time needed to create and approve schedules, they rely on demand forecasts that are made months in advance. As a result, unforeseen changes in demand and other disruptions can lead to schedule infeasibilities that need to be addressed before the day of operation. The CSS can make schedule adjustments up to three weeks in advance. The adjustments are limited to a pre-defined set of actions such as extending shift times — within the allowed overtime limits or re-assigning locations of shifts. Any time after the three week mark, they must ask the security officer's permission to tweak their schedule. Therefore, final schedule adjustments take place on a rolling basis three weeks before the schedule goes live, and the schedule is published to security staff immediately after the final adjustments have been made. If adjustments are needed closer to the day of operation, the CSS will discuss with the people on shift how to best address these, for example, by approving additional leave or cancelling overtime.

In addition to the rolling 3-week-out adjustments, the schedules are reviewed by the CSS and a representative of the security officers roughly on a monthly basis (depending on time availability) to ensure it is still up to standard or make minor adjustments (that do not need union or airport operator approval) if necessary.

### 4.2.2 Problem Specifics

This section gives an overview of all problem specifics, such as the security hall layout (Section 4.2.2.1), contractual structure (Section 4.2.2.2) and other work regulations (Section 4.2.2.3) that directly impact the scheduling rules that need to be adhered to.

#### 4.2.2.1 Security Hall Layout

The security hall at Edinburgh Airport consists of 6 lanes with luggage scanners that are paired up to share a metal detector/body scanner (Figure 4.2) — the combination of a pair of lanes and a metal detector is also referred to as a cell.

Passengers typically arrive at the security hall by scanning their boarding passes and joining a main queue. Staff then guide them to the various operational lanes to keep waiting times to a minimum. Depending on how many passengers are expected to arrive throughout the day, additional lanes will be opened (if not all six are operational yet), or lanes will be closed down. Which lanes are opened/closed next always follows the same pattern, which is 1,6,2,5,3,4 — this pattern is also referred to as *lane ordering*. For example, Lane 1 will be the first to open and the last to close, whereas Lane 4 will

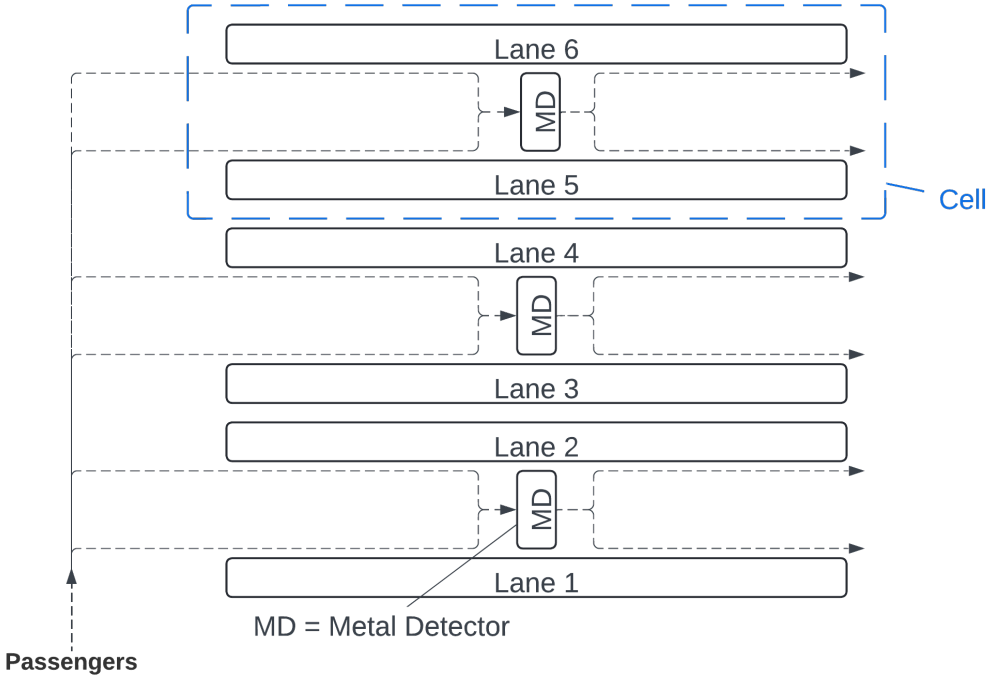



Figure 4.2: Security Hall Layout with Lanes and Cell Structure.

be the last to open and first to close. This pattern is partly due to the special status of lanes 1 and 6 as they are used as family and special assistance (Lane 1) and fast track (Lane 6) lanes and, therefore, need to be open most of, if not all, the time.

In order for an additional lane to open, it needs to be staffed by security officers. The number of required officers depends on the cell structure (i.e. opening the first lane in a cell requires a different number of officers than opening the second lane in a cell). With the current scanners, 12 officers are required to open a cell (i.e. open the first lane in a cell and the metal detector), and five additional officers (17 total) are needed to operate both lanes in a cell.

For a fully staffed cell, the current expected throughput per hour is approximately 800 passengers. However, this value is dependent on many factors, such as the time of year or the types of travellers going through security. Typically, in the winter, passenger throughput is lower since people tend to wear winter clothes and boots, which take more time to take off than summer attire. However, in the summer (particularly during school holidays), more families with children travel, who take longer to get through security compared to regular business travellers.

At the time of writing this thesis, Edinburgh Airport is in the process of rolling

out new hand luggage scanners (Department for Transport, 2022) and currently has one new scanner in operation. The introduction of these new scanners comes with expected changes to the staffing requirements of the security hall. At the moment, the CSS estimates that two to three additional officers will be needed to operate a cell, leading to an expected throughput of 200 additional passengers per hour. However, the exact numbers remain to be determined as more scanners are set up and the staff becomes more familiar with their operation.

Operating the lanes and metal detectors makes up the bulk of the tasks that security officers have to cover. However, there are some additional positions that need to be covered throughout the day and night. These are called *post/extra* and cover supervisory roles within the airport's security hall and external posts in other areas of the airport. These shifts differ from the security lane ones in that they are constantly required and, therefore, have fixed start and end times. They are sometimes referred to as *constant shifts* and split into three shift types (morning, afternoon and night). In contrast, security hall shifts are referred to as *variable shifts* and are split into just morning and afternoon shifts.

#### 4.2.2.2 Contractual Structure

Edinburgh Airport's security officers are split into multiple contracts and arranged into teams within the contracts — ideally, teams will always work together and not be split up. Team sizes can differ between contracts, but all teams that share a contract must have the same team size. This ensures that teams of the same contract can rotate through the same schedule, which is essential to ensure the roster's longevity.

The contracts differ in working regulations such as weekly hours, allowed shift types or number of workdays per week. They specify:

CC1 Maximum weekly work hours

CC2 Maximum shift duration

CC3 Minimum shift duration

CC4 Permitted shift types

CC5 Permitted work days

CC6 Maximum number of work days per calendar week (Monday - Sunday).

#### 4.2.2.3 Other Work Regulations

In addition to the contract-specific constraints, there also exists a set of general constraints that apply to all security officers equally. These rules originate from various sources, such as the potentially very long timeframe the roster will be used for, which enforces the use of cyclic rosters. Other sources include (union agreed/enforced) staff preferences, such as minimizing single days off over the roster, or the system setup, such as opening and closing times of the security hall and their impact on shift start and end times.

GC1 All Schedules have to be cyclic to ensure that they can be used for as long as needed.

GC2 All workforce demand (constant and variable) needs to be satisfied — there can be over scheduling but no under scheduling

GC3 Forward rotation has to be enforced at all times, meaning that consecutive shifts (without a day off in between) have to either be of the same type or the second shift has to be later in the day than the first.

GC4 There can be at most five consecutive work days.

GC5 There have to be two consecutive days off in any interval of three calendar weeks.

GC6 There can be no single days off in the schedule.

GC7 The start and end times of constant shifts match the corresponding shift type's defined start and end times.

GC8 If necessary, start/end times of shifts can extend up to two hours beyond the limits of their respective shift type.

GC9 Variable shifts start and end with a 15-minute briefing.

GC10 Variable shifts start and end within security hall opening times.

GC11 The average shift length in the roster cannot be greater than the maximum average shift length.

On top of this list of general constraints, there are some special cases to be considered as well. Firstly, there are some constraints that the CSS would like to

enforce, but these often lead to infeasibilities in the roster. Two such examples are avoiding maximum day work stretches and not scheduling single days off (GC6). Instead of completely removing them from the roster, they can be set up as an objective function that needs to be minimised.

O1 Keep maximum day work stretches to a minimum

O2 Keep single days off to a minimum (instead of GC6)

All shifts assigned to security officers also need to contain breaks to adhere to working regulations. Breaks are assigned based on shift duration. The airport has a set of break patterns that consist of one to three breaks with durations between 15-45 minutes. Each break pattern corresponds to a range of shift durations that are required to have this pattern. For example, there will be one pattern for shifts ranging from 0-4 hours, the next pattern applies to shifts of duration 4.25-5 hours, and so on (see Table 4.1).

| Shift Duration (hours) | Break 1 (min.) | Break 2 (min.) | Break 3 (min.) |
|---|---|---|---|
| $\leq 4$ | 15 | 0 | 0 |
| $4.25 - 5$ | 30 | 0 | 0 |
| $5.25 - 7$ | 30 | 15 | 0 |
| $7.25 - 8$ | 30 | 30 | 0 |
| $8.25 - 9$ | 45 | 30 | 0 |
| $9.25 - 10$ | 45 | 45 | 0 |
| $10.25 - 11$ | 30 | 30 | 30 |
| $11.25 - 12$ | 45 | 30 | 30 |

Table 4.1: Break patterns for different ranges of shift durations.

In addition, all work periods within a shift need to be at least 1.5 hours long, which leads to two break-specific constraints.

BC1 Break patterns have to match the prescribed pattern for the shift length.

BC2 There needs to be at least 1.5 hours of work time between breaks and between a break and shift start/end.

When using the roster tool, the CSS can pre-define some shift and days-off assignments in the form of a *partial roster*. The roster tool must not change these

assignments — except for a small buffer to adjust shift times. When a partial roster is provided, the roster tool will use the remaining staff's availability to fill the holes and satisfy any demand not met by the partial roster. It is important to note that any assignments in the partial roster must comply with all rules and regulations that apply to the final roster.

PC1 Partial roster assignments need to be unchanged

PC2 If necessary, shift times can be changed by 30 minutes from the partial roster inputs.

Overall, the rules outlined in this section must be adhered to if the resulting roster is considered suitable for use by Edinburgh Airport's security officers. In order to ensure this, they are added as constraints or objective functions to the mathematical formulation of the rostering problem that is presented in the following Section (4.3).

## 4.3 Problem Formulation

This section provides the mathematical formulations of the models that comprise Edinburgh Airport's roster tool. It starts with an overview of the decomposition approach (Section 4.3.1) and the demand aggregation process (Section 4.3.2) before presenting the models to fix the rotating schedules (Section 4.3.3), finalize tours (Section 4.3.4) and assign breaks (Section 4.3.5).

### 4.3.1 Decomposition Approach

Rostering Problems are notoriously difficult to solve (Ernst et al., 2004b), something that is only exacerbated by problem size, and Edinburgh Aiport's staff scheduling problem is no exception to this. As a result, the problem had to be decomposed into several smaller parts that could be solved in a reasonable amount of time — in this case, reasonable means minutes rather than hours as one of the applications of the final product would be its use during discussions with union representatives. Figure 4.3 provides an overview of the particular decomposition approach used to model the airport's scheduling problem.

### 4.3.2 Prep - RT1: Workloads and Model Week creation (RT1)

The first step in the scheduling process involves determining the expected workloads for the upcoming season. It is performed by the airport's analytics team rather than directly by the roster tool but is still important to the scheduling process overall, which is why it is included in Figure 4.3 with a different background colour. During this step, the airport's analytics team accumulates the flight schedules provided by airlines and passenger arrival data from previous seasons to predict the passenger arrival patterns for the upcoming season. However, having expected passenger arrival numbers is not enough to start producing rosters. The analysts need to first translate expected passenger arrival numbers into the expected number of open lanes that are needed to cope with the arriving passengers. To do this, they use the *expected passenger throughput*, which is an estimate of how many passengers can pass through a lane in a given timeframe (e.g., 15 minutes). Passenger throughputs will vary from lane to lane (e.g., Lane 1, which is used for people with reduced mobility or families, has a lower passenger throughput than a regular lane) and by season (e.g., in winter, passenger throughput is normally lower because people tend to wear more clothes

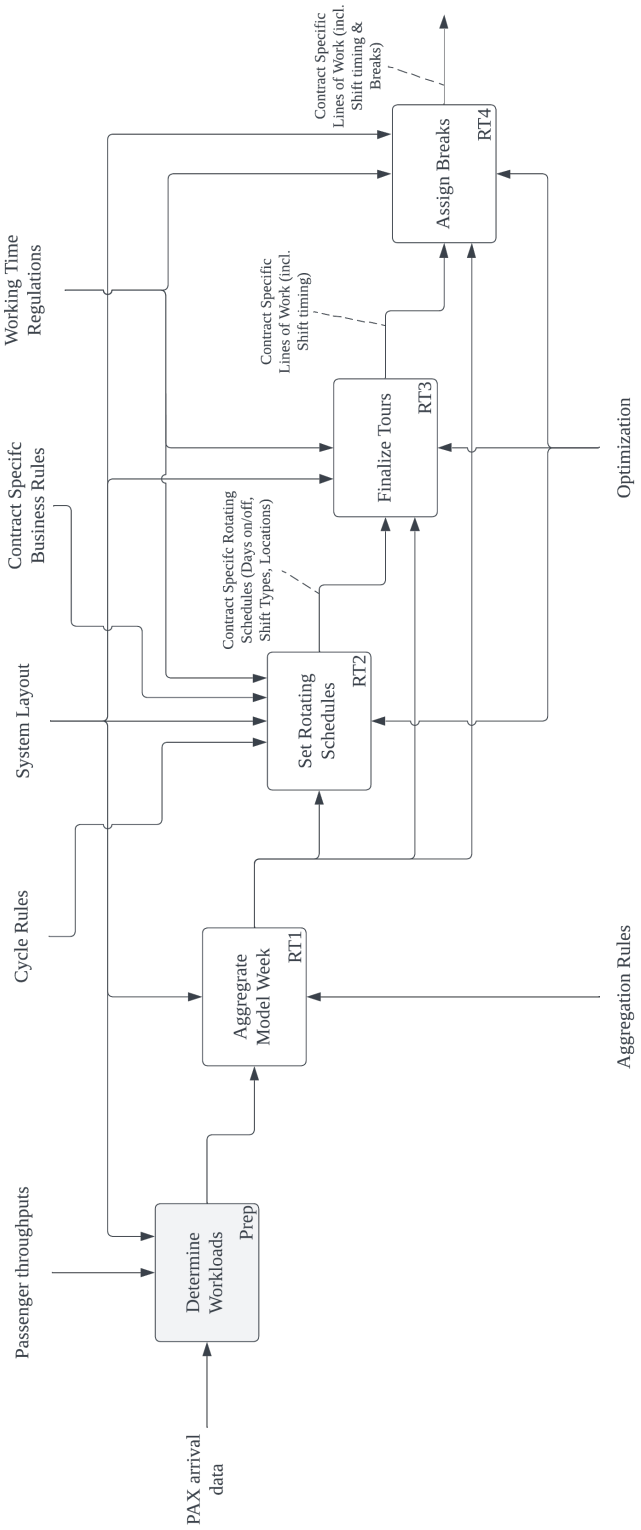



Figure 4.3: Roster Tool Decomposition.

which need to be taken off during security checks). Combining the expected passenger arrival numbers with passenger throughputs, lane ordering, and the resulting workforce requirements to open/close an additional lane will lead to the expected workforce requirements of variable shifts in the security hall. Adding the workforce requirements for constant shifts to those for variable shifts produces the expected workforce requirements for the season. This final demand forecast provides the CSS (and now the roster tool) with the expected staff demand for each 15-minute interval — also referred to as *bins* — in the upcoming season.

This is usually the point at which the CSS takes over to create the schedules. However, for the roster tool, an additional step (module RT1, Figure 4.3) is necessary. The way rotating schedules work is that the available workforce numbers are identical from a week-to-week perspective. So, to create such a schedule, a *model* week that represents a typical weekly demand pattern in the forecast is needed. Typically, a model is created with aggregation rules that best reflect the user's need. This could be the average/maximum/minimum demand observed over the season or an entirely user-specific function. In the case of Edinburgh Airport, the aim is to cover as much (if not all) of the expected demand as possible in the forecast — the airport would rather have an overstaffed security hall without queues than an understaffed security hall with long queues of disgruntled passengers. So, the model uses an aggregation rule that picks for every day and bin of the model week the $m^{\text{th}}$ percentile of demand that occurs in the demand forecast for that bin and weekday. To fully avoid any understaffing according to the demand forecast, the aggregation level should be $m = 100$. However, the demand forecast is not entirely accurate, and changes to demand patterns will occur. As a result, disregarding the highest peaks of demand by using an aggregation level below 100 (e.g., in the range of 80-90) can lead to better-performing rosters. The roster tool is formulated in such a way that the CSS can play with the aggregation level to compare the resulting rosters and pick the one that suits all requirements the best.

### 4.3.3 RT2: Set Rotating Schedules

Once the model week is defined, the roster tool starts to build the roster. The first step of which is setting the rotating schedule. The following problem parameters are needed to formulate this problem.

$nl$ number of locations.

$nc$ number of contracts.

$nt$ number of teams (lines).

$nr$ number of shift types.

$g$ line length — most often seven days (one working week, which is normally the 'planning period' in rostering parlance).

$x$ integer used for defining the consecutive days off requirement. A typical form is to ensure that at least every interval of $x$ calendar weeks in the schedule includes at least one occurrence of two consecutive days off.

$w\_up$ maximum length of consecutive working days.

$w\_id$ ideal length of consecutive working days.

$\mathscr{C} = \{1,\ldots,nc\}$ set of all contracts .

$\mathscr{T} = \{1,\ldots,nt\}$ set of all teams — teams of the same contract have consecutive indices.

$\mathscr{D} = \{1,\ldots,g\}$ list of days in one line. $\mathscr{D}_c^+$ represents the set of permissible work days for contract $c \in \mathscr{C}$ and $\mathscr{D}_c^-$ represents the set of days that teams of contract $c \in \mathscr{C}$ are not allowed to work, where $\mathscr{D}_c^+ \cup \mathscr{D}_c^- = \mathscr{D}$ and $\mathscr{D}_c^+ \cap \mathscr{D}_c^- = \{\}$.

$\mathscr{A}_c$ set of permissible shift types for contract $c \in \mathscr{C}$ — often: morning ($M$), afternoon ($A$) and night ($N$). An additional option for a 'day off'(denoted by $O$) also exists. $\mathscr{A}_c^+$ denotes the set of all permissible shift types for contract $c$ including days off. We also consider $\bigcup_{c=1}^{nc} \mathscr{A}_c = \mathscr{A}$ and $\mathscr{A} \cup \{O\} = \mathscr{A}^+$.

$\mathscr{L} = \{1 \ldots nl\}$ set of all locations of shifts, where $\mathscr{L}^+$ denotes all locations including days off (denoted by $O$).

$d\_up_c$ maximum number of working days per line for teams under contract $c \in \mathscr{C}$.

$ts_c$ team size for contract $c \in \mathscr{C}$ .

$ft_c$, $lt_c$ (indices of the) first and last team on contract $c \in \mathscr{C}$. These indices are needed to model the cyclic nature of the constraint (i.e. once a team has reached the last line of the schedule for their contract ($lt_c$) it continues on to the first line of the contract ($ft_c$)).

$cr_i$ contract of team $i \in \mathscr{T}$.

$ld_l$ constant workforce demand of location $l \in \mathscr{L}$.

$lc_l$ binary variable indicating whether location $l \in \mathscr{L}$ corresponds to a constant shift (1) or a variable shift (0).

$R$ requirement matrix — each element $R_{sh,j}$ encodes the highest workforce demand in the model week for the timeframe corresponding to shift type $sh \in \mathscr{A}$ on day $j \in \mathscr{D}$ — the highest demand has to be used here to ensure that enough security officers are assigned for that shift type to cover all demand in the model week during the next scheduling step (RT3).

$PS_{i,j}$ $\in \mathscr{A}^+, 1 \leq i \leq nt, 1 \leq j \leq g$ encodes the shift type assigned to line $i$ on day $j$ in the partial roster input. If no shift or day off is assigned, the value is -1, which indicates to the roster tool that it can assign any shift type or day off — within the given constraints.

$PL_{i,j}$ $\in \mathscr{L}^+, 1 \leq i \leq nt, 1 \leq j \leq g$ encodes the location assigned to line $i$ on day $j$ in the partial roster input. If no shift or day off is assigned, the value is -1, which indicates to the roster tool that it can assign any location or day off — within the given constraints.

The decision variables for the problem represent the rotating schedule of shift types and the associated shift locations (variable shift or constant shift):

$S$ $nt \times g$ matrix where each element $S_{i,j} \in \mathscr{A}^+$, $1 \leq i \leq nt$, $1 \leq j \leq g$ encodes the shift type assigned to line $i$ on day $j$.

$L$ $nt \times g$ matrix where each element $L_{i,j} \in \mathscr{L}^+$, $1 \leq i \leq nt$, $1 \leq j \leq g$ encodes the location of the shift assigned in $S_{i,j}$

$v\_off_i$ binary indicator encoding whether team $i \in \mathscr{T}$ has been allocated two consecutive days off in their starting week (1) or not (0).

To help model the rotation through the lines of teams of the same contract, two functions are introduced as well:

$$u_c(i) = \begin{cases} lt_c - (ft_c - i - 1) \bmod (lt_c - ft_c + 1), & i < ft_c \\ (i - lt_c - 1) \bmod (lt_c - ft_c + 1) + ft_c, & ft_c \leq i \end{cases}$$

$$v_c(i,j) = \begin{cases} (i,j), & 1 \leq j \leq g, \forall i \in \mathscr{T} \\ (u_c(i + \lfloor j-1/g \rfloor)),\ (j-1) \bmod g + 1), & g < j, \forall i \in \mathscr{T} \end{cases}$$

Both definitions make use of modular arithmetic. The length of a line is normally $g = 7$ (the seven days of the calendar week). With that being said, function $u_c(i)$ maps the rotation of teams through subsequent weeks of the schedule, ensuring that the schedule for the days of the immediately subsequent week is the one found in the very next row (line) of the roster of the corresponding contract. Function $v_c(i, j)$ instead makes sure that, for any team $i$, after following the indications of row (line) $i$ for the first week, the shift/day off for the first day of the following week is the first on the row (line) immediately next according to the roster of the corresponding contract, and so on.

Armed with the above concepts/notation, the following basic constraint satisfaction problem can be formulated:

$$L_{i,j} = O \leftrightarrow S_{i,j} = O \qquad i \in \mathscr{T}, j \in \mathscr{D} \tag{4.1}$$

$$\sum_{i \in \mathscr{T}} (ts_{cr_i}(S_{i,j} = sh)) = R_{sh,j} \qquad j \in \mathscr{D}, sh \in \mathscr{A} \tag{4.2}$$

$$\sum_{i \in \mathscr{T}} (S_{i,j} = sh)(ts_{cr_i}(lc_{L_{i,j}} = 1)) = \sum_{l \in \mathscr{L}} ld_l \qquad j \in \mathscr{D}, sh \in \mathscr{A} \tag{4.3}$$

$$PS_{i,j} = sh \rightarrow S_{i,j} = sh \qquad j \in \mathscr{D}, i \in \mathscr{T}, sh \in \mathscr{A}^+ \tag{4.4}$$

$$PL_{i,j} = l \rightarrow L_{i,j} = l \qquad j \in \mathscr{D}, i \in \mathscr{T}, l \in \mathscr{L}^+ \tag{4.5}$$

Constraints (4.1) ensures that the days-off entries in the shift type matrix ($S$) and the location matrix ($L$) match each other. Demand satisfaction (GC2) is enforced by constraint (4.2) and constraint (4.3) ensures that the correct distribution of variable and constant shifts is assigned. Constraints (4.4) - (4.5) ensure that the partial roster assignments are enforced (PC1).

In addition to the basic demand satisfaction problem above, the following constraints and objectives can be added as required by the CSS. All of the constraints are set up to ensure the cyclic nature of the roster (GC1) — meaning they also hold when a security officer moves from the last line in the schedule to the first.

CC4 Permitted shift types

$$S_{i,j} \in \mathscr{A}^+_{cr_i} \qquad i \in \mathscr{T}, j \in \mathscr{D} \tag{4.6}$$

CC5 Permitted work days

$$S_{i,j} \in \mathscr{A}^+_{cr_i} \qquad i \in \mathscr{T}, j \in \mathscr{D}^+_{cr_i} \tag{4.7}$$

$$S_{i,j} = 0 \qquad i \in \mathscr{T}, j \in \mathscr{D}^-_{cr_i} \tag{4.8}$$

CC6 Maximum number of work days per calendar week (Monday - Sunday)

$$\sum_{j \in \mathscr{D}} (S_{i,j} \neq 0) \leq d\_up_{cr_i} \qquad i \in \mathscr{T} \tag{4.9}$$

GC3 Forward rotation has to be enforced at all times, meaning that consecutive shifts (without a day off in between) have to either be of the same type or the second shift has to be later in the day than the first.

$$S_{i,j} = A \rightarrow S_{v_{cr_i}(i,j+1)} \neq M \qquad i \in \mathscr{T}, j \in \mathscr{D} \tag{4.10}$$

$$S_{i,j} = N \rightarrow S_{v_{cr_i}(i,j+1)} \neq A \qquad i \in \mathscr{T}, j \in \mathscr{D} \tag{4.11}$$

$$S_{i,j} = N \rightarrow S_{v_{cr_i}(i,j+1)} \neq M \qquad i \in \mathscr{T}, j \in \mathscr{D} \tag{4.12}$$

GC4 There can be at most w_*up* consecutive work days.

$$\text{sliding\_sum}(1, w\_up, w\_up+1, [S_{u_c(i),j} \neq 0 | i \in ft_c \ldots lt_c + 1, j \in \mathscr{D}]) \quad c \in \mathscr{C} \tag{4.13}$$

GC5 There have to be two consecutive days off in any interval of $x$ calendar weeks..

$$v\_off_i = 1 \leftrightarrow \exists_{j \in 1 \ldots g-1} (S_{i,j} \neq 0 \wedge S_{i,j+1} \neq 0) \qquad i \in \mathscr{T} \tag{4.14}$$

$$\text{sliding\_sum}(1, x, x, [v\_off_{u_c(i)} | i \in ft_c \ldots lt_c + x]) \qquad c \in \mathscr{C} \tag{4.15}$$

GC6 There can be no single days off in the schedule.

$$S_{i,j} \neq 0 \wedge S_{v_{cr_i}(i,j+1)} = 0 \rightarrow S_{v_{cr_i}(i,j+2)} = 0 \qquad i \in \mathscr{T}, j \in \mathscr{D} \tag{4.16}$$

The CSS has the option to change the objective of this model from *constraint satisfaction* to:

O1 Keep maximum day work stretches to a minimum

$$\min \sum_{i \in \mathscr{T}, j \in \mathscr{D}} (\text{count}_{d \in 0 \ldots w\_up-1} (S_{i,j+d} \neq 0) = w\_up) \tag{4.17}$$

O2 Keep single days off to a minimum (instead of GC6)

$$\min \sum_{i \in \mathscr{T}, j \in \mathscr{D}} (S_{v_{cr_i}(i,j-1)} \neq 0 \wedge S_{i,j} = 0 \wedge S_{v_{cr_i}(i,j+1)} \neq 0) \tag{4.18}$$

### 4.3.4 RT3: Finalize Tours

With the rotating schedules set, the next step in the process is to 'finalize the tours' (i.e. set the start and end times of shifts). As part of this model, the following rules need to be considered and adhered to.

CC1 Maximum weekly work hours

CC2 Maximum shift duration

CC3 Minimum shift duration

GC7 The start and end times of constant shifts match the corresponding shift type's defined start and end times.

GC8 If necessary, start/end times of shifts can extend up to two hours beyond the limits of their respective shift type.

GC9 Variable shifts start and end with a 15-minute briefing.

GC10 Variable shifts start and end within security hall opening times.

GC11 The average shift length in the roster cannot be greater than the maximum average shift length.

PC2 If necessary, shift times can be changed by 30 minutes from the partial roster inputs.

The main goal of this model is to decide the start and end times of shifts, which involves calculating the number of staff that are working at any time and their weekly hours. More formally, the decision variables of the model are:

$W$ Working matrix — each element $W_{b,j}$ encodes the number of employees working during bin $b \in \mathscr{B}$ on day $j \in \mathscr{D}$

$N$ Non-productive paid time (NPPT) matrix — each element $N_{b,j}$ encodes the surplus workforce during bin $b \in \mathscr{B}$ for day $j \in 1 \dots g$

$T$ start ($T_{s,start}$) and end ($T_{s,end}$) time of each shift $s \in \mathscr{S}$

$sd_s$ duration of shift $s \in \mathscr{S}$

$sd\_avg_c$ Average shift duration for contract $c \in \mathscr{C}$

$wh_i$ weekly hours worked by team $i \in \mathscr{T}$

$ns_c$ number of shifts worked by contract $c \in \mathscr{C}$

Paired with the following list of problem parameters the airport's Tour Scheduling Problem can be formulated:

$nb$ number of Bins (time intervals) in a day

$\mathscr{B}$ set of bins

$ns$ number of Shifts

$\mathscr{S}$ set of all shifts

$\mathscr{S}_p \subset \mathscr{S}$ set of all shifts that were pre-assigned in the partial roster

$c\_start_{sh}$ start time of constant shifts of shift type $sh \in \mathscr{A}$

$c\_end_{sh}$ end time of constant shifts of shift type $sh \in \mathscr{A}$

$v\_start_{sh}$ earliest start time of variable shifts of shift type $sh \in \mathscr{A}$

$v\_end_{sh}$ latest end time of variable shifts of shift type $sh \in \mathscr{A}$

$sd_{\max_c}$ maximum shift duration for teams of contract $c \in \mathscr{C}$

$sd_{\min_c}$ minimum shift duration for teams of contract $c \in \mathscr{C}$

$wh_{\max_c}$ maximum weekly hours worked by teams of contract $c \in \mathscr{C}$

$wh\_avg_{\max_c}$ maximum average weekly hours of teams in contract $c \in \mathscr{C}$

$wh\_avg_{\min_c}$ minimum average weekly hours of teams in contract $c \in \mathscr{C}$

$sl_s$ location of shift $s \in \mathscr{S}$, where $sl_s \in \mathscr{L}$

$sj_s$ day of shift $s \in \mathscr{S}$, where $sj_s \in \mathscr{D}$

$sa_s$ shift type of shift $s \in \mathscr{S}$, where $sa_s \in \mathscr{A}^+$

$sc_s$ binary indicator whether shift $s \in \mathscr{S}$ is variable (0) or constant (1)

$sr_s$ number of employees assigned to shift $s \in \mathscr{S}$

$D$ demand matrix — each entry $D_{b,j}$ encodes the workforce requirement for bin $b \in \mathscr{B}$ on day $j \in \mathscr{D}$

$SH$ Shift roster — each entry $S_{i,j} \in \mathscr{S}$ encodes which shift is worked by team $i \in \mathscr{T}$ on day $j \in \mathscr{D}$

$PT$ start ($PT_{s,start}$) and end ($PT_{s,end}$) time of each shift $s \in \mathscr{S}$ that was pre-assigned in the partial roster. If shifts were not pre-assigned in the partial roster, the entries are $-1$.

$a\_max$ maximum allowed adjustment (number of bins) to pre-assigned shift start and end times in the partial roster.

The objective at this stage is to minimize non-productive paid time (NPPT)— which essentially represents overscheduling in the roster.

$$\min \sum_{j \in 1 \ldots g} \sum_{b \in \mathscr{B}} N_{b,j} \tag{4.19}$$

$$PT_{s,start} - a\_max \leq T_{s,start} \leq PT_{s,start} + a\_max \quad s \in \mathscr{S}_p \tag{4.20}$$

$$PT_{s,end} - a\_max \leq T_{s,end} \leq PT_{s,end} + a\_max \quad s \in \mathscr{S}_p \tag{4.21}$$

$$sd_s = T_{s,end} - T_{s,start} + 1 \quad s \in \mathscr{S} | sa_s \neq night \tag{4.22}$$

$$sd_s = T_{s,end} + (nb - T_{s,start}) + 1 \quad s \in \mathscr{S} | sa_s = night \tag{4.23}$$

$$T_{s,start} < T_{s,end} \quad s \in \mathscr{S} | sa_s \neq night \tag{4.24}$$

$$T_{s,start} > T_{s,end} \quad s \in \mathscr{S} | sa_s = night \tag{4.25}$$

$$T_{s,start} = c\_start_{sa_s} \quad s \in \mathscr{S} | sc_s = 1 \tag{4.26}$$

$$T_{s,end} = c\_end_{sa_s} \quad s \in \mathscr{S} | sc_s = 1 \tag{4.27}$$

$$T_{s,start} \geq v\_start_{sa_s} \quad s \in \mathscr{S} | sc_s = 0 \tag{4.28}$$

$$T_{s,end} \leq v\_end_{sa_s} \quad s \in \mathscr{S} | sc_s = 0 \tag{4.29}$$

$$sd_s \leq sd_{\max_{cr_i}} \quad s \in \mathscr{S}, i \in \mathscr{T} | SH_{i,sj_s} = s \tag{4.30}$$

$$sd_s \geq sd_{\min_{cr_i}} \quad s \in \mathscr{S}, i \in \mathscr{T} | SH_{i,sj_s} = s \tag{4.31}$$

$$wh_i = \sum_{j \in \mathscr{D}} (SH_{i,j} \neq 0) sd_{SH_{i,j}} \quad i \in \mathscr{T} \tag{4.32}$$

$$wh_i \leq wh_{\max_{cr_i}} \quad i \in \mathscr{T} \tag{4.33}$$

$$ns_c = \sum_{j \in \mathscr{D}} \sum_{i \in ft_c \ldots lt_c} (SH_{i,j} \neq 0) \quad c \in \mathscr{C} \tag{4.34}$$

$$sd\_avg_c = \sum_{j \in \mathscr{D}} \sum_{i \in ft_c \ldots lt_c} \frac{(SH_{i,j} \neq 0) sd_{SH_{i,j}}}{ns_c} \quad c \in \mathscr{C} \tag{4.35}$$

$$sd\_avg_c \leq wh\_avg_{\max_c} \quad c \in \mathscr{C} \tag{4.36}$$

$$sd\_avg_c \geq wh\_avg_{\min_c} \quad c \in \mathscr{C} \tag{4.37}$$

$$W_{b,j} = \sum_{s \in \mathscr{S}} \Big( (sj_s = j)(sc_s = 0)(T_{s,start} < b) (T_{s,end} > b)(sr_s) \Big) \quad b \in \mathscr{B}, j \in \mathscr{D} \tag{4.38}$$

$$W_{b,j} \geq D_{b,j} \quad b \in \mathscr{B}, j \in \mathscr{D} \tag{4.39}$$

$$N_{b,j} = W_{b,j} - D_{b,j} \quad b \in \mathscr{B}, j \in \mathscr{D} \tag{4.40}$$

Constraints (4.20) and (4.21) ensure that pre-defined shifts from the partial roster are not adjusted by more than the maximum allowed number of bins. Constraints (4.22) and (4.23) define the duration of shifts and Constraints (4.24) and (4.25) ensure that the start and end times of shifts are ordered correctly. Note, that the definition and order of night shifts are defined differently to other shifts. This is because night shifts pass midnight, which must be accounted for as part of the formulation. Constraints (4.26) and (4.27) set the start and end times of constant shifts, while constraints (4.28) and (4.29) limit the start and end times of variable shifts. Constraints (4.31) and (4.30) ensure that the duration of each shift $s \in \mathscr{S}$ is within the contractual limits of each team $i \in \mathscr{T}$ assigned to work that shift. Constraint (4.32) defines the weekly hours worked by each team $i \in \mathscr{T}$ and constraint (4.33) puts an upper limit on the number of weekly hours assigned to each team. Constraint (4.34) assigns the number of shifts worked by all teams under contract $c \in \mathscr{C}$ which is then used to calculate the average shift duration for each contract (Constraint (4.35)). The average shift duration for each contract is limited by constraints (4.36) and (4.37). Constraint (4.38) evaluates the number of employees working variable shifts for each time bin $b \in \mathscr{B}$ on each day $j \in \mathscr{D}$ while constraint (4.39) ensures that all variable demand is covered. Finally, constraint (4.40) evaluates the non-productive paid time for each time bin $b \in \mathscr{B}$ on each day $j \in \mathscr{D}$ which is minimized as part of the objective of this problem.

### 4.3.5 RT4: Assign Breaks

The last step of the roster tool is to assign breaks. Which break pattern needs to be assigned depends on the shift duration. As breaks reduce the number of available staff members to cover demand, just assigning them without the possibility of adjusting

shift durations can lead to infeasibility. As a result, the model is allowed to also extend shifts from the assignments made by the previous module to accommodate the required breaks. This means that all the constraints outlined in the previous section also apply to this model. In addition, the following two rules need to be adhered to:

BC1 Break patterns have to match the prescribed pattern for the shift length.

BC2 There needs to be at least 1.5 hours of work time between breaks and between a break and shift start/end.

The additional problem parameters for this model are:

$ns_c$ number of shifts worked by contract $c \in \mathscr{C}$

$nsd$ number of possible shift durations

$\mathscr{SD}$ set of all possible shift durations

$x$ minimum buffer (in bins) between breaks and the start/end of a shift or between two consecutive breaks.

$T$ start ($T_{s,start}$) and end ($T_{s,end}$) time of each shift $s \in \mathscr{S}$ as assigned by the tour scheduling model.

$sd_s$ duration of shift $s \in \mathscr{S}$ as assigned by the tour scheduling model.

$nbp$ total number of break patterns

$\mathscr{P}$ $= [1 \dots nbp]$ set of all break patterns — ordered by the minimum shift duration that requires the

$min\_sd_p$ minimum shift duration for which break pattern $p \in \mathscr{P}$ is suitable pattern

$nb_p$ number of breaks in break pattern $p \in \mathscr{P}$

$bn_{\max}$ maximum number of breaks per shift

$BP$ $nbp \times bn_{\max}$ matrix where each element $BP_{p,n}$ encodes the duration of break number $n \in 1 \dots bn_{\max}$ in break pattern $p \in \mathscr{P}$. The matrix is assigned 0 for $n > nb\_p$

The decision variables here consist of the new shift times and work/break patterns in addition to the values needed to calculate the average shift duration. Formally, the list of decision variables is:

$W\_ext$ Working matrix — each element $W\_ext_{b,j}$ encodes the number of employees working during bin $b \in \mathscr{B}$ on day $j \in \mathscr{D}$

$WB$ Work/Break matrix — each element $WB_{b,s}$ encodes the state (0 = off, 1= working, 2 = briefing, 3 = break) for all bins $b \in \mathscr{B}$ and each shift $s \in \mathscr{S}$

$T\_ext$ start ($T\_ext_{s,start}$) and end ($T\_ext_{s,end}$) time of each shift $s \in \mathscr{S}$

$sd\_ext_s$ duration of shift $s \in \mathscr{S}$

$sd\_ext\_avg_c$ Average shift duration for contract $c \in \mathscr{C}$

$wh\_ext_i$ weekly hours worked by team $i \in \mathscr{T}$

$bp_s$ break pattern assigned to shift $s \in \mathscr{S}$

$BT_{s,n}$ start bin of break $n \in 1 \dots bn_{\max}$ of shift $s \in \mathscr{S}$

Using the above presented notation, the constraints of the model can be expressed as follows.

$$T\_ext_{s,start} = T_{s,start} \quad s \in \mathscr{S} | sc_s = 1 \tag{4.41}$$

$$T\_ext_{s,end} = T_{s,end} \quad s \in \mathscr{S} | sc_s = 1 \tag{4.42}$$

$$T\_ext_{s,start} \leq T_{s,start} \quad s \in \mathscr{S} | sc_s = 0 \tag{4.43}$$

$$T\_ext_{s,end} \geq T_{s,end} \quad s \in \mathscr{S} | sc_s = 0 \tag{4.44}$$

$$T\_ext_{s,start} \geq v\_start_{sa_s} \quad s \in \mathscr{S} | sc_s = 0 \tag{4.45}$$

$$T\_ext_{s,end} \leq v\_end_{sa_s} \quad s \in \mathscr{S} | sc_s = 0 \tag{4.46}$$

$$sd\_ext_s = T\_ext_{s,end} - T\_ext_{s,start} + 1 \quad s \in \mathscr{S} \tag{4.47}$$

$$sd\_ext_s \leq sd_{\max_{cr_i}} \quad s \in \mathscr{S}, i \in \mathscr{T} | SH_{i,sj_s} = s \tag{4.48}$$

$$wh\_ext_i = \sum_{j \in \mathscr{D}} (SH_{i,j} \neq 0) sd_{SH_{i,j}} \quad i \in \mathscr{T} \tag{4.49}$$

$$wh\_ext_i \leq wh_{\max_{cr_i}} \quad i \in \mathscr{T} \tag{4.50}$$

$$sd\_ext\_avg_c = \sum_{j \in \mathscr{D}} \sum_{i \in ft_c \dots lt_c} \frac{(SH_{i,j} \neq 0) sd\_ext_{SH_{i,j}}}{ns_c} \quad c \in \mathscr{C} \tag{4.51}$$

$$sd\_ext\_avg_c \leq wh\_avg_{\max_c} \quad c \in \mathscr{C} \tag{4.52}$$

$$bp_s = \max_{p \in \mathscr{P}} \{\min\_sd_p \leq sd\_ext_s\} \quad s \in \mathscr{S} \tag{4.53}$$

$$BT_{s,n} = 0 \leftrightarrow (nb_{bp_s} < n) \quad n \in 1 \dots bn_{\max}, \; s \in \mathscr{S} | nb_{bp_s} < bn_{\max} \tag{4.54}$$

$$\text{strictly\_increasing}[BT_{s,n} | n \in 1 \dots nb_{bp_s}] \quad s \in \mathscr{S} \tag{4.55}$$

$$T\_ext_{s,start} \leq BT_{s,1} + x \quad s \in \mathscr{S} \tag{4.56}$$

$$BT_{s,nb_{bp_s}} + BP_{bp_s,nb_{bp_s}} + x \leq T\_ext_{s,end} \quad s \in \mathscr{S} \tag{4.57}$$

$$BT_{s,n} + BP_{bp_s,n} + x < BT_{s,n+1} \quad s \in \mathscr{S}, \; n \in 1 \dots nb_{bp_s} - 1 | nb_{bp_s} > 1 \tag{4.58}$$

$$WB_{b,s} = 0 \leftrightarrow (b < T\_ext_{s,start}) \vee (T\_ext_{s,end} < b) \quad b \in \mathscr{B}\_ext, s \in \mathscr{S} \tag{4.59}$$

$$WB_{b,s} = 2 \leftrightarrow (b = T\_ext_{s,start}) \vee (T\_ext_{s,end} = b) \quad b \in \mathscr{B}\_ext, s \in \mathscr{S} | sc_s = 0 \tag{4.60}$$

$$WB_{b,s} = 1 \leftrightarrow (b = T\_ext_{s,start}) \vee (T\_ext_{s,end} = b) \quad b \in \mathscr{B}\_ext, s \in \mathscr{S} | sc_s = 1 \tag{4.61}$$

$$WB_{b,s} = 3 \leftrightarrow (BT_{s,n} \leq b) \wedge (b \leq BT_{s,n} + BP_{bp_s,n}) \quad s \in \mathscr{S}, n \in 1 \dots nb_{bp_s} \tag{4.62}$$

$$WB_{b,s} = 1 \leftrightarrow (T\_ext_{s,start} < b) \wedge (b < BT_{s,1}) \quad s \in \mathscr{S} \tag{4.63}$$

$$WB_{b,s} = 1 \leftrightarrow (BT_{s,nb_{bp_s}} < b) \wedge (b < T\_ext_{s,end}) \quad s \in \mathscr{S} \tag{4.64}$$

$$WB_{b,s} = 1 \leftrightarrow (BT_{s,n-1} + BP_{bp_s,n-1} < b) \wedge (b < BT_{s,n}) \quad s \in \mathscr{S}, \; n \in 2 \dots nb_{bp_s} | nb_{bp_s} > 1 \tag{4.65}$$

$$W\_ext_{b,j} = \sum_{s \in \mathscr{S}} \Big( (sj_s = j)(sc_s = 0)(WB_{s,b} = 1)(sr_s) \Big) \quad b \in \mathscr{B}, j \in \mathscr{D} \tag{4.66}$$

$$W\_ext_{b,j} \geq D_{b,j} \quad b \in \mathscr{B}, j \in 1 \dots g \tag{4.67}$$

Constraints (4.41) - (4.42) ensure that the start and end times of constant shifts are unchanged after break assignment and constraints (4.43) - (4.44) ensure that variable shifts are only extended beyond the times set by the tour scheduling model. Constraints (4.45) - (4.46) limit the start and end times of shifts to be within Security hall opening hours. The new shift durations are defined by constraint (4.47) and constraint (4.48) ensures that the new shift durations stay below the pre-defined maximum shift duration. The weekly hours for each team are defined by constraint (4.49) and limited to stay below the maximum allowed weekly hours by constraint (4.50). The average weekly hours for each contract are calculated by constraint (4.51) and limited to be below the maximum average shift duration by constraint (4.52).

Constraint (4.53) assigns each shift the required break pattern based on the shift duration. If a shift requires less than the maximum allowed breaks, constraint (4.54) ensures that the excess breaks in the Break Time matrix $BT$ are assigned the value 0 and constraint (4.55) guarantees that the assigned start times in $BT$ are increasing. Constraints (4.56) - (4.57) limit breaks to occur between the start and end of a shift plus a buffer of $x$ bins. Similarly, constraint (4.58) ensures that there is a buffer of $x$ bins between breaks if more than one break is assigned.

Finally, constraints (4.59) - (4.65) set the Work/Break matrix, constraint (4.66) calculates the Working matrix using the Work/Break matrix and constraint (4.67) ensures that all workforce demand is satisfied.

The aim at this stage of the roster tool is to feasibly assign breaks. This means that breaks should be added to shifts with the least amount of changes made to shift durations. Since shifts can only be extended by the model (but not shortened), this can be expressed in the objective function as minimising the total shift duration after assigning breaks.

$$\min \sum_{s \in \mathscr{S}} (sd\_ext_s) \tag{4.68}$$

## 4.4 The Roster Tool

This section gives an overview of the development history of Edinburgh Airport's Roster Tool (Section 4.4.1) and an introduction to the implementation and use of the prototype (Section 4.4.2) before ending in a discussion on the necessary factors to ensure uptake of such a tool (Section 4.4.3).

### 4.4.1 Development History

Edinburgh Airport's journey to finding a purpose-built roster tool for their security workforce scheduling problem started in 2019 with a consulting project aimed at building an Excel-based roster tool that used metaheuristics coded in VBA. The idea was to replicate the CSS's approach to creating rosters and automating it. As a result, the approach heavily relied on system configurations such as lane ordering to create shifts and assign their start and end times.

This project was a success as it fulfilled the brief to produce an Excel-based roster tool that replicated the CSS's steps. However, the tool developed did not solve the airport's problem. It quickly realised that, while the tool did what was initially requested, the actual problem of producing rosters was far more extensive than initially presented. The main issues with this 'failed' tool were expressed as a lack of flexibility and the missing break assignment module, and the tool was labelled as *unusable*.

The work in this thesis started in 2020 without direct consultation with the airport due to COVID and related issues the airport had to deal with. Based on the feedback provided for the initial approach, the idea was to move from metaheuristics to MIP/CP, which would, in theory, provide the required flexibility of changing contracts and days-off requirements without needing to change the underlying model or solution approach. In addition, an initial break assignment model was added, which focused solely on matching the total assigned break duration to the requirement based on the shift duration — no pre-defined break patterns were used.

With the first version of this new approach implemented, an initial meeting with the airport was organised to discuss the collaboration for the duration of the PhD. The CSS was initially very sceptical about the continuation of the project — a very understandable reaction based on the failed consulting project and the resulting cost to the airport — but agreed to provide additional information on the scheduling problem and feedback on the approach. It quickly became apparent that the so-called *lack of*

*flexibility* of the initial tool went far beyond not being able to change contract numbers or the number of weeks that required consecutive days off. What the CSS really wanted was a roster tool that could be used for any scheduling problem at the airport — which was out of scope for the PhD. It was agreed that while the end goal would be the development of a *tool for all scheduling problems*, the project would initially focus on the security scheduling problem.

Over the duration of the PhD meetings with the CSS took place to discuss the latest developments of the roster tool. Examples include:

- The implementation of new break assignment rules.
- Taking away the direct influence of lane ordering on the shift start/end times and moving to a more general approach of ensuring enough members of staff were on shift to cover the lane demand at all times.
- Introducing the option to select permissible workdays.
- The introduction of additional objectives in the 'Set Rotating Schedules' model (module RT2) to allow for the minimization of single days off and five-day work stretches.
- The creation of a variety of outputs from the roster tool.
- The option of starting from a partial roster.

In the final year of the PhD, the roster tool development reached a point that made it possible to provide the airport with a first prototype to try out. An overview of the structure of this prototype and the different inputs and outputs is provided in the following Section 4.4.2.

### 4.4.2 The Roster Tool Prototype

The implementation of the roster tool[1] matches the description provided in Chapter 1 (Section 1.4, Figure 1.2) and is shown again in Figure 4.4. The input data for the roster is provided in the form of an Excel sheet that the CSS can manipulate to some extent, and the final output is again provided in the form of an Excel document (see Appendix 4.A for more details). In between the inputs and outputs, a python based

[1]The complete user handbook provided to the CSS alongside the prototype can be found in Appendix 4.A. It describes all the inputs and possible changes the CSS can make.

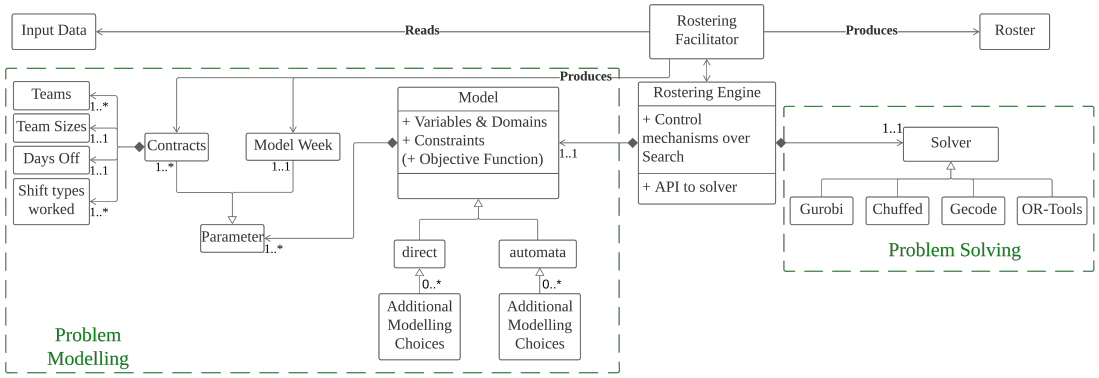



Figure 4.4: UML diagram of roster tool implementation as shown in Chapter 1.

Rostering Facilitator translates the inputs into the required problem parameters (e.g. model week creation — module RT1 in Figure 4.3) for the scheduling models, calls on MiniZinc (Nethercote et al., 2007) models to solve each of the rostering stages (modules RT2-RT4 in Figure 4.3), and translates the MiniZinc outputs into the required format for the CSS.

#### 4.4.2.1 Roster Outputs

The most important output to the scheduler is the roster itself. Figure 4.6 gives an example of an output roster for five contracts. It shows for all teams (0-53) what time their shifts start and end, if they are working, and the total weekly hours they have been scheduled. In this particular example, the fourth contract is set to only be for weekdays. Consequently, all Saturday and Sunday entries are empty — representing days off. To help the CSS check days-off and shift type-related regulations, they are also provided with versions of this roster that only show the scheduled shift type and the shift number (Figure 4.7).

The shift numbers are particularly important as they provide the CSS with a way to identify each shift in the provided output lists of shift details and break details.

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | **Shift Number** | **Day** | **Shift Type** | **Constant** | **Teamsize** | **Start** | **End** | **Duration** |
| 2 | 1 | Monday | Afternoon | 0 | 8 | 11:30:00 | 18:15:00 | 6.75 |
| 3 | 2 | Monday | Morning | 0 | 8 | 03:15:00 | 13:15:00 | 10 |
| 4 | 3 | Monday | Morning | 0 | 8 | 04:30:00 | 14:15:00 | 9.75 |
| 5 | 4 | Monday | Morning | 0 | 8 | 03:15:00 | 12:15:00 | 9 |
| 6 | 5 | Monday | Afternoon | 0 | 8 | 12:00:00 | 17:00:00 | 5 |

Figure 4.5: Example output of shift details for five shifts.

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** | **Saturday** | **Sunday** | **Weekly Hours** |
| 2 | **0** | 12:15:00-18:45:00 | | 04:30:00-14:30:00 | 03:15:00-08:30:00 | 19:45:00-03:30:00 | | 03:15:00-07:15:00 | 33.5 |
| 3 | **1** | | 03:15:00-08:30:00 | 03:15:00-07:15:00 | 04:30:00-13:15:00 | | | 04:15:00-14:15:00 | 28 |
| 4 | **2** | 03:15:00-13:00:00 | 04:30:00-14:30:00 | | | 03:15:00-13:15:00 | 03:15:00-09:30:00 | 03:30:00-12:15:00 | 44.75 |
| 5 | **3** | 04:15:00-14:15:00 | 11:30:00-20:00:00 | | 04:45:00-14:30:00 | 03:15:00-11:15:00 | 04:30:00-08:30:00 | 19:45:00-03:30:00 | 48 |
| 6 | **4** | | | | 11:30:00-20:00:00 | 11:30:00-20:15:00 | 19:45:00-03:30:00 | | 25 |
| 7 | **5** | 03:15:00-12:15:00 | | | | 04:15:00-14:00:00 | 04:15:00-13:45:00 | 11:30:00-19:45:00 | 36.5 |
| 8 | **6** | 19:45:00-03:30:00 | | 19:45:00-03:30:00 | 19:45:00-03:30:00 | | 11:30:00-20:00:00 | 11:30:00-20:00:00 | 40.25 |
| 9 | **7** | 11:15:00-15:15:00 | 12:00:00-20:00:00 | | | 03:15:00-11:45:00 | 03:15:00-11:45:00 | 12:00:00-18:00:00 | 35 |
| 10 | **8** | 12:30:00-22:30:00 | 19:45:00-03:30:00 | | | 12:00:00-22:00:00 | | 03:15:00-07:15:00 | 31.75 |
| 11 | **9** | 03:15:00-13:15:00 | 03:15:00-11:45:00 | 11:15:00-20:00:00 | | | 11:15:00-21:15:00 | | 37.25 |
| 12 | | | | | | | | | |
| 13 | | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** | **Saturday** | **Sunday** | **Weekly Hours** |
| 14 | **10** | | | 11:15:00-20:00:00 | 19:45:00-03:30:00 | | 03:15:00-11:45:00 | 13:45:00-22:15:00 | 33.5 |
| 15 | **11** | | | 04:45:00-10:00:00 | 03:15:00-11:45:00 | 19:45:00-03:30:00 | 19:45:00-03:30:00 | 19:45:00-03:30:00 | 37 |
| 16 | **12** | | | | 03:15:00-11:45:00 | 03:15:00-11:45:00 | 03:30:00-10:45:00 | 03:15:00-12:15:00 | 33.25 |
| 17 | **13** | | 11:30:00-20:00:00 | | 11:30:00-20:00:00 | 11:30:00-20:00:00 | 11:30:00-20:00:00 | | 34 |
| 18 | **14** | | | 03:30:00-13:30:00 | | 11:30:00-20:00:00 | | 11:30:00-20:00:00 | 27 |
| 19 | **15** | 13:15:00-18:15:00 | 11:30:00-21:00:00 | | | 03:45:00-09:15:00 | 11:15:00-18:30:00 | 11:30:00-20:00:00 | 35.75 |
| 20 | **16** | 18:30:00-23:00:00 | 12:45:00-17:45:00 | | 03:45:00-10:15:00 | 11:30:00-18:30:00 | 11:30:00-21:30:00 | | 33 |
| 21 | **17** | 11:30:00-16:00:00 | 11:30:00-20:00:00 | 11:30:00-20:00:00 | | | 03:15:00-12:30:00 | 19:45:00-03:30:00 | 38.5 |
| 22 | **18** | | 04:00:00-09:45:00 | 03:15:00-08:00:00 | 03:15:00-11:30:00 | 03:15:00-13:15:00 | 14:45:00-23:00:00 | | 37 |
| 23 | **19** | 12:45:00-20:30:00 | | | 12:30:00-19:00:00 | 19:45:00-03:30:00 | | 04:15:00-14:15:00 | 32 |
| 24 | **20** | | 05:00:00-15:00:00 | 03:15:00-11:45:00 | 12:00:00-18:30:00 | 11:45:00-19:30:00 | 13:45:00-23:30:00 | | 42.5 |
| 25 | **21** | 11:30:00-20:00:00 | | 03:15:00-11:45:00 | | | 03:15:00-11:30:00 | 11:45:00-21:00:00 | 34.5 |
| 26 | **22** | 11:30:00-20:00:00 | | 19:45:00-03:30:00 | | 04:15:00-14:15:00 | 03:15:00-11:45:00 | 03:30:00-13:15:00 | 44.5 |
| 27 | **23** | 04:30:00-14:15:00 | | | | 04:00:00-13:30:00 | 04:15:00-10:15:00 | 03:15:00-11:30:00 | 33.5 |
| 28 | | | | | | | | | |
| 29 | | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** | **Saturday** | **Sunday** | **Weekly Hours** |
| 30 | **24** | 12:45:00-19:15:00 | 13:15:00-23:00:00 | | 11:15:00-20:45:00 | | 03:45:00-10:45:00 | 03:30:00-13:30:00 | 42.75 |
| 31 | **25** | | | 04:30:00-13:15:00 | 19:45:00-03:30:00 | | | 03:15:00-11:45:00 | 25 |
| 32 | **26** | 03:15:00-11:45:00 | 03:15:00-08:00:00 | 05:00:00-14:15:00 | | 11:15:00-17:45:00 | 11:30:00-20:00:00 | 12:15:00-19:45:00 | 45 |
| 33 | **27** | | 04:15:00-14:00:00 | 03:15:00-11:45:00 | 03:15:00-11:45:00 | 03:15:00-11:45:00 | | | 35.25 |
| 34 | **28** | 03:15:00-11:45:00 | 03:15:00-11:45:00 | 03:15:00-10:45:00 | 15:00:00-23:30:00 | | | | 33 |
| 35 | **29** | 03:45:00-11:45:00 | 04:30:00-14:30:00 | | 04:30:00-14:15:00 | 03:15:00-11:45:00 | | | 36.25 |
| 36 | **30** | 11:45:00-20:15:00 | 11:15:00-16:30:00 | | 03:15:00-11:45:00 | 11:30:00-20:00:00 | | | 30.75 |
| 37 | **31** | | 03:15:00-08:00:00 | 11:30:00-20:00:00 | 12:30:00-19:00:00 | 15:45:00-22:00:00 | 19:45:00-03:30:00 | | 33.75 |
| 38 | **32** | 19:45:00-03:30:00 | 19:45:00-03:30:00 | | 04:00:00-14:00:00 | 12:30:00-18:45:00 | | | 31.75 |
| 39 | **33** | 11:30:00-20:00:00 | 19:45:00-03:30:00 | 19:45:00-03:30:00 | | | | 03:15:00-11:45:00 | 32.5 |
| 40 | **34** | 03:15:00-11:45:00 | 03:15:00-11:45:00 | 12:45:00-21:45:00 | 11:30:00-20:00:00 | | | 11:45:00-20:30:00 | 43.25 |
| 41 | **35** | 19:45:00-03:30:00 | | | | 04:30:00-08:45:00 | | 03:15:00-11:45:00 | 20.5 |
| 42 | **36** | | | 11:30:00-20:00:00 | 12:30:00-20:00:00 | 14:45:00-20:15:00 | 11:45:00-21:15:00 | 11:30:00-20:15:00 | 39.75 |
| 43 | **37** | | 03:15:00-11:45:00 | 03:15:00-08:30:00 | 03:15:00-11:30:00 | | 03:15:00-12:15:00 | 15:45:00-23:15:00 | 38.5 |
| 44 | | | | | | | | | |
| 45 | | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** | **Saturday** | **Sunday** | **Weekly Hours** |
| 46 | **38** | 04:00:00-10:00:00 | 04:30:00-13:45:00 | 03:15:00-12:30:00 | 03:45:00-13:00:00 | 03:15:00-08:15:00 | | | 38.75 |
| 47 | **39** | 12:15:00-21:00:00 | 17:00:00-23:30:00 | 13:00:00-22:15:00 | 11:45:00-21:00:00 | 13:45:00-23:30:00 | | | 43.5 |
| 48 | **40** | 03:15:00-11:45:00 | 16:00:00-23:00:00 | 13:45:00-23:30:00 | 13:30:00-23:30:00 | | | | 35.25 |
| 49 | **41** | 03:15:00-12:00:00 | 11:30:00-19:15:00 | 14:15:00-23:30:00 | 11:30:00-16:00:00 | 12:45:00-17:15:00 | | | 34.75 |
| 50 | **42** | 16:00:00-21:15:00 | 19:00:00-23:30:00 | 14:00:00-23:00:00 | 13:15:00-20:30:00 | 13:00:00-22:00:00 | | | 35 |
| 51 | **43** | 03:45:00-13:45:00 | 12:45:00-18:00:00 | | 03:15:00-12:30:00 | 03:15:00-13:00:00 | | | 34.25 |
| 52 | **44** | 03:15:00-10:45:00 | 14:30:00-19:15:00 | 14:30:00-23:30:00 | 11:15:00-20:15:00 | | | | 30.25 |
| 53 | **45** | 03:30:00-10:45:00 | 03:15:00-11:45:00 | 05:00:00-15:00:00 | 03:15:00-11:45:00 | 03:30:00-13:30:00 | | | 44.25 |
| 54 | | | | | | | | | |
| 55 | | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** | **Saturday** | **Sunday** | **Weekly Hours** |
| 56 | **46** | | | | 03:15:00-09:30:00 | | 12:00:00-16:00:00 | | 10.25 |
| 57 | **47** | 03:45:00-11:15:00 | 04:15:00-13:00:00 | 03:15:00-11:45:00 | 11:15:00-15:30:00 | | 04:45:00-14:15:00 | 04:45:00-13:45:00 | 47.5 |
| 58 | **48** | | | 04:45:00-14:45:00 | 11:15:00-16:00:00 | | 12:15:00-21:45:00 | 13:30:00-23:30:00 | 34.25 |
| 59 | **49** | 14:00:00-22:30:00 | | 03:15:00-12:00:00 | 11:15:00-16:00:00 | | 03:15:00-12:15:00 | 03:15:00-11:45:00 | 39.5 |
| 60 | **50** | 11:30:00-20:00:00 | 16:00:00-22:00:00 | 11:30:00-20:00:00 | | | 04:15:00-13:45:00 | 14:00:00-22:45:00 | 41.25 |
| 61 | **51** | | 11:15:00-19:00:00 | 13:45:00-23:00:00 | 13:00:00-20:15:00 | | | 04:45:00-14:45:00 | 34.25 |
| 62 | **52** | 04:30:00-14:30:00 | 03:15:00-11:45:00 | 13:30:00-23:30:00 | 14:45:00-23:30:00 | | 12:00:00-16:00:00 | 11:45:00-18:30:00 | 48 |
| 63 | **53** | | 11:45:00-20:45:00 | 12:45:00-19:30:00 | 14:45:00-23:30:00 | 11:30:00-20:00:00 | | | 33 |

Figure 4.6: Sample Output Roster.

| | | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday | Sunday |
|---|---|---|---|---|---|---|---|---|
| 0 | | Afternoon | | Morning | Morning | Night | | Morning |
| 1 | | | Morning | Morning | Morning | | | Morning |
| 2 | | Morning | Morning | | | Morning | Morning | Morning |
| 3 | | Morning | Afternoon | | Morning | Morning | Morning | Night |
| 4 | | | | | Afternoon | Afternoon | Night | |
| 5 | | Morning | | | | Morning | Morning | Afternoon |
| 6 | | Night | | Night | Night | | Afternoon | Afternoon |
| 7 | | Afternoon | Afternoon | | | Morning | Morning | Afternoon |
| 8 | | Afternoon | Night | | | Afternoon | | Morning |
| 9 | | Morning | Morning | Afternoon | | | Afternoon | |
| | | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** | **Saturday** | **Sunday** |
| 10 | | | | Afternoon | Night | | Morning | Afternoon |
| 11 | | | | Morning | Morning | Night | Night | Night |
| 12 | | | | | Morning | Morning | Morning | Morning |
| 13 | | | Afternoon | | Afternoon | Afternoon | Afternoon | |
| 14 | | | | Morning | | Afternoon | | Afternoon |
| 15 | | Afternoon | Afternoon | | | Morning | Afternoon | Afternoon |
| 16 | | Afternoon | Afternoon | | Morning | Afternoon | Afternoon | |
| 17 | | Afternoon | Afternoon | Afternoon | | | Morning | Night |
| 18 | | | Morning | Morning | Morning | Morning | Afternoon | |
| 19 | | Afternoon | | | Afternoon | Night | | Morning |
| 20 | | | Morning | Morning | Afternoon | Afternoon | Afternoon | |
| 21 | | Afternoon | | Morning | | | Morning | Afternoon |
| 22 | | Afternoon | | Night | | Morning | Morning | Morning |
| 23 | | Morning | | | | Morning | Morning | Morning |
| | | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** | **Saturday** | **Sunday** |
| 24 | | Afternoon | Afternoon | | Afternoon | | Morning | Morning |
| 25 | | | | Morning | Night | | | Morning |
| 26 | | Morning | Morning | Morning | | Afternoon | Afternoon | Afternoon |
| 27 | | | Morning | Morning | Morning | Morning | | |
| 28 | | Morning | Morning | Morning | Afternoon | | | |
| 29 | | Morning | Morning | | Morning | Morning | | |
| 30 | | Afternoon | Afternoon | | Morning | Afternoon | | |
| 31 | | | Morning | Afternoon | Afternoon | Afternoon | Night | |
| 32 | | Night | Night | | Morning | Afternoon | | |
| 33 | | Afternoon | Night | Night | | | | Morning |
| 34 | | Morning | Morning | Afternoon | Afternoon | | | Afternoon |
| 35 | | Night | | | | Morning | | Morning |
| 36 | | | | Afternoon | Afternoon | Afternoon | Afternoon | Afternoon |
| 37 | | | Morning | Morning | Morning | | Morning | Afternoon |
| | | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** | **Saturday** | **Sunday** |
| 38 | | Morning | Morning | Morning | Morning | Morning | | |
| 39 | | Afternoon | Afternoon | Afternoon | Afternoon | Afternoon | | |
| 40 | | Morning | Afternoon | Afternoon | Afternoon | | | |
| 41 | | Morning | Afternoon | Afternoon | Afternoon | Afternoon | | |
| 42 | | Afternoon | Afternoon | Afternoon | Afternoon | Afternoon | | |
| 43 | | Morning | Afternoon | | Morning | Morning | | |
| 44 | | Morning | Afternoon | Afternoon | Afternoon | | | |
| 45 | | Morning | Morning | Morning | Morning | Morning | | |
| | | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** | **Saturday** | **Sunday** |
| 46 | | | | | Morning | | Afternoon | |
| 47 | | Morning | Morning | Morning | Afternoon | | Morning | Morning |
| 48 | | | | Morning | Afternoon | | Afternoon | Afternoon |
| 49 | | Afternoon | | Morning | Afternoon | | Morning | Morning |
| 50 | | Afternoon | Afternoon | Afternoon | | | Morning | Afternoon |
| 51 | | | Afternoon | Afternoon | Afternoon | | | Morning |
| 52 | | Morning | Morning | Afternoon | Afternoon | | Afternoon | Afternoon |
| 53 | | | Afternoon | Afternoon | Afternoon | Afternoon | | |

| | | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday | Sunday |
|---|---|---|---|---|---|---|---|---|
| 0 | | 1 | | 73 | 108 | 147 | | 210 |
| 1 | | | 37 | 74 | 109 | | | 211 |
| 2 | | 2 | 38 | | | 148 | 180 | 212 |
| 3 | | 3 | 39 | | 110 | 149 | 181 | 213 |
| 4 | | | | | 111 | 150 | 182 | |
| 5 | | 4 | | | | 151 | 183 | 214 |
| 6 | | 5 | | 75 | 112 | | 184 | 215 |
| 7 | | 6 | 40 | | | 152 | 185 | 216 |
| 8 | | 7 | 41 | | | 153 | | 217 |
| 9 | | 8 | 42 | 76 | | | 186 | |
| | | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** | **Saturday** | **Sunday** |
| 10 | | | | 77 | 113 | | 187 | 218 |
| 11 | | | | 78 | 114 | 154 | 188 | 219 |
| 12 | | | | | 115 | 155 | 189 | 220 |
| 13 | | | 43 | | 116 | 156 | 190 | |
| 14 | | | | 79 | | 157 | | 221 |
| 15 | | 9 | 44 | | | 158 | 191 | 222 |
| 16 | | 10 | 45 | | 117 | 159 | 192 | |
| 17 | | 11 | 46 | 80 | | | 193 | 223 |
| 18 | | | 47 | 81 | 118 | 160 | 194 | |
| 19 | | 12 | | | 119 | 161 | | 224 |
| 20 | | | 48 | 82 | 120 | 162 | 195 | |
| 21 | | 13 | | 83 | | | 196 | 225 |
| 22 | | 14 | | 84 | | 163 | 197 | 226 |
| 23 | | 15 | | | | 164 | 198 | 227 |
| | | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** | **Saturday** | **Sunday** |
| 24 | | 16 | 49 | | 121 | | 199 | 228 |
| 25 | | | | 85 | 122 | | | 229 |
| 26 | | 17 | 50 | 86 | | 165 | 200 | 230 |
| 27 | | | 51 | 87 | 123 | 166 | | |
| 28 | | 18 | 52 | 88 | 124 | | | |
| 29 | | 19 | 53 | | 125 | 167 | | |
| 30 | | 20 | 54 | | 126 | 168 | | |
| 31 | | | 55 | 89 | 127 | 169 | 201 | |
| 32 | | 21 | 56 | | 128 | 170 | | |
| 33 | | 22 | 57 | 90 | | | | 231 |
| 34 | | 23 | 58 | 91 | 129 | | | 232 |
| 35 | | 24 | | | | 171 | | 233 |
| 36 | | | | 92 | 130 | 172 | 202 | 234 |
| 37 | | | 59 | 93 | 131 | | 203 | 235 |
| | | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** | **Saturday** | **Sunday** |
| 38 | | 25 | 60 | 94 | 132 | 173 | | |
| 39 | | 26 | 61 | 95 | 133 | 174 | | |
| 40 | | 27 | 62 | 96 | 134 | | | |
| 41 | | 28 | 63 | 97 | 135 | 175 | | |
| 42 | | 29 | 64 | 98 | 136 | 176 | | |
| 43 | | 30 | 65 | | 137 | 177 | | |
| 44 | | 31 | 66 | 99 | 138 | | | |
| 45 | | 32 | 67 | 100 | 139 | 178 | | |
| | | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** | **Saturday** | **Sunday** |
| 46 | | | | | 140 | | 204 | |
| 47 | | 33 | 68 | 101 | 141 | | 205 | 236 |
| 48 | | | | 102 | 142 | | 206 | 237 |
| 49 | | 34 | | 103 | 143 | | 207 | 238 |
| 50 | | 35 | 69 | 104 | | | 208 | 239 |
| 51 | | | 70 | 105 | 144 | | | 240 |
| 52 | | 36 | 71 | 106 | 145 | | 209 | 241 |
| 53 | | | 72 | 107 | 146 | 179 | | |

Figure 4.7: Output rosters of shift types and shift numbers.

The shift details sheet (Figure 4.5) stores information for all shifts, including the weekday and shift type they are scheduled for, whether the shift is constant or variable, how many employees are assigned, as well as the start/end times and shift duration. How many shifts are scheduled over the week varies from solution to solution and depends on many factors, such as the initial rotating roster that is created by module RT2 and how many smaller or larger teams are assigned work on a certain day. In the roster used for the example outputs in this section, a total of 241 shifts are used (which can be seen in the output Roster in Figure 4.7). The Start Hours output is additional information that is needed by the Python code to create the output figures shown later in this section.

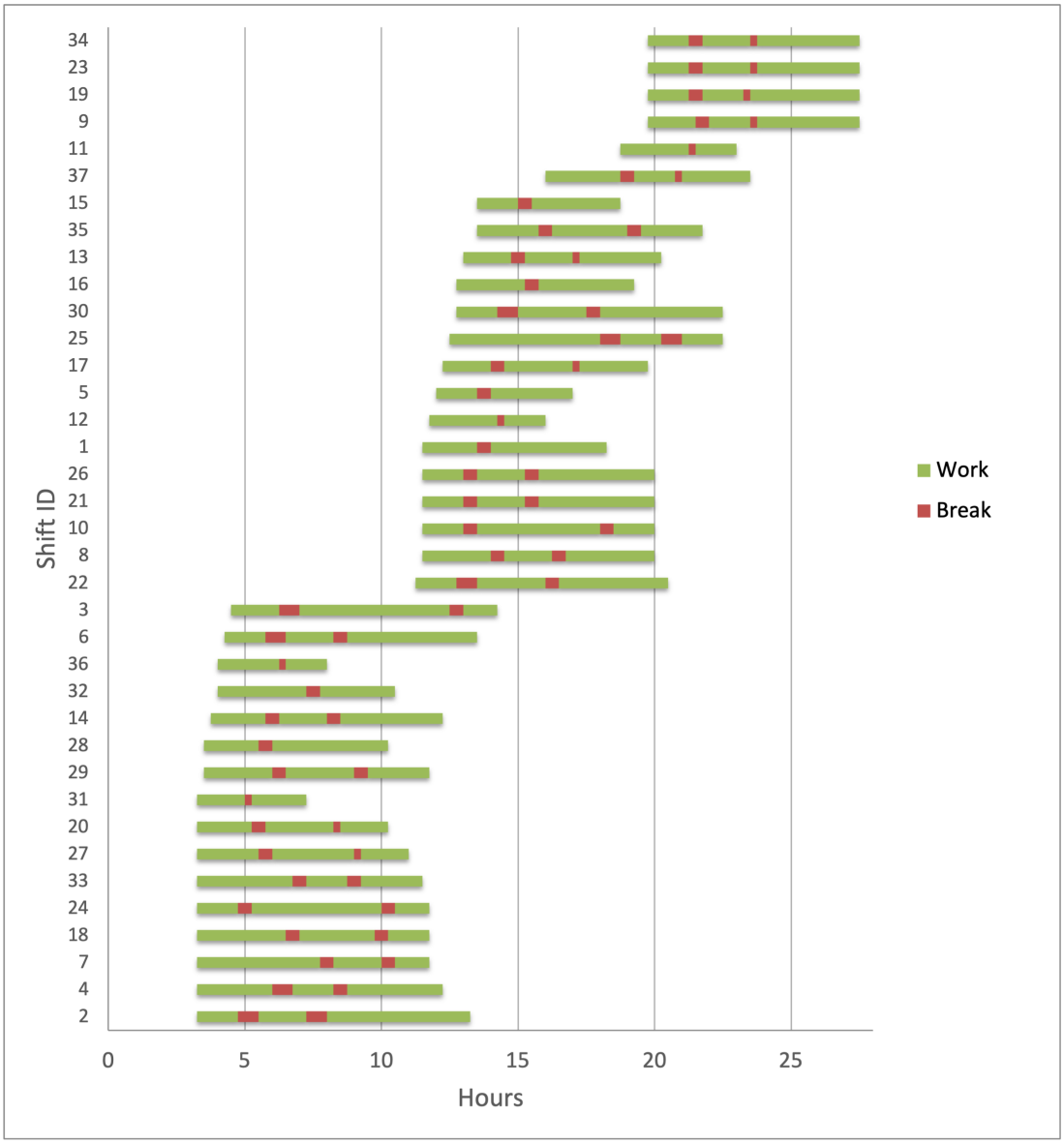



Figure 4.8: Graphical work/break overview.

Figure 4.8 shows an example of a graphical output of all shifts scheduled on Monday in the example roster — red refers to breaks, and green represents work time. This output is accompanied by a break details table that gives the start/end time of each shift and the associated start/end of all breaks. It is important to note that night shifts in this image pass the 24-hour mark. This is to facilitate the modelling of break start and end times in the break assignment module. Any value above 24 represents the early hours of the next day.

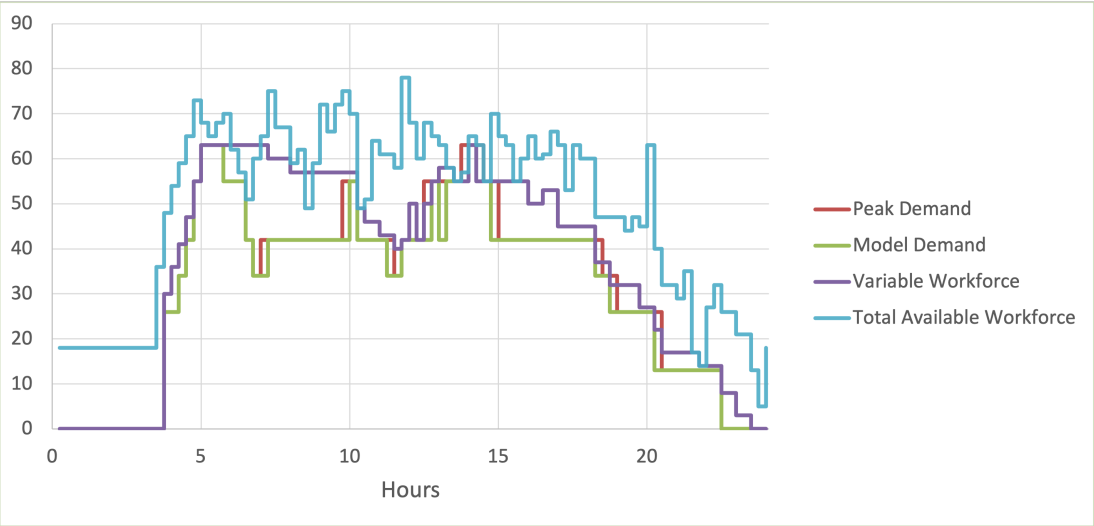



Figure 4.9: Available Workforce throughout the day.

Finally, the roster tool provides a graphical representation of demand and the available workforce — again for each weekday separately. Figure 4.9 shows an example graph for Monday shifts. It shows the peak (maximum predicted demand in the demand forecast), the model week demand, the number of employees scheduled to work variable shifts and the total available workforce. The difference between the variable workforce count and the total available workforce count is that the latter accounts for scheduled breaks — security officers on a break cannot work to cover demand — while the former does not. This is why the blue 'total available workforce' line will sometimes dip below the purple 'variable workforce' line.

### 4.4.3 Ensuring Uptake

The most important factor determining if the final roster tool will be used is its ability to solve the problem at hand and produce *palatable* (i.e. feasible and 'appealing' to the CSS and workforce) solutions. To ensure this is the case, I worked closely with the

CSS to model the security officers' scheduling problem and the associated constraints. I took on the provided feedback and implemented the requested features to make the tool more applicable to the CSS's needs — something that was confirmed after each round of feedback. However, having a working tool does not mean it will be used in the long run. To ensure this is the case, additional considerations need to be satisfied. The first one is guidance on how to use the tool after the collaboration has ended. If the CSS has a tool but does not know how to use it to its full extent, it will not stay useful for long and might be abandoned in the long run. To stop this from happening, I created a handbook to accompany the tool. The handbook is attached in Appendix 4.A and outlines all features and possible settings, describes the steps needed to run the tool and explains the roster tool outputs..

Another important aspect of ensuring the roster tool's uptake is integration into the airport's existing systems. The tool is currently coded on a MacBook, relying on Apple's operating system to run. This cannot be integrated into the airport's systems. A one-time project with a software engineer/developer would be needed to transfer the roster tool to a Windows-based operating system, if not to build an application with a more user-friendly interface for the CSS.

Finally, the airport needs to be able to maintain the roster tool and implement reasonable changes to the tool — such as adding individual constraints to the optimization modules (RT2-RT4). This is only possible if a person working at the airport is familiar with OR/MS techniques and/or can be trained in the modelling language to make the required changes. Given that the airport has an analytics team with several analysts and data engineers on site, this problem should be easy to overcome.

Overall, several aspects must be considered when trying to ensure the tool's uptake. However, the airport clearly needs such a tool and has the resources and staff to ensure that all the above-listed requirements are satisfied in the long run.

## 4.5 Airport Feedback and Future Developments

At the time of completing this thesis, the CSS has been provided with two versions of the prototype to try out and give feedback on. The overarching review has been that while the tool is the best solution they have seen so far (including attempts at implementing commercial tools), they can still not use it as it is. The main reason the CSS keeps bringing up is the continued *lack of flexibility* and the requirement to consistently be able to produce multiple rosters with the same set of rules. While the latter can be obtained through repeated runs of the tool, automation is required to ensure a better user experience. The *lack of flexibility* of the roster is primarily related to ever-changing rules in the way contracts work (even since the start of this collaboration), and the additional requirements that are put on the tool — which were not there at the beginning and as a result make the modelling and implementation of the problem more complex. However, this point is not entirely negative but rather a confirmation of the CSS's growing confidence in the tool's usability and, thus, higher expectations of what it should be able to do in the long run.

Ultimately, the airport needs a tool to automate its scheduling process and relieve some of the CSS's workload so the CSS can focus on other aspects of the workforce planning process. The collaboration that was formed as part of the work for this thesis has now come to an end with a prototype that cannot yet be used by the airport. However, there have been early talks to discuss a potential continuation of the work to complete adapting the tool to fit the airport's standards.

# Appendix

## 4.A EAL - Roster Tool Handbook

# EAL Roster Tool- Handbook

### Getting Started – Finding the Folder

The EAL Roster Tool implementation can be found as a folder in the Favourites bar in Finder (see Figure 1). Finder is located at the very left in the Menu Bar at the bottom of the screen.

The roster tool folder holds all the important components of the tool, which are split into subfolders.

1. Data_in: holds the input Excel Sheet
2. Data_out: hold the output Excel Sheet
3. Python: holds all required Python files
4. Minizinc: holds all required Minizinc files
5. Additional Files needed for running the Python scripts:
    a. get-pip.py
    b. poetry.lock
    c. pyproject.toml

In order to use the Roster Tool only the Data_in and Data_out folders need to be accessed directly!

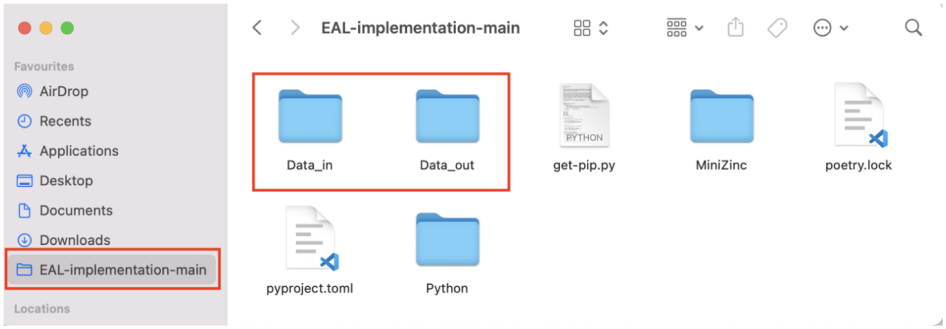



Figure 1 Locating the Roster Tool folders in Finder

### Data_in – input Data

Data_in holds the Excel Sheet "input Data March 23" that works as the Data input for the Roster Tool. It is extremely important that the file name, tab names and greyed out areas in this excel sheet **are not changed**. Otherwise, the python scripts will not work. The file is organised as follows:

1. **Demand Forecast**: holds the Lane forecast for the planning period. To change the forecast, simply copy and paste a new forecast with the same format into the sheet. Very Important! The format of the date needs to remain the same for the new forecast
2. **Contract Details:** holds the contractual details for all contracts. This includes:
    a. **Contract:** numerical indicator of each contract
    b. **Hours:** the contracted weekly work hours
    c. **Team size:** number of people in a team for this contract

   d. **Min team size:** minimum team size (this is for the purpose of automatic team creation - not in use at the moment so no need to know the value)
   e. **Team number:** how many teams work under the contract
   f. **Max team number:** maximum number of teams (this is for the purpose of automatic team creation - not in use at the moment so no need to know the value)
   g. **Max Weekly Hours:** maximum permissible weekly hours
   h. **Min Shift Duration:** minimum shift duration (in hours)
   i. **Max Shift Duration:** maximum shift duration (in hours)
   j. **Days per week worked:** maximum number of days worked per calendar week
   k. **Morning:** indicator whether the contract can work morning shifts (1- can work the shift, 0- cannot work the shift)
   l. **Afternoon:** indicator whether the contract can work afternoon shifts (1- can work the shift, 0- cannot work the shift)
   m. **Night:** indicator whether the contract can work afternoon shifts (1- can work the shift, 0- cannot work the shift)
   n. **Monday-Sunday:** indicator whether the contract can work shifts on that day (1- can work the shift, 0- cannot work the shift)

   Each line in the sheet refers to one contract and the lines can be added/removed as needed to escribe the entire workforce.

3. **Input Roster:** this sheet gives the option to add a partial roster. The lines are set up to match the current contractual set-up. If the number of teams in a contract change, the input lines need to be adjusted here. The lines on the sheet do not need to be changed for the roster tool to work. They are there as visual guidance to distinguish the contracts from each other. There are several ways to indicate shifts in the partial roster:
   **a. Indicate the shift type:** Enter “Morning”, “Afternoon” or “Night” and that team will be allocated a shift that matches that particular shift type
   **b. Indicate shift start/end times:** Enter the start and end time of the shift in the following format: HH:MM:00-HH:MM:00. Please ensure that shift start and end times are always in quarter hour steps. (i.e. you can enter 14:30:00 or 14:45:00 but not 14:34:00). The tool cannot guarantee that these times will be kept to exactly. In the parameters Tab, there is an option to adjust how many bins each of the shifts in the partial roster can be adjusted by.
   **c. Indicate a day of:** Enter “OFF” to ensure that the team won’t be assigned a shift on that day
   **d. Leave cell empty:** This indicates to the tool that this day is available to be assigned either a shift or a day off.
4. **Demand Details:** holds all the information about constant demand and how variable demand is added up
   a. **Demand:** workforce demand for each location, depending on how many lanes are open
   b. **Cumulative demand:** cumulative demand depending on how many lanes are open
   c. **Demand separate:** workforce demand for each separate location
   d. **Constant:** indicator whether the location is constant(1) or variable (0)
   e. **Night:** indicator whether the location exists for night shift (1- yes, 0-no)
   f. **Location:** location name

It should be possible to add more lanes to this sheet. However, this has not been tested yet so might cause some unforeseen problems

5. **Shift Details:** holds the information about start and end times (as 15 minute time bins) for shift types. Here the constant start and end times are used as they are given. The variable open and close times act as boundaries. This means that the actual shift times will be anywhere between those values (with a buffer allowance to go over the limit – the buffer is set in the "Parameter" sheet)
6. **Break Requirements:** Break patterns are based on a maximum of three breaks per shift. The columns store the following information:
    a. **ShiftDuration:** the minimum duration of a shift that requires this pattern
    b. **BreakNumber:** how many breaks are required
    c. **Break1-Break3:** duration of each break (in minutes – please stick to 15 minute steps). If the break is not required, the input is 0. For example, if only 2 breaks are required, then Break3 should have entry 0.
7. **Parameters**: additional problem parameters for the Tool.
    a. **Aggregation Level:** indicator of how much variation in the deman forecast should be included in the model week. This is a percentage so please enter a value between 0-100.
    b. **Min average shift length:** minimum average shift duration. This is applied to each contract separately to allow for fairness over all contracts
    c. **Max average shift length:** maximum average shift duration. This is applied to each contract separately to allow for fairness over all contracts
    d. **2 days off every n weeks:** input value for n
    e. **Breakslack:** minimum duration (in 15-minute bins) between breaks
    f. **Buffer:** buffer (in 15-minute bins) to go over the start and end limits for shift types
    g. **Max consecutive days:** maximum allowed consecutive work days
    h. **Ideal consecutive days:** ideal consecutive work days
    i. **Shift Adjustment:** by how many bins can the Roster Tool adjust shift times that are provided in the Input Roster. It is possible that the tool cannot find a solution without changing the provided shift times, so it is important to leave a slack of 2 bins (30 minutes).
    j. **Shift types:** number of shift types
    k. **Locations:** number of "different" locations – here all security lanes are counted as one location.
8. **A4 constraint Dashboard:** this sheet allows for the switching on/off of rules for the roster. Constraints that start with an O refer to objectives. There can only be one objective selected at a time and the cells will turn red if too many objectives are switched on. In addition, there must always be one objective switched on.
   constraints that start with a C refer to roster rules. These can be switched/on of freely unless they clash with the selected objective (cells will turn red again if a disallowed combination is selected).
   The last option in the Dashboard (B1) let's you switch the break assignment on and off. Break assignment can take a very long time (10+ minutes) so there is an option to turn break assignment off and make it easier to play around with the other rules and settings without needing to wait for breaks.

## Data_out – the final roster

Once the roster tool has been used – see the next section for running instructions – the final roster outputs can be found in this folder. If the tool is used without the break assignment options, there will be two output files titled "EAL_Outputs_NoBreaks_VApril24.xlsx" and "Shift Diagrams No Breaks April 24.xlsx". If the break assignment options is used, there will be two additional files titled "EAL_Outputs _VApril24.xlsx" and "Shift Diagrams April 24.xlsx" which makes it possible to compare shifts before and after break assignment. To avoid losing the roster please store it in a different folder or rename the file before running the Roster Tool. Any existing file of that name will be overwritten by the Roster Tool.

The output "EAL_Outputs _VApril24.xlsx" is arranged as follows. The "No breaks" version has the same structure with some exceptions that are indicated were necessary:

1. **Model Week – Lanes:** Lane demand of the model week that was used to produce the roster
2. **Model Week – Workforce Demand:** Workforce demand of the model week that was used to produce the roster
3. **Final Roster:** Shift times and days off for all contracts with the corresponding weekly work hours
4. **Rough Roster**: allocated shift types and days off
5. **Constant Roster:** allocated shift types and days off with an indicator whether the allocated shift is constant or not
6. **Shift Roster:** Shift numbers of the shifts assigned to each team at each day of the week. The shift numbers can be used to get more shift details in the "Shift Details" tab
7. **Daily Hours Roster:** Roster with the shift durations that were assigned to each team.
8. **Shift Details:** Shift details for all shifts in the roster. The information includes
    a. Shift number
    b. Day of shift
    c. Rough type
    d. Constant indicator
    e. Team size
    f. Start time
    g. End time
    h. Start Hour (just another expression for start time that is needed in the roster tool)
    i. Duration
9. **Work Break Patterns:** Visual representation of each shift over the whole day. 1 – work, 2- briefing, 3- break. – this sheet will be missing in the no breaks versions
10. **Break Details:** this sheet includes for every shift: – this sheet will be missing in the no breaks versions
    a. Start time
    b. Start/end time of break 1
    c. Start/end time of break 2
    d. Start/end time of break 3
    e. Shift end

The output file “Shift Diagrams No Breaks April 24.xlsx” (and the no breaks version) gives (for every day of the week) a diagram of the shift durations throughout the day (with breaks for the “normal” version) and a plot of the workforce demand and the scheduled workforce.

## Using the Roster Tool

Once the input data is set-up as required it is time to open up and run the python script to obtain the outputs. The python script can be run through the App “VS Code” which is located in the Menu bar at the bottom of the screen (Figure 2).

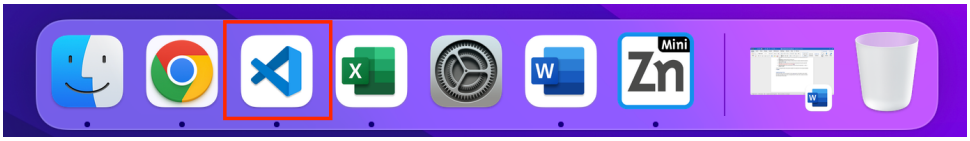


Figure 2 Locating VS code in the Menu Bar

Once opened, VS code will likely look as shown in Figure 3 below. If it does not, it will look as shown in Figure 4. If that is the case, please skip step 1 below.

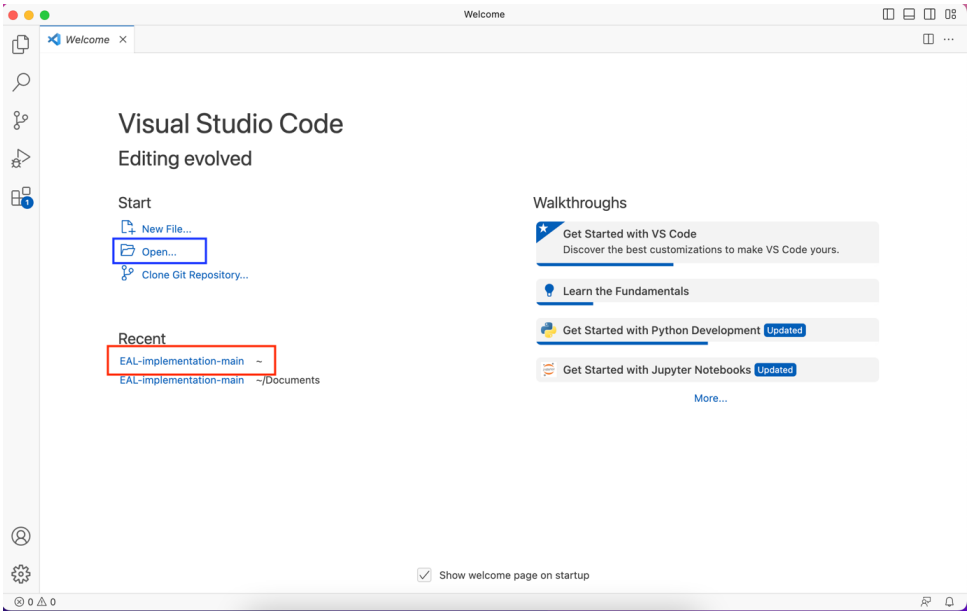



Figure 3 VS Code without a selected folder.

Steps to opening and running the roster tool in VS Code.

1. There are many ways to open the folder containing the Roster Tool.
   1.1. If the tool has been used recently it will appear under “Recent” (highlighted by the red box).
   1.2. If it does not appear under recent, the folder can also be opened via the “Open” option (highlighted by the blue box”)
      1.2.1. The “Open” option will open a new finder menu (as shown in Figure 1). Navigate to the “EAL-implementation-main” folder (so that all the subfolders can be seen on the screen) and then click “open” at the bottom.
2. Once opened, the VS Code App will look as shown in Figure 4.

3. In most cases, the required Python file "EAL_Roster_Tool_March23.ipynb" will open automatically on the main screen. This can be verified by checking the name in the top tab of the open file (green box in fig.4).
   3.1. If the file is not open, it can be selected from the Explorer window on the left (blue box in fig.4).
   3.2. If the file cannot be seen, then you need to open the "Python" drop-down.
      3.2.1.You can find the Python drop-down above the of the blue box shown in figure 4.
      3.2.2.Once the dropdown is open, select the file "EAL_Roster_Tool_March23.ipynb".
4. The final step before running the code is to check that the correct "kernel" is selected (red box in fig 4). This should happen automatically.
   4.1. However, if the selection does not say ".venv (Python ……)" click on the box and select .venv from the dropdown menu that appears at the top of the screen.
5. Finally, run the Roster Tool by clicking on "Run All" (purple box in fig 4.). This will run the code in the grey boxes of the file (one after another). Each box will initially have a little clock in the bottom left corner. Once the box start to run, the clock will turn to a rotating arrow. Finally, a green tick with a time (in seconds) will appear when the box has run to completion. In addition, there will be output messages after some of the boxes. Wait until all boxes have a green tick. This indicates that the roster has run to completion.
6. Once the file has run to completion, go back to the Finder and navigate to the "Data_out" folder to find the roster outputs. See section Data_out for more information about the output file.

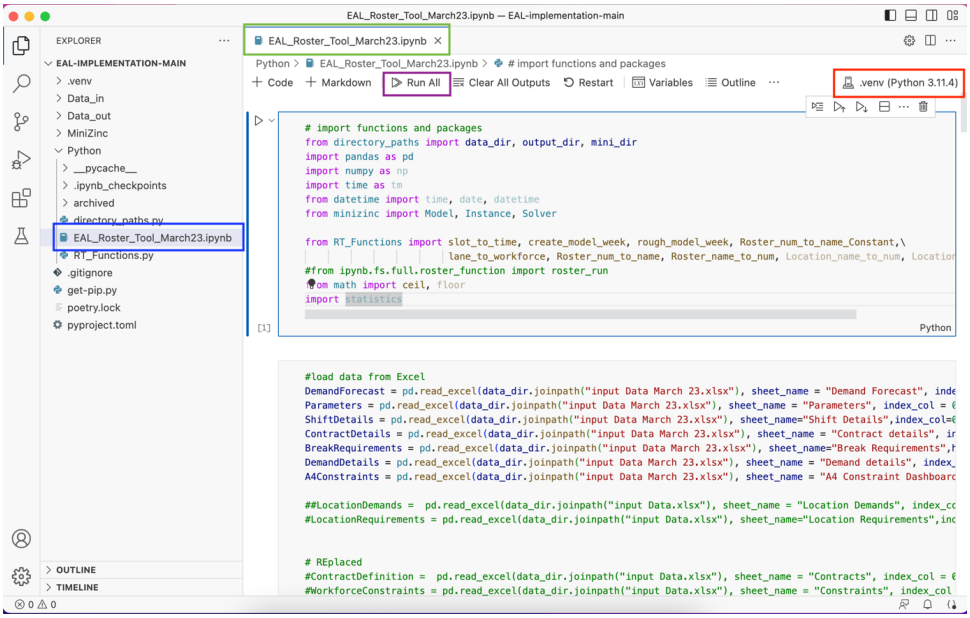



**Figure 4 VS Code with Roster Tool files opened.**

If you make changes to the input file you need to restart (button is located to the right of "run All") after the changes have been saved in the input file.

# Bibliography


Ackoff, R. L. (1967). Management misinformation systems. *Management science*, 14(4):B–147.

Ackoff, R. L. (1978). *The Art of Problem Solving Accompanied by Ackoff's Fables*. John Wiley & Sons.

Akbari, M., Zandieh, M., and Dorri, B. (2013). Scheduling part-time and mixed-skilled workers to maximize employee satisfaction. *International Journal of Advanced Manufacturing Technology*, 64(5-8):1017–1027.

Akl, A. M., El Sawah, S., Chakrabortty, R. K., and Turan, H. H. (2022). A Joint Optimization of Strategic Workforce Planning and Preventive Maintenance Scheduling: A Simulation–Optimization Approach. *Reliability Engineering & System Safety*, 219:108175.

Alfares, H. K. (2004). Survey, Categorization, and Comparison of Recent Tour Scheduling Literature. *Annals of Operations Research*, 127(1-4):145–175.

Alfares, H. K. (2007). A simulation approach for stochastic employee days-off scheduling. *International Journal of Modelling and Simulation*, 27(1):9–15.

Algethami, H. and Landa-Silva, D. (2017). Diversity-Based Adaptive Genetic Algorithm for a Workforce Scheduling and Routing Problem. *2017 IEEE Congress on Evolutionary Computation (CEC)*.

Atlason, J., Epelman, M. A., and Henderson, S. G. (2004). Call center staffing with simulation and cutting plane methods. *Annals of Operations Research*, 127(1-4):333–358.

Babor, M., Senge, J., Rosell, C. M., Rodrigo, D., and Hitzmann, B. (2021). Optimization of no-wait flowshop scheduling problem in bakery production with modified pso, neh and sa. *Processes*, 9.

Baker, K. R. (1976). Workforce Allocation in Cyclical Scheduling Problems: A Survey. *Operational Research Quarterly (1970-1977)*, 27(1):155.

Barghi, B. and Sikari, S. S. (2022). Meta-heuristic Solution with Considering Setup Time for Multi-Skilled Project Scheduling Problem. *Operations Research Forum*, 3(16).

Bartholdi, J. J. (1981). A Guaranteed-Accuracy Round-Off Algorithm for Cyclic Scheduling and Set Covering. *Operations Research*, 29(3):501–510.

Bartholdi, J. J., Orlin, J. B., and Ratliff, H. (1980). Cyclic Scheduling via Integer Programs with Circular Ones. *Operations Research*, 28(5):1074–1085.

BBC (2020). Edinburgh Airport to cut third of its workforce. `https://www.bbc.co.uk/news/uk-scotland-edinburgh-east-fife-53609665` [Accessed: (03/09/2024)].

BBC (2022a). Bin strikes continue as unions query pay offer. `https://www.bbc.co.uk/news/uk-scotland-62635331` [Accessed: (03/09/2024)].

BBC (2022b). Train services in South West cancelled in rail strikes. `https://www.bbc.co.uk/news/uk-england-devon-64485000` [Accessed: (03/09/2024)].

BBC (2024a). Hundreds of Heathrow Border Force staff walk out. `https://www.bbc.co.uk/news/articles/c93pek7n7pvo` [Accessed: (03/09/2024)].

BBC (2024b). UCU strike dates: Which universities are affected? `https://www.bbc.co.uk/news/education-59415694` [Accessed: (03/09/2024)].

Bechtold, S. E. (1981). Work force scheduling for arbitrary cyclic demands. *Journal of Operations Management*, 1(4):205–214.

Becker, T. (2020). A decomposition heuristic for rotational workforce scheduling. *Journal of Scheduling*, 23(5):539–554.

Becker, T., Schiffer, M., and Walther, G. (2022). A General Branch-and-Cut Framework for Rotating Workforce Scheduling. *INFORMS Journal on Computing*, 34(3):1548–1564.

Becker, T., Steenweg, P. M., and Werners, B. (2019). Cyclic shift scheduling with on-call duties for emergency medical services. *Health Care Management Science*, 22(4):676–690.

Bennett, B. T. and Potts, R. B. (1968). Rotating Roster for a Transit System. *Transportation Science*, 2(1):14–34.

Bessiere, C. (2006). Constraint Propagation. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, pages 29–83.

Bixby, R. E. (2012). A Brief History of Linear and Mixed-Integer Programming Computation. *Documenta Mathematica · Extra Volume ISMP*, pages 107–121.

Brucker, P., Qu, R., and Burke, E. (2011). Personnel scheduling: Models and complexity. *European Journal of Operational Research*, 210(3):467–473.

CAA (2017). Operating resilience of the uk's aviation infrastructure and the consumer interest. `https://www.caa.co.uk/our-work/publications/documents/content/cap1515/` [Accessed: 2.11.2021].

Calvet, L., Juan, A. A., Fernandez-Viagas, V., and Framinan, J. M. (2016). Combining simulation with metaheuristics in distributed scheduling problems with tochastic processing times. In *Proceedings of the 2016 Winter Simulation Conference*, pages 2347–2357.

Cassandras, C. G. and Lafortune, S., editors (2008). *Languages and Automata*, pages 53–131. Springer US, Boston, MA.

Cheng, C.-H., Fu, M. C., and Shi, L. (2002). Feature Article: Optimization for simulation: Theory vs. Practice. *INFORMS Journal on Computing*, 14(3):192–215.

Chuffed (2022). The chuffed cp solver. `https://www.gov.uk/holiday-entitlement-rights` [Accessed: (02/08/2022)].

Dantzig, G. B. (1954). Letter to the Editor—A Comment on Edie's "Traffic Delays at Toll Booths". *Journal of the Operations Research Society of America*, 2(3):339–341.

De Bruecker, P., Van Den Bergh, J., Beliën, J., and Demeulemeester, E. (2015). Workforce planning incorporating skills: State of the art. *European Journal of Operational Research*, 243(1):1–16.

Dehghanimohammadabadi, M., Rezaeiahari, M., and Seif, J. (2023). Multi-Objective Patient Appointment Scheduling Framework (MO-PASS): a data-table input simulation–optimization approach. *Simulation*, 99(4):363–383.

Department for Transport and Civil Aviation Authority (2023). Aviation statistics: data tables (AVI). `https://www.gov.uk/government/statistical-data-sets/aviation-statistics-data-tables-avi` [Accessed: (12/09/2024)].

Department for Transport, T. (2022). Passengers to benefit from biggest shake-up of airport security rules in decades. `https://www.gov.uk/government/news/passengers-to-benefit-from-biggest-shake-up-of-airport-security-rules-in-decades` [Accessed: (28/03/2024)].

Dietz, D. C. (2011). Practical scheduling for call center operations. *Omega*, 39(5):550–557.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41.

Doumic, M., Perthame, B., Ribes, E., Salort, D., and Toubiana, N. (2017). Toward an integrated workforce planning framework using structured equations. *European Journal of Operational Research*, 262(1):217–230.

Edinburgh Airport (2024a). Facts and figures. `https://corporate.edinburghairport.com/about-us/facts-and-figures` [Accessed: (28/08/2024)].

Edinburgh Airport (2024b). Framework for future access to Edinburgh Airport. `https://corporate.edinburghairport.com/framework-for-future-access-to-edinburgh-airport` [Accessed: (03/09/2024)].

Erkinger, C. and Musliu, N. (2017). Personnel scheduling as Satisfiability Modulo Theories. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 614–621.

Ernst, A. T., Jiang, H., Krishnamoorthy, M., Owens, B., and Sier, D. (2004a). An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research*, 127(1-4):21–144.

Ernst, A. T., Jiang, H., Krishnamoorthy, M., and Sier, D. (2004b). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153:3–27.

Eurocontrol (2024). EUROCONTROL Forecast 2024-2030. `https://www.eurocontrol.int/publication/eurocontrol-forecast-2024-2030` [Accessed: (12/09/2024)].

Figueira, G. and Almada-Lobo, B. (2014). Hybrid simulation–optimization methods: A taxonomy and discussion. *Simulation Modelling Practice and Theory*, 46:118–134.

Forrester, J. W. (2007). System dynamics—a personal view of the first fifty years. *System Dynamics Review*, 23(2-3):345–358.

Gatwick Airport (2024). Northern Runway plans. `https://www.gatwickairport.com/company/future-plans/northern-runway.html` [Accessed: (12/09/2024)].

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549.

Glover, F. and McMillan, C. (1986). The general employee scheduling problem. An integration of MS and AI. *Computers and Operations Research*, 13(5):563–573.

Gök, Y. S., Padrón, S., Tomasella, M., Guimarans, D., and Ozturk, C. (2023). Constraint-based robust planning and scheduling of airport apron operations through simheuristics. *Annals of Operations Research*, 320:795–830.

Gök, Y. S., Tomasella, M., Guimarans, D., and Ozturk, C. (2020). A Simheuristic Approach for Robust Scheduling of Airport Turnaround Teams. In *Proceedings - Winter Simulation Conference*, volume 2020-December, pages 1336–1347. Institute of Electrical and Electronics Engineers Inc.

Goldsman, D. and Goldsman, P. (2015). Discrete-Event Simulation. In Loper, M. L., editor, *Modeling and Simulation in the Systems Engineering Life Cycle*, pages 103–109. Springer London.

Goodarzian, F., Abraham, A., Ghasemi, P., Di Mascolo, M., and Nasseri, H. (2021). Designing a green home healthcare network using grey flexible linear programming: Heuristic approaches. *Journal of Computational Design and Engineering*, 8(6):1468–1498.

Google Developers (2022). OR-Tools. https://www.gov.uk/holiday-entitlement-rights [Accessed: (02/08/2022)].

Guimarans, D., Arias, P., and Mota, M. M. (2015). Large Neighbourhood Search and Simulation for Disruption Management in the Airline Industry. In *Applied Simulation and Optimization*, pages 169–201. Springer International Publishing.

Guo, J. and Bard, J. F. (2024). Air traffic controller scheduling. *Computers & Industrial Engineering*, 191:1–21.

Gurobi (2024). Gurobi software. https://www.gurobi.com [accessed: 02/03/2024].

Haddock, J. and Mittenthal, J. (1992). SIMULATION OPTIMIZATION USING SIMULATED ANNEALING. *Computers ind. Engng*, 22(4):387–395.

Hajad, M., Tangwarodomnukun, V., Jaturanonda, C., and Dumkum, C. (2019). Laser cutting path optimization using simulated annealing with an adaptive large neighborhood search. *International Journal of Advanced Manufacturing Technology*, 103(1-4):781–792.

Han, J., Chen, C., Tiong, R. L. K., Wu, K., and Chew, D. K. H. (2024). Strategic Workforce Planning for Production of Prefabricated Bathroom Units: An Advanced Markovian Approach. *Journal of Construction Engineering and Management*, 150(8).

Herroelen, W. and Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165(2):289–306.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems*. The MIT Press.

IBM (2024). Ibm ilog cplex optimization studio. https://www.ibm.com/products/ilog-cplex-optimization-studio?mhsrc=ibmsearch_a&mhq=CPLEX [Accessed:15/08/2024].

Ingels, J. and Maenhout, B. (2015). The impact of reserve duties on the robustness of a personnel shift roster: An empirical investigation. *Computers and Operations Research*, 61:153–169.

Ingels, J. and Maenhout, B. (2017). Employee substitutability as a tool to improve the robustness in personnel scheduling. *OR Spectrum*, 39(3):623–658.

Ingels, J. and Maenhout, B. (2018). The impact of overtime as a time-based proactive scheduling and reactive allocation strategy on the robustness of a personnel shift roster. *Journal of Scheduling*, 21(2):143 – 165.

Ingels, J. and Maenhout, B. (2019). Optimised buffer allocation to construct stable personnel shift rosters. *Omega (United Kingdom)*, 82:102–117.

Jacquillat, A. and Odoni, A. R. (2015). An integrated scheduling and operations approach to airport congestion mitigation. *Operations Research*, 63(6):1390–1410.

Jacquillat, A. and Odoni, A. R. (2018). A roadmap toward airport demand and capacity management. *Transportation research. Part A, Policy and practice*, 114:168–185.

Jaillet, P., Loke, G. G., and Sim, M. (2022). Strategic Workforce Planning Under Uncertainty. *Operations Research*, 70(2):1042–1065.

Jnitova, V., Elsawah, S., and Ryan, M. (2017). Review of simulation models in military workforce planning and management context. *Journal of Defense Modeling and Simulation*, 14(4):447–463.

Juan, A. A., Faulin, J., Grasman, S. E., Rabe, M., and Figueira, G. (2015). A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems. *Operations Research Perspectives*, 2:62–72.

Juan, A. A., Keenan, P., Martí, R., Mcgarraghy, S., Panadero, J., Carroll, P., and Oliva, D. (2023). A review of the role of heuristics in stochastic optimisation: from metaheuristics to learnheuristics. *Annals of Operations Research*, 320:831–861.

Keskin, M., Çatay, B., and Laporte, G. (2021). A simulation-based heuristic for the electric vehicle routing problem with time windows and stochastic waiting times at recharging stations. *Computers & Operations Research*, 125:1–15.

Kiermaier, F., Frey, M., and Bard, J. F. (2016). Flexible cyclic rostering in the service industry. *IIE Transactions*, 48(12: Operations Engineering & Analytics):1139–1155.

Kiermaier, F., Frey, M., and Bard, J. F. (2020). The flexible break assignment problem for large tour scheduling problems with an application to airport ground handlers. *Journal of Scheduling*, 23(2):177–209.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220(4598):671–680.

Kisialiou, Y., Gribkovskaia, I., and Laporte, G. (2018). Robust supply vessel routing and scheduling. *Transportation Research Part C: Emerging Technologies*, 90:366–378.

Kisialiou, Y., Gribkovskaia, I., and Laporte, G. (2019). Supply vessel routing and scheduling under uncertain demand. *Transportation Research Part C: Emerging Technologies*, 104:305–316.

Kızıloğlu, K. and Sakallı, Ü. S. (2023). Integrating Flight Scheduling, Fleet Assignment, and Aircraft Routing Problems with Codesharing Agreements under Stochastic Environment. *Aerospace*, 10(12).

Kletzander, L. and Musliu, N. (2022). Hyper-Heuristics for Personnel Scheduling Domains. In *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling*.

Kletzander, L., Musliu, N., Gärtner, J., Krennwallner, T., and Schafhauser, W. (2019). Exact methods for extended rotating workforce scheduling problems. *Proceedings of the 29th International Conference on Automated Planning and Scheduling, ICAPS*, pages 519–527.

Komarudin, Guerry, M. A., De Feyter, T., and Vanden Berghe, G. (2013). The roster quality staffing problem - A methodology for improving the roster quality by modifying the personnel structure. *European Journal of Operational Research*, 230(3):551–562.

Kostanek, E. and Khoreva, V. (2018). Multi-generational Workforce and Its Implication for Talent Retention Strategies. In *Psychology of Retention*, pages 203–221. Springer International Publishing, Cham.

Kuo, Y. H. (2014). Integrating simulation with simulated annealing for scheduling physicians in an understaffed emergency department. *HKIE Transactions Hong Kong Institution of Engineers*, 21(4):253–261.

Laporte, G. (1999). The art and science of designing rotating schedules. *Journal of the Operational Research Society*, 50(10):1011–1017.

Laporte, G. and Pesant, G. (2004). A general multi-shift scheduling system. *Journal of the Operational Research Society*, 55:1208–1217.

Larsen, R. and Pranzo, M. (2019). A framework for dynamic rescheduling problems. *International Journal of Production Research*, 57(1):16–33.

Lau, H. C. (1996). On the complexity of manpower shift scheduling. *Computers Ops Res*, 23(1):93–102.

Law, A. M. (2015). *Simulation Modeling and Analysis*. McGraw-Hill Education, New York, 5 edition.

Levner, E., Kats, V., Alcaide López De Pablo, D., and Cheng, T. C. (2010). Complexity of cyclic scheduling problems: A state-of-the-art survey. *Computers and Industrial Engineering*, 59:352–361.

Liao, S., Van Delft', C., Koole, G., Dallery, Y., and Jouini, O. (2009). Call center capacity allocation with random workload. In *2009 International Conference on Computers and Industrial Engineering, CIE 2009*, pages 851–856.

Llort, N., Lusa, A., Martínez-Costa, C., and Mateo, M. (2019). A decision support system and a mathematical model for strategic workforce planning in consultancies. *Flexible Services and Manufacturing Journal*, 31(2):497–523.

Loper, M. L. and Register, A. (2015). Introduction to Modeling and Simulation. In *Modeling and Simulation in the Systems Engineering Life Cycle*, pages 3–16.

Lusby, R., Dohn, A., Range, T. M., and Larsen, J. (2012). A column generation-based heuristic for rostering with work patterns. *Journal of the Operational Research Society*, 63(2):261–277.

MacDonald, L. and Paul, J. A. (2024). A risk analytics model for strategic workforce planning: readiness of enlisted military personnel. *Annals of Operations Research*, 338(1):513–533.

Madigan, D. J. and Kim, L. E. (2021). Towards an understanding of teacher attrition: A meta-analysis of burnout, job satisfaction, and teachers' intentions to quit. *Teaching and Teacher Education*, 105:103425.

Maenhout, B. and Vanhoucke, M. (2009). The impact of incorporating nurse-specific characteristics in a cyclical scheduling approach. *Journal of the Operational Research Society*, 60(12):1683–1698.

Maenhout, B. and Vanhoucke, M. (2023). Dynamic personnel rescheduling: insights and recovery strategies. *Journal of Scheduling*, 27:1–27.

Martin, S., Ouelhadj, D., Smet, P., Berghe, V., and Özcan, E. (2013). Cooperative search for fair nurse rosters. *Expert Systems with Applications*, 40:6674–6683.

MiniZinc Challenge 2021 results (2021). MiniZinc challenge 2021 results.

Morgenstern, M. S., Pires, F. G., Freitas, E. P., and Silva, L. A. L. (2020). Time-limited search and optimization of the spatial positioning of agents in virtual environments. In *IEEE Symposium Series on Computational Intelligence (SSCI)*.

Mörz, M. and Musliu, N. (2004). Genetic algorithm for rotating workforce scheduling problem. In *ICCC 2004 - Second IEEE International Conference on Computational Cybernetics, Proceedings*, pages 121–126.

Musliu, N., Gärtner, J., and Slany, W. (2002). Efficient generation of rotating workforce schedules. *Discrete Applied Mathematics*, 118(1-2):85–98.

Musliu, N., Schutt, A., and Stuckey, P. J. (2018). Solver Independent Rotating Workforce Scheduling. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10848 LNCS:429–445.

Musliu, N., Schutt, A., and Stuckey, P. J. (2019). Index of /staff/musliu/benchmarks.

Nasri, M., Metrane, A., Hafidi, I., and Jamali, A. (2020). A robust approach for solving a vehicle routing problem with time windows with uncertain service and travel times. *International Journal of Industrial Engineering Computations*, 11(1):1–16.

Neri, F. and Cotta, C. (2012). A Primer on Memetic Algorithms. In Neri, F., Cotta, C., and Moscato, P., editors, *Handbook of Memetic Algorithms*, volume 379, chapter 4. Springer.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. (2007). MiniZinc: Towards a standard CP modelling language. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4741 LNCS:529–543.

Olya, M. H., Badri, H., Teimoori, S., and Yang, K. (2022). An integrated deep learning and stochastic optimization approach for resource management in team-based healthcare systems. *Expert Systems With Applications*, 187.

Ouelhadj, D. and Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, 12(4):417–431.

Panadero, J., Juan, A. A., Ghorbani, E., Faulin, J., and Pagès-Bernaus, A. (2024). Solving the stochastic team orienteering problem: comparing simheuristics with the sample average approximation method. *International Transactions in Operational Research*, 31(5):3036–3060.

Parker, S. R. and Marriott, J. A. (1999). "Personnel Forecasting Strategic Workforce Planning" A Proposed Simulation Cost Modeling Methodology. In *Proceedings of the 1999 Winter Simulation Conference*, pages 1410–1414.

Pereira, I. and Madureira, A. (2013). Self-Optimization module for Scheduling using Case-based Reasoning. *Applied Soft Computing Journal*, 13(3):1419–1432.

Pesant, G. (2004). A regular language membership constraint for finite sequences of variables. In *International conference on principles and practice of constraint programming*, pages 482–495. Springer.

Petropoulos, F., Laporte, G., Aktas, E., Alumur, S. A., Archetti, C., Ayhan, H., Battarra, M., Bennell, J. A., Bourjolly, J. M., Boylan, J. E., Breton, M., Canca, D., Charlin, L., Chen, B., Cicek, C. T., Cox, L. A., Currie, C. S., Demeulemeester, E., Ding, L., Disney, S. M., Ehrgott, M., Eppler, M. J., Erdosğan, G., Fortz, B., Franco, L. A., Frische, J., Greco, S., Gregory, A. J., Hämäläinen, R. P., Herroelen, W., Hewitt, M., Holmström, J., Hooker, J. N., Işık, T., Johnes, J., Kara, B. Y., Karsu, ö., Kent, K., Köhler, C., Kunc, M., Kuo, Y. H., Letchford, A. N., Leung, J., Li, D., Li, H., Lienert, J., Ljubić, I., Lodi, A., Lozano, S., Lurkin, V., Martello, S., McHale, I. G., Midgley, G., Morecroft, J. D., Mutha, A., Oğuz, C., Petrovic, S., Pferschy, U., Psaraftis, H. N., Rose, S., Saarinen, L., Salhi, S., Song, J. S., Sotiros, D., Stecke, K. E., Strauss, A. K., Tarhan, I., Thielen, C., Toth, P., Van Woensel, T., Berghe, G. V., Vasilakis, C., Vaze, V., Vigo, D., Virtanen, K., Wang, X., Weron, R., White, L., Yearworth, M., Yıldırım, E. A., Zaccour, G., and Zhao, X. (2023). Operational Research: methods and applications. *Journal of the Operational Research Society*, 75:423–617.

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435.

Purnomo, H. W. and Bard, J. F. (2007). Cyclic preference scheduling for nurses using branch and price. *Naval Research Logistics (NRL)*, 54(2):200–220.

Raychaudhuri, S. (2008). Introduction to Monte Carlo simulation. In *2008 Winter Simulation Conference*, pages 91–100. IEEE.

Rocha, M., Oliveira, J. F., and Carravilla, M. A. (2013). Cyclic staff scheduling: optimization models for some real-life problems. *Journal of Scheduling*, 16(2):231–242.

Rocha, M., Oliveira, J. F., and Carravilla, M. A. (2014). A constructive heuristic for staff scheduling in the glass industry. *Annals of Operations Research*, 217(1):463–478.

Ropke, S. and Pisinger, D. (2006). An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, 40(4):455–472.

Rossit, D. G., Vigo, D., Tohmé, F., and Frutos, M. (2019). Visual attractiveness in routing problems: A review. *Computers & Operations Research*, 103:13–34.

Schulte, C., Lagerkvist, M., and Tack, G. (2022). Gecode: generic constraint development environment. `www.gecode.org` [Accessed: (02/08/2022)].

Schwemmer, J., Günsel, C., Kühn, M., and Schmidt, T. (2023). Workforce Rostering for Decentrally Controlled Production Systems: A Simulation-based Optimization Framework using a Genetic Algorithm. *Logistics Research*, 16(1).

Simon, H. A. (2019). *The Sciences of the Artificial, reissue of the third edition with a new introduction by John Laird*. MIT press.

Smit, Y., den Hengst, F., Bhulai, S., and Mehdad, E. (2023). Strategic Workforce Planning with Deep Reinforcement Learning. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 13811 LNCS, pages 108–122. Springer Science and Business Media Deutschland GmbH.

Sokol, Marc Barry, . e. and Tarulli, Beverly, e. (2024). *Strategic Workforce Planning: Best Practices and Emerging Directions*. New York, NY : Oxford University Press.

Steenweg, P. M., Schacht, M., and Werners, B. (2021). Evaluating shift patterns considering heterogeneous skills and uncertain workforce availability. *Journal of Decision Systems*, 30(1):27–49.

Stuckey, P. J., Feydy, T., Schutt, A., Tack, G., and Fischer, J. (2014). The minizinc challenge 2008-2013. *AI Magazine*, 5:55–60.

Stuckey, P. J., Marriott, K., and Tack, G. (2022). The minizinc handbook. `https://www.gov.uk/holiday-entitlement-rights` [Accessed: (23/03/2022)].

Sutton, C., Prowse, J., McVey, L., Elshehaly, M., Neagu, D., Montague, J., Alvarado, N., Tissiman, C., O'Connell, K., Eyers, E., Faisal, M., and Randell, R. (2023). Strategic workforce planning in health and social care – an international perspective: A scoping review.

TAV Technologies (2023). Resolving Airport Capacity Constraints Through Flow Management. https://airportindustry-news.com/resolving-airport-capacity-constraints-through-flow-management/ [Accessed: (12/09/2024)].

The Port Authority of New York and New Jersey (2019). 2019 Airport Traffic Report. https://www.gov.uk/holiday-entitlement-rights [Accessed: (08/08/2022)].

Triska, M. and Musliu, N. (2011). A Constraint Programming Application for Rotating Workforce Scheduling. In Mehrotra, K., Mohan, C., Oh, J., and Varshney, P.K.Ali, M., editors, *Developing Concepts in Applied Intelligence. Studies in Computational Intelligence*, volume 363, pages 83–88. Springer, Berlin, Heidelberg.

Turan, H. H., Elsawah, S., Jalalvand, F., and Ryan, M. J. (2020). Solving Strategic Military Workforce Planning Problems with Simulation-optimization. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1620–1625. IEEE.

UK Civil Aviation Authority (2024). CAA Aviation Trends Quarter 2 2024. https://www.caa.co.uk/media/akqfqyxo/aviation-trends-2024-q2.pdf [Accessed: (12/09/2024)].

UK Government (2024). Holiday entitlement. https://www.gov.uk/holiday-entitlement-rights [Accessed: (02/04/2024)].

Van Den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., and De Boeck, L. (2013). Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367–385.

van Hoeve, W.-J. and Katriel, I. (2006). Chapter 6 - Global Constraints. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, pages 169–208.

van Twist, R., van den Akker, M., and Hoogeveen, H. (2021). Synchronizing transportation of people with reduced mobility through airport terminals. *Computers and Operations Research*, 125.

Wallace, M. (2020). *Building decision support systems: using MiniZinc*. Springer.

Whitaker, E. T. (2015a). Agent-Based Simulation. In Loper, M., editor, *Modeling and Simulation in the Systems Engineering Life Cycle*, pages 139–155. Springer London.

Whitaker, E. T. (2015b). System Dynamics. In Loper, M. L., editor, *Modeling and Simulation in the Systems Engineering Life Cycle*, pages 157–165. Springer London.

Wickert, T. I., Smet, P., and Vanden Berghe, G. (2021). Quantifying and enforcing robustness in staff rostering. *Journal of Scheduling*, 24:347–366.

Wiesflecker, J. (2022). Benchmark instances used for experiments. https://www.gov.uk/holiday-entitlement-rights.

Wiesflecker, J., Tomasella, M., and Archibald, T. W. (in press - 2024). Simheuristics for strategic workforce planning at a busy airport. In *2024 Winter Simulation Conference (WSC)*. IEEE.

Willis, G., Cave, S., and Kunc, M. (2018). Strategic workforce planning in healthcare: A multi-methodology approach. *European Journal of Operational Research*, 267(1):250–263.

WorldData.info (2024). The 30 largest airports and airlines in the United Kingdom. `https://www.worlddata.info/europe/united-kingdom/airports.php` [Accessed: (28/08/2024)].

Yadav, N. and Tanksale, A. (2023). A multi-objective approach for reducing Patient's inconvenience in a generalized home healthcare delivery setup. *Expert Systems with Applications*, 219.

Zeng, L., Zhao, M., and Liu, Y. (2019). Airport ground workforce planning with hierarchical skills: a new formulation and branch-and-price approach. *Annals of Operations Research*, 275(1):245–258.

Zhang, D., Wang, X., Li, S., Ni, N., and Zhang, Z. (2018). Joint optimization of green vehicle scheduling and routing problem with time-varying speeds. *PLoS ONE*, 13(2).

Zhang, S., Chen, M., Zhang, W., and Zhuang, X. (2020). Fuzzy optimization model for electric vehicle routing problem with time windows and recharging stations. *Expert Systems with Applications*, 145:113123.

Zhou, Y., Yang, J. j., and Huang, Z. (2020). Automatic design of scheduling policies for dynamic flexible job shop scheduling via surrogate-assisted cooperative co-evolution genetic programming. *International Journal of Production Research*, 58(9):2561–2580.

Zografos, K. G., Madas, M. A., and Androutsopoulos, K. N. (2017). Increasing airport capacity utilisation through optimum slot scheduling: review of current developments and identification of future needs. *Journal of Scheduling*, 20:3–24.