Enhancing Decision Support Systems for Airport Slot Allocation

Sha Wang

A thesis presented for the degree of

Doctor of Philosophy

School of Electronic Engineering and Computer Science
Queen Mary University of London

August 2022

I, Sha Wang, confirm that the research included within this thesis is my own work or that where it has been carried out in collaboration with, or supported by others, that this is duly acknowledged below and my contribution indicated. Previously published material is also acknowledged below.

I attest that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge break any UK law, infringe any third party's copyright or other Intellectual Property Right, or contain any confidential material.

I accept that the College has the right to use plagiarism detection software to check the electronic version of the thesis.

I confirm that this thesis has not been previously submitted for the award of a degree by this or any other university.

Signature: Sha Wang

Date: 23/08/2022

Details of collaboration and publications:

The work reported in this thesis has been supported by the Engineering and Physical Sciences Research Council (EPSRC) through Programme Grant EP/M020258/1 'Mathematical Models and Algorithms for Allocating Scarce Airport Resources (OR-MASTER)'.

During the development of this thesis, one publication was produced.

S. Wang, J. H. Drake, J. Fairbrother and J. R. Woodward, "A Constructive Heuristic Approach for Single Airport Slot Allocation Problems," 2019 IEEE Symposium Series on Computational Intelligence (SSCI), 2019, pp. 1171-1178, doi: 10.1109/SSCI44817.2019.9002892.

Acknowledgement

Undertaking this PhD has been a challenging and rewarding experience for me, especially during a global pandemic. Without the support and guidance I received from many people, I would not have been able to complete it.

First, I would like to thank my supervisor, Dr John Woodward, for all the support and encouragement he gave me for academic research and my personal development. I would also like to thank Dr John Drake (University of Leicester) for his constructive guidance and valuable comments on my research. My PhD would not have been fruitful without his input. In addition, I would like to thank Prof Edmund Burke for leading the OR-MASTER project. I gratefully acknowledge the funding received for my PhD from the Queen Mary University of London.

I would particularly like to thank the researchers from Lancaster University. Prof Konstantinos Zografos has provided me with airport slot request data to run experiments and offered me strategic and professional insights into the research topic. My biggest thank you is to Dr Jamie Fairbrother, with whom I co-authored a paper. A lot of work in this thesis is inspired by his previous work. My thanks also go to Dr Fotios Katsigiannis and Dr Robert Shone for giving me valuable feedback.

Moreover, my thanks go to all academics in my research group, especially Dr Jun Chen, Dr Michal Weiszer, Dr Xinwei Wang and Yujian Gan. I appreciate their help during the entire time of my PhD. I would like to thank my previous colleagues, Dr Simon Rawles, Dr Pol Arias and Dr Sachchida Chaurasia, who made the time in the group pleasant.

I would also like to thank my dear friends, especially the Stringer family, Dr Yan Yang, Ranulf Doswell, Yueyue Si, Dr Tong Bai, Jianshu Qiao and people from the QM Chinese Christian fellowship. Thank you for being there for me through all the ups and downs in life.

Last but not least, I am deeply grateful to my family for their unconditional love and support. I will always owe my achievements to my loving and caring family.

Abstract

Due to the growing imbalance between air traffic demand and airport capacity at congested airports, airlines must secure slots to operate flights at capacity-constrained airports. In practice, slot allocation is performed by independent slot coordinators at each airport according to a set of principles and regulations. As a result, the current decision-making system is considered inefficient and does not take adequate account of the complexity of real-world problems. Therefore, optimisation techniques are needed to improve airport capacity management and slot allocation.

This thesis aims to contribute to single airport slot allocation research by providing an in-depth analysis of the slot request data and developing new models and solution algorithms to deal with large-scale slot allocation problems. First, we propose a new model considering slot rejections (SASA-R) based on the maximum acceptable displacement of slots to support the decision-making of rejecting slots. In addition, we analyse the impact of changing the current slot allocation rules on slot allocation results. Second, we propose a two-stage approach that aims to solve large-scale slot allocation problems. A greedy constructive heuristic is developed to generate feasible solutions in a short time. This initial feasible solution is then improved by an adaptive large neighbourhood search heuristic (ALNS). A novel related destroy operator is designed specifically for this problem. The results show high-quality solutions can be obtained within a few hours for the problem instance tested, while a commercial optimisation solver does not return a feasible solution after several days of computation. Third, we propose a flexible slot allocation model to allocate slots individually on different days of the week. This model enhances existing models by enabling coordinators to explore the trade-off between schedule regularity and flexibility. The results show that the flexible scheduler can simultaneously reduce the number of rejected slots and schedule displacement.

Contents

List of Tables

List of Figures

Glossary

ad hoc slot An allocated slot which is not eligible for historic precedence. vii, 10, 11, 54, 68, 125

airport capacity Maximum number of passengers or aircraft that can be accommodated in a certain period of time. 1

equivalent seasons Consecutive summer seasons (two summers) or consecutive winter seasons (two winters) as opposed to two consecutive seasons (a summer and a winter season). 8, 9, 11

historic precedence The principle whereby airlines are entitled to a series of slots that were operated at least 80% of the time during the period allocated in the previous equivalent season, also known as the 'grandfather rights'. 9, 10, 11

Level 3 A Level 3 airport is also known as a coordinated airport, where the demand for flight operations exceeds the airport declared capacity significantly. 1, 83

schedule displacement The time difference between requested and allocated time of slot. 4, 67, 70, 74, 76, 78, 81, 82, 84, 85, 88, 89

series of slots At least 5 slots allocated for the same or approximately same time on the same day-of-the-week, distributed regularly in the same season. vii, 4, 10, 12, 54, 68, 69, 70

slot The permission granted for an airline to use the full range of airport infrastructure at a coordinated airport at a specific date and time. 1

slot pool The slots available at a Level 3 airport at initial allocation after unchanged historic slots are allocated, including any newly created slots. 11, 12, 66, 149

slot coordination The generic term encompassing slot allocation at a Level 3 airport. 7, 8, 38

slot coordinator The organisation or individual responsible for slot allocation at a Level 3 airport. 2, 3, 8, 9, 10, 19, 20, 21, 45, 48, 57, 82

waitlist A list of slot requests with no slot allocated. 10, 45

Acronyms

ALNS Adaptive Large Neighbourhood Search. 90, 102

API Application Programming Interface. 74, 110, 133

GAP Generalised Assignment Problem. 39

GP Genetic Programming. 42

IATA International Air Transport Association. 1, 7, 26, 50, 54

ILP Integer Linear Programming. 19, 29

ILS Iterated Local Search. 36

LNS Large Neighbourhood Search. 42, 102

MIP Mixed Integer Programming. 24, 74, 111, 133

MRGAP Multi-Resource Generalised Assignment Problem. 39

SAL Slot preliminary Allocation List. 9

SASA-R Single Airport Slot Allocaiton- with Rejections. 4, 5, 6, 144, 145, 149

SC Slot conference. 10

SHL Slot Historics List. 8, 9

SOSTA Simultaneous Optimisation of the airport SloT Allocation. 35

SSIM Standard Schedules Information Manual. 48

TFI Timing Flexibility Indicator. 13, 24

TILP Truncated Integer Linear Programming. 36

VNS Variable Neighbourhood Search. 36

WASG Worldwide Airport Slot Guidelines. 2, 7

Chapter 1

Introduction

1.1 Background and Motivations

According to a recent report from EUROCONTROL (2022), the number of global flights in 2050 is expected to reach 16 million annually in the most-likely scenario. That is a 44% increase compared to 11 million in 2019. The growth of flights, however, is constrained by the maximum capacity available at several extremely congested airports. Furthermore, due to the high level of uncertainty and the COVID-19 pandemic, it is predicted that the total airport capacity increase by 2040 is only a few flights; as a result, less than half a million flights will not be accommodated in the most-likely scenario.

Due to the severe imbalance between air traffic demand and airport capacity at congested airports, the International Air Transport Association (IATA) introduced 'slots' to measure airport runway capacity and allocate limited airport capacity to competing airlines. Airport slots, or slot, refers to the permission granted for a planned flight operation to use a Level 3's (also known as 'coordinated' airports) full range of infrastructure at a specific time and date (WASG, 2020). Airports are designated as Level 3 if

demand exceeds the declared capacity of the airport significantly. Slot allocation is currently practiced at 200 Level 3 airports of the busiest airports outside the United States, serving over 1.5 billion passengers - 43% of global traffic each year (IATA, 2019). In Europe, the slot allocation is governed by Regulation 95/93/EEC (Commission, 1993) and Worldwide Airport Slot Guidelines (WASG).

Slot allocation is currently performed by independent slot coordinators at each airport according to a set of principles and regulations. As a result, the current slot allocation system is considered inefficient and does not consider all the real-world complexity. There is increasing research on the reformation of the slot allocation rules, and more attention is being paid to enhancing the decision-making system for slot allocation by optimisation techniques. However, research on slot allocation currently suffers from several main limitations: (1) lack of in-depth analysis of slot request data and understanding of the difference in slot demand among Level 3 airports; (2) oversimplified objectives and constraints of slot allocation model that do not consider the interests of all stakeholders and the real-world complexity; (3) limited research on the impact of changes in regulatory rules on slot allocation; (4) lack of consideration of the uncertainties involved in airport declared capacity assessment; (5) lack of capability to deal with slot allocation at very congested single airports or airport-network level problems. Therefore, there is enormous room to improve slot allocation results by developing more intelligent decision-making frameworks.

1.2 Research Objectives and Scope

My research aims to contribute to airport slot allocation research by better understanding the real-world slot request data and the features of the prob-

lem to develop new models and solution algorithms for single airport slot allocation problems. My research objectives include three aspects:

- To improve the existing mathematical models of single airport slot allocation problems to support slot coordinators to make better decisions, such as rejecting slot requests, implementing different slot allocation priority regimes and flexible slot allocation.
- To develop faster solution algorithms for the existing slot allocation models, such as a constructive heuristic to generate feasible solutions quickly and speed up the solving time of the optimisation solvers.
- To develop solution algorithms for solving large-scale slot allocation problems, where exact methods are computationally impractical. In addition, the solution method should also be easy to adapt to different models and instances. Finally, to investigate the trade-off between solving the slot allocation model using exact and heuristic methods.

While the research focuses on single airport slot allocation problems, the research outcomes are also helpful for solving network-wide airport slot allocation problems. A constructive heuristic, for example, can provide feasible solutions for each airport in an airport network in a short time.

1.3 Contributions of this Thesis

The main contributions can be summarised as follows:

- An analysis of the real-world airport slot request data, investigating the underlying properties of the problem being solved and the complex interactions between the key component of the model (Chapter 3).

- Development of a new Single Airport Slot Allocation model with consideration of slot Rejections (SASA-R). This is the first investigation of slot rejections under different maximum displacement thresholds (Chapter 4).

- Development of a greedy constructive heuristic which can generate feasible solutions to the SASA-R model quickly (Chapter 5).

- Development of a self-adaptive improvement heuristic based on the large neighbourhood search algorithm with a novel related destroy operator. The results show high-quality solutions can be obtained within a few hours for the tested instance, while a commercial optimisation solver does not return a feasible solution after three days of computation (Chapter 5).

- Development of a flexible slot allocation model which allocates a series of slots individually on different days of the week and the time difference of slots can be controlled. The model provides a trade-off between schedule regularity and schedule flexibility. The flexible scheduler can significantly reduce the number of rejected slots and schedule displacement in lower priority classes (Chapter 6).

1.4 Publications

During the development of this thesis, one publication was produced. It is related to the work presented in Section 5.2 in Chapter 5.

- S. Wang, J. H. Drake, J. Fairbrother and J. R. Woodward, "A Constructive Heuristic Approach for Single Airport Slot Allocation Problems," 2019 IEEE Symposium Series on Computational Intelligence

(SSCI), 2019, pp. 1171-1178, doi: 10.1109/SSCI44817.2019.9002892.

One manuscript was submitted to Evo*2023 (Evostar 2023). It is related to the work presented in Section 5.2 in Chapter 5.

- D. Melder, J. H. Drake, S. Wang,"An Evolutionary Hyper-Heuristic for Airport Slot Allocation".

Two manuscripts are in preparation for submission to the Transportation Science journal.

- Chapter 3

  D. Melder, S. Wang, J. H. Drake, "An analysis of real-world airport slot allocation problems", (2022). Manuscript in preparation.

- Chapter 5

  S. Wang, J. H. Drake, J. R. Woodward,"An adaptive large neighbourhood search heuristic for solving large-scale slot allocation problems", (2022). Manuscript in preparation.

1.5 Structure of the Thesis

The main body of this thesis is organised as follows: Chapter 2 provides an overview of the latest slot allocation guidelines and reviews existing research on slot allocation problems for both single airport and airport networks. In addition, related research areas and solution methodologies are reviewed. In chapter 3, three real-world slot request data sets are analysed. The results provide insights into the variation of slot demand at different coordinated airports. Chapter 4 proposes a new single airport slot allocation model (SASA-R), which considers slot rejections under the maximum allowable slot displacement threshold. The model is then used to investigate

the possible changes to the current slot allocation rules. Chapter 5 proposes a two-stage solution method to solve the SASA-R model. It consists of a greedy constructive heuristic and an adaptive large neighbourhood search improvement heuristic. Chapter 6 presents a new flexible slot allocation model (flexible scheduler). The solution obtained from this model can provide a better trade-off between schedule regularity and flexibility. Finally, the thesis ends with a summary of key results and future work in Chapter 7.

Chapter 2

Background and Related Work

This chapter provides an overview of the Worldwide Airport Slot Guidelines (WASG)(WASG, 2020) and a critical review of existing literature on single airport and airport network-wide slot allocation problems. In addition, we give a brief review of the related research areas and solution methodologies. The limitations of the existing literature are discussed throughout this chapter, and future research needs are summarised at the end.

2.1 Overview of the Worldwide Airport Slot Guidelines

The Worldwide Airport Slot Guidelines work as the industry standard for strategic slot coordination at capacity-constrained airports, also known as Level 3 airports, where the airport capacity is insufficient to meet the demand of airlines and other air carriers. The latest WASG was published by the International Air Transport Association (IATA), Airports Council In-

ternational, and the Worldwide Airport Coordinators Group in June 2020. These guidelines are produced jointly by airports, airlines, and slot coordinators and revised regularly to reflect the proven best practice of the slot coordination procedure. The following sections will discuss slot allocation policy, process, priority precedence, and other regulatory requirements.

2.1.1 General policies of slot allocation

The prime objective of slot coordination is to ensure the most efficient management and allocation of airport slots to maximise benefits to air passengers while considering the interests of airports and airlines. Hence, these guidelines are developed to ensure reliable, consistent, and flexible schedules to meet consumers' demands. Meanwhile, slots at congested airports should be allocated openly, fairly, transparently, and non-discriminatory by the independent slot coordinator appointed for each airport.

2.1.2 Slot allocation process

The strategic slot allocation is conducted twice a year for the summer or winter scheduling seasons. Due to seasonal travel demands, some airports are only slot coordinated for the summer or winter scheduling season. The slot allocation process follows the coordination timetable (Chapter 10 in WASG (2020)). Fig. 2.1 gives an example of the 'Calendar of Coordination Activities' for the summer scheduling season of 2019. Below are the key dates and activities of the slot allocation process:

- Slot Historics List (SHL): More than six months before the start of the upcoming scheduling season, slot coordinators review airlines' utilisation of slots in the previous equivalent seasons (e.g., winter to winter)

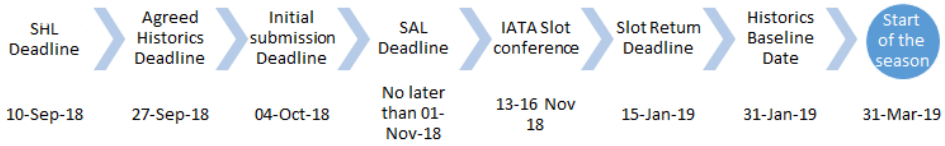


Figure 2.1: Calendar of Coordination Activities

to identify slots that are qualified for the historic precedence for the next scheduling season.

- Agreed Historics Deadline: Once they receive the SHL messages, airlines must confirm their precedence status by the Agreed Historics Deadline. Any disagreement regarding the status of historic precedence must be resolved with slot coordinators before the Initial Submission Deadline.

- Confirm coordination parameters and declared capacities: One to two weeks before the Initial Submission Deadline, airports must provide coordinators with final coordination parameters for the next scheduling season. Coordination parameters consist of all functional limitations at the airport (i.e., runway, terminal, and apron capacities), airspace capacities, and actual utilisation of airport declared capacity of the busiest week of the previous equivalent seasons.

- Initial Submission Deadline: About five to six months before the start of the next scheduling season, airlines submit their slot requests for their planned services to slot coordinators. Descriptions of slot requests will be given in Section 3.1.

- Slot preliminary Allocation List (SAL) Deadline: The initial slot allocation is conducted between the Initial Submission Deadline and the SAL Deadline. Within this time frame, usually 2 to 3 weeks, slots

are allocated at each airport independently. All airlines are informed about their allocated slots at the same time, about two weeks prior to the start of the slot conference. If the initial demand for slots exceeds the capacity limit available for allocation, a 'No slot' waitlist is created to hold rejected slots.

- Slot conference (SC): The initial slot allocation result will be discussed at the slot conference, which is attended by airline representatives, slot coordinators, airport representatives and observers. Airports can request new slots or make adjustments to their allocated slots in order to improve their schedules. Slots can also be exchanged or transferred between airlines. In addition, the cancellation of unwanted slots is encouraged to improve the resource utilisation rate and slot allocation efficiency.

- Series Return Deadline: Airlines must return a series of slots they do not intend to operate by this deadline to give coordinators sufficient time to replace them with other series of slots or slots for ad hoc slot services on the waitlist.

- Historics Baseline Date: It is the start date used for determining eligibility for historic precedence for the following scheduling season. All cancellations made after this date are considered slot misuse in the 80% utilisation rate calculation for the historic precedence

2.1.3 Slot allocation rules

The initial slot allocation only concerns slots for series services. Series services refer to a regular flight operating for at least five weeks for the same time on the same day of the week in the same scheduling season. Services

shorter than five weeks are regarded as ad hoc slot services and not assigned any slots in the initial allocation stage.

Precedence rules for slot allocation

According to the latest slot allocation rules (WASG (2020) section 8.1), when allocating slots, the following slot precedence must be observed.

- Unchanged historic slots: Slot allocation goes first to unchanged historical slots. They have not changed operations from the previous equivalent seasons or have changes that do not impact the coordination parameters (e.g., a change in flight number). We refer to these slots as *historic* slots herein. In order to qualify for the historic precedence, also known as the 'grandfather rights', airlines must use at least 80% of the series of *historic* slots in the previous equivalent seasons so that they can have the same slots for the next scheduling season.

- Slot pool: Once all *historic* slots have been allocated, a slot pool will be established to hold the remaining available slots for the following: i) *Change-to-historic* slots are slots with changes to historical slots that impact the coordination parameters (e.g., a change in flight time, aircraft or terminal); ii) *New entrants* carriers can request for no more than six slots (for three pairs of arrival and departure flight movements) on any day of the scheduling season. New entrants that are offered slots within one hour before or after the requested slots but do not accept this offer by the end of the first day of the slot conference day will no longer be eligible for the new entrants status; iii) *Others* are the remaining slots requested by non-new-entrant carriers for additional slots to the ones they already hold or by carriers who lost their historical slots.

- The 50/50 rule: 50% of the slots in the slot pool must be allocated to *New entrants* and the other 50% to *Others* unless requests in such class are less than 50%.

It should be emphasised that the latest slot allocation priority rules differ from those of the previous guidelines, where slots are allocated hierarchically in the order of *Historic*, *Change-to-historic*, *New entrants* and *Others*. According to the latest slot allocation rules (WASG (2020) section 8.1), after allocating all *Historic* slots, *Change-to-historic*, *New entrants* and *Others* slots must be allocated holistically and fairly. That is, these three priority classes should compete for available slots in the slot pool at the same time and be processed as a single batch.

Apart from the four primary priority classes, several additional criteria can be used for 'tie-breaking' purposes when allocating slots with the same precedence. Consideration should be given to the following factors in no particular order: year-round operations, the effective period of operation, aircraft size, connectivity, operational factors, type of consumer service, and market.

Other requirements for slot allocation

The following requirements for slot allocation must be considered when modelling the slot allocation problems:

- Schedule regularity: A flight operating for at least five weeks in the scheduling season must be given a series of slots at the same time of the day. Unless the airline indicates it is acceptable, coordinators should not offer different slots on different days of the week to the same flight (see WASG 10.10.2).

- Timing flexibility: Airlines must indicate the time range of slots they are willing to accept by specifying the TFI in their submitted requests. If the requested slots are unavailable, coordinators must offer slots between the requested slots and the historical slots or within any timing flexibility range indicated by the airline.

- Turnaround times: Turnaround operations are the activities conducted for preparing an inbound aircraft for its next outbound flight. The *requested turnaround time* refers to the time difference between a pair of slots requested for the arrival and departure flights operated by the same aircraft. Coordinators must not offer inconsistent turnaround times of the requested turnaround time while respecting minimum turnaround times and avoiding any increase in turnaround times.

2.2 Single Airport Slot Allocation Problems

The single airport slot allocation problem can be considered as an extension of the Multi-Resource Generalised Assignment Problem (MRGAP) (Gavish and Pirkul, 1991), which involves allocating a number of slots to a number of airlines' requests, with the aim of either maximising the airlines' preferences and/or being fair to all airlines. Meanwhile, various types of constraints must be satisfied. In this section, we will first review the problem constraints in Section 2.2.1 and the optimisation objectives in Section 2.2.2. After this, we will provide a literature review of the single airport slot allocation model and solution approaches in Section 2.2.3 and Section 2.2.4.

2.2.1 Constraints

Table 2.1 provides an overview of the constraints of single airport slot allocation problems considered in the existing literature.

Table 2.1: Constraints for single airport slot allocation problems: (1)Runway; (2)Apron; (3)Terminal Passengers; (4)Turnaround time; (5)Slot preferences; (6)Accessibility

Literature	(1)	(2)	(3)	(4)	(5)	(6)
Zografos et al. (2012)	✓			✓		
Zografos and Jiang (2016)	✓			✓		✓
Zografos et al. (2018)	✓			✓	✓	
Ribeiro et al. (2018)	✓			✓		
Ribeiro et al. (2019b)	✓	✓	✓	✓		
Ribeiro et al. (2019a)	✓	✓	✓	✓		
Zografos and Jiang (2019)	✓			✓		
Wang et al. (2019)	✓			✓		
Androutsopoulos et al. (2020)	✓			✓		
Fairbrother et al. (2020)	✓			✓	✓	
Fairbrother and Zografos (2021)	✓			✓		
Jiang and Zografos (2021)	✓			✓		
Jorge et al. (2021)	✓			✓		
Katsigiannis and Zografos (2021)	✓	✓	✓	✓	✓	
Katsigiannis et al. (2021)	✓			✓		

(1) Runway capacity

Airport runway capacity represents the primary resource of an airport. The declared runway capacity describes the number of aircraft movements (landing or take-off) that can be accommodated during a specific period, in other words, the number of slots available for slot allocation. The declared runway capacity can be fixed for a day or the entire scheduling season or varying for periods such as peak hours, weekends or holidays (Ribeiro et al., 2019b). In addition, there can be individual capacities for different types of movements.

However, due to the concentration of slot demand during the most popular periods, the runway capacity is typically expressed in a rolling capacity form. For example, there can be limits on the number of slots available for arrivals per hour, rolling every 15 minutes (e.g., 9:00-10:00, 9:15-10:15, etc.).

(2) Apron capacity

At some airports, the capacity of apron stands has created a bottleneck in the airport capacity. Katsigiannis and Zografos (2021) and Ribeiro et al. (2019b) have taken this into account by limiting the number of parked aircraft of a specific type (light, medium, heavy) at a terminal during a fixed period of a day or rolling time horizons.

(3) Terminal passenger capacity

Slots available for allocation may be limited by the terminal's passenger capacity, which is affected by several factors, including passenger types (Schengen/Non-Schengen, Domestic/International, etc.) and aircraft movement types (arrival or departure). Katsigiannis and Zografos (2021) considered the terminal passenger capacity in the form of rolling time horizons of a day, and Ribeiro et al. (2019b) used a fixed passenger capacity for each day.

(4) Turnaround times

Minimum turnaround time constraints ensure that an inbound aircraft has sufficient time to be prepared for its next outbound flight. The minimum turnaround time for low-cost flights is typically 30 minutes and an hour and a half for large aircraft and premium airlines. Due to operating costs and environmental impact, coordinators should avoid offering a pair of slots

with an increase in the requested turnaround times. However, only a few studies considered the upper limit for the turnaround time. Katsigiannis and Zografos (2021) limits the turnaround time for each pair of flights to the requested turnaround time. Ribeiro et al. (2018, 2019b) investigated the trade-off between the schedule displacement and changes in requested turnaround times. Results suggested that allowing a 5-minute change in the requested turnaround time can significantly reduce the total schedule displacement.

(5) Airline slot preferences

Airline slot preference constraints are motivated by improving the acceptability and utilisation of slots. In cases where airlines specify the range of slots they are willing to accept, it is only feasible to offer them slots within that range. In the absence of airlines' preferences information, Pellegrini et al. (2012, 2017) adopted the maximum displacement constraint to limit the maximum displacement of individual slots. Zografos et al. (2018) simulated airlines' tolerance for a given maximum displacement and minimised the number of violated assignments. Katsigiannis and Zografos (2021) constructed artificial slot flexibility range parameters for requests with different priorities and proposed a membership function that penalises large slot displacement. Finally, Fairbrother et al. (2020) developed a slot exchange mechanism which allows airlines to specify preferences in a flexible manner subject to a displacement budget.

(6) Airport accessibility

The airport accessibility constraints ensure that a minimum number of slots must be allocated to flights connecting small, remotely located airports with

major hub airports. This constraint has only been considered in Zografos and Jiang (2016).

2.2.2 Objectives

The results of slot allocation are measured by schedule efficiency and schedule fairness. Many previous studies have focused on improving schedule efficiency through minimising the schedule displacement, while some other research have simultaneously considered schedule fairness. Only a few papers in the literature (Zografos et al., 2012; Fairbrother and Zografos, 2021) considered one objective for the problem, more studies have considered multiple objectives. Table 2.2 provides an overview of the different objectives considered in the existing literature.

Table 2.2: Optimisation objectives for single airport slot allocation problems: to minimise (1)rejected requests; (2)rejected slots; (3)maximum displacement; (4) schedule displacement; (5)displaced slots; to maximise (6)inter-airline fairness

	(1)/(2)	(3)	(4)	(5)	(6)
Zografos et al. (2012)			✓		
Zografos and Jiang (2016)			✓		✓
Zografos et al. (2018)		✓	✓		
Ribeiro et al. (2018)	✓	✓	✓	✓	
Ribeiro et al. (2019b)		✓	✓	✓	
Ribeiro et al. (2019a)		✓	✓		
Zografos and Jiang (2019)			✓		✓
Androutsopoulos et al. (2020)		✓	✓		
Fairbrother et al. (2020)			✓		✓
Fairbrother and Zografos (2021)			✓		
Jiang and Zografos (2021)			✓		✓
Jorge et al. (2021)	✓	✓	✓		✓
Katsigiannis et al. (2021)		✓	✓		✓
Katsigiannis and Zografos (2021)	✓	✓	✓		

(1) Minimising the number of rejected requests

The number of rejected requests was modelled as a minimisation objective in Katsigiannis and Zografos (2021) model.

(2) Minimising the number of rejected slots

This objective concerns the number of rejected slots rather than the number of requests. A request typically consists of a group of slots on different days of the season. All slots in the same request must be allocated or rejected at the same time (Ribeiro et al., 2018; Jorge et al., 2021).

(3) Minimising the maximum displacement

The displacement of an individual slot refers to the absolute time difference between the time of the requested and allocated slots. The maximum displacement indicates the worst case of displacement across all slots or requests, as all slots submitted in the same request will have the same displacement. Therefore, minimising the maximum displacement will improve schedule acceptability and inter-flight fairness.

(4) Minimising the schedule displacement

Schedule displacement is the sum of the displacement of all slots, and it is the most common metric to quantify schedule efficiency. However, Androutsopoulos et al. (2020) argues that large slot displacement should be penalised over small displacement. Thus, they proposed a squared schedule displacement cost function, in which the displacement of each slot equals the square of the absolute time difference between the allocated and requested slots.

(5) Minimising the number of displaced slots

Airlines may be offered displaced slots when their requested slots are not available. They have a chance to reject the offer or negotiate with slot coordinators during the slot conference. This objective is motivated by the consideration of improving slot acceptability to simplify the negotiation process during the slot conference (Ribeiro et al., 2018).

(6) Maximising schedule fairness

In addition to schedule efficiency, fairness is an essential criterion for slot allocation outcomes and has recently been extensively studied in the literature. The inter-flight fairness measures seek to distribute schedule displacement fairly among flights (e.g., to minimise the maximum displacement of all slots). The inter-airline fairness measures seek to distribute schedule displacement fairly among airlines based on their demand for slots or contribution to the congestion Jiang and Zografos (2021); Fairbrother et al. (2020).

In addition to the above objectives, Jorge et al. (2021) proposed several new objectives according to the additional principles of slot allocation in the WASG, such as minimising the schedule displacement for long operation duration services or long-haul flights.

2.2.3 Review of existing models

Single airport slot allocation problems have been modelled as Integer Linear Programming (ILP) in the literature. To our knowledge, Zografos et al. (2012) proposed the first model for this problem with a single objective to minimise the schedule displacement. The model was solved for three small to medium-sized coordinated airports using an exact method and a row-

generation algorithm. Results suggested that the schedule displacement can be reduced by between 14% and 95% compared to the slot allocation result made by slot coordinators.

This model was later extended by Zografos and Jiang (2016) to incorporate schedule fairness objectives. The objectives are to minimise the schedule displacement and maximise schedule fairness, considering airport accessibility. First, the schedule displacement is weighted by aircraft seat number and flight distance. Next, a relative fairness indicator is calculated for each airline, which is the ratio of the proportion of displacement allocated to this airline to the number of slots requested by this airline. The fairness objective is to minimise the variance of relative fairness for all airlines. Finally, airport accessibility was incorporated to ensure sufficient slots are allocated to flights connecting a small local airport to a hub airport. Zografos and Jiang (2019) developed and solved a bi-objective model which minimises the schedule displacement and the maximum deviation of the fairness indicator for all airlines from the average. This model was solved hierarchically using the ϵ-constraint method and a row generation approach developed by Zografos et al. (2012). Results showed that there is a trade-off between schedule fairness and schedule displacement. However, the proposed fairness metric could potentially result in a large displacement to airlines that requested more slots, thus not considered fair to established airlines. Recently, Jiang and Zografos (2021) investigated three inter-airline fairness metrics: absolute, relative, and the Gini index. Each fairness metric was combined with the schedule displacement objective, resulting in three different bi-objective models. The ϵ-constraint method was used to solve the bi-objectives models and generate efficient Pareto frontiers for alternative fairness objectives. After this, a voting mechanism based on the majority rule was used by

airlines to decide the preferable Pareto frontier. Then the slot coordinator selects the preferable acceptable solution on the preferable frontier. Results suggested that the model with the relative fairness metric yields the best solution. However, the preferable acceptable solution depends significantly on the relative weights of schedule displacement and fairness objectives set by airlines and slot coordinators.

Ribeiro et al. (2018) proposed a novel multi-objective Priority-based Slot Allocation Model (PSAM) which captured all the primary slot allocation requirements of the guidelines. The model adopted a weighted-based objective function that considered four factors with decreasing weights: the number of rejected slots, maximum displacement, total schedule displacement and the number of displaced slots. A lexicographic approach was employed to solve the model sequentially for each slot priority. Results showed that the developed modelling and formulation strengthening approach could provide optimal solutions for a medium-sized airport examined in a few minutes. However, the solving time of the model was considered sensitive to the weights of the different objectives.

Ribeiro et al. (2019b) extended the previous model to a new model PSAM-ATR that incorporates apron and terminal capacity and ignores the number of rejected slots objective. In addition, a weight-based approach is used to integrate slot priorities. Results suggested that PSAM-ATR can be solved optimally or near-optimally at a very busy airport (more than 200,000 flight movements per year) in a few hours of computation. Most importantly, the most significant contribution of this research is that it examined several potential changes to the current slot allocation rules. The changes include flexibility in setting the turnaround time, considering the impact of displacement on passengers, slot time flexibility for historical slots, weighted priori-

ties, prioritising new-entrant slots over change-to-historic slots, relaxation of schedule regularity constraints and adjusting the airport's declared capacity. Results showed that minor adjustments to the rules could significantly improve airport capacity utilisation and to better meet airlines' needs. Ribeiro et al. (2019a) proposed a novel mat-heuristic solution approach based on large-scale neighbourhood search to solve the model previously presented in Ribeiro et al. (2019b). The details of this algorithm will be discussed later in 2.2.4. Results suggested that solutions can be found to be within 2-5% of the optimum after 30 minutes and within 0-0.03% of the optimum after 10 hours of computation. In contrast, with a commercial solver, the optimality gap was 5-10% after two days and 0.5-2% after 7 days. The result looks promising for dealing with the busiest airports such as the tested Lisbon Airport with over 200,000 annual flight movements.

Jorge et al. (2021) also extended the PSAM to integrate priorities for year-round operations. Nine optimisation objectives associated with the WASG additional criteria (see 2.1.3) were formulated and replaced previous objectives of the PSAM. Due to the computational time limit, the solution generation aimed at finding a good set of efficient solutions instead of the Pareto frontier. A decision-making tool was also developed to assist coordinators in comparing and analysing different efficient solutions. Moreover, solutions can be visualised and ranked by the tool. Different solutions with the same objective function value can be compared by analysing how slot requests swap across a pair of such solutions and the corresponding displacement of these requests.

Another stream of literature focuses on investigating the trade-off between different objectives. Zografos et al. (2018) proposed and solved a bi-objective model to investigate the trade-off between schedule displacement

and maximum displacement. Results suggested that only a small sacrifice of schedule displacement could reduce the maximum displacement significantly and improve the schedule acceptability. Another proposed model allows one to investigate the trade-off between schedule displacement and the number of violated slot assignments under various declared capacity levels and values of maximum acceptable slot displacement (simulated as a constant parameter for all slots). Findings suggested that tolerating 15 to 30 minutes of slot displacement can substantially benefit the reduction of both violated slot assignments and schedule displacement. Findings suggested that tolerating 15 to 30 minutes of slot displacement can substantially benefit the reduction of both violated slot assignments and schedule displacement. Androutsopoulos et al. (2020) proposed a bi-objective model to examine the trade-off between the total schedule displacement and the squared maximum slot displacement. The underlying motivation is to promote airlines' acceptability and utilisation of allocated slots by penalising largely displaced slots. The solution approach is a hybrid algorithm based on the Feasibility Pump heuristic(Fischetti et al., 2005) and Large Neighbourhood Search. Also, new problem instances have been generated based on real slot request data for a Greek Regional Airport.

Some other research focused on slot allocation with flexibility. Fairbrother and Zografos (2021) proposed a new model called segmentation scheduler. By using this model, slots in different segments of the season are allocated independently. Meanwhile, the range of slot times on different segments for each request is limited in order to maintain the desired level of schedule regularity. In addition, the idea of changing the series threshold was tested. Results suggested that optimal solutions are not sensitive to the different segmentation methods but are much more sensitive to the number

of segments and the flexibility range parameter. Only a small number of requests received varying slots across segments. Due to the computational complexity of this flexible slot allocation model, it has only been solved for a small coordinated airport. Katsigiannis and Zografos (2021) modelled a flexible slot allocation problem using a Mixed Integer Programming (MIP) which incorporates airlines' schedule flexibility preferences and dynamic allocation of the runway, terminal's passenger and apron capacity. A two-stage solution framework was proposed. In the first stage, the model is solved without the flexibility to minimise lexicographically the number of rejected requests, maximum displacement and schedule displacement for each priority group hierarchically. In the second stage, a Timing Flexibility Indicator (TFI) model was formulated. A weighted objective function was employed to minimise the number of requests satisfying airlines' flexibility preferences. It is reported that the joint consideration of dynamic allocation of capacity and TFI improved all schedule quality metrics significantly and are better than when considering them individually. However, by considering the TFI, higher priority slots are observed to receive an increased maximum displacement than lower priority slots.

Androutsopoulos and Madas (2019) extended the basic strategic slot allocation model to incorporate fairness constraints. The model assumes that of a fair schedule, the total schedule displacement imposed on each airline should reflect the additional schedule displacement caused by scheduling flights of this particular airline. In other words, airlines that plan to operate more flights during peak hours should be assigned higher displacement weights. The displacement weighting scheme does not only consider the number of slots requested per airline and provides a pre-specified displacement share of each airline in the model.

Fairbrother et al. (2020) proposed a two-stage slot scheduling mechanism that incorporates schedule efficiency, fairness and airlines' preferences. The mechanism consists of first constructing a fair reference schedule and then adjusting this using airlines' displacement preferences. A new demand-based fairness measure was proposed, which requires that the total displacement allocated to an airline should be proportionate to its number of requested slots in peak periods rather than simply the number of requests it makes. Numerical tests demonstrated that the demand-based fairness approach provides a better trade-off for schedule displacement versus fairness and requires less computational time when compared to the non-demand-based fairness approach. In the second stage, airlines propose preferred displacement for each of its requests subject to constraints that ensure the total preferred displacement is at least equal to its allocated displacement in the fair reference schedule, also referred to as displacement budget. This allows airlines to redistribute the displacement budget from requests that should be prioritised to requests that are not a priority. The proposed approach was tested using real request data and simulated airline displacement preference data. Results suggested that schedule fairness can be significantly improved with only small increases in schedule displacement. In addition, the displacement budget mechanism seems to be effective in terms of schedule improvements requested by airlines.

Katsigiannis et al. (2021) proposed another multi-objective formulation. In this model, three previously proposed objectives were considered: schedule displacement, maximum displacement and demand-based fairness. With the proposed multi-level solution approach, one can explore the trade-offs between the objectives of the various slot allocation hierarchies and their impact on non-hierarchy schedules. The authors also proposed a multi-level

solution framework. This approach finds a set of solutions rather than a single optimal solution for each level (priority class) and solves the corresponding multi-objective model. Once all priority classes are processed, solutions for each level will be aggregated to complete solutions and dominated solutions will be filtered out. It was found that allowing for small increases of the objective value for historic and change-to-historic slots could result in a more efficient overall schedule and significantly improve the objective value of the new-entrant and other non-new-entrant requests. However, the computational time for the proposed approach was relatively long even when solving small-sized problems. Hence, specialised heuristics are required for solving the multi-level, multi-objective problem at larger airports.

To support the assessment of the potential performance of strategic flight schedules, Lambelho et al. (2020) proposed an machine-learning based approach which evaluates a schedule with respect to predicted flight delays and cancellations six months before flight execution. This research provides a means to rank strategic schedules and an indication on the potential performance of the IATA guidelines-compliant schedules.

2.2.4 Review of existing solution methods

Incorporating slot priorities

The four primary slot priorities considered in the literature are *historic*, *change-to-historic*, *new entrants* and *others*. Slots requested for public service obligations are considered as a distinct priority class and was given the highest priority for slot allocation by Katsigiannis and Zografos (2021). Within each priority class, Jorge et al. (2021) prioritised year-round operations over single-season operations. Ribeiro et al. (2018) explicitly formulated slot displacement constraints for two different types of *change-to-*

historic slots. Zografos et al. (2018); Zografos and Jiang (2019) allocated *historic* and *change-to-historic* slots as a whole due to a lack of slot priority information. It is worth mentioning that almost all of the research reviewed in this chapter was conducted before the latest WASG was released, in which *change-to-historic*, *new-entrants* and *others* are required to be allocated holistically with equal priority. Thus, the existing slot allocation models may not reflect the latest priority rules accurately.

Slot priorities can be incorporated by solving the slot allocation model in a sequential manner or weighted-based manner. By using the sequential approach, the model is solved lexicographically (Ribeiro et al., 2018; Zografos et al., 2012, 2018) or hierarchically (Fairbrother et al., 2020; Fairbrother and Zografos, 2021; Jiang and Zografos, 2021). Both approaches solve a number of sub-problems formed by each priority class. In the hierarchical approach, once the sub-problem for the higher priority class has been solved, the decision variables for the higher priority class are fixed and the remaining capacity for lower priority classes is updated to maintain solution feasibility. In the lexicographic approach, once the sub-problem for a higher priority class has been solved, only the optimal objective function value is fixed and it is added as a constraint to the sub-models for lower priority classes to ensure that the slot allocation result remains optimal for the higher priority class. It is possible that there exists more than one optimal solution for each priority class, therefore fixing the values of decision variables might constrain the slot allocation for lower priority classes more than necessary. Lastly, in the weighted-based approach, the objective function of each priority class is weighted according to its priority (Ribeiro et al., 2019b). The advantages of this method are: the model only needed to be solved once rather than multiple times for each priority class considered, and one can

easily examine the impact of revising the order of priorities by assigning higher weights for slots that needed to be prioritised. Experiment results showed that the weighted-based approach obtained the same solution as the sequential approach but in a significantly shorter time.

Dealing with more than one objective

Most existing single airport slot allocation models considered more than one objective. The optimal (or most preferred) solution for these models is known as the Pareto optimal solution (also called efficient or non-dominated solution), whose performance cannot be improved without degrading the performance of at least one of the other objectives. A Pareto frontier (or efficient frontier) refers to the set of Pareto optimal solutions (Mavrotas, 2009). Three methods have been used in literature to deal with models with more than one objective.

The first method is the weighted sum method (Marler and Arora, 2010). It is the simplest way to transform a multi-objective model into a single objective model. Ribeiro et al. (2018, 2019b,a) used different weighting parameters to reflect the relative importance of each objective in the objective function. By modifying the weights, different solutions on the Pareto optimal frontier can be achieved. However, the major drawback of this method is that the appropriate weights need to consider all stakeholder's interests and be consistent with the slot allocation rules.

The second method used is the ϵ-constraint method, which was first proposed by Haimes (1971). Specifically, this method selects one of the objectives to optimise and converts the other(s) into model constraints (Zografos et al., 2018; Zografos and Jiang, 2019; Jiang and Zografos, 2021). By systematically adjusting the values of the objective functions not be-

ing optimised, one can generate or approximate the Pareto frontier of the multi-objective model Jorge et al. (2021).

The third method is the lexicographic method, which has also been used to solve slot allocation models for different priority classes. This method assumes that the objectives can be ranked in the order of importance and therefore largely depends on the weights of each objective determined by decision-makers Katsigiannis and Zografos (2021).

Exact Methods

Exact methods are developed to find the optimal solution to an optimisation problem. The optimality of the solution can be proved. One of the exact algorithms, 'Branch-and-Bound', was proposed as early as the 1960s by Land and Doig. It has been widely applied to solve NP-hard problems. In addition, another exact algorithm 'Cutting Plane' works by solving the linear programming relaxation of the given Integer Linear Programming (i.e., discarding the integer constraints). The algorithm iteratively adds linear inequality constraints, called a cut, that leads to an integer solution without excluding any integer feasible points until an optimal integer solution is found.

Previous research has shown that for small to medium-sized instances, the exact method can be a very good strategy as it can possibly find optimal solutions in an acceptable amount of time. When solving large-scale problems (e.g., problems with a large number of decision variables or highly constrained), exact methods become computationally impractical. Zografos et al. (2012); Zografos and Jiang (2019) proposed an exact solution approach that first reduces the number of scheduling days by representing days having exactly the same slot requests as one single day. Next, a row

generation procedure was used to only generate capacity and turnaround time rows when they are violated by the optimal solution for the linear relaxation of the model. Fairbrother et al. (2020); Fairbrother and Zografos (2021) implemented a similar algorithm in which the capacity constraints are added as lazy constraints via a callback to the optimisation solver. Results showed that these solution approaches are only effective when applied to small to medium-sized airports. However, the performance can be further improved by using the Reduce-and-Split cut generation (Andersen et al., 2005). Ribeiro et al. (2018) developed new constraints to restrict the feasible region of the linear relaxation. It was reported that by using this method, the required computation time was reduced significantly.

Heuristics

A heuristic is considered as any solution approach that employs a practical method to generate sufficiently good solutions in a reasonable amount of time, but optimality cannot be guaranteed. Therefore, heuristics are often employed when the model cannot be solved by exact methods in an acceptable amount of time. Two types of heuristics exist in the literature: constructive heuristics and improvement heuristics. Constructive heuristics aim to construct a complete feasible solution, while improvement heuristics aim at improving the quality of the feasible solution based on perturbation algorithms (Pearl, 1984).

Several constructive heuristics have been proposed in the current literature. Some of them are purely based on heuristics, such as the greedy constructive heuristic proposed by Wang et al. (2019). This work will be introduced later in Section 5.2). The approach depends on different request ordering heuristics, by which requests are ordered according to static metrics

or dynamic metrics. A greedy allocation algorithm was developed to allocate slots to each request with the minimum slot displacement. The allocation algorithm is similar to the one presented in (Fairbrother and Zografos, 2021). Fairbrother and Zografos (2021) proposed a constructive heuristic and an improvement heuristic which iteratively modifies a non-flexible solution to a flexible solution. Both these approaches can produce feasible solutions in a very short period of time, but the solution quality is not good enough. However, it could speed up the solving process of the exact solver therefore significantly reducing the required computation time. Other constructive heuristics proposed in the literature are based on matheuristic (i.e., mathematical programming with heuristics) (Boschetti et al., 2009). Ribeiro et al. (2019a) proposed a group-based constructive heuristic. Firstly, requests are ordered in decreasing order of operating days. Secondly, requests are divided into a pre-defined number of groups each having roughly the same number of slots. Then, the model was solved sequentially for each group by using an exact solver. Results suggested that the effectiveness of this constructive heuristic is sensitive to the number of groups. There is a trade-off between the number of groups and the required computation time. Androutsopoulos et al. (2020) presented a constructive heuristic based on the Feasibility Pump (FP) heuristic (Fischetti et al., 2005). Instead of ordering requests first, all unscheduled requests are selected for scheduling by the roulette wheel selection mechanism. The selection probability of each request is defined by a regret metric and a remaining capacity metric. The underlying assumption is that the higher the regret cost and less remaining capacity at the closest feasible time interval, the higher probability of this request being scheduled earlier. Note that only a pre-defined number of requests (10%) are scheduled before the FP is applied. The FP aims to extend the par-

tial feasible solution to a complete feasible solution by solving a restricted version of the model and computes high-quality heuristic solutions.

Only a few improvement heuristics have been developed to deal with large-scale problems. Ribeiro et al. (2019a) developed an improvement heuristic based on the Large-scale Neighbourhood Search technique. A 'destroy and repair' process was implemented to iteratively re-optimise part of the current solution. The destroy operator selects a subset of requests that have been scheduled within a time window. The repair operator then solves the model only for the selected subset of requests by using an exact solver. Time windows are selected according to a probabilistic distribution which will be updated after each iteration. Every time a sub-problem cannot be solved to optimality or improved within a time limit, this sub-problem is considered to be computationally intensive or reached to optimality. Therefore, the number of requests (in the currently selected time window) that will be selected when this time window is next visited will be reduced by a factor. Meanwhile, the probability corresponding to this time window will also be reduced according to a parametric function. If a new optimal solution was found for the sub-problem, the values of all algorithm parameters remain unchanged. Experimental tests have demonstrated the benefits of using a large-scale neighbourhood search approach which allows the flexibility to swap slots in a larger neighbourhood. However, this algorithm involves a good amount of calibration parameters and only when each parameter is set to be around its 'sweet spot', the algorithm converges in a reasonable time. Androutsopoulos et al. (2020) proposed another solution approach based on the large neighbourhood search. The destroy operator removes a random percentage of requests at each iteration (20%-40%) and the repair operator applies the FP heuristic described earlier.

2.3 Slot Allocation Problems for Airport Network

Slot allocations are currently done individually at each coordinated airport. Therefore, it does not explicitly consider the inter-dependencies of slots allocated to each flight at the origin and destination airports. Several studies have identified and investigated on allocating slots simultaneously for an airport network.

2.3.1 Constraints

In order to allocate slots simultaneously and coherently at airports in a network, additional constraints must be taken into account in addition to those considered for single airport slot allocation problems 2.1.

Route requirements

A problem with slot allocation at each airport independently can arise when flights connecting two coordinated airports receive no slot or only receive slots at the origin or destination airports. Therefore, the flight coherency constraints must be considered to ensure that two coupled slots of the same flight at the origin and destination airports must be both allocated or none of them is allocated (Benlic, 2018; Pellegrini et al., 2012, 2017).

En-route duration requirements

The en-route duration requirements ensure that the time difference of the coupled slots assigned to a flight connecting two airports must be no longer than a given limit (Pellegrini et al., 2012) or fixed to the requested time (Benlic, 2018). In the model proposed by Pellegrini et al. (2017), the block time (i.e., the total amount of time a flight takes, from leaving the departure gate, "off-blocks", to arriving at the destination gate, "on-blocks") is

required to be within a pre-defined time range.

Airspace sectors capacity

Airspace capacity has only been modelled so far by Pellegrini et al. (2012) which describes the maximum number of aircraft movements per unit of time interval (typically an hour). It is motivated by achieving a high level of safety and contributing to the sustainable development of air transportation.

2.3.2 Objectives

Similar to single airport slot allocation models, schedule displacement is also the most commonly used objective for problems at the airport network level. Pellegrini et al. (2012, 2017) proposed a model which aims to first minimise the number of unaccommodated flights and second to minimise the schedule displacement cost. The displacement cost penalises both slot displacement and flight duration increase. Meanwhile, displacing slots is preferred over the increase in flight duration, and displacing one slot by two time intervals is more expensive than displacing the two slots for two arrival movements by one time interval each. In addition, Pellegrini et al. (2017) proposed two models with schedule fairness objectives. One can be interpreted as minimising the weighted sum of the average costs paid by airlines and the cost paid by the most penalised airlines. The second model takes into account airlines' size. Specifically, the costs of each airline are normalised by dividing by the number of airline's requests.

2.3.3 Review of existing models

Castelli et al. (2011b) proposed a mathematical model which considered the minimisation of schedule displacement cost and schedule fairness simul-

taneously. Pellegrini et al. (2012) proposed a model for the Simultaneous Slot Allocation Problem (SSAP). The model was tested on three randomly generated instances that assemble airport networks with various numbers of airports and flights and a hub and spoke structure. However, the model did not consider slot priorities and was only tested on a single day using artificial data. Pellegrini et al. (2017) proposed a multi-objective integer linear programming model named Simultaneous Optimisation of the airport SloT Allocation (SOSTA) and solved it exactly for a network of 120 European airports on the busiest day of 2013. It is worth mentioning that several model variants were proposed and tested. For example, the objective function can be reformulated to a quadratic displacement cost function. Compared to the model with linear displacement cost, the number of rejected requests was not changed, and more requests received slots with less displacement. Two formulations with inter-airline fairness consideration were proposed. The fairness is measured by first the maximum request rejection cost for all airlines and second the maximum total schedule displacement cost for all airlines. Benlic (2018) extended the single airport slot allocation model by Zografos et al. (2012) to the network level. The model was solved by a two-phase heuristic approach. Experiments were conducted on a set of artificial instances. According to the experimental results, taking into account the en-route constraints does not significantly affect the schedule displacement compared to allocating slots at each airport independently. Although the proposed heuristic gives us some confidence in solving large-scale realistic network-wide slot allocation problems, the generated instances and the proportional extrapolation of declared capacities are oversimplistic so they do not resemble a real-world problem precisely.

2.3.4 Review of existing solution methods

Exact Methods

Pellegrini et al. (2012) introduced the Truncated Integer Linear Programming (TILP) algorithm to solve the slot allocation model for a single day. It solves the SSAP model in a predetermined computation time and returns the optimal or the best solution found. Results suggested that TILP could not deal with large instances containing a realistic number of flights (about 30,000 in a day). It could only deal with small and medium instances (with 650 and 1,000 flights a day respectively) due to a large amount of computational time and memory requirements.

The SOSTA model (Pellegrini et al., 2017) was tested on an airport network with 120 airports and 32,655 slot requests on the busiest day in 2013. The proven optimal solution can be found by using a CPLEX solver in a few minutes. Sensitivity analysis pointed out that the SOSTA model is not badly affected by the increasing imbalance between demand and capacity, and does not significantly suffer from different weights of cost functions.

Heuristics

Castelli et al. (2011a) proposed an ant colony algorithm to allocate slots in an airport network. Results suggested that the algorithm is able to solve large instances in a short time. Pellegrini et al. (2012) solved the SSAP model by using two meta-heuristics: Iterated Local Search (ILS) (Lourenço et al., 2010) and Variable Neighbourhood Search (VNS) (Hansen and Mladenović, 2001). Both are based on two local search procedures, one aims to minimise the number of rejected flights, and the other one aims to minimise the scheduling cost of all accommodated flights. To avoid getting stuck in

the same locally optimal solution, the ILS relies on perturbing the current local optimal and iteratively calling the two local search procedures after starting from the modified solution. In contrast, the VNS explores distant neighbourhoods of the current solution, and searches from there to a new one if and only if an improvement is found. Results suggested that VNS works best on small and medium-sized instances and ILS is the best performing for large instances. Benlic (2018) developed a two-phase heuristic solution approach. In the first phase, a constructive heuristic generates an initial feasible solution with the aim of maximising the number of accommodated requests. Requests that result in infeasibility are excluded from consideration in the next phase. In the second phase, an iterative heuristic improves the initial solution in terms of the total schedule displacement by applying a destroy and repair procedure. Specifically, the constructive heuristic starts from a partially infeasible solution that contains requests violating one or more constraints. Then a set of conflicting requests were randomly selected and rescheduled to new slots. A tabu search-based algorithm was used in the rescheduling process to prohibit allocating conflicting requests back to the previously allocated time for the next few iterations. The improvement heuristic selects a set of displaced requests on a random basis and iteratively reschedules them to random slots to improve the solution quality. It has been demonstrated that the order in which the requests are allocated is important as it has a big impact on the overall quality and feasibility of final solutions.

2.4 Related Research Area and Solution Methodologies

2.4.1 Airport declared capacity modelling

Airport capacity declarations play a fundamental role in slot coordination. A thorough demand and capacity analysis is essential to assess the slot capacity declarations and capacity deliverability and should be undertaken regularly. An underestimation of the declared capacity can prevent airlines from accessing additional capacity and slots at their current airports or expanding into new markets. Overestimated declared capacity means more flights are scheduled than an airport can accommodate, which will contribute to a sharp rise in flight delays. Unfortunately, this process is lacking at multiple airports today (IATA, 2021). A number of research have been found focusing on developing analytical modelling to appropriately declare airport capacity, and the trade-offs between over-scheduling and under-scheduling (Barnhart et al., 2012; Ball et al., 2007; Odoni et al., 2011). In some studies, airport capacity and slot allocation are examined under stochastic conditions. Corolli et al. (2014) proposed two stochastic programming models for the airport network slot allocation problem under uncertainty. Scala et al. (2021) introduced a novel approach that combines the optimisation and simulation techniques for solving and evaluating a slot allocation problem under uncertainty.

2.4.2 Integration with other airport operations

The decision of slot allocation can affect the planning decision for other airport operations, such as crew scheduling, gate assignment and tactical slot allocation. Crew scheduling refers to the assignment of airport personnel

to maintain a schedule so they can meet organisational goals. Once slots have been assigned to aircraft or flights, a crew scheduler needs to develop a schedule such as assigning pilots to specific planes with considerations of the slot allocation solution, employee preferences and personnel cost, etc (Kohl and Karisch, 2004; Kasirzadeh et al., 2017). Karsu et al. (2021) formulated a mixed-integer model for the airport gate assignment problem to minimise the total walking distance of traveling passengers. Jacquillat and Odoni (2015) proposed an integrated approach to jointly consider strategic slot allocation efficiency and slot utilisation at the tactical level. Zeng et al. (2021) proposed a slot allocation mechanism to reduce the operational delay in the strategic slot allocation phase. The actual operation delay is estimated based on the historical operation data.

2.4.3 Multi-Resource Generalised Assignment Problem

The airport slot allocation problem studied in this thesis belongs to the Generalised Assignment Problem (GAP) which deals with assigning a number of tasks to a number of agents with minimum cost (Ross and Soland, 1975). Each task can only be assigned to one agent, and multiple tasks are allowed to be assigned to one agent subject to the resource availability of the agent in performing their tasks. Ross and Zoltners (1979) and Lourenço and Serra (1998) studied an extension of this problem which is referred to as the Multi-Resource Generalised Assignment Problem (MRGAP). It considers different types of resources that are required by each agent to perform each given task.

The assignment problem is a combinatorial optimisation problem with NP-hard complexity (Sahni and Gonzalez, 1976; Johnson and Garey, 1979). Many of the assignment problems including the MRGAP are modelled as

Integer Linear Programming (ILP). An ILP problem is a mathematical optimisation program in which all constraints and the objective function must be linear in the decision variables, and all the decision variables can only take integer values. The GAP has applications in many fields including transportation and routing, scheduling, telecommunication, production planning, supply chain and logistics, facility location, etc. For a survey of GAP variations and applications see Öncan (2007). Particularly, aviation-related applications have been found in airline fleet assignment (Barnhart et al., 2002), crew pairing (Kasirzadeh et al., 2017) and crew rostering problems (Kohl and Karisch, 2004).

2.4.4 Heuristic solution methodologies

Meta-heuristics

A meta-heuristic is a class of search strategies designed to guide the search process of a simple heuristic (Burke et al., 2014). Features that distinguish a meta-heuristic from a simple heuristic are as follows: (i) the framework of a meta-heuristic can be easily adapted to different optimisation problems, by virtue of its general nature, and can still meet the expectations of computation time and solution quality. In contrast, most simple heuristics are problem-dependent as they need to be designed according to the particularities of the problem; (ii) meta-heuristics have the capability to escape from a perpetual cycle of local optima, while simple heuristics usually adopt greedy algorithms which make the locally optimal choice at each step, thus are likely to get trapped in a local optimal solution (Yang, 2010). A selection of widely used meta-heuristic algorithms is briefly introduced below.

- **Tabu Search**: Tabu search was first introduced by Glover in 1986 and is motivated by the idea of modelling human memory processes. The primary feature of tabu search is to constrain an embedded heuristic from revisiting recently visited areas of the search space by maintaining a 'tabu list'. The tabu list may record recently visited solutions that must be avoided or specific move features that are not allowed for the next search step. This strategy allows the non-improving solutions to be accepted, thereby escaping from locally optimal solutions. However, the global optimal solution may not always be found, depending on parameter settings such as the size of the tabu list, size of the neighbourhood, move operator, etc (Gendreau, 2003). Recently, tabu-based algorithms have been applied to a network-wide slot allocation problem by Benlic (2018). More applications of tabu search in aircraft scheduling and assignment problems can be found in the following papers (Xu and Bailey, 2001; Atkin et al., 2008; Soykan and Rabadi, 2016)

- **Simulated Annealing**: Simulated Annealing (SA) was first applied in combinatorial optimisation by Kirkpatrick et al. in 1983 and is motivated by the similarity between the annealing process of solid materials and general combinatorial optimisation problems (Osman, 1995). The search process starts from a random state and picks a random move at each step. If the selected move improves the quality of the solution, then it is always accepted as the new current solution. Otherwise, it is accepted as the new current solution with some probability of less than 1. The probability decreases exponentially with the degradation level of the move, which is the amount by which the solution is worsened (Delahaye et al., 2019). SA-based algorithms have been applied

to solve many related problems including the airport gate assignment problem (Ding et al., 2005; Drexl and Nikulin, 2008), aircrew rostering problem (Lučić and Teodorović, 2007) and railway crew scheduling problem (Hanafi and Kozan, 2014).

- **Large Neighbourhood Search**: The meta-heuristic Large Neighbourhood Search (LNS) was first proposed by Shaw (1998). Large neighbourhood search methods explore a complex neighbourhood by use of heuristics. The neighbourhood of a solution is typically defined as the set of solutions that can be reached by first applying the destroy method and then the repair method (Pisinger and Ropke, 2019). A destroy method destructs part of the current solution while a repair method rebuilds the destroyed solution. Diversification and intensification are two critical factors for the destroy methods, they can be accomplished by using random destroy, worst or critical destroy (Vaz, 2015; Guimarans et al., 2015), related destroy (Shaw, 1998; Pillac et al., 2013; Praseeratasang et al., 2019) and history-based destroy methods (Pisinger and Ropke, 2007). The repair methods can be based on well-performing problem-specific heuristics or approximation algorithms or exact algorithms. Some examples are presented in (Bent and Van Hentenryck, 2004; Raidl and Puchinger, 2008; Hemmelmayr et al., 2012; Chen et al., 2018).

Genetic Programming

Genetic Programming (GP) is an evolutionary algorithm-based and domain-independent methodology to automatically find computer programs that perform user predefined tasks. GP iteratively transforms a population of computer programs into a new generation of programs by applying mecha-

nisms simulating the evolutionary process of natural selection (Koza, 1992). It has been successfully applied to solve many combinatorial optimisation problems with the goal of automating the heuristic design process. Some previous works employed GP as the hyper-heuristic to evolve local search heuristics ((Burke et al., 2010b; Drake et al., 2014)). An alternative application of GP is to evolve a heuristic function, which provides decision-making support for a specific construction heuristic (Burke et al., 2007b).

Hyper-heuristics

The development of hyper-heuristics is motivated by the goal of automating the design of heuristic algorithms to solve complex combinatorial optimisations. Hyper-heuristics can be considered as heuristics that select predefined low-level heuristics or generate new heuristics rather than searching for solutions directly. No single heuristic performs well in all respects. For some problem instances, simple heuristics cannot provide feasible solutions (Ross et al., 2002), or during the solution construction process, the state of the problem change so the heuristic in use might become less appropriate than another heuristic for the current problem state (Burke et al., 2007b). Hence, hyper-heuristic selection approaches are considered good strategies for intelligently selecting potential combinations or permutations of low-level heuristics.

According to Burke et al. (2010a), hyper-heuristics can be classified into two categories: heuristic selection and heuristic generation. Within each category, hyper-heuristics can be further classified as constructive or improvement hyper-heuristics in terms of the low-level heuristics employed. Recently, there are many successful studies in the literature which implemented hyper-heuristics. However, no previous work in the literature has

used hyper-heuristic approaches to solve the airport slot allocation problem. Examples of selection hyper-heuristics can be found in (Burke et al., 2007b) which investigated a hyper-heuristic selection framework for exam and course timetabling problems. More recently, Pour et al. (2018) employed a choice function hyper-heuristic framework for a maintenance task allocation problem. Examples based on hyper-heuristic generation can be found in (Burke et al., 2010b) which employed genetic programming to evolve 2-D strip packing heuristics. Drake et al. (2014) introduced a hyper-heuristic generation approach based on genetic programming for the knapsack problem.

Both simple heuristics and meta-heuristics can be used as high-level heuristics to select the most suitable low-level heuristic at each decision point. For example, tabu search can be used as the high-level heuristic to search for permutations of low-level heuristics in order to avoid repeating the trivial search. However, Iterated Local Search (ILS) and Variable Neighbourhood Search (VNS) are considered more effective than tabu search in terms of traversing the search space of heuristics (Burke et al., 2010a). In addition, early choices in the solution construction process affect later decisions, which means that low-level heuristics selected at the beginning have a greater impact on the overall quality of the solution (Qu and Burke, 2009).

In summary, given a set of low-level construction heuristics, hyper-heuristics approaches can be developed based on the following four aspects to (1) randomly select low-level heuristics or generate random lists of low-level heuristics, (2) select the best performing low-level heuristic at each step based on preliminary evaluation of low-level heuristics, (3) search a space of low-level heuristics based on meta-heuristics, e.g., tabu search, simulated annealing algorithm and (4) select low-level heuristics based on learning mechanisms

which take into account the historical performances of low-level heuristics (Chakhlevitch and Cowling, 2008).

2.5 Future research needs

Previous research has shown that airport slot allocation outcomes can be improved in many aspects. Since the main focus of this study is to enhance the decision support system for single airport slot allocation, we will discuss the future research needs on this aspect.

Slot rejection needs to be further investigated. At super congested airports, slot coordinators may reject some of the requests or put them on a waitlist when developing the initial slot allocation plan. This waitlist is then used for replacing any cancelled flights and returned request series. The current slot allocation regulations lack decision support for request rejection and slot reallocation. Only a few studies have considered minimising the number of rejected slots, but no requests were reported to be rejected in the existing literature. Fairbrother et al. (2020) raised the issue of fair distribution of rejections to airlines. This is a potential research direction that needs attention. In chapter 4, we investigate slot rejections due to unacceptable large slot displacement.

Priority of slot allocation. Incorporating different slot allocation priority rules affects both the modelling of the problem and the solution algorithm. The recently reformed regulations have stressed the importance of holistic slot allocation to further promote airline competition. Future research needs to provide more insight into this issue Odoni (2021). In chapter 4, we compare the holistic slot allocation with the hierarchic slot allocation and discuss the impact of the change of the rules on the slot allocation results.

Heuristics are needed to enhance the capability of existing slot allocation models. Almost all previous works have addressed the need to develop faster algorithms for solving the slot allocation problem at large airports. To our knowledge, Ribeiro et al. (2019a) proposed the only matheuristic solution method based on the large neighbourhood search to solve large-scale problems. In chapter 5, we propose a two-stage solution approach including a greedy constructive heuristic and an adaptive large neighbourhood search heuristic to enhance the model's capability to solve large-scale problems.

Flexible slot allocation needs to be further investigated. Toward improving the flexibility and efficiency of slot allocation, more research should be conducted on slot allocation mechanisms in the future. In chapter 6, we investigate the trade-off between schedule regularity and flexibility by using a newly proposed model.

Chapter 3

A Data Analysis of Real-world Slot Requests

Real-World slot request data is crucial in the research of slot allocation for better understanding actual airport slot demand, developing realistic slot allocation models, and validating proposed models and solution algorithms. Although significant research attention has been dedicated to solving slot allocation problems, relatively little work so far has sought to characterise and understand the underlying properties of the problem being solved. Therefore, this chapter presents an analysis of the real-world slot request data from three small and medium-sized coordinated airports.

The remainder of this chapter is as follows, we first introduce the key components of airlines' slot requests in Section 3.1. Section 3.2 provides in-depth data analysis of the airport slot demand patterns, exploring the relationships across different airports. Finally, in Section 3.3 we study the demand-capacity imbalance at each airport to better understand the underlying properties of the problem being solved.

3.1 Introduction of Slot Requests

In this section, we describe the components of a typical slot request provided by an airline to the slot coordinator. Table 3.1 shows an example of slot requests submitted by an arbitrary airline for the summer season of an arbitrary year. Each request is identified by a unique request code, and it typically includes two groups of slots, each group is for a series of arrival or departure flight movements. Note that two slots are required per flight operation, one for take-off and one for landing.

The summer season starts on the last Sunday in March and continues through to the Saturday of the last weekend in October. The 'startDate' and 'endDate' indicate the first and last planned operation day. The action code (column 'AC') indicates the type of operation, which also shows the priority of slot allocation. There are four primary priority classes: *Historic* (H), *Change-to-historic* (CH), *New entrants* (NE) and *Others* (OT). The operating days requested in a week are indicated by a 7 digits sequence 'Wdays'. For example, 1234500 represents Monday to Friday. The frequency indicator 'ReqFreq' indicates how often the operation will be carried out, for example, a 2 indicates once every two weeks. The requested arrival and departure time ('ReqArr' and 'ReqDep') is represented by 4 digits in 24-hour format. The overnight indicator 'OV' is a binary variable indicating whether or not the departure flight takes place on the same day as the arrival. The aircraft type and its seat number are provided in 'Aircraft' and 'Seats'. Finally, the origin, previous, next and final destination airport associated with each request is provided in the last four columns. For further information on the data contained in a slot request file, interested readers are referred to the Standard Schedules Information Manual (SSIM) documentation (IATA et al., 2020).

Table 3.1: Example of slot requests provided by airlines

Req code	Airline	startDate	endDate	AC	Wdays	ReqFreq	ReqArr	ReqDep	OV
a1_0001	AIR_1	25MAY	02OCT	NE	1000000	2	0800	0830	False
a1_0002	AIR_1	25MAY	01SEP	H	1234000	1	1400	1500	False
Req code	ArrFlight	DepFlight	Aircraft	Seats	Origin	Last	Next	Destination	-
a1_0001	5541	5542	319	150	VIE	VIE	VIE	VIE	-
a1_0002	4302	4303	738	186	DUS	DUS	PAD	PAD	-

A number of attributes can be derived from the original requests to provide more insights into the characteristics of slot demand at an airport, as demonstrated in Table 3.2. Firstly, the requested arrival and departure time ('ReqArr' and 'ReqDep') are converted to time interval indexes using this equation: time interval index= $hour * 60/ct + minute/ct$, where ct is the length of coordination time interval of the slot coordinated airport. The index is then rounded down to the nearest integer. Secondly, the number of operating days in a week 'NWdays' is derived directly from the operating days 'Wdays'. Thirdly, the start date, end date and frequency together determine the total number of operating weeks in the season 'Nweeks'. In addition, weeks in the season are indexed sequentially from 1 to 30. Thus, 'startWeek' indicates the index of the first operation week. Fourthly, 'Ndays' indicates the total number of operating days in the season, which equals half of the total number of slots 'Nslots' for a given request. Finally, 'reqTurn' represents the requested turnaround time which is the time difference between the requested departure and arrival time.

3.2 Statistical Analysis of Airport Attributes

In this section, we present a summary of the characteristics of the three real-world airports (Airport 1, 2, and 3), based on attributes calculated as described in Section 3.1 above. We analyse the demand patterns throughout

Table 3.2: Slot request attribute details

Attributes	Example	Variable type	Descriptions
reqCode	2009-A1-001	string	unique request ID
priority	H	categorical	request priority
airline	air_001	string	airline code
startDate	2009-03-29	date	first operation day
endDate	2009-05-29	date	last operation day
daySplit	0	binary	overnight indicator
ReqFreq	1	integer	frequency of operations
reqArr	34	integer	arrival slot index
reqDep	38	integer	departure slot index
Wdays	[0,3,4]	list	list of operating days
NWdays	3	integer	operating days
Nweeks	30	integer	number of operating weeks
startWeek	4	integer	start week index
Ndays	90	integer	total operating days of the season
reqTurn	4	integer	requested turnaround time

the season, first aggregated by days, then split by day of the week. We then examine the frequencies in terms of the number of operating days in a week, total operation weeks and total operating days. Following this, we look at the request priorities, presenting the breakdown of requests with regard to slot precedence. Next, we present and discuss the time distribution of flight operations throughout the day and the turnaround time distribution. Finally, we look at the associations between key attributes of slot requests.

Table 3.3 provides an overview of the request data from three real-world data sets initially presented by Zografos et al. (2012). Although the three airports can be considered small or medium-sized in terms of the annual number of flights and passengers, they were all categorised as coordinated airports by IATA as the demand for slots significantly exceeds the capacity

of these airports. In addition to the number of airlines, airlines' requests and requested slots, we also show the number of distinct scheduling days. Zografos et al. (2012) noted that some calendar days have identical sets of operations, which can be represented by a single day to simplify the model. However, this case seems to occur very rarely in the three data sets given that the total scheduling days in the season is 210 days.

Table 3.3: Summary of request data sets from the three real-world airports

Airport	Airlines	Requests	Slots	Scheduling days	Passengers for the season (million)
1	44	449	15,386	179	1.30
2	80	932	34,714	201	5.41
3	80	1,087	47,676	204	8.74

3.2.1 Slot demand characteristics

Figure 3.1 (left) shows the aggregated number of requested slots on each day of the season. We observe that slot demand follows a similar trend for the three airports. The number of requested slots is relatively small at the beginning of the season, then rises steadily from April to May, and stabilises at the highest level between June and September before it dropped back to a lower level. We observe a strong seasonality of slot demand within each week, corresponding to the peaks in the left figure. In order to further investigate the patterns in daily demand, the slot demand is split by day of the week in Fig. 3.1 (right). Interestingly, different patterns can be seen for the three airports. For example, Airport 3 is most busy on Mondays whereas Airport 1 is least busy on Mondays. This pattern can be explained partly by the primary type of travellers at these airports. Business travellers typically fly early and late in the week, and leisure travellers are typically flying at or around the weekend.

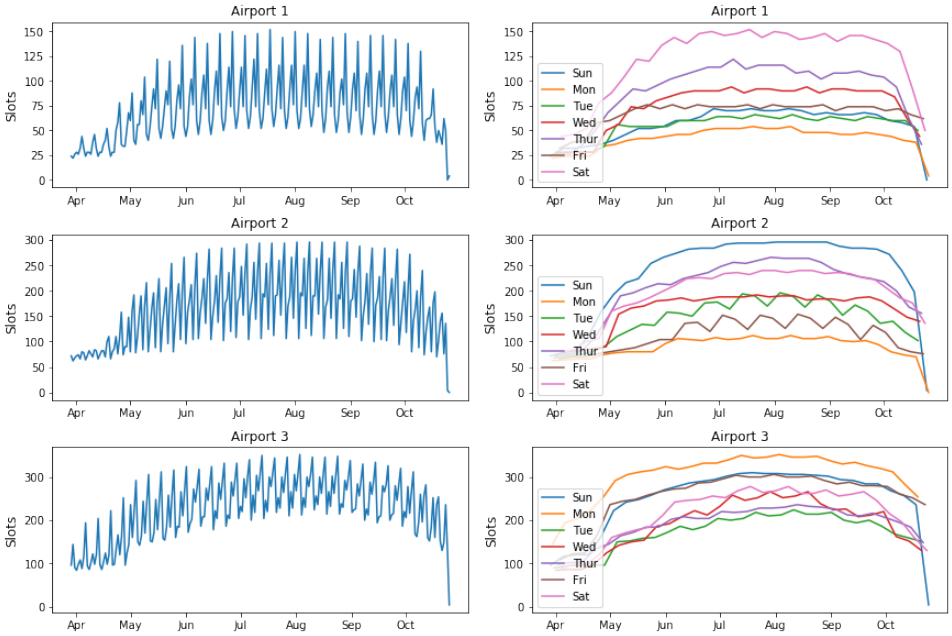
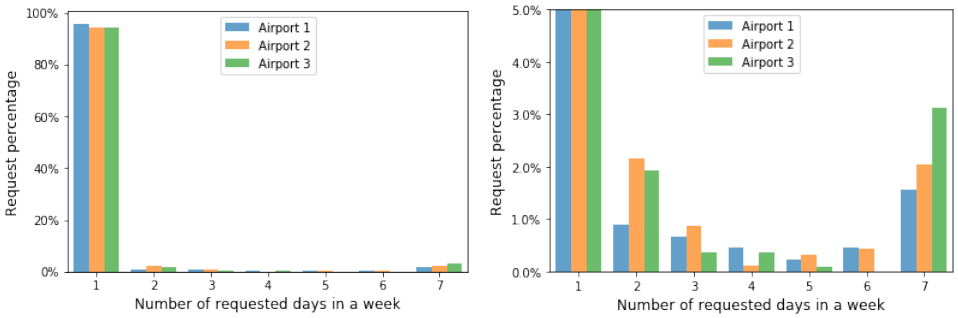


Figure 3.1: Number of slots requested for each day in the season

3.2.2 Flight operation frequency characteristics

We can measure frequency by the following three attributes of a request:

i. 'NWdays': Figure 3.2 (a) shows a histogram of the number of operating days for each request, with the y-axis indicating the percentage of requests. It can be seen that around 95% of flights only run once a week. The zoomed-in Fig. 3.2(b) shows that the remaining 5% of requests have multiple requested operating days during a week. For example, 3% of requests are for daily flights in Airport 3. Note that flights operating on multiple days in a week are scheduled at the same time on different days. Since airport 3 has a particularly higher proportion of daily flights, the associations between days will make it more difficult to meet the schedule regularity requirement when allocating slots. This motivates us to consider a flexible slot allocation to allocate slots on different days of the week individually. This idea will be further investigated in 6.1.



(a) Original (b) Zoom in

Figure 3.2: Histogram of the number of operating days for flight operations for the three airports

ii. 'Nweeks': Figure 3.3 shows the distribution of the number of operating weeks of operations across the season. We observe that Airport 3 has a larger percentage of requests that span a large number of weeks compared to Airport 1 and 2, with more than 10% of flights being weekly operations throughout the season. In contrast, over 20% of flights at Airports 1 and 2 only operate in one week of the season. The descriptive statistics of the number of the operating week are shown in Table 3.4. We performed a Mann-Whitney U test (at the 95% confidence interval) for the average value of 'Nweeks'. The results indicate that the average number of requested operating weeks at Airport 3 is statistically greater than the other two airports.

Table 3.4: Descriptive statistics of operating weeks

Nweeks	Airport 1	Airport 2	Airport 3
count	449.00	932.00	1087.00
mean	12.69	13.28	16.36
std	10.47	10.37	10.36
min	1.00	1.00	1.00
25%	1.00	3.00	6.00
50%	9.00	10.00	18.00
75%	23.00	24.00	26.00
max	30.00	30.00	30.00

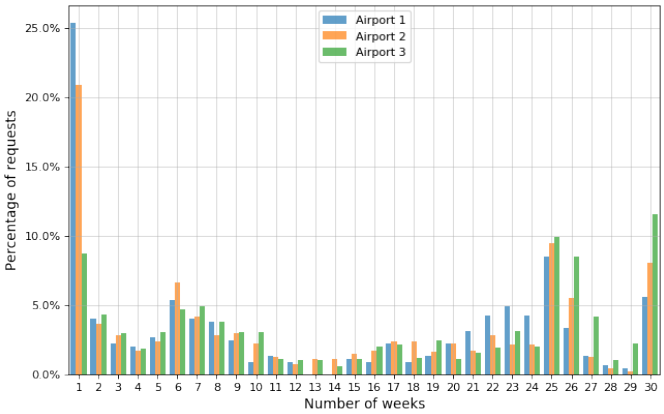


Figure 3.3: Histogram of the number of operating weeks in the season for flight operations for the three airports

Lastly, we look at slots requested for ad hoc slot and series flight operations. Recall the IATA threshold for series services is 5 weeks. Any requests for less than 5 weeks will be treated as ad hoc slot requests and thus scheduled individually after the initial slot allocation. Figure 3.4 shows a bar chart (left) that compares the number of ad hoc slot requests (operating less than 5 weeks) and series of slots requests (at least 5 weeks). As demonstrated in Figure 3.4 (right), the number of individual slots belonging to ad hoc slot requests is insignificant compared to slots in series at all three airports.

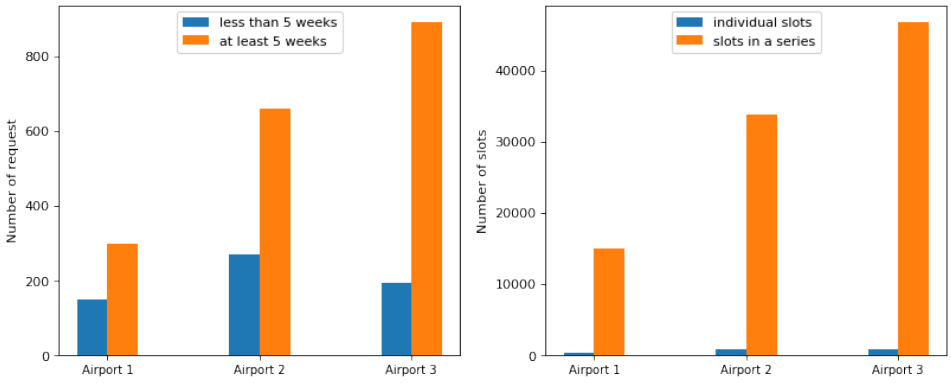


Figure 3.4: ad hoc slot requests vs. series of slots

3.2.3 Slot priority characteristics

Figure 3.5 presents the proportional distribution of slot priority groups for the flight operations at the three airports. Recall that slots are categorised into four priority classes. We observe that the proportion of all historical slots (*Historic* and *Change-to-historic* slots) for the three airports is similar, ranging between 41.4% to 45.7%. *New entrants* account for less than 10% of requested slots at all three airports. The *Others* group is the largest group, containing around 50% of the total number of requested slots at each airport. When considering a slot allocation problem, the priority order is typically managed by solving the model in a sequential manner, with the model solved for several sub-problems of each of the priority classes in order of priority. As a result, the number of slots requested by the higher priority groups will constrain the slots available for the lower priority groups later on. For example, in an airport with a large proportion of historical operations, competition for new entrants and others will be intense, because there is limited capacity remaining after all of the historical slots are allocated.

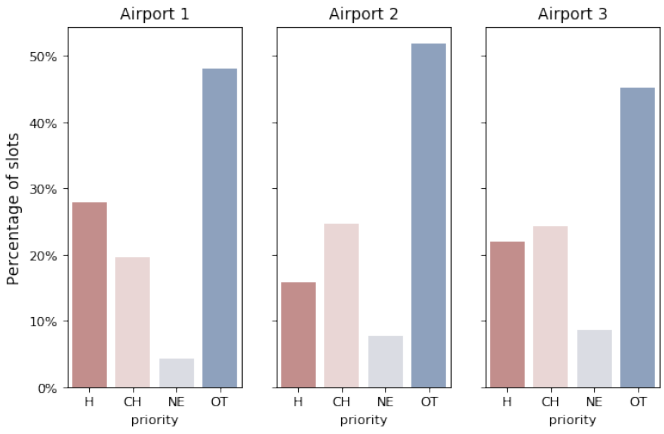


Figure 3.5: Distribution of slot priorities across four categories for the three airports

3.2.4 Time distribution of operations and turnaround time

Figure 3.6 plots the distribution of the aggregated number of requested slots during the day. This shows that flights are unevenly distributed throughout the day, as we can observe several arrival or departure peaks at different times for different airports. We also observe a peak of departures at 4 am at Airport 3. This can be explained by the fact that some flights arrive late at night and have to wait until the next morning to depart. Understanding the time distribution of slots will help us to develop effective methods to decompose the overall problem into smaller sub-problems based on particular time periods.

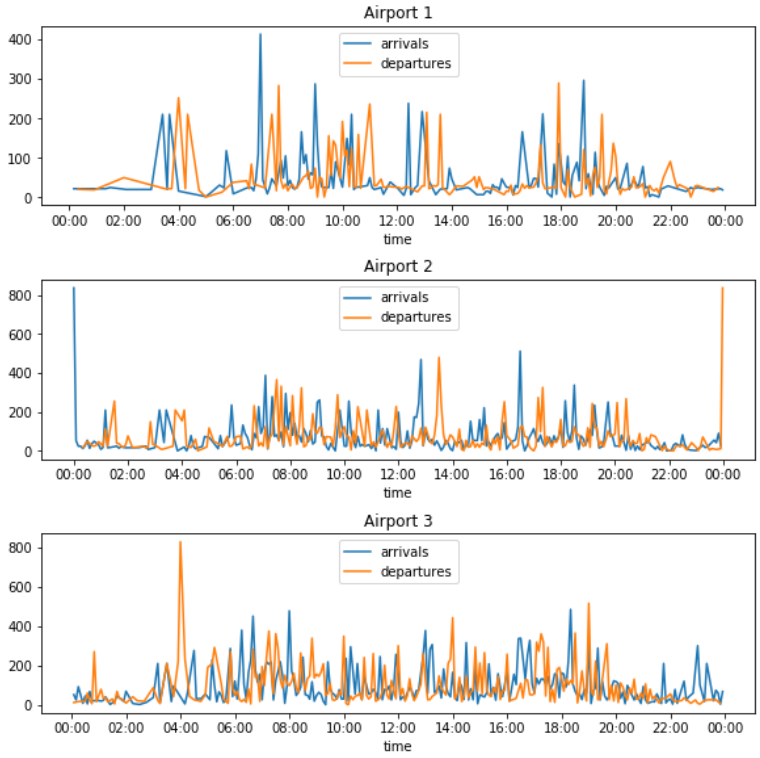


Figure 3.6: Time distribution of the aggregated number of requested slots for the three airports

Closer inspection of Fig. 3.6 shows that departures tend to lag approxi-

mately one hour behind arrivals in all three airports. The time difference between a pair of arrival and departure flights is referred to as the turnaround time, which is usually about 30 minutes for low-cost flights and an hour and a half for large aircraft and premium airlines, as can be seen in Figure 3.7 (upper). However, some flights require several hours of ground time before their departure the next day, as shown in Figure 3.7 (bottom). Note that slot coordinators should respect the minimum turnaround time requirements and avoid any increase in turnaround time where possible.

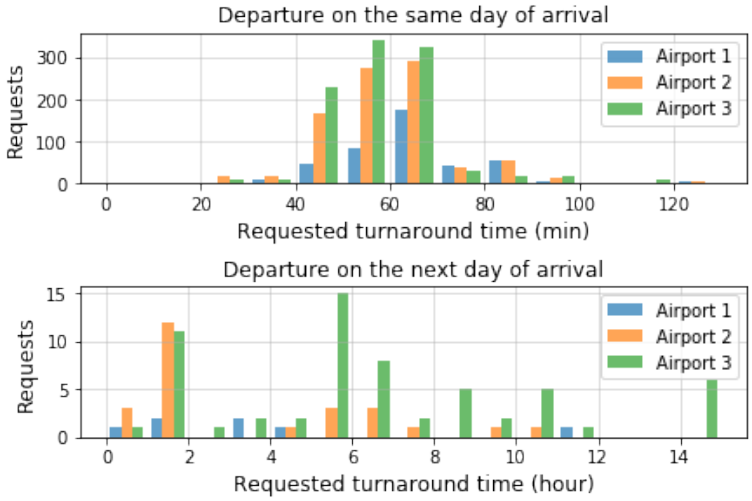


Figure 3.7: Requested turnaround time

3.2.5 Associations between key attributes of slot requests

In this section, we provide more analysis of request characteristics. Table 3.5 provides a detailed breakdown of the number of operation days (per week) requested based on a given frequency of operation. In Airport 1, all requests that operate for more than one day are operated weekly (i.e. 'ReqFreq = 1). In other words, if a request operates on multiple operation days (e.g. Mon, Tues and Fri) it will use these days every week across the dates requested.

Next, we look at the association between the operation days and op-

Table 3.5: Number of requests operating with specific weekly frequencies and operating days

	Airport 1			**Airport 2**			**Airport 3**		
	Frequency								
Operating days	1	2	3	1	2	3	1	2	3
1	393	14	23	808	1	66	956	7	60
2	4	0	0	17	0	0	21	0	0
3	3	0	0	7	0	0	4	0	0
4	2	0	0	1	0	0	4	0	0
5	1	0	0	2	0	0	1	0	0
6	2	0	0	3	0	0	0	0	0
7	7	0	0	18	0	0	34	0	0

eration weeks. As demonstrated in Figure 3.8, we observe that flights operating on multiple days in a week tend to operate for long periods with more frequent operations (every week). Requests that operate on a single day per week have a large range of request lengths. However, when requests have multiple operation days, a smaller range (with longer request weeks) is present. Specifically, the distributions of operation weeks when an operation has only one operation day will likely differ from the distribution when an operation has seven operating days. We calculate the Spearman correlation coefficients between these two variables for each airport. The result supports that the operation length and requested operation days are positively correlated.

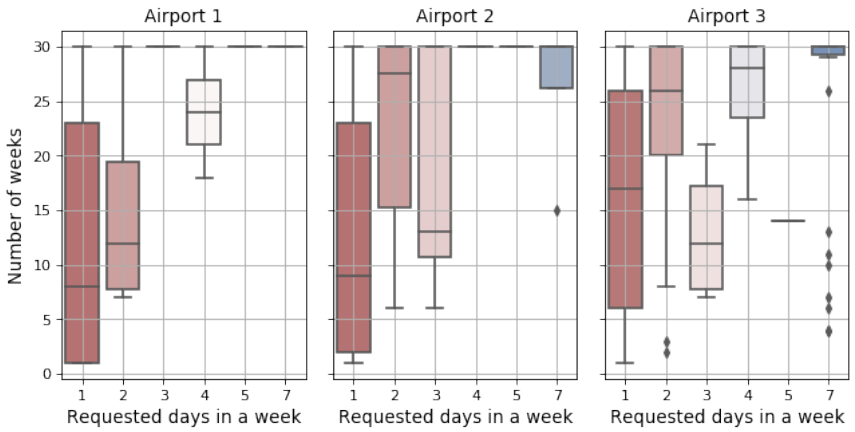


Figure 3.8: Requested operation weeks given operating days

Lastly, we look at the request characteristics of different priorities. Figure 3.9 shows the distribution of the operation weeks by priorities. Since the operation weeks do not follow the normal distribution, we performed Levene's test to test the equality of variances. If the assumption of equal variances is violated, we then perform Welch's ANOVA to test if the mean values for each priority group are significantly different. Otherwise, the One-way ANOVA test is performed. Lastly, we perform the Games-Howell test after Welch's ANOVA or Tukey's posthoc test after the One-way ANOVA to tell exactly which group means are different. Table 3.6 presents the statistical significance test results.

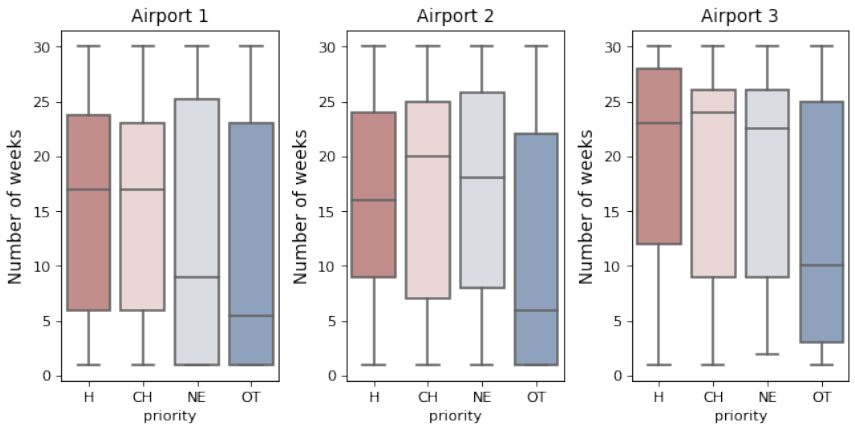


Figure 3.9: Operation weeks by request priorities

Table 3.6: ANOVA test significant p-values at 0.05

	Levene's test	Welch's ANOVA	One-way ANOVA	Games-Howell	Tukey 's
Airport 1	0.4870	-	0.0030		H-OT: 0.0061
					CH-OT: 0.0396
Airport 2	0.0110	0.0000	-	H-OT: 0.0010	
				CH-OT: 0.0010	
				NE-OT: 0.0283	
Airport 3	0.0000	0.0000	-	H-OT: 0.0010	
				CH-OT: 0.0010	
				NE-OT: 0.0010	

From the above table, we observe that the p-values from the ANOVA test are less than 0.05 for all three airports, which means that the requested operation weeks are significant differences between the four priority groups. For Airport 1, the average operation length of *Historical* and *Change-to-historic* are significantly larger than *Others*. For Airports 2 and 3, the average operation length of *Others* is significantly lower than the other three priority groups. Similar analyses have been conducted to analyse the slot time distribution of different priority groups, as presented in Fig. 3.10. The results showed that there is no significant difference in the requested time of different priority groups.

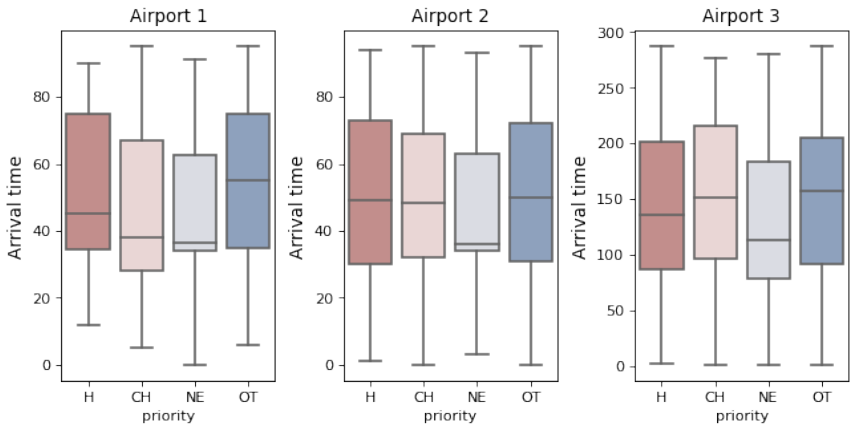


Figure 3.10: Request time of each priority class

3.3 Demand-Capacity Imbalance indicators

The imbalance between demand and capacity is considered an important factor determining how difficult it is to find a feasible solution for a given slot allocation problem instance. One way of measuring the demand-capacity imbalance is to look at the violated capacity constraints when all flight operations are allocated to their preferred slots. In this case, the resulting solution (referred to as the *default schedule*) is expected to be infeasible. However, it can provide some insight into where the main bottlenecks of the problem instance lie. Table 3.7 shows the percentage of violated constraints of the default schedule for the three airports. The higher the percentage, the further an initial infeasible solution is from being feasible. We can see that the violated constraints for arrivals in a rolling 60-minute period (column 'Arr./60min') are much higher than that of departures at all three airports. This is because the capacity limits for arrivals are much tighter than those on departures, as can be seen in Table 3.8. It is evident from the default schedule that capacity constraints are redundant during non-peak periods since only a small percentage of them are violated.

Table 3.7: Percentage of violated capacity constraints of the default schedule

Airport	Total/15min	Arr./60min	Dep./60min	Arr./10min	Dep./10min
1	1.48	5.17	2.29	-	-
2	3.99	7.58	0.94	-	-
3	8.97	7.84	4.09	0.23	0.43

Table 3.8: Declared capacity constraints at the three airports

Airport	Total/15min	Arr./60min	Dep./60min	Arr./10min	Dep./10min
1	3	4	6	-	-
2	5	8	12	-	-
3	5	10	12	5	5

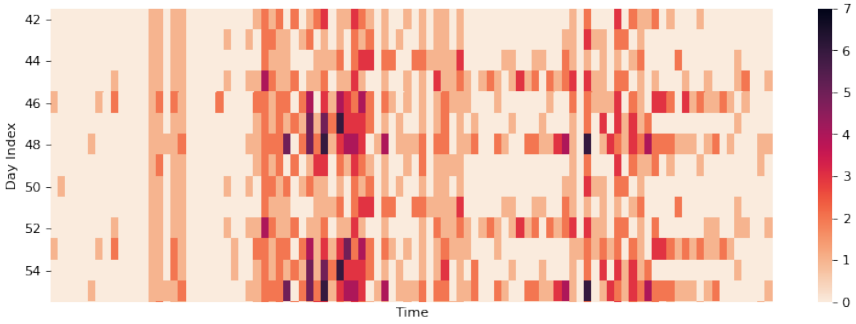


Figure 3.11: Heatmap of requested slots at Airport 1 during the busiest two weeks of operation. The colour scales indicate the number of slots requested for each time period.

Another way of measuring the demand-capacity imbalance is to look at congested time periods in which the slot demand exceeds the capacity. Slot requests made for congested periods are expected to be rescheduled or rejected if there are no available slots close enough to the requested slots. In Fig. 3.11, we plot a heat map of slot demand for the busiest 2 weeks. Each square represents a time period defined by the time interval (x-axis) and the day of the season (y-axis). The colour of the squares shows the number of slots in that time period. In this example, congested time periods are those with values greater than 3 (declared capacity for each time period). Table 3.9 compares the percentage of congested time periods of the entire season for the three airports. Note that we calculate the congested periods for arrivals and departures separately since they have individual declared capacities. We observe that the numbers are much higher in Airport 3 than in the other two airports. Most of the time periods are unsaturated, indicating that the capacity constraints are redundant during these periods. The result motivated us to add capacity constraints as lazy constraints through a callback of the exact solver.

Table 3.9: Proportion of congested time periods of the default schedule

Airport	% of Arrival congested	% of Departure congested
1	10.80	5.17
2	14.36	4.98
3	22.15	18.03

The difficulty of solving the slot allocation problem for a single airport is determined by a number of factors. Obviously, the number of slot requests is one of the factors of problem complexity. However, in cases where the number of requests is equal, insufficient airport declared capacity and severe imbalance between slot demand and airport capacities are also thought to significantly increase the difficulty of solving the problem.

To conclude, in this chapter we have explored slot allocation data for three real-world airports, identifying underlying structural properties that have not previously been discussed in the literature. We have examined the distribution of flights in a scheduling season, on a weekly and daily basis. We observed that the three airports have different flight operation features, different peak hours and congestion levels. Through the analysis conducted on schedule-coordinated airports, we identify three relationships that capture detailed slot demand patterns. The relationships that have been identified are operation length and operation frequency given the number of operation days, and operation week based on operation length. The data analysis results demonstrate the difference in slot demand patterns at different airports, which should be taken into account when developing mathematical models and solution algorithms for the slot allocation problem. Due to the lack of real-world request data, this analysis also provides a basis for algorithm development and the generation of artificial data sets.

Chapter 4

Optimising Slot Allocation Considering Slot Rejection and Schedule Efficiency

We propose a new model for the single airport slot allocation problem in this chapter. This model allows us to investigate slot rejections under different maximum allowable displacement thresholds. Meanwhile, the proposed model can be used to analyse possible changes to the current slot allocation rules to provide insights into how these rules affect slot allocation and can be modified in the future.

The remainder of this chapter is structured as follows: Section 4.1 introduces the model formulations. Next, Section 4.2 presents the experimental results of solving the model for one of the coordinated airports and discusses the experiment results. Next, we perform sensitivity analysis in Section 4.3 to examine the sensitivity of the optimal solutions to the changes in requested turnaround times, alternative displacement cost functions, and changes in the slot allocation priority rules. Finally, summaries and conclu-

sions of this study are presented in Section 4.5.

4.1 A slot allocation model considering slot rejection

Slot rejections have been considered in the literature (Ribeiro et al., 2018; Jorge et al., 2021), and the number of rejected requests or slots has been modelled as an objective to be firstly minimised in corresponding models. However, no slot rejections were reported in previous experiments. The reason may be that the capacity of the airport under study is sufficient to meet the demand for slots, or available slots can always be found regardless of slot displacement constraints. This motivates us to further investigate slot rejections under different maximum allowable displacement thresholds. Specifically, the motivation is twofold. First, even at airports where the capacity is sufficient to meet the demand for slots, airlines may receive slots distant in time from their desired slots and, therefore, unlikely to accept or utilise those allocated slots. This will have a significant effect on slot allocation at extremely congested airports. For example, in Amsterdam's Schiphol airport, the number of wait-listed slots (i.e., rejected in the initial slot allocation stage and may be reconsidered later) represented approximately 16% of the initially requested slots in the summer season of 2017 (Odoni, 2021). Therefore, we investigate slot rejections affected by different maximum acceptable displacement thresholds. Second, the *50/50* priority rules may require the consideration of slot rejections. For example, when *new entrants* or *Others* request more than 50% of the slots in the slot pool, some requests have to be rejected.

Therefore, we propose a new model for the single airport slot allocation

problems, called SASA-R, which considers slot rejections under different maximum allowable displacement thresholds. The model proposed in this chapter is built upon the model proposed by Zografos et al. (2012); Ribeiro et al. (2018). It takes as input the airport declared capacities (see Table 3.8) and slot requests from airlines (see detailed description in Table 3.1). It then produces a schedule that first minimises the total number of rejected slots and second the schedule displacement of all slots. The model formulations are presented as follows:

Notations used in the model

$\mathcal{T} = \{0, ..., T-1\}$: set of time intervals equal in length, indexed by t. T is the total number of time intervals of a day

$\mathcal{D} = \{0, ..., D-1\}$: set of days, indexed by d. D is the total number of days of the summer or winter scheduling season

$\mathcal{M}_{arr} \subset \mathcal{M}$: set of arrival requests, indexed by i

$\mathcal{M}_{dep} \subset \mathcal{M}$: set of departure requests, indexed by j

$\mathcal{P} \subset \mathcal{M}_{arr} \times \mathcal{M}_{dep}$: set of paired requests, indexed by (i, j) such that j is the departure that follows the arrival i operated by the same aircraft

$\mathcal{M} = \mathcal{M}_{arr} \cup \mathcal{M}_{dep}$: set of all requests, indexed by m

$\mathcal{C}$ ={Arr., Dep., Total}: set of flight movement types, indexed by c

$\mathcal{L}$: set of time lengths associated with the declared capacity, indexed by l (e.g., 15 minutes, 1 hour)

The set $\mathcal{T}$ consists of indexes of time intervals of a day. The length of each time interval equals the airport coordination time interval, typically 5, 10 or 15 minutes and the corresponding number of time intervals on each day is 288, 144, or 96 respectively. Note that slots allocated to airlines are

represented by time interval indexes. The set $\mathcal{D}$ consists of indexes of all days of the scheduling season. We now clarify the definition of a **request** used in this chapter. In general, a slot request or request refers to the request initially submitted by airlines, which usually includes two groups of slots for paired arrival and departure flights. These two groups are defined as two separate requests and the initial submitted request is referred to as **a request pair** (a pair of requests, or paired requests). Note that the request pair $(i, j) \in \mathcal{P}$, are included as two separate requests in the set $\mathcal{M}$, such that each $m \in \mathcal{M}$ represents an arrival or departure request. Within each request, slots requested for the same time on the same day of the week for at least 5 weeks form a **series of slots**. If the number of weeks is less than 5, the request is treated as **ad hoc slot** request. Recall that WASG requires that all series of slots must be processed together. Moreover, a request may include multiple series of slots for the same time on different days of the week, they must also be processed at the same time. Lastly, the types of flight movements and the time duration over which the capacity constraints apply are represented in sets $\mathcal{C}$ and $\mathcal{L}$.

Parameters used in the model

B_m^d: equals to 1 if request m has requests on day d, and 0 otherwise

$T_{i,j}^{min}$: minimum turnaround time for the request pair $(i, j) \in \mathcal{P}$

$T_{i,j}^{max}$: maximum turnaround time for the request pair $(i, j) \in \mathcal{P}$

C_{dtl}^{arr}: arrival capacity for time interval t, day d and time duration l

C_{dtl}^{dep}: departure capacity for time interval t, day d and time duration l

C_{dtl}^{total}: total capacity (arrival and departure) for time interval t, day d and time duration l

ϕ_m: maximum allowable slot displacement for request m, $0 < \phi_m < T$

τ_m: requested slot of request m, $\tau_m \in \mathcal{T}$

Note that the turnaround time parameters: $T_{i,j}^{min}$, $T_{i,j}^{max}$ are not explicitly provided by airlines but are considered in this model to ensure a reliable turnaround time for each pair of flight movements. The impact of changing the turnaround times parameters will be studied in Section 4.3.1.

Decision variables

$$x_m^t = \begin{cases} 1, \text{if request } m \text{ is allocated with slot(s) at } t \\ 0, \text{otherwise} \end{cases}$$

$$z_m = \begin{cases} 1, \text{if request } m \text{ is rejected} \\ 0, \text{otherwise} \end{cases}$$

First, the decision variables x_m^t are binary variables that capture whether a request m is allocated with an individual slot or a series of slots at t or not. Note that t is a time interval index, corresponding to a 15 minutes time interval. The variables x_m^t ensure that all slots requested in m are handled together. The second binary decision variable z_m indicates whether a request m is rejected or not. If m is rejected, it means no slots are assigned to this request. The paired arrival and departure request $(i, j) \in \mathcal{P}$ should always be rejected or scheduled at the same time. The logical relationship between these decision variables will be defined in the constraints section.

Objective function

$$\min w_1 \sum_{m \in M} \sum_{d \in D} B_m^d z_m + w_2 \sum_{m \in M} \sum_{d \in D} \sum_{t \in T_m} B_m^d f_m^t x_m^t \tag{4.1}$$

The objective function 4.1 of the model consists of two objectives. The

first objective is the total number of rejected slots. We use the number of rejected slots instead of rejected requests because slots requested in the same request are either all allocated or none are, so rejecting requests with fewer slots is preferable to rejecting requests with more slots. The second objective is the schedule displacement of all accommodated requests, excluding rejected requests. Specifically, the displacement of a slot, or slot displacement, is the absolute difference between the requested and allocated slot, denoted by Eq. (4.6). The schedule displacement of a request is its slot displacement multiplied by its total number of operating days of the season. Note that every slot in the same request will have the same slot displacement. For example, suppose a request for a series of slots at 11 am on ten different days is allocated with a series of slots at 9 am, the slot displacement would be 2 hours, and the displacement of this request would be 20 hours. Note that displacement can also be presented in time intervals.

The two objectives are scaled into a single objective function 4.1, using the weighted sum method (Marler and Arora, 2010). The number of rejected slots and schedule displacement is multiplied by two user-supplied weights w_1 and w_2. Based on existing literature (Ribeiro et al., 2018; Jorge et al., 2021) and our communication with airlines, we consider that minimising the number of rejected slots is more important than minimising the schedule displacement. Therefore, we set the weights w_1 to 10^8 and w_2 to 1 to ensure that the number of rejected slots is minimised first. The value of weights is determined by some preliminary experiments to see the magnitude of the value of the two objective functions.

Constraints

$$T_m = \{t \in T | \tau_m - \phi_m \leq t \leq \tau_m + \phi_m\} \tag{4.2}$$

$$\sum_{t \in T_m} x_m^t \leq 1, \forall m \in M \tag{4.3}$$

$$1 - \sum_{t \in T_m} x_m^t = z_m, \forall m \in M \tag{4.4}$$

$$z_i = z_j, \forall (i,j) \in P \tag{4.5}$$

$$f_m^t = |t - \tau_m| \tag{4.6}$$

$$(1 - z_m) T_{i,j}^{min} \leq \sum_{t \in T_j} t x_j^t - \sum_{t \in T_i} t x_i^t \leq T_{i,j}^{max}, \forall (i,j) \in P \tag{4.7a}$$

$$(1 - z_m) T_{i,j}^{min} \leq \sum_{t \in T_j} (T + t) x_j^t - \sum_{t \in T_i} t x_i^t \leq T_{i,j}^{max}, \forall (i,j) \in P \tag{4.7b}$$

$$\sum_{i \in M_{arr}} \sum_{t}^{t+L_c-1} x_i^t \leq C_{dt}^{arr}, \forall d \in D, t \in T | t < T - L_c + 1 \tag{4.8}$$

$$\sum_{j \in M_{dep}} \sum_{t}^{t+L_c-1} x_j^t \leq C_{dt}^{dep}, \forall d \in D, t \in T | t < T - L_c + 1 \tag{4.9}$$

$$\sum_{m \in M} \sum_{t}^{t+L_c-1} x_m^t \leq C_{dt}^{total}, \forall d \in D, t \in T | t < T - L_c + 1 \tag{4.10}$$

Equation 4.2 defines the range of time intervals that is available to request m, denoted by T_m. This ensures that the maximum slot displacement of m is no greater than ϕ_m, as m can only be scheduled no earlier than $\tau_m - \phi_m$ and no later than $\tau_m + \phi_m$, where τ_m is the requested slot of m,

$\tau_m \in \mathcal{T}$. ϕ_m is a non-negative parameter that can take different values for different requests, or take the same value for all requests as the schedule-wide maximum allowable displacement threshold. Constraints 4.3 ensure that for each request, at most one time interval within the time window T_m can be allocated to this request. Therefore, requests are allowed to be rejected when $\sum_{t \in T_m} x_m^t = 0$. If m is rejected, z_m equals 1, therefore constraints 4.4 are satisfied, and vice versa. Constraints 4.5 ensure that a pair of arrival and departure requests are accommodated or rejected at the same time. Eq. (4.6) defines the slot displacement of m, which is the absolute difference between the allocated slot t and the requested slot τ_m. Constraints 4.7a ensure that the slot allocated to a departure request j must be at least $T_{i,j}^{min}$ time intervals later than the slot allocated to its paired arrival request i, and no more than $T_{i,j}^{max}$ intervals. If the departure flight is on the next day of its arrival, constraints 4.7a can be replaced by 4.7b. Note that the term $1 - z_m$ ensures if paired requests are rejected, turnaround time constraints are still satisfied. Constraints 4.8 to 4.10 are rolling capacity constraints for arrival, departure and both types of flight movements respectively. The formulation of these capacity constraints is similar to the ones presented in Ribeiro et al. (2018). Figure 4.1 illustrates how the rolling capacity constraints are checked. We plot two arbitrary days of the season. The x-axis denotes the set of time intervals on a day. The y-axis represents the number of arrival movements. A bar's height indicates the arrival movements that have been scheduled into the corresponding time interval. Suppose the capacity for arrival movements in any 4 consecutive time intervals is 3, that is for any following time period: 0 to 3, 1 to 4, ..., 4 to 7, a maximum of 3 arrival movements can be scheduled. All constraints are satisfied on the first day. For the other day, the capacity constraints are violated by 2 in

time periods 1 to 4 and 2 to 5.

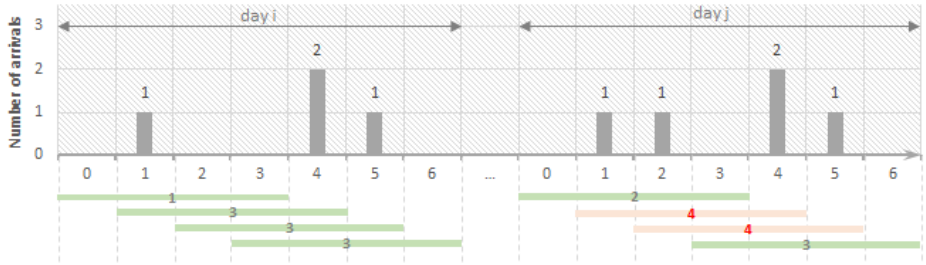


Figure 4.1: Example of rolling capacity constraints

4.2 Experiments

4.2.1 Data and set-up

The proposed model has been applied to all three airports. We only provide the results for Airport 2 in detail in this chapter. Results for the other two airports are presented in the Appendix A.1 and discussed in Section 4.4. We first solve the SASA-R model without consideration of slot priorities, and the resulting solution is referred to as the *non-hierarchical* solution. We will incorporate slot priorities by solving the model hierarchically and holistically in Section 4.3.2.

Regarding the model parameters, the declared capacity for Airport 2 is presented in Table 3.8. The minimum turnaround time parameter is set to be the minimum value of the requested turnaround time and 1 hour. This allows for some flexibility in shortening the unnecessary turnaround time. The maximum turnaround time parameter is set to the requested turnaround time, which means that the assigned turnaround time must be no longer than the requested turnaround time. Sensitivity analysis of changes in requested turnaround times will be discussed later in Section 4.3.1.

All integer linear programs in this chapter are solved using Gurobi 9.0.2 on a computer with an Intel Core i5-8365U processor. Models are implemented using Python and Gurobi's Python API (gurobipy). A branch-and-cut method proposed by Fairbrother et al. (2020) was used to speed up the solution time of the model. Since the capacity constraints are only active during specific congested time periods of the season, they are added as lazy constraints through a callback function of Gurobi (the Gurobi LazyConstraints parameter is set to value 1). The rest of the Gurobi solver parameters have been left at their default settings, such as the TimeLimit parameter is Infinity and the Relative MIP optimality gap is 0.0001.

4.2.2 Solution metrics

In addition to the optimisation objectives: the number of rejected slots and schedule displacement, we propose more schedule efficiency metrics to compare different solutions:

(1) The number of rejected requests is denoted by 'RejReqs' and calculated by 4.11

$$\sum_{m \in M} z_m \tag{4.11}$$

(2) The maximum slot displacement is denoted by 'MaxDisp' and calculated by 4.12. Note that m may be rejected or allocated with its requested slot. In both cases, the displacement of m will be zero

$$\max_{m \in M} \sum_{t \in T_m} f_m^t x_m^t \tag{4.12}$$

(3) The number of displaced requests is denoted by 'DispReqs' and calculated by 4.13. y_m indicates whether request m is displaced or not

$$y_m = \begin{cases} 1, \text{if } \sum_{t \in T_m} f_m^t x_m^t > 0 \text{ allocated slot is earlier or later than } \tau_m \\ 0, \text{if } \sum_{t \in T_m} f_m^t x_m^t = 0 \ m \text{ is rejected or allocated with requested slots at } \tau_m \end{cases}$$

$$\sum_{m \in M} y_m \tag{4.13}$$

(4) The number of displaced slots is denoted by 'DispSlots' and calculated by 4.14

$$\sum_{m \in M} \sum_{d \in D} B_m^d y_m \tag{4.14}$$

(5) The displacement per displaced slot is denoted by 'Disp/Slot' and calculated by 4.15

$$\sum_{m \in M} \sum_{d \in D} \sum_{t \in T_m} B_m^d f_m^t x_m^t / \sum_{m \in M} \sum_{d \in D} B_m^d y_m \tag{4.15}$$

4.2.3 Experiment results

In this section, we discuss the *non-hierarchical* results of solving the SASA-R model for Airport 2. We first do not limit the maximum slot displacement by setting $\phi_m = None, \forall m \in \mathcal{M}$. Next, we solve the model with various thresholds of maximum slot displacement.

Table 4.1: Non-hierarchical results for Airport 2. Values in parentheses are the relative change rate to the solution when slot displacement is not constrained

Threshold	RejSlots	RejReqs	SchedDisp	MaxDisp	DispReqs	DispSlots	Disp/Slot (min)	Time (sec)
None	0	0	12,930	4h30m	510	4,998	38.8	1081
3h	0	0	13,000 (0.5%)	3h	518	4,987	39.1 (0.7%)	555
2h30m	0	0	13,072 (1.0%)	2h30m	512	5,043	38.9 (0.1%)	393
2h	0	0	13,099 (1.2%)	2h	504	4,991	39.4 (1.4%)	351
1h30m	0	0	13,597 (5.1%)	1h30m	553	5,381	37.9 (-2.4%)	161
1h	18	2	17,842 (37.9%)	1h	558	7,451	35.9 (-7.5%)	87
30m	456	52	14,066 (8.7%)	30m	612	9,199	22.9 (-40.9%)	13
0	3,318	342	0	0	0	0	-	1

The first column in Table 4.1 shows the pre-defined thresholds to the maximum slot displacement. Column 'MaxDisp' shows the maximum slot displacement in the optimal solutions. The number of rejected slots and schedule displacement is presented in columns 'RejSlots' and 'SchedDisp', respectively. Slot rejections occur when the maximum slot displacement is limited to one hour and 18 slots (of a pair of requests) are not assigned. When the threshold is reduced to 30 minutes, 456 slots (of 52 requests) are rejected. As expected, schedule displacement increases as the threshold decreases, but only significantly when the threshold is sufficiently low (1 hour 30 minutes). A trade-off exists between the schedule displacement and the maximum slot displacement threshold. For example, the schedule displacement is 37.9% larger with a maximum 1-hour slot displacement than without it. With regards to the displacement per slot, it increased slightly before decreasing significantly as the threshold decreased. The reason for the increase is that allowing a small number of slots to be largely displaced can benefit the minimisation of the schedule displacement. However, when the threshold is too low, there are fewer available slots for each request. Thus, more slots have to be rejected. The displacement per slot is therefore reduced because it only takes into account accommodated requests. Furthermore, we noticed that the displacement per slot was at its lowest level at 38 minutes without rejecting any slots and dropped significantly due to slot rejections. Therefore, tightening the maximum displacement threshold will not reduce the average slot displacement, only if some slots are rejected. For example, when the threshold is 30 minutes, the average slot displacement dropped to 23 minutes due to 456 slots (1.3% of total requested slots) being rejected. Lastly, the experiment demonstrated that lower maximum slot displacement thresholds lead to less computation time as the search space is reduced by

restricting the range of slots for each request.

4.3 Sensitivity analysis

In this section, we perform sensitivity analysis to analyse how changes in turnaround times, displacement cost functions and slot allocation priority rules affect schedule efficiency.

4.3.1 Sensitivity to the changes in requested turnaround times

We now investigate the sensitivity of the optimal solutions to the changes in the requested turnaround times (the time difference between the pair of requested slots). Six scenarios are tested. Scenario A requires that the requested turnaround time of each pair of requests must be strictly respected. This is a scenario that resembles the WASG rules, which state that an increase in the requested turnaround time should be avoided as much as possible due to operating costs and environmental impact. Scenario B is the default assumption in the model and experiments in this chapter. The minimum turnaround time parameter is set to the minimum value of the requested turnaround time and 1 hour to allow flexibility in reducing the long requested turnaround time. Meanwhile, an increase to the requested turnaround time is not permitted. Scenario C allows a 15 minutes increase of the requested turnaround time. Scenario D and E consider a flexible minimum turnaround time, similar to scenario B, and permit a 15-minute and 30-minute expansion of the requested turnaround times, respectively. Lastly, scenario F only considers the minimum turnaround time constraints with flexibility.

Results are shown in Table 4.2. The second column indicates the allowable reduction and increases in the requested turnaround times. The

Table 4.2: Sensitivity of non-hierarchical results to changes in the requested turnaround times

Scenarios	Changes	SchedDisp	DispReqs	DispSlots	Disp/Slot (min)	Time (sec)
MaxDisp threshold = 2h, RejSlots=0						
A	0,0	13,788	540	5,200	39.8	250
B	flexible, 0	13,099 (-5%)	504	4,991	39.4 (-1%)	230
C	0, +15 min	11,928 (-13%)	538	5,230	34.2 (-14%)	172
D	flexible, +15 min	11,447 (-17%)	510	5,121	33.5 (-16%)	173
E	flexible, +30min	10,631 (-23%)	447	4,450	35.8 (-10%)	110
F	flexible, none	9,539 (-31%)	402	3,795	37.7 (-5%)	45
MaxDisp threshold = 1h, RejSlots= 18						
A	0,0	18,726	594	7,924	35.4	90
B	flexible, 0	17,842 (-5%)	558	7,451	35.9 (1%)	83
C	0, +15 min	16,166 (-14%)	560	7,868	30.8 (-13%)	27
D	flexible, +15 min	15,980 (-15%)	542	7,926	30.2 (-15%)	20
E	flexible, +30min	14,125 (-25%)	482	6,910	30.7 (-14%)	14
F	flexible, none	12,750 (-30%)	413	5,815	32.9 (-7%)	11

maximum slot displacement threshold is set to 1 and 2 hours. Results show that the number of rejected slots did not change as the turnaround time parameters changed. However, the results demonstrate a trade-off between changes in the requested turnaround times and schedule displacement. We observe that schedule displacement can be reduced by 5% to 31% with small to large changes in the requested turnaround time. Note that even slight flexibility (e.g., scenario C allows a 15-minute increase in turnaround time) can lead to a 13% improvement in the schedule displacement and a 14% reduction in the average slot displacement. Moreover, allowing changes to the requested turnaround time tends to shorten the computation time significantly, as indicated in the last column. Results suggest that forcing the allocated turnaround time to be exactly the same as the requested might be unnecessary. A small degree of turnaround time flexibility can improve schedule efficiency and reduce the computation time of finding the optimal solutions.

Finally, we analyse the allocated turnaround times when the maximum allowable turnaround time is not limited (scenario F), as done in several papers (Zografos et al., 2012; Fairbrother et al., 2020). Table 4.3 compares the results under scenario F and the coordinated schedule created by airport coordinators. In the coordinated schedule, 97% of request pairs received pairs of slots with the same turnaround time as they requested. However, one request pair was given 11 hours less time for a turnaround than requested by the coordinator. Another pair of requests received 45 minutes more time for turnarounds. When the maximum turnaround time is not limited, 17% fewer request pairs received slots with the same turnaround time as they requested. Meanwhile, the increase in requested turnaround times is more significant as the maximum displacement threshold increases (up to 2 hours and 45 minutes). In addition, the range of changes in turnaround times is smaller than in the coordinated schedule. The reason for this is, on the one hand, restricted by the maximum slot displacement thresholds and, on the other hand, driven by the objective of minimising the schedule displacement.

In summary, it is necessary to limit the maximum allowable turnaround times. Otherwise, the increase in the requested turnaround time will be significant. However, restricting the allocated turnaround time to be the same as the requested one may be unnecessary.

Table 4.3: Allocated vs. requested turnaround times

	% of requests with no changes in requested turnaround times	**max. reducing**	**max. increase**
coordinated	97%	11 hours 15 minutes	45 minutes
scenario F (maxDisp=2 hour)	81%	1 hour	2 hours 45 minutes
scenario F (maxDisp=1 hour)	80%	1 hour	1 hour 45 minutes

4.3.2 Sensitivity to the priority rules

We now solve the SASA-R model according to the previous WASG priority rules. The result is referred to as the *hierarchical* results. In this manner, slots are allocated sequentially from the highest to the lowest priorities in the order of *Historics*, *Change-to-historics*, *New-entrants* and *Others*. Specifically, the model is firstly solved for *Historics* requests. Once the optimal solution for the higher priority class is found, the values of decision variables are fixed, the capacities are updated, and we continue to solve the model for the next highest priority class until all priority classes are processed. We do not limit the maximum slot displacement for any priority group in the *hierarchical* allocation. The results are shown in Table 4.4. No slots were rejected in this case. The maximum slot displacement comes from the *Others* at 4 hours and 15 minutes. These results will be compared with the *holistic* results introduced in the following.

Table 4.4: Hierarchical results for Airport 2, without maximum slot displacement thresholds

	Threshold	SchedDisp	MaxDisp	DispReqs	DispSlots	Disp./slot (min)	Time (sec)
H	–	0	0	0	0	0	
CH	–	548	45m	42	357	23	
NE	–	492	15m	8	492	15	
OT	–	17,681	4h15m	388	6,107	43	
All		**18,721**	**4h15m**	**438**	**6,956**	**40**	**100**

We now perform the *holistic* allocation as suggested in the latest WASG regulations. After allocating all the *Histotic* slots, the remaining slots are allocated holistically. That is, the *Change-to-historic*, *New entrants* and *Others* are processed simultaneously as a single batch. In order to compare the results of the *holistic* and *hierarchical* allocation, we set the maximum displacement thresholds for each priority class in the *holistic* allocation model to the maximum slot displacement in the *hierarchical* results. The results

of *holistic* allocation are reported in Table 4.5.

Several observations can be made. First, the total schedule displacement achieved by the *holistic* allocation is 21% less than that of the *hierarchical* allocation. Same reduction rate for the displaced slots. The average slot displacement remained at the same level. Second, as expected, the *holistic* allocation would reduce the schedule displacement of the *New-entrants* by 71% and *Others* by 36%, but increase that of the *Change-to-historics* significantly by 483%. This is due to assigning *New-entrants* and *Others* slots equal priority with *Change-to-historics* slots. Lastly, we observe that the maximum displacement in the *holistic* results is 15 minutes less than that of the *hierarchical* allocation. This reduction stems from the *Others*. However, the displacement per slot increased by 14%, showing that fewer *Others* slots are displaced but with a larger displacement of individual slots. In summary, holistic allocation tends to reduce the schedule displacement, displaced slots and potentially the maximum displacement. However, the required computation time for *holistic* allocation also increased. It is expected that for large-scale airports, the time required for *holistic* allocation would be significantly longer than the *hierarchical* allocation as the latter divides the problem into smaller problems. For example, it takes more than 40 hours to solve the model holistically than hierarchically for a larger instance Table A.8.

Table 4.5: Holistic results for Airport 2. Values in parentheses are the relative change rate to the hierarchical results

	Threshold	**SchedDisp**	**MaxDisp**	**DispReqs**	**DispSlots**	**Disp./slot (min)**	**Time (sec)**
H	0	0	0	0	0	0	
CH	45m	3,193 (483%)	45m	143	1,924 (439%)	25 (8%)	
NE	15m	142 (-71%)	15m	14	142 (-71%)	15 (0%)	
OT	4h15m	11,393 (-36%)	4h	320	3,450 (-44%)	50 (14%)	
All		**14,728 (-21%)**	**4h**	**477**	**5,516 (-21%)**	**40 (0%)**	**133**

In Table 4.6, we present the slot allocation results produced by the airport slot coordinator, which is referred to as the coordinated schedule. It should be noted that the coordinated schedule does not satisfy all the capacity constraints considered in the SASA-R model. Therefore, we are unable to compare the results directly. However, we can still conclude that the slot allocation results obtained by the optimisation-based model are significantly better than the coordination results in terms of schedule displacement, maximum slot displacement and average slot displacement. However, displaced requests and slots are lower in the coordinated results than in the model results. Possible reasons for this could be that, in practice, coordinators usually process slot requests one at a time according to their priorities, thus not considering the complete set of requests simultaneously. Therefore, a request that is processed in the early stage is more likely to have zero or minimal displacement. Whereas later processed requests will have to be allocated with slots that are distant from their desired slots.

Table 4.6: Slot coordination results for Airport 2

	SchedDisp	**MaxDisp**	**DisReqs**	**DisSlots**	**Disp./slot (min)**
H	100	30m	2	50	30
CH	3,397	3h15m	44	636	80
NE	1,462	3h15m	16	320	68
OT	27,734	21h45m	255	3,494	120
All	**32,693**	**21h45m**	**317**	**4,500**	**110**

4.3.3 Sensitivity to the displacement costs

In this section, we investigate two alternative objective functions with different costs of displacement. The first one concerns weighting the displacement of the individual slot by the relative number of seats of the aircraft that uses the slot. The second one employs a squared displacement cost in the objec-

tive function.

Weighted objective function

With severe congestion, Level 3 airports have seen airlines using more aircraft with a larger seating capacity to increase the number of passengers served per flight movement. The current slot allocation regulations do not consider the size of the aircraft or the number of passengers as a primary criterion of slot allocation. However, previous studies (Jorge et al., 2021; Odoni, 2021) have paid attention to this issue. To further investigate this, we proposed another weighted objective function in which each slot is weighted by the relative number of seats of the aircraft using that slot. The weight for each slot request m, denoted by α_m, is calculated by dividing the number of seats in the aircraft associated with m by the number of seats in the largest aircraft at that airport (see 4.16, s_m denotes the number of seats of the aircraft associated with request m). The new weighted objective function is denoted by 4.17, which incorporates both the aircraft size (estimating the number of passengers) and the duration of the operation. Note that this is already considered in the objective function. The number of operating days $\sum_{d\in D} B_m^d$ can be seen as a weight to the displacement of each individual slot.

$$\alpha_m = s_m / \max_{m\in M} s_m \tag{4.16}$$

$$\min w_1 \sum_{m\in M}\sum_{d\in D} \alpha_m B_m^d z_m + w_2 \sum_{m\in M}\sum_{d\in D}\sum_{t\in T_m} \alpha_m B_m^d f_m^t x_m^t \tag{4.17}$$

$$\sum_{m \in M} \sum_{d \in D} s_m B_m^d z_m \tag{4.18}$$

In Table 4.7, we report the results of solving the SASA-R model with the new weighted objective function 4.17 and constraints 4.2 to 4.10. Aircraft are divided into 5 groups according to their number of seats. The first column shows the estimated total number of passengers that could not be served due to slot rejections, which is calculated by 4.18. As expected, when using the weighted objective function, the number of passengers affected by slot rejections is 12,900. This number is only 27% of that without consideration of aircraft size. However, the schedule displacement increased by 6% and the displaced slots increased by 8%. Although the displacement allocated to smaller aircraft increases, with a particular increase for the smallest aircraft with no more than 100 seats, the average slot displacement for all requests remains the same.

Table 4.7: Holistic results for Airport 2, with the weighted objective function. Values in parentheses are the relative change rate to the results without consideration of aircraft size. '-' indicates the number is compared against zero value

Seats	**RejPassengers**	**RejSlots**	**SchedDisp**	**DispSlots**	**Disp./slot (min)**
100	4,440 (-)	120 (-)	4,222 (91%)	2,206 (147%)	29 (-22%)
101 - 160	5,220 (-40%)	36 (-40%)	4,576 (6%)	1,592 (7%)	43 (0%)
161 - 200	3,240 (-70%)	18 (-70%)	8,856 (-2%)	3,508 (-8%)	38 (6%)
201 -300	0 (-)	0 (-)	2,225 (-25%)	847 (-34%)	39 (11%)
301 +	0 (-100%)	0 (-100%)	208 (-43%)	99 (-38%)	32 (-6%)
Total	**12,900 (-73%)**	**174 (-8%)**	**20,087 (6%)**	**8,252 (8%)**	**37 (0%)**

A squared displacement cost

We next study the cost of slot displacement. The current literature assumes that the cost of allocating a slot to a request is equal to the time differ-

ence between the requested slot and the allocated slot. However, the larger the difference, the more costly for airlines to adjust their flight schedule to utilise that slot. Therefore, we consider using a squared displacement cost to penalise the large displacement of individual slots. The squared cost of displacement is calculated by 4.19.

$$f_m^t = |t - \tau_m|^2 \tag{4.19}$$

We first solve the model with the default linear cost of displacement 4.6 and then replace this with the squared cost of displacement 4.19. Finally, we compare the results with the coordinated schedule in Table 4.8. In order to get comparable results, no maximum slot displacement constraints are considered. Results suggest that using the squared cost of displacement would result in a 31% increase in the schedule displacement and a 121% increase in the number of displaced slots. However, both the maximum slot displacement and the average slot displacement dropped greatly by more than two hours and 16 minutes. In the coordination results, 39% fewer requests did not receive their requested slots, but the schedule displacement and the average slot displacement are significantly worse than the optimisation solution. These results again demonstrate the benefit of using optimisation-based slot allocation approaches.

Table 4.8: Non-hierarchical results with the linear and squared cost of displacement. Values in parentheses are the relative change rate to the results for linear cost of displacement

Cost of displacement	**SchedDisp**	**MaxDisp**	**DisReqs**	**DisSlots**	**Disp./slot (min)**
1 Linear	12,930	4h30m	510	4,998	39
2 Squared	16,891 (31%)	2h15m	662 (30%)	11,047 (121%)	23(-41%)
3 Coordinated	32,693 (153%)	18h	312 (-39%)	4,570 (-9%)	107 (174%)

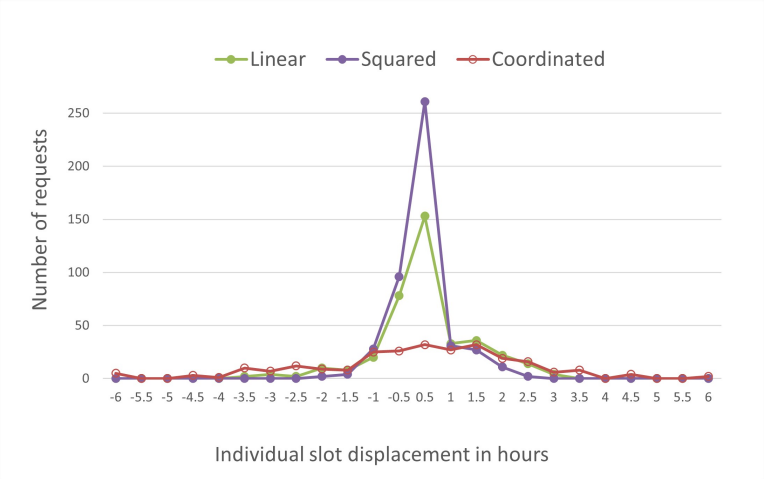


Figure 4.2: Distribution of slot displacement

Fig. 4.2 shows the distribution of slot displacement. A negative (positive) value on the x-axis indicates the allocated slot is earlier (later) than the requested slot. For requests that received their requested slots, the slot displacement would be zero, so not plotted in the figure. The first observation is that with the squared cost of displacement, the variance of slot displacement has significantly reduced compared to the results of the linear cost of displacement. 78% requests have zero displacements, and 95% requests received slots within 30 minutes of their requested slots, whereas the corresponding percentages are 82% and 92.6% in the linear cost of displacement results. The displacement of slots is more widely dispersed in the coordination results, ranging from 18 hours earlier to 4 hours later than the requested slots. Note that the schedule developed by the coordinators does not satisfy all the capacity and turnaround time constraints considered in the optimisation model. Another observation is that there are slightly more requests allocated with slots later than their requested slots than with slots earlier than their requested slots. It is worth investigating different displacement costs for allocating slots earlier or later than the requested

slot.

4.4 Discussion of Model Scalability

Appendix A.1 presents the results of solving the SASA-R model for Airport 1 (small) and Airport 3 (medium-sized airport). Regarding the slot allocation sensitivity to the changes in requested turnaround time (Table A.5), slot allocation priority rules (Table A.2) and displacement cost functions (Table A.4), similar conclusions can be drawn for Airport 1 as the results discussed in the previous section. Due to the small problem size for Airport 1, the time to solve the model exactly was insignificant in all experiments. For Airport 3, the largest instance tested, results demonstrated that the exact solver takes a significantly longer time to find the optimal solution when the model is solved hierarchically (Table A.7). The results of holistic allocation in Table A.8 show that the solver terminated after 41 hours due to being out of memory and returned the best-known solution with an optimality gap of about 6%. Recall that Airport 3 has 37% more requested slots than Airport 2 (Table 3.3). In addition, the proportions of congested arrival and departure time periods at Airport 3 are 1.5 to 3.6 times more as Airport 2 (Table 3.9). Moreover, the time interval for slot allocation is 15 minutes for Airport 2 and 5 minutes for Airport 3, which increases the number of decision variables and constraints significantly. As a result, the model cannot be solved in a short time for Airport 3. This motivated us to develop more effective algorithms to improve the scalability of the model.

4.5 Conclusions

In this chapter, we proposed a new model (SASA-R) for the single airport slot allocation problem considering slot rejections and displacement penalties. Unlike previous models, the maximum slot displacement is modelled as a constraint rather than a minimisation objective. As a result, the maximum displacement of each slot can be limited individually to guarantee the worst case of slot displacement is acceptable. Meanwhile, the model allows us to investigate slot rejections under different maximum allowable displacement thresholds. To our knowledge, this is the first analysis of slot rejections in the literature. However, due to the computational complexity of the model, it can only be solved for the two small airports we studied. For the medium-sized airport, the model can only be solved with a low maximum displacement threshold in an acceptable amount of time. Therefore, we develop a heuristic solution algorithm in the next chapter to solve this model for a medium-sized airport.

Sensitivity analysis was performed to analyse the impact of possible changes to the current WASG rules. Results suggest that allowing a small change in the requested turnaround time can significantly reduce the schedule displacement and the average slot displacement and reduce the computation time of solving the model. In addition, results demonstrated that it is necessary to limit the maximum allowable turnaround time. It is worth considering using different turnaround time parameters for different requests according to the factors that affect the aircraft turnaround operations, such as aircraft type, route, airline service level, etc.

Regarding priorities of slot allocation, we found that *holistic* allocation can lead to better schedule displacement, maximum displacement and the number of displaced slots compared to the *hierarchical* allocation. However,

the results of *change-to-historic* are much more degraded due to having equal priority with *New entrants* and *Others*. Results also show that *New entrants* cannot always benefit from the *holistic* allocation due to the different priority distribution at different airports.

Lastly, we tested two new objective functions. One involves weighting the objectives by the relative number of aircraft seats associated with each request. Results show that by using the weighted-based objectives, the estimated number of passengers associated with slot rejections can be considerably reduced. The other objective function uses a squared cost of displacement to penalise large slot displacement. Results suggest that the variance of slot displacement is significantly reduced compared to the linear cost of displacement results. Given the same maximum displacement threshold, the squared cost of displacement leads to a 40% reduction in average slot displacement and a 30% increase in the schedule displacement.

Chapter 5

A two-stage solution method for a single airport slot allocation model

Previous experiments have demonstrated that existing slot allocation models can only be solved for small to medium-sized airports. This chapter proposes a two-stage solution approach that aims to tackle large-scale problems. It is developed to solve the single airport slot allocation model proposed in Section 4.1. We first introduce the general framework of the two-stage solution method in Section 5.1. Next, Section 5.2 presents the greedy constructive heuristic, which is developed to generate feasible solutions. In Section 5.3, we introduce an adaptive large neighbourhood search heuristic (ALNS) to improve the initial feasible solution by iteratively applying a so-called 'destroy and repair' method. The experiment results and sensitivity analysis results are reported in Section 5.4. Lastly, the conclusions about the solution algorithm are presented in Section 5.4.

5.1 A Two-stage Solution Method

We introduce the general framework of the two-stage solution approach in Algorithm 3, which includes a greedy constructive heuristic and an improvement heuristic based on the large neighbourhood search. The algorithm takes a list of slot requests and airport coordination parameters as inputs and returns the best solution that can be found as output. In the first stage, the constructive heuristic generates an initial feasible solution x_0. After this, the global best solution x_{best} and the current solution x are initialised with x_0. In the second stage, the improvement heuristic iteratively destroys and repairs the incumbent solution x (line 4). In each iteration (lines 3-12), one of the four destroy operators is chosen according to a probability based on their historical performance. The selected destroy operator $destroy(.)$ removes a variable number of requests from the schedule. The repair method $repair(.)$ then attempts to improve the solution over the set of removed requests while fixing the value of decision variables for the other requests and returns a new complete solution x^*. Next, the new solution's cost (the objective function value) is evaluated. If a new global best solution is found, x_{best} is updated (lines 5-7). The new solution x^* is only accepted as the new current solution for the next iteration if the solution acceptance criteria are satisfied. Otherwise, x^* is rejected, which means that the current solution is retained. Note that since the partial solution is repaired by an exact method, the new solution obtained is at least as good as the current solution. The iteration process terminates when the stopping criteria are satisfied (e.g., the limit of computation time, the maximum number of iterations or the algorithm converged) and the best solution will be returned.

Algorithm 1 General Solution Framework

Input: set of requests, airport coordination parameters
Output: best solution x_{best}

$x_0 \leftarrow$ An initial feasible solution generated by Algorithm 2
incumbent solution $x, x_{best} \leftarrow x_0$
while stopping criterion is not met **do**
    $x^* \leftarrow repair(destroy(x))$
    if $\text{cost}(x^*) < \text{cost}(x_{best})$ **then**
        $x_{best} \leftarrow x^*$
    end if
    if solution acceptance criteria is met **then**
        $x \leftarrow x^*$
    end if
end while
return x_{best}

5.2 A Greedy Constructive Heuristic

Due to the complexity of slot allocation constraints, it is difficult to construct a feasible solution to the problem. First, schedule regularity must be maintained by allocating a series of slots simultaneously rather than individually. Second, declared runway capacities for a rolling time period significantly increase the number of capacity constraints. Third, there is also interdependence between paired arrival and departure slots due to the turnaround time constraints. Fourth, requests must be assigned slots on the same day as they are made. All these constraints result in a significant interaction among decision variables and make this problem highly constrained. In this section, we present a greedy constructive heuristic that aims to generate solutions that satisfy all the constraints in a short time while considering the solution quality. We introduce the solution construction procedure in Section 5.2.1 and describe the four request ordering heuristics in Section 5.2.2 and the greedy allocation algorithm in Section 5.2.3.

5.2.1 Solution construction procedure

The greedy constructive heuristic starts with an empty solution and gradually constructs a complete feasible solution by allocating slots to each request one at a time. Algorithm 1 presents the framework of the greedy constructive heuristic. The algorithm takes a list of requests and airport coordination parameters as inputs and returns a schedule that satisfies all constraints as an output. The solution is constructed by using a request ordering heuristic first and then a greedy allocation algorithm. Specifically, two types of ordering heuristics are used, a detailed description will be given in 5.2.2. If the ordering heuristic employed is static ordering, then it is executed at the beginning (line 2), before the allocation of any slot. If requests are ordered by a dynamic ordering heuristic, it is executed at the beginning of each iteration to provide a changing order of requests based on information from the current solution (line 4). Note that lines 2 and 4 are only active when the corresponding ordering heuristic is employed. Given the order of requests, the constructive heuristic iteratively schedules requests one at a time until all requests have been considered (lines 3-9). Within each iteration, the first request m in the ordered request list is selected and passed to the greedy allocation algorithm. Note that there are two possible outcomes of the greedy allocation algorithm: a request may receive a series of slots or no slot is allocated. Once all requests have been processed, a complete solution will be returned and rejected requests will not be considered again by the constructive heuristic.

5.2.2 Request ordering heuristics

Intuitively thinking, scheduling requests according to a specific order can lead to a better initial feasible solution than a random order. Therefore,

Algorithm 2 Framework of the constructive heuristic

Input: a list of requests, coordination parameters
Output: initial feasible solution x_0

initialising an empty solution $x \leftarrow \emptyset$
call the static ordering heuristic (list of requests)
while list of requests is not empty **do**
    call the dynamic ordering heuristic (list of requests, x)
    $m \leftarrow$ first request in the sorted list of requests
    remove m from the list
    call Algorithm 3 to find slots for m (m, x)
    update current solution x and remaining capacities
end while
$x_0 \leftarrow x$

the assumption of the request ordering heuristics is that requests that are more difficult to find feasible slots should be scheduled first with the hope that the easier requests can fit around the difficult ones and maintain the feasibility at the same time (Burke et al., 2014). Two types of request ordering heuristics are developed. A static ordering heuristic generates an order of requests based on the request's attributes, whereas a dynamic ordering heuristic produces an order of requests based on the current solution information, therefore the order changes during the solution construction procedure. As both ordering heuristics involve random factors, the constructive heuristic is stochastic. Next, we introduce each of the ordering heuristics.

Static ordering heuristics

Most days first (MD) orders requests in a non-increasing order by the total number of days each request is made for. This is similar to the current practice of airport slot coordinators. Requests with the same number of operating days are ordered randomly.

Dynamic ordering heuristics

Most conflicts first (MC) orders requests in a non-increasing order by the number of conflicts each request has with the current solution. Specifically, for each request m, requesting slot t on a set of days $\mathcal{D}_m$, we calculate the number of conflicts. It equals the total number of movements that have already been scheduled within the time window from t to t+1 hour, summed by $\mathcal{D}_m$, in the current solution. All requests start with zero conflict with the initial empty solution. Hence, *MC* is designed to start with a pre-selected request. This request is randomly selected from requests with the most number of days. Requests with the same value of the conflict metric are ordered randomly. The order of requests produced by *MC* is changing as the conflict metrics are updated every time a new request has been scheduled and the current solution is updated. Note that conflict metrics are only calculated for requests that have not been allocated slots.

Least remaining capacity first (LR) orders requests in a non-decreasing order by the remaining capacity left for each request. The remaining capacity of any time period of a day refers to how many more movements can be accommodated in that time period without breaking any capacity constraints. For each request m, requesting slot t on a set of days $\mathcal{D}_m$, the remaining capacity metric is calculated by summing the remaining capacities of the time period t to $t + 1$ hour for all days in $\mathcal{D}_m$. The remaining capacity for each time period of the season is updated when a new request has been processed. Similar to *MC*, this heuristic starts with a randomly selected request with the most number of days.

Largest displacement first (LD) orders requests in a non-increasing order by the best slot displacement of scheduling it to the current solution. The best slot displacement is the absolute time difference between the requested

slot and the closest available slot in the current solution. If the requested slot for a request is available, then the best slot displacement will be zero. Requests with the largest displacement will be scheduled first. This heuristic also starts with a randomly selected request with the most number of days.

Note that the above ordering heuristics can be combined or used individually. For example, we can use the *Most days first* as the first ordering criteria and *Most conflict first* as the second criteria. The performance of different ordering heuristics will be discussed in Section 5.2.4 after the introduction of the greedy allocation algorithm.

5.2.3 Greedy allocation algorithm

The greedy allocation algorithm takes a request which is determined by the ordering heuristics as input. Then it allocates a series of slots with the minimum slot displacement to this request or rejects it. The pseudo-code of the greedy allocation algorithm is given in Algorithm 2. At the beginning of the algorithm, several variables are initialised to hold key information (line 2). Given a request m, requesting for a series of slots at τ, the greedy allocation algorithm first checks time intervals no earlier than τ in the order of τ, τ+1, τ+2, ..., until the last time interval of the day T, or when the maximum allowable displacement is met. The first feasible slot that can be found would have the best displacement by far and this slot and the corresponding slot displacement will be stored in variables t' and $forward = |t' - \tau|$ for later comparison. After searching forward (lines 1-13), the algorithm checks time intervals earlier than τ in the order of τ-1, τ-2, ..., until the first time interval of the day, or the maximum allowable displacement is reached. Similarly, the first feasible slot that can be found is stored in t'', and the corresponding displacement is recorded by $backward = |t'' - \tau|$ (lines 14-25). After search-

ing forward and backward, if no feasible slots can be found on both sides, this request will be rejected and the paired request (arrival or departure request) will also be rejected and removed from the solution (lines 26-28). Otherwise, the displacement of allocating slots t' and t'' to this request are compared. The one with the smaller slot displacement will be assigned to the request. If they have the same displacement, any one of them can be allocated to the request (lines 30-33).

5.2.4 Experiment results

The greedy constructive heuristic was applied to solve the SASA-R model proposed in 4.1. We evaluate the performance of different ordering heuristics on the quality of the initial feasible solution obtained. All ordering heuristics were tested on each of the three airports, and 101 runs with distinct seeds were carried out. Results for the three airports are shown in Table 5.1,Table 5.2 and Table 5.3 respectively. The following are the conclusions:

- The greedy constructive heuristic is able to produce feasible solutions in less than 1 minute for all three airports. These feasible solutions can also be useful to initialise the optimisation solver to accelerate the speed of solving.
- There is no specific ordering heuristic that performs better than others in all three instances. However, the random ordering performed significantly worse than other manually designed ordering heuristics, demonstrating that the order of requests to be processed has a significant impact on the solution quality.
- Our hypothesis that requests that are more difficult to schedule should be handled first is experimentally supported. Results show that re-

Algorithm 3 Greedy allocation algorithm

Input: request m (requested slot time at τ), incumbent solution x
Output: solution x_0

$t \leftarrow \tau$
$t' \leftarrow None$, $t'' \leftarrow None$, $forward \leftarrow \infty$, $backward \leftarrow \infty$
while $t \leq q$ (last time interval of a day) **do**
    if turnaround constraints are met when m is scheduled at t **then**
        if capacity constraints are met when m is scheduled at t **then**
            $t' \leftarrow t$
            $forward \leftarrow |t' - \tau|$
            break
        else $t \leftarrow t + 1$
        end if
    else $t \leftarrow t + 1$
    end if
end while
$t \leftarrow \tau - 1$
while $t \geq 0$ **do**
    if turnaround constraints are met when m is scheduled at t **then**
        if capacity constraints are met when m is scheduled at t **then**
            $t'' \leftarrow t$
            $backward \leftarrow |t'' - \tau|$
            break
        else $t \leftarrow t - 1$
        end if
    else $t \leftarrow t - 1$
    end if
end while
if t' is *None* **and** t'' is *None* **then**
    reject request m
    reject the paired request j
else
    if $forward \leq backward$ **then**
        allocate slot series at t' to m
    else allocate slot series at t'' to m
    end if
end if
updates the current solution x_0 and the remaining capacity

quests with the most number of operating days of the season should be scheduled first in general. However, using *MD* alone leads to a

larger deviation of the solution, since requests with the same number of operating days are scheduled randomly.

- The standard deviation of all solution metrics tends to be reduced when *MD* is used as the first ordering criterion and another ordering heuristic as the second ordering criterion.

- As expected, *LD* takes the longest time to construct a feasible solution, as it calculates the best displacement of each request when it is considered. Surprisingly, this heuristic only provided the best solution for Airport 1. This indicates that the *LD* heuristic might be too greedy as it makes decisions purely based on what the best displacement of each request at the time is.

Table 5.1: Mean number of solution metrics generated by the greedy construction heuristic (with standard deviations in parentheses), Airport 1

1st criteria	2nd criteria	SchedDisp	MaxDisp	DispSlots	Time (sec)
MD	-	8,902 (205)	13 (1)	2,521 (57)	1
MD	*MC*	8,897 (53)	12 (0)	2,665 (22)	2
MD	*LR*	8,884 (55.1)	12 (0)	2,660 (23)	3
MD	*LD*	8,604 (0)	12 (0)	2,611 (0)	10
random	-	19,923 (4,689)	16 (4.8)	3,949 (446)	2

Table 5.2: Mean number of solution metrics generated by the greedy construction heuristic (with standard deviations in parentheses), Airport 2

1st criteria	2nd criteria	SchedDisp	MaxDisp	DispSlots	Time (sec)
MD	-	19,050 (140)	21 (0)	4,387 (19)	3
MD	*MC*	19,031 (19)	21 (0)	4,425 (0)	6
MD	*LR*	19,039 (20)	21 (0)	4,425 (0)	9
MD	*LD*	19,033 (92)	21 (0)	4,447 (14)	9
random	-	48,600 (10,341)	26 (7)	8,262 (663)	3

Table 5.3: Mean number of solution metrics generated by the greedy construction heuristic (with standard deviations in parentheses), Airport 3

1st criteria	2nd criteria	SchedDisp	MaxDisp	DispSlots	Time (sec)
MD	-	135,443 (2,870)	106 (3)	11,406 (100)	27
MD	*MC*	126,719 (0)	100 (0)	10,986 (0)	22
MD	*LR*	126,719 (0)	100 (0)	10,986 (0)	36
MD	*LD*	133,682 (1,403)	106 (0)	11,676 (101)	47
random	-	316,086 (47,642)	103 (12)	16,083 (629)	17

5.2.5 A group-based constructive heuristic

This section replicates the group-based constructive heuristic proposed by Ribeiro et al. (2019a). The assumptions of this method are the same as the greedy constructive heuristic, which is that requests with more number of operating days should be prioritised for slot allocation because they are more constrained by the schedule regularity requirements. Thus, requests are first ordered in a non-increasing order by the number of days (same as the *Most days first* ordering heuristic). Next, all requests are divided into a pre-defined number of groups, each group has an equal number of slots. The problem is then divided into several sub-problems, which are solved sequentially in the order of the number of days by using an optimisation solver. For details of this method, readers are referred to Ribeiro et al. (2019a). All integer linear programs in this chapter are solved using Gurobi 9.0.2 on a computer with an Intel Core i5-8365U processor.

We tested this approach using the largest data set we have, and we provide the solver with a feasible solution obtained by the greedy constructive heuristic. Results are shown in Table 5.4. We also compare these results with the best solution obtained by the greedy constructive heuristic (line 'Greedy CH'). As we can see, when requests are divided into two groups, the number of slots in each group is very large. The solver returned a solution with an optimality gap of 2.73% after 37 hours of computation. From

2 to 3 groups, the required computation time was reduced to 51 minutes, with a 5.5% increase in the schedule displacement. However, if the number of groups is too high, the schedule displacement increases significantly. The greedy constructive heuristic is equivalent to dividing the problem into n groups (n is the number of requests). Although it can generate feasible solutions in a couple of seconds, the schedule displacement is 50% larger than the result from the 2 groups.

The results suggest that the group-based approach can generate better solutions than the greedy constructive heuristic. The best trade-off between the computation time and solution quality can be achieved when the number of groups parameter is around the "sweet spot". In this case, dividing requests into four groups seems to achieve the best trade-off. The greedy constructive heuristic can generate feasible solutions much quicker but the solution quality is not very good. While the goal of the constructive heuristic is not to reach optimality, it is worth leveraging the structure of the problem to obtain a good initial solution as the starting point for the improvement heuristic.

Table 5.4: Solutions generated by the group-based constructive heuristic for Airport 3. Values in parentheses are the relative change rate to the results of 2 groups

#groups	MaxDisp	SchedDisp	Opt.Gap(%)	DispSlots	Run time
2	-	75,447	2.73%	-	37 hours
3	51	79,571 (5.5%)	0	11,131	51 min
4	51	87,738 (15.4%)	0	10,929	18 min
5	77	92,350 (19.3%)	0	10,797	14 min
6	97	98,941 (25.4%)	0	10,912	9 min
7	100	102,521 (27.4%)	0	10,719	8 min
Greedy CH	100	126,719 (50.0%)	-	10,986	22 sec

5.3 An Adaptive Large Neighbourhood Search Heuristic for solution improvement

The slot allocation problem by its nature is a highly constrained combinatorial optimisation problem, and it is the kind of problem which can be easily decomposed into a set of sub-problems, each consisting of a subset of slot requests. Moreover, it is very difficult to explicitly define the neighbourhood structure of a feasible solution. The features of this problem are considered to be well suited for the Large Neighbourhood Search (LNS) heuristic, in which the neighbourhood structure is typically defined by a **destroy and repair** method. Hence, we developed a self-adaptive improvement heuristic based on the LNS heuristic, called the Adaptive Large Neighbourhood Search (ALNS) heuristic. Specifically, the ALNS heuristic explores a large neighbourhood of the current solution by iteratively removing a set of requests from the current schedule (known as the destroy phase) and later in the repair phase, the removed requests are rescheduled with the aim of improving the solution quality.

5.3.1 Destroy operators

The performance of the ALNS heuristic significantly depends on the destroy operators. The underlying goal of them is to choose a set of requests that their solution is most likely to be improved. Four destroy operators are developed in the ALNS heuristic. Two of them, *random* and *worst* destroy operators, are widely used in the literature. The *time window* operator was used by Ribeiro et al. (2019a) and we modified it. The *related* destroy operator is a novel one which is tailored to this problem. Each destroy operator will be described in the following.

- ***random*** destroy operator: in order to prevent the search algorithm from getting stuck in a locally optimal region, the search is diversified by using the random destroy operator. In every iteration, the random destroy operator selects a variable number of requests that include $q_r\%$ of total requested slots at random so that the neighbouring size of the current solution can be controlled by the percentage parameter q_r. In our tests, q_r is set to a random number between 10% and 20%.

- ***worst*** destroy operator: it selects a variable number of requests that include q_w % of the total requested slots with the largest schedule displacement in the current solution. In our tests, q_w is set to a random number between 10% and 20%.

- ***time window*** destroy operator: this operator selects the set of requests currently located in a certain time window. Each time window is determined by two variables: centre time point o and radius r. For example, the time window associated with $o = 11$ and $r = 1$ hour is a 2-hour time window from 10 to 12. The *time window* destroy operator selects the centre time point randomly from a fixed set of time points that are separated by 1 hour from each other (e.g., 9:00, 10:00, ...). The radius r is an algorithm parameter that controls the size of the time window. Recall that requests are unevenly distributed across the day, when the time window locates in peak periods, a large number of requests would be chosen by this operator, leading to a very large part of the solution being destroyed. In this case, the benefit of the LNS heuristic is eliminated. Therefore, the number of selected requests is controlled by another algorithm parameter q_{tw} to avoid an unnecessarily large number of requests in that time window being destroyed.

- ***related*** destroy operator: The goal of this operator is to identify a set of interrelated requests by certain criteria to give a higher opportunity for the interchange of slots. We consider two requests to be related if the scheduling of one request may affect the scheduling of another request. For example, the pair of arrival and departure requests are related due to turnaround time constraints. The following part of this section provides a detailed description of this operator.

Two requests are defined as directly or 1st-degree related if their operating days coincide and the slots they are allocated are close in time. Let t_i be the slot time assigned to request i in the current solution, and $\mathcal{D}_i$ be the set of operating days of i. According to the definition, two requests a and b are 1st-degree related if $\mathcal{D}_a \cap \mathcal{D}_b \neq \emptyset$ and $|t_a - t_b| <= \alpha$, where α is a pre-defined threshold, called relatedness threshold, that measures how close in time is considered as related.

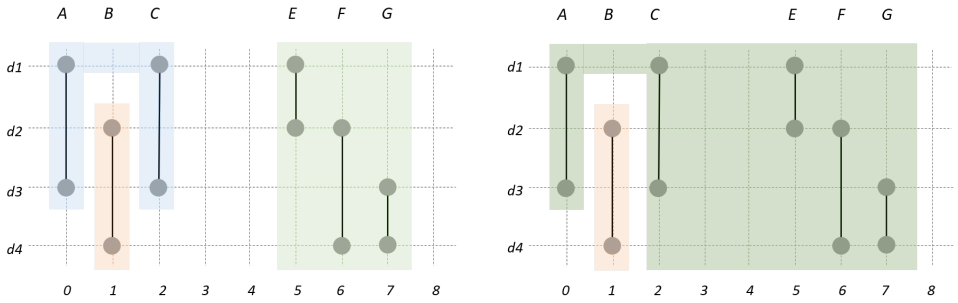


Figure 5.1: Illustration example of the relatedness of requests under different relatedness threshold: $\alpha = 2$ (left), $\alpha = 3$ (right); related requests are covered in the same colour

To make this clear, we use an example in Figure 5.1 to illustrate the relatedness between six requests A to E. Each line segment represents a request, including a series of slots (slots are represented by dots). The horizontal and vertical coordinates of each dot are time interval and day

index. In the left figure, the relatedness threshold α is set to two-time intervals. Under this scenario, request A and C are directly related, and request B is not directly related to A or C as the days associated with them are not coincide, and are thus unlikely to affect each other's schedule. Similarly, requests E and F, F and G are 1st-degree related. When we increase the relatedness threshold to three-time intervals, as shown in the right figure, requests C and E become related as their slot time difference falls below the relatedness threshold.

It is worth noting that even if two requests do not directly relate, they can still affect each other's schedule by a third request that is directly related to both of them. This is similar to the degrees of connection in a social media network. Specifically, for an arbitrary request, requests which are directly related to its 1st-degree relations are defined as 2nd-degree related (e.g., requests E and G in the left figure). Similarly, its 3rd-degree related requests are those directly related to its 2nd-degree relations (e.g., requests C and G in the right figure). Note that the pair of arrival and departure requests are always defined as 1st-degree related.

Fig. 5.2 illustrate how the related destroy operator works. It takes two parameters as inputs, the relatedness threshold parameter α and the degree of relatedness k (e.g., 1st-degree, 2nd-degree, etc), and it returns a set of requests for the destroy and repair procedure. The operator first selects an initial request m_0 randomly. Next, the operator identifies requests that are directly related to the initial request, denoted by $\mathcal{M}_0^{1st}$. The operator returns $\mathcal{M}_0^{1st}$ when k is 1st-degree. If k is set to 2nd-degree, the operator continues to identify requests that are 2nd-degree related to the initial request (1st-degree related to any requests in $\mathcal{M}_0^{1st}$), denoted by $\mathcal{M}_0^{2nd}$, and all requests within 2nd-degree relations to m_0 will be returned, which is the set of requests

$\mathcal{M}_0^{1st} \cup \mathcal{M}_0^{2nd}$.

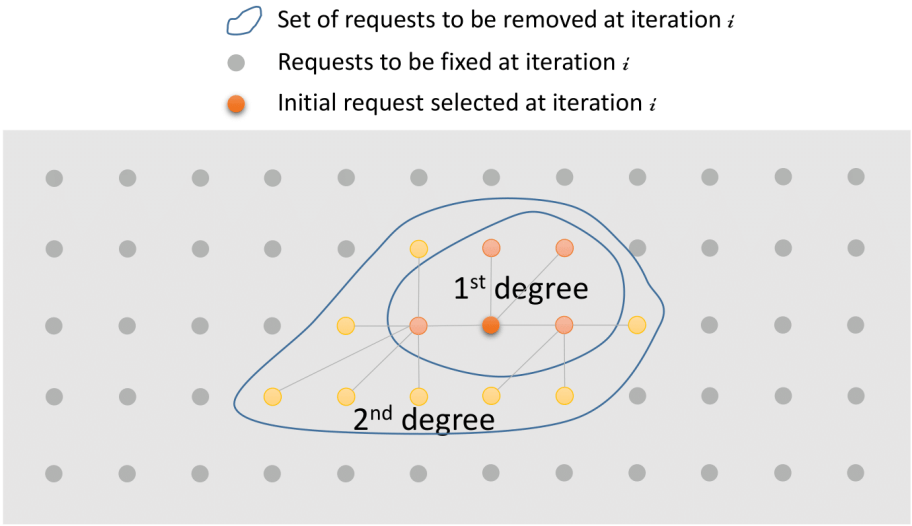


Figure 5.2: Related destroy operator

Finally, it is worth emphasising that the time window destroy operator selects a group of requests merely based on their scheduled time but does not necessarily consider if they are constrained by common constraints. For example, two requests from the same time window but on different days of the season are unlikely to exchange slots. In contrast, the related destroy operator only selects a set of related requests, thus the free capacities released by one request are more likely to benefit other related requests.

5.3.2 Repair method

The repair method takes as inputs the set of requests selected by one of the destroy operators and returns a complete solution as output. Let $\mathcal{M}^{\mathcal{R}}$ be the set of requests selected by the destroy operator, and $\mathcal{M}^{\mathcal{S}}$ be the set of remaining requests, $\mathcal{M}^{\mathcal{S}} \cup \mathcal{M}^{\mathcal{S}} = \mathcal{M}$. The partial solution to $\mathcal{M}^{\mathcal{S}}$ and $\mathcal{M}^{\mathcal{R}}$ are denoted by $x(\mathcal{M}^{\mathcal{S}})$ and $x(\mathcal{M}^{\mathcal{R}})$ respectively. The repair method

aims to solve the model formed by the set of removed requests $\mathcal{M}^{\mathcal{R}}$ and the remaining capacities $\mathcal{C}\backslash\mathcal{C}^{\mathcal{M}^{\mathcal{R}}}$ by using the Gurobi optimiser. Note that the full set of requests is still included in the model to maintain global feasibility. The solver returns the optimal solution for the set of removed requests $x^*(\mathcal{M}^{\mathcal{R}})$ or the best solution can be found in the given time limit. The complete solution $x(\mathcal{M})$ is constructed by concatenating the partial solution $x(\mathcal{M}^{\mathcal{S}})$ and $x^*(\mathcal{M}^{\mathcal{R}})$.

5.3.3 Adaptive heuristics

We now introduce the framework and the adaptive layer of the ALNS heuristic. It extends the large neighbourhood search by allowing multiple destroy and repair methods to be used within the same search. All four destroy operators introduced earlier in Section 5.3.1 are used, and we use an exact method for repairing. The framework of the ALNS heuristic is described in Algorithm 4. Line 1-6 presents the algorithm initialising. The current solution is initialised with a given initial feasible solution, which can be generated by the greedy constructive heuristic or the group-based constructive heuristic. Then we create a list of destroy operators, and each one's score is stored in the score list ω. Next, an empty set $\mathcal{V}$ is created to record the set of requests that have been selected by the destroy operators. Based on this information, the percentage of requests that have been considered can be obtained, denoted by the request coverage rate ρ. In lines 7-23, the current solution is iteratively destroyed by one of the destroy operators and repaired by using an optimisation solver. At each iteration, one destroy operator j is selected by a roulette wheel mechanism (more detail will be given in the following). Then the operator j identifies a subset of requests for improvement. Request coverage information $\mathcal{V}$ and ρ are updated accordingly. Next, we

initialise the optimisation solver by setting the initial feasible solution to the current solution. In addition, the free capacities released from the destroyed requests are updated to ensure global feasibility. The solver then solves the restricted version of the model in a given amount of time. It returns the optimal solution or the best solution that can be found within that time limit. If the solver-returned solution is better than the current solution over the removed and repaired set of requests, it means a better solution is obtained as the schedule of the remaining requests is fixed. This improved solution will be accepted immediately as the new current solution. The score associated with this destroy operator will be increased as a reward. If the current solution cannot be improved, it will remain the starting point of the next iteration. Note that the reason for this could be the current solution is local optimum or more computation time is required. However, it can also indicate that the applied destroy operator is not effective at that point during the search process. Therefore, the score associated with this destroy operator will be decreased according to the score updating mechanism explained next.

Different destroy operators are selected by a roulette wheel mechanism. If we have k operators with scores $\omega_j, j = 1, 2, ...k$, then operator j is selected with probability $\omega_j / \sum_{j=1}^{k} \omega_j$. In the ALNS heuristic, each destroy operator j is assigned the same score in the beginning ($w_j = 100, \forall j \in 1, 2, 3, 4$), so they have the same probability of being selected. The scores are adjusted dynamically as the search progresses according to their historical performance. The mechanism to update the score is simple. Every time operator j leads to a new global best solution, j is rewarded and its score will increase by 5%, therefore having a higher probability of being chosen in the next iteration. If the solution is not improved by using operator j, there will

be a penalty and the score of j will decrease by 5%. Therefore, the ALNS heuristic adapts to the state of the search and the instance at hand.

Finally, it is important to mention the strategies that we used to further diversify the search procedure and to avoid the same part of the solution being considered repetitively. First, the *worst* destroy operator is forbidden from being applied in two consecutive iterations. Second, the percentage of requests that have been selected by destroy operators is tracked during the search procedure. When this percentage is low, we want to prioritise exploring new areas of solution space. Hence, the *related* operator first selects a request that has not been visited and starting from that request, a set of related requests are removed. When the request coverage rate has reached the pre-defined level (e.g., 98%), the related destroy operator selects the initial request from all requests with probability. The more times a request has been selected, the less probability it will be chosen again.

5.4 Experiments

This section presents the experiment results of solving the single airport slot allocation model (Eq. (4.1) to Eq. (4.10)) using the two-stage solution method. The minimum allowable turnaround time parameter is set to 1 hour or the requested turnaround time if it is less than 1 hour. We permit 15 minutes increase in the requested turnaround time. The model has been solved for the largest instance we have. Since the focus of this chapter is on heuristic solution algorithms that are capable of solving large-scale problems, we do not consider request priorities in the experiments. This makes the problem even harder to solve since all requests need to be scheduled at the same time. The priority of slot allocation can be easily addressed by solving the problem of each priority class hierarchically or holistically. Therefore,

Algorithm 4 ALNS heuristic framework

Input: initial feasible solution x_0, algorithm parameters (see Table 5.6)
Output: best solution x_{best}

initialising the current solution: $x \leftarrow x_0$
initialising a list of destroy operators: $des. = [1, 2, ..., j]$
initialising scores for destroy operators: $\omega = [w_1, w_2, ..., w_j]$
set of visited requests: $\mathcal{V} \leftarrow \emptyset$
percentage of visited requests: $\rho \leftarrow 0$
iteration index $i \leftarrow 0$
while the stopping criteria is not met **do**
    select one destroy operator j according to $\omega_j, \mathcal{V}, \rho$
    $\mathcal{M}^{\mathcal{R}} \leftarrow j(x)$
    $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{M}^{\mathcal{R}}$
    update ρ
    $\mathcal{M}^{\mathcal{S}} \leftarrow \mathcal{M} \backslash \mathcal{M}^{\mathcal{R}}$
    update remaining capacity for $\mathcal{C} \backslash \mathcal{C}^{\mathcal{M}^{\mathcal{S}}}$
    solver initial solution $\leftarrow$ current solution x
    $x^*(\mathcal{M}^{\mathcal{R}}) \leftarrow$ solver
    if $\text{cost}(x^*(\mathcal{M}^{\mathcal{R}})) < \text{cost}(x(\mathcal{M}^{\mathcal{R}}))$ **then**
        $x' \leftarrow x^*(\mathcal{M}^{\mathcal{R}}) \cup x(\mathcal{M}^{\mathcal{S}})$
        update the current solution $x \leftarrow x'$
        reward destroy operator j: $w_j^{i+1} \leftarrow w_j^i * \mu^+$
    else penalise destroy operator j: $w_j^{i+1} \leftarrow w_j^i * \mu^-$
    end if
    $i \leftarrow i + 1$
end while
$x_{best} \leftarrow x$

the priority of slot allocation can be disregarded with no loss of generality.

All integer linear programs in this chapter are solved using Gurobi 9.0.2 on a computer with an Intel Core i5-8365U processor. Models are implemented using Python and Gurobi's Python API (gurobipy). A branch-and-cut method proposed by Fairbrother et al. (2020) was used to speed up the solution time of the model. Since the capacity constraints are only active during certain congested time periods of the season, they are added as lazy constraints through a callback function of Gurobi (the Gurobi LazyConstraints parameter is set to value 1). The rest of the Gurobi solver

parameters have been left at their default settings, such as the TimeLimit parameter is Infinity and the Relative MIP optimality gap is 0.0001.

Table 5.5 presents the results from solving the model using the Gurobi Optimiser. As expected, ignoring the priority constraints increases the computational time significantly, as all requests are scheduled at the same time. For small instances (Airports 1 and 2), the model can be solved to optimality in a couple of minutes. For Airport 3, Gurobi Optimiser runs out of memory after a few days of computation. It could not find a feasible solution and only provided the lower bound of solutions. As a result of this, we developed the ALNS heuristic to solve this instance.

Table 5.5: Results of Gurobi optimiser

Airport	Gap(%)	Time	SchedDisp	Rows	Columns
1	0.0	133 seconds	6,073	3,084	98,698
2	0.0	18 minutes	12,930	6,342	202,945
3	best bound	3 days	53,850	7,830	751,681

5.4.1 Data and set-up

The solution approach is implemented with the baseline inputs reported in Table 5.6. The initial feasible solution was generated by the greedy constructive heuristic with the *most days first* ordering heuristic. Note that this method is stochastic as requests with the same number of days are scheduled in random order. In order to focus on the performance of the improvement heuristic, we set the random seed to 1 in order to get the same solution each time for the initial feasible solution. The solution quality is measured by the schedule displacement, which is 132,785 time intervals (equivalent to 11,065 hours). Note that we do not know the optimal solution to this problem, we only know the lower bound of the schedule displacement at 53,850-time intervals, so this will be used to benchmark the solution obtained by using

heuristic methods. The *random* and *worst* destroy operators remove roughly 10% to 20% of the total requested slots at each iteration, the percentage is set to be a random number between this range. Note that slots from the same requests and paired requests must be removed together so the percentage is approximated. Next, the relatedness threshold α for the *related* operator is set to 1 hour. The degree of relation is set to within the 2nd degree based on some preliminary experimental results. The request coverage rate ρ is set to 0.98, which means that the *related* destroy operator will prioritise selecting requests that have not been considered until 98% of all requests have been considered at least once. Lastly, considering that some areas of the solution can be very difficult to improve, we limit the Gurobi optimiser solving time to 10 minutes in each iteration. The solver returns the best solution that can be found within the given time.

Table 5.6: Inputs of the ALNS heuristic and baseline values

	Parameters	Symbols	Baseline values
greedy CH	initial feasible solution	x_0	SchedDisp = 137,285
random destroy	% slots to be selected	q_r	rand(0.10, 0.20)
worst destroy	% slots to be selected	q_w	rand(0.10, 0.20)
related destroy	relatedness threshold	α	1 hour
related destroy	degree of relatedness	k	within 2nd-degree
related destroy	request coverage rate	ρ	0.98
time window destroy	time window radius	r	30 minutes
solver repair	time limit	t_l	10 minutes

We perform 10 runs of the ALNS heuristic on each instance, given the same initial feasible solution. The algorithm terminates when the convergence criteria are satisfied. It is the relative reduction in the schedule displacement in the last 10 iterations calculated by Eq. (5.1), where i is the current iteration number. $obj(x)^i$ is the schedule displacement in the i^{th}

iteration. ϵ^i (5.1) is the convergence value in the i^{th} iteration. The convergence criterion is not applied until the tenth iteration and the number of iterations was found to be adequate for the algorithm to converge.

$$\epsilon^i = \frac{obj(x)^{i-9} - obj(x)^i}{obj(x)^{i-9}} < 0.001 \tag{5.1}$$

5.4.2 Experiment results

Table 5.7 reports the average iterations, average schedule displacement (in the number of time intervals), average gap to the lower bound and the range of this gap over 10 runs. The ALNS heuristic yields solutions within 6.5% to the lower bound in about four hours, with the gap ranging from 6.43% to 6.61%. This shows the algorithm is able to provide high-quality solutions much faster than using the exact method. The improvement heuristic shows significant variability in early iterations. For example, after 1 hour, the range of gap to the lower bound is 17.97% to 42.33%. This is due to the random selection of requests and the impact of the request coverage strategy. Recall that when less than 98% of requests have been selected by destroy operators, the *related* destroy operator only selects the initial request from requests that have not been considered. Once the request coverage rate is above 98%, all

Table 5.7: ALNS heuristic results

Run time	Ave. Iterations	Ave. SchedDisp	Gap (%)	Gap Range (%)
30m	4	87,600	62.79	27.89-70.28
1h	22	67,340	25.05	17.97-42.33
1h 30m	30	63,011	17.01	10.55-26.63
2h	45	61,416	14.05	12.36-16.58
2h 30m	76	59,078	9.71	8.96-10.34
3h	86	57,802	7.34	6.66-7.72
3h 30m	95	57,396	6.58	6.41-7.52
4 h	104	57,337	6.48	6.43-7.03

requests will be selected according to probability. The more times a request has been selected, the less probability it will be chosen again. The schedule displacement tends to stabilise after two and a half hours and the algorithm converged within four hours.

We now compare the performance of the four destroy operators. The box plot Fig. 5.3 shows the computation time for each destroy operator. On average, the *related* destroy takes about 10 seconds to identify a set of related requests (within 2nd-degree relation). The other operators only take less than one second. The number of request pairs that are selected by each operator in each iteration is presented in Fig. 5.4. Recall that the *random* and *worst* operators select 10% to 20% of total requested slots in each iteration, however, the average number of request pairs removed by the *worst* destroy operator is significantly lower than the *random* operator, indicating that requests experiencing large displacement tend to have more slots requested in them. The *related* destroy operator selects the largest number of request pairs but a large variance was observed, due to the request relatedness structure. The *time window* operator selects the smallest number of paired requests given the time window size of 1 hour.

Figure 5.3: Running time of destroy operators

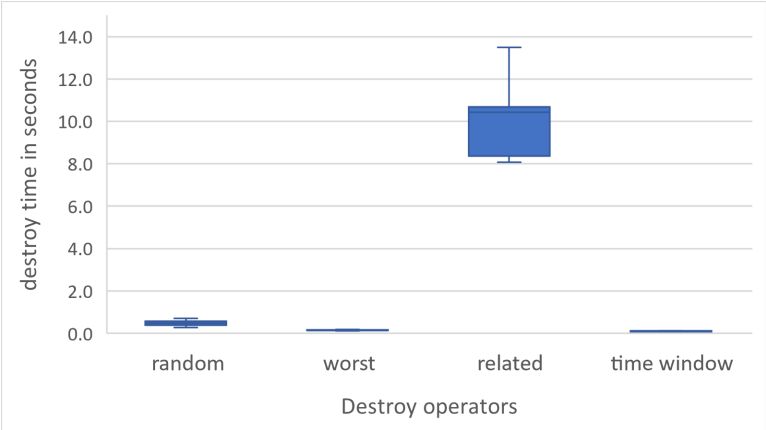


Figure 5.4: Request pairs removed by destroy operators

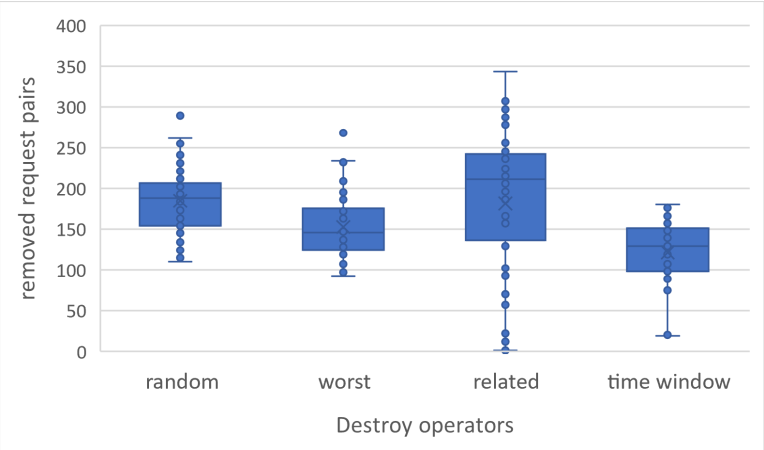


Fig. 5.5 shows the computation time to repair the partial solutions when different destroy operators are applied in each iteration. Note that the solving time limit for the solver is set to 10 minutes (600 seconds). The *worst* destroy operator results in the longest computation time for the solver to repair, averaging about 3 minutes. The *related* destroy operator causes the second longest time, about 80 seconds on average. However, a significant time variation can be observed for both the *worst* and *related* destroy operators. Based on these results, it appears that requests with large schedule displacements or those that are interrelated are harder to improve. Finally, we compare the improvement of schedule displacement in each iteration in Fig. 5.6a. Results show that the *worst* destroy operator can lead to a significant improvement of the schedule displacement in a single iteration. However, large improvements tend to occur in early iterations. Fig. 5.6b shows the zoomed-in part of Fig. 5.6a with the y-axis limited to o to -5000 time intervals. The median value of the improvements for *worst* destroy is 316, 166 and 146 for the *related* and *time window* destroy operators and only 61 for the *random* operator.

Figure 5.5: Solver computation time for the application of different destroy operators

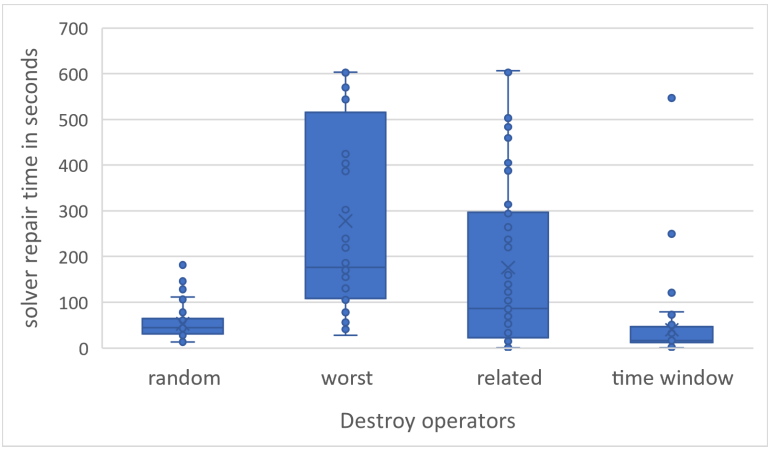
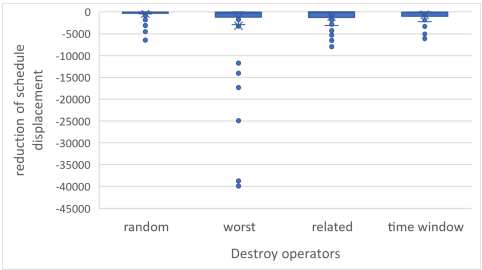
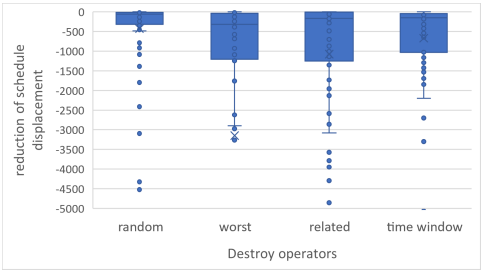


Figure 5.6: Reduction of schedule displacement in each iteration



(a) Original



(b) Zoom in

In summary, the two-stage solution method can provide high-quality solutions within 6.5% to the lower bound of the optimal solution in about four hours. In contrast, the Gurobi optimiser only returns a lower bound after three days of computations, even with a warm-start feasible solution obtained from the greedy constructive heuristic.

5.4.3 Sensitivity analysis to the algorithm parameters

The sensitivity tests aim to investigate the solution approach's effectiveness, test its sensibility to input parameters, and assess its computational

tractability. Parameters in this section are varied one at a time while the rest are set to their baseline values (see Table 5.6).

Sensitivity to initial feasible solutions. We first analyse the sensitivity of the ALNS heuristic to the quality of the initial solution ('starting point'). We know from results in Table 5.4 that the greedy constructive heuristic yields much worse feasible solutions than the group-based constructive heuristic. However, the former only takes a few seconds to generate feasible solutions, and the latter takes much longer time, depending on the number of groups. The baseline initial feasible solution is yielded by the greedy constructive heuristic with the *most days first* request ordering heuristic in 30 seconds. For better initial solutions, we choose the solutions obtained from the group-based constructive heuristics when the number of groups is 3, 5 and 6. The results are shown in Table 5.8. Column 'Initial Gap' shows the gap between the initial feasible solution to the lower bound and 'Time' indicates the computation time to get the feasible solutions. Not as expected, the worse starting point leads to a better best solution and the algorithm seems to converge more quickly when starting with a better initial solution. This indicates that the group-based constructive heuristic may lead to a locally optimal solution since it divides the problem into several small problems and solves each of them exactly. Moreover, the group-based constructive heuristic takes longer time than the greedy constructive heuristic to generate feasible solutions. Therefore, it is more advantageous to use the greedy heuristic for construction because this problem seems to favour a greater emphasis on solution improvement over construction (Burke et al., 2007a).

Table 5.8: Sensitivity to initial feasible solutions

Initial Sol.	Initial Gap (%)	Time	Ave. Gap (%)	Gap Range (%)	Convergence time
Greedy CH (baseline)	155.94	30 sec	6.48	6.43-7.03	4.00(3.89-4.85)
G =6	87.73	9 min	10.43	8.95-11.92	3.69 (2.82-4.56)
G= 5	71.49	14 min	14.67	8.95-20.40	2.04 (1.98-2.10)
G= 3	47.92	51 min	8.29	7.31-10.73	2.24 (1.11-3.66)

Solution degradation without each of the four destroy operators. We now examine the impact of removing one destroy operator at a time on the performance of the ALNS heuristic. Table 5.9 reports the average gap between the best solution and the lower bound and the gap range when each destroy operator is removed from use.

Firstly, the algorithm's coverage speed does not change much without the random destroy operator. However, the best solutions obtained are slightly worse than using all four operators. In addition, early iterations show a significant variation in solutions. Secondly, without the worst destroy operator, the algorithm converges much quicker in 2 hours, but the best solution is 6% worse than when all four destroy operators are used. It shows that the algorithm converges in local optima quickly and struggles to improve the solution without removing the worst parts of the solution. Thirdly, without the related destroy operator, the algorithm's coverage speed remains the same, but the best solution is degraded by 5%. Lastly, the solution degradation without the time window destroy operator is significant, resulting in a 24.60% gap between the best solution and the lower bound. The results demonstrate the advantages of using four different operators within the ALNS heuristics. The four destroy operators complement each other and work with different neighbourhood structures to diversify and intensify the search at the same time.

Table 5.9: Solution degradation without each of the four destroy operators

Run time	w/o random		w/o worst		w/o related		w/o time window	
	Gap (%)	Range (%)	Gap (%)	Range (%)	Gap (%)	Range (%)	Gap (%)	Range (%)
30 min	70.09	51.96-88.21	62.42	51.72-71.44	62.06	59.57-91.93	79.74	69.82-86.90
1 hour	68.67	37.52-78.48	30.06	15.12-54.50	43.55	30.47-55.21	48.16	36.61-53.33
1 h 30 min	42.22	21.88-66.55	22.34	16.99-33.53	24.80	14.48-32.36	31.92	26.62-36.61
2 h	27.22	18.50-34.51	12.63	10.84-13.61	21.42	15.68-29.99	25.49	22.65-29.20
2 h 30 min	23.68	17.06-33.11	-	-	18.90	13.14-25.94	24.64	22.37-28.39
3 h	16.37	12.33-23.61	-	-	15.59	15.43-16.51	24.60	23.91-25.20
3h 30m	11.58	9.58-13.04	-	-	14.14	13.80-14.29	-	-
4h	7.92	7.31-8.52	-	-	11.70	11.32-12.01	-	-

Sensitivity to the relatedness threshold α. α is a parameter of the *related* destroy operator, describing the time difference between a pair of directly related requests. The larger the value of α, the larger set of requests would be considered interrelated. A large value of α results in a very large neighbourhood to search, and the solution is time costly to be improved. If α is too small, a small group of requests will be removed, resulting in limited flexibility to swap slots across time intervals. Table 5.10 presents the results with α varies from 30 minutes to 90 minutes. Results suggest that the value of α affects the algorithm's convergence speed because the larger the value of α, the larger the neighbourhood size; thus more computationally intensive the model is at each iteration. In our test, $\alpha = 1$ hour achieved the best trade-off between the best solution quality and the algorithm's convergence time.

Table 5.10: Sensitivity to relatedness threshold α

α	Ave. SchedDisp	Gap(%)	Gap Range (%)	Convergence time
30 min	57,741	7.23	7.15-7.34	4.5 hours
1 hour (baseline)	57,337	6.48	6.43-7.03	4.0 hours
90 min	57,494	6.77	6.73-7.99	5.4 hours

Sensitivity to time window radius r. r is a parameter of the *time window* destroy operator, determining the size of the time window. The

larger the value of r, the bigger the time window and the larger the number of requests selected for improvement. Table 5.11 presents the results with r varies from 30 minutes to 60 minutes, and the corresponding time window is 1 hour, 1.5 hours and 2 hours. Results suggest that the value of r mainly affects the algorithm's convergence speed but does not significantly affect the best solution quality.

Table 5.11: Sensitivity to time window radius r

r	Ave. SchedDisp	Gap(%)	Gap Range (%)	Convergence time
30 min (baseline)	57,337	6.48	6.43-7.03	4.00 hours
45 min	57,507	6.97	6.66-8.65	4.51 hours
60 min	57,494	6.55	6.39-6.76	5.23 hours

Sensitivity to the adaptive layer of the ALNS. Finally, we assess the solution algorithm performance with and without the adaptive layer, which involves the probabilistic selection of the four destroy operators according to their past success in solution improvement. Results in Table 5.12 show that when the four destroy operators are selected randomly in each iteration, the algorithm takes a significantly longer time to converge, but the ultimate solution quality does change much from the results of the ALNS heuristic. The results validate the benefit of using multiple competing destroy operators and choosing them by probabilities based on their historical performance.

Table 5.12: Sensitivity to the adaptive layer of the ALNS heuristic

	Ave. SchedDisp	Gap(%)	Gap Range (%)	Convergence time
Adaptive LNS	57,337	6.48	6.43-7.03	4.00 hours
LNS	57,501	6.78	6.46-7.05	5.17 hours

The major findings from the sensitivity analysis are: First, the initial solution to the ALNS heuristic as the starting point can impact the perfor-

mance of the algorithm. A better quality feasible solution does not guarantee that a better final solution will be achieved by improvement heuristics. In our problem, better results are achieved by using the greedy constructive heuristic. And the group-based constructive heuristic can lead to local optima solutions and make it more difficult to improve by the ALNS heuristic. Second, the results demonstrate the benefit of using the four different destroy operators within the same search as they complement each other and lead to the best solutions together. Third, results validate the adaptive LNS can speed up the algorithm's convergence speed significantly. Finally, there seems to exist a 'best' value for the relatedness parameter of the related destroy operator that will lead to the best solutions. In contrast, the time window size of the time window destroy operators mainly affect the speed of algorithm convergence.

5.5 Conclusions

In this chapter, we proposed an original two-stage solution method based on the large neighbourhood search heuristic with a self-adaptive mechanism. Results suggest that the two-stage solution method can produce high-quality solutions within 6.5% to the lower bound in about four hours for a medium-sized airport. The exact method could only return a lower bound of the optimal solution after 3 days of computation with a warm-start solution produced by the greedy constructive heuristic.

The greedy constructive heuristic can generate feasible solutions for all three airports tested in less than one minute. The hypothesis of the request ordering heuristic that requests that are most difficult to find feasible slots should be scheduled first is experimentally supported. Although the initial solution quality is much worse than the solutions yielded from the group-

based constructive heuristic, it can lead to a better best solution together with the improvement heuristic. This indicates that the greedy constructive heuristic may not lead to many local optima.

Sensitivity analysis results also demonstrated the benefit of using four different operators within the ALNS heuristics. The four destroy operators complement each other and work with different neighbourhood structures to diversify and intensify the search simultaneously. Moreover, it is advantageous to use a self-adaptive heuristic as a hyper-heuristic to select which destroy operator as a lower heuristic to use.

In summary, the effectiveness of this algorithm benefits from the neighbourhood structure defined within the LNS heuristic. The algorithm takes advantage of the features of the problem itself and uses the relatedness between the decision variables to determine the neighbourhood structure. In addition, the algorithm only involves a small number of parameters, and only the relatedness parameters affect the best solution quality. Results also show that the ALNS heuristic can provide high-quality solutions even with various parameter settings and low-quality initial solutions.

The ALNS can be further improved from the following aspects: the first is to incorporate heuristic repair operators to accelerate the repairing phase. Several meta-heuristics can be used at the top level of ALNS to help the heuristic escape a local minimum. The second is to improve the adaptive heuristic by considering more complex reward and penalise mechanisms for each operator or each pair of destroy and repair operators based on how efficiently they improve the solution quality in each iteration.

We are motivated by the positive results of the ALNS heuristic to test its scalability and apply it on large instances to enhance the capability of existing models. More experiments need to be done to investigate if certain

problem features affect the algorithm's ability to find good solutions.

Chapter 6

A Flexible Scheduler for Single Airport Slot Allocation Problems

Allocating slots at the same time of the day to a regular flight helps airlines to create a regular schedule. However, it may lead to more slot rejections at very congested airports and a larger schedule displacement, ultimately affecting capacity utilisation negatively. This chapter investigates the possible ways that can provide flexibility to slot allocation while maintaining schedule regularity. An introduction is first given to the current schedule regularity rules in Section 6.1. Afterward, a novel single airport slot allocation model, called flexible scheduler, is proposed in Fig. 6.1. The model is then tested on real-world data to investigate the trade-off between schedule regularity and flexibility. We also tested the idea of changing the *series threshold* as another way to improve schedule flexibility in Section 6.3. In Section 6.4, we discuss the experiment results and make some conclusions about the proposed flexible slot allocation approach.

Table 6.1: Example of how requests are processed under the current schedule regularity rules (non-flexible) and with schedule flexibility

Schedule	Request	Time	Days in the week	Week index	Ad hoc/series	Slots
non-flexible	A	10:45	Mon - Sun	2,3	ad hoc	14
	B	10:45	Mon	1,2,3,4,5	series	5
	C	10:45	Mon, Fri	1,2,...,8	series	16
flexible	C-01	10:45	Mon	1,2,...,8	series	8
	C-02	10:45	Fri	1,2,...,8	series	8

6.1 Discussion of Schedule Regularity

According to the latest WSAG guidelines WASG (2020), requests are treated as ad hoc slot or series based on the operation duration associated with them. If the operation duration is less than five weeks (known as the *series threshold*), slots are allocated on an individual basis. Otherwise, slots are allocated in series. To make it more clear, we explain this through an example presented in Table 6.1. Request A, which can be a request for arrival or departure flights, is made for a group of slots at 10:45 on every day of the 2^{nd} and 3^{rd} week of the season, and it comprises a total of 14 slots. Since the number of weeks of request A is less than five weeks, it may not be assigned slots at the same time on different days. Request B is made for slots on a series of Mondays for five weeks, consisting of a total of 5 slots. Despite having fewer individual slots than request A, request B is qualified to receive a series of slots at the same time as it is made for a longer operation duration. Request C involves slots on two days of the week and eight weeks of the season, including a total of 16 slots. It is also treated as a series request, therefore it must be assigned a series of slots at the same time.

As we can see, allocating a series of slots at the same time of the day to a request can arise a problem called slot blocking, which has been identified in previous studies (Fairbrother et al., 2020; EUROPE, 2022). This means that

if one slot in a series of slots is not available, the entire series of slots cannot be allocated to that request, and this potentially increases the schedule displacement. Therefore, there is a need to investigate flexible slot allocation approaches to mitigate the blocking problem and improve schedule efficiency. In spite of the fact that flexible allocation does not fully comply with the current schedule regularity rules, we can obtain valuable insights into the trade-off between schedule regularity and schedule flexibility.

Several approaches have been proposed in light of schedule flexibility in the literature (Ribeiro et al., 2019b; Fairbrother et al., 2020; Odoni, 2021). The first method is to allocate slots to requests on different days independently. The problem with this approach is that the allocated slot on one day may be hours apart from the slot allocated on another day, and the times may also change frequently. Considering that passengers have regular travel needs and airlines need to develop regular flight schedules, frequent changes in slot times should be avoided. Therefore, it is important to restrict the changes in slot times to maintain a desired level of schedule regularity. The second method is to segment the entire season into several time periods, and then a request can be assigned different slots on different segments of the scheduling season. The third method is to increase the *series threshold* so that fewer requests have to be assigned slots in series. The rationale behind this is that requests with a shorter operation duration may block requests with a longer operation duration, so the schedule displacement can be increased. In addition, airlines may request a longer operation duration than they intend to operate in order to be eligible to get a series of slots at the same time. Test results of previous studies showed that the schedule displacement is not significantly sensitive to changes in the *series threshold* and the segmentation methods. Allocating slots for each day of the season

independently will make the problem too large to solve, and the schedule regularity is also difficult to maintain. This motivates the flexible slot allocation approach proposed in this chapter, which will be introduced in the next section.

Before we introduce the model formulation, we provide some analysis results of the request data of the instance under consideration. Table 6.2 shows the number of paired requests that fall into each group of the number of operating days in a week and the number of operating weeks. We observe that nearly 95% of requests are made for only one day in a week. However, requests with a large number of operating weeks, between 26 to 30, tend to have slots requested for multiple days in a week (i.e., 17 paired requests are made for every single day of the week for at least 26 weeks, each involving more than 180 slots), and they are most restricted by the schedule regularity constraints. Therefore, we believe that schedule efficiency will likely be improved if requests for multiple operating days in a week have more flexibility in their schedules.

Table 6.2: Number of request pairs with different operating days in a week and number of operating weeks of the season

	number of operating days in a week						
No. weeks	1	2	3	4	5	6	7
[1,5]	293	0	0	0	0	0	0
[6,10]	172	1	2	0	0	0	1
[11,15]	45	3	3	0	0	1	0
[16,20]	92	4	0	0	0	0	0
[21,25]	170	0	0	0	0	0	0
[26,30]	103	9	2	1	2	2	17

Table 6.3 shows the number of request pairs that fall into each group of operating days in a week and request types (ad hoc or series), given different *series threshold*. At the current threshold of five weeks, 652 paired requests are made for series operations and they must be allocated slots in

Table 6.3: Number of request pairs with different operating days in a week and request types, under different series threshold

		number of operating days in a week							
series threshold		1	2	3	4	5	6	7	total
< 5	ad hoc	271	0	0	0	0	0	0	271
≥ 5	series	604	17	7	1	2	3	18	652
< 10	ad hoc	445	1	1				1	448
≥ 10	series	430	16	6	1	2	3	17	475
< 15	ad hoc	497	4	5				1	507
≥ 15	series	378	13	2	1	2	3	17	416
< 20	ad hoc	581	8	5	0	0	1	1	596
≥ 20	series	294	9	2	1	2	2	17	327
< 25	ad hoc	684	8	5					697
≥ 25	series	191	9	2	1	2	2	17	224
< 30	ad hoc	839	10	5			1	1	856
≥ 30	series	10	7	2	1	2	2	7	31

series. The higher the threshold, the more requests are treated as ad hoc so are allocated individual slots. However, even when the *series threshold* is 30 weeks, there are still 31 request pairs that have to be allocated slots in series, out of which 21 are for multiple days in a week. This result motivates us to test the idea of increasing the series threshold and providing more schedule flexibility to requests for multiple operating days in a week.

6.2 A Flexible Scheduler for Single Airport Slot Allocation Problems

In this section, we present a new slot allocation model called the flexible scheduler. By using this model, a request can be allocated with slots at different times on different days of the week, subject to the range of slot time difference being no greater than a pre-defined flexibility range parameter. We can consider again the request C in Table 6.1, which contains two series of slots C-01 and C-02, each compromising 8 slots on a series of Mondays and Fridays respectively. The flexible scheduler schedules C-01 and C-02

individually and makes sure that the time difference between slots for C-01 and C-02 does not exceed the flexibility range. As far as we know, none of the previous studies has considered this, so we test this idea of schedule flexibility in this research.

We now introduce the formulations of the flexible scheduler. The flexible scheduler is extended from the model proposed in Section 4.1 (hereafter referred to as the non-flexible model). To avoid repetition, we use the same notation as in the non-flexible model. Following are the additional notations used to define the flexible scheduler.

Additional notations and parameters used in the model

$\mathcal{K}$: set of days of the week $\{0, 1, 2, 3, 4, 5, 6\}$, index by k

$\mathcal{D}_m$: set of days of request m, e.g., {0,1,...,90}

$\mathcal{K}_m$: set of days of the week on which request m is applied, $\mathcal{K}_m \subset \mathcal{K}$

$\mathcal{D}_m^k$: set of days on the k^{th} day of the week, $\mathcal{D}_m^k \subset \mathcal{D}_m$

γ: flexibility range parameter, $\gamma > 0$

The flexible scheduler distinguishes which days of the week are associated with each request. Hence, a number of sets are used to achieve this. The set $\mathcal{K}$ consists of integers from 0 to 6, representing Monday to Sunday respectively. Next, the set $\mathcal{K}_m$ indicates on which days of the week a request m has requested slots (e.g., $\mathcal{K}_m = \{0, 6\}$ indicates m is made for slots on Monday and Sunday). The set $\mathcal{D}_m$ is the set of days of the season on which the request m is applied. $\mathcal{D}_m^k$ is a subset of $\mathcal{D}_m$, corresponding to the series of slots on the k^{th} day of the week, if request m is made for that day of the week. Lastly, γ is a model parameter that represents the maximum allowable range of slot times difference between different days of the week.

γ is non-negative.

Decision variables

$$x_m^{kt} = \begin{cases} 1, \text{if request } m \text{ is assigned slots at } t, \text{ on the } k^{th} \text{ day of the week} \\ 0, \text{otherwise} \end{cases}$$

$$z_m^k = \begin{cases} 1, \text{if request } m \text{ is rejected, on the } k^{th} \text{ day of the week} \\ 0, \text{otherwise} \end{cases}$$

The binary decision variable x_m^{kt} captures whether a request m is allocated with slots at t, on the k^{th} day of the week or not. Note that the number of decision variables associated with each request m equals the number of operating days of m, ranging from 1 to 7. The second decision variable z_m^k is also binary, which indicates whether request m is rejected on the k^{th} day of the week or not. It is important to note that a decision to satisfy or reject is applied to each series of slots on the same day of the week rather than to all series of slots together. Furthermore, the paired arrival and departure slots are always allocated or rejected at the same time.

Objective function

$$\min w_1 \sum_{m \in M} \sum_{k \in k_m} |D_m^k| z_m^k + w_2 \sum_{m \in M} \sum_{k \in k_m} \sum_{t \in T_m} |D_m^k| f_m^{kt} x_m^{kt} \tag{6.1}$$

The objective 6.1 of the flexible scheduler is to first minimise the total number of rejected slots, and second the total schedule displacement. This objective function has the same form as the objective function of the non-flexible model (detailed description is provided in Section 4.1).

Constraints

$$T_m = \{t \in T | \tau_m - \phi_m \leq t \leq \tau_m + \phi_m\} \tag{6.2}$$

$$\sum_{t \in T_m} x_m^{kt} \leq 1, \forall m \in M, \forall k \in k_m \tag{6.3}$$

$$1 - \sum_{t \in T_m} x_m^{kt} = z_m^k, \forall m \in M, \forall k \in k_m \tag{6.4}$$

$$z_i^k = z_j^k, \forall (i,j) \in P, \forall k \in k_m \tag{6.5}$$

$$f_m^{kt} = |t - \tau_m|, \forall m \in M, \forall k \in k_m \tag{6.6}$$

$$(1 - z_i^k) T_{i,j}^{min} \leq \sum_{t \in T_j} t x_j^{kt} - \sum_{t \in T_i} t x_i^{kt} \leq T_{i,j}^{max}, \forall k \in k_m, \forall (i,j) \in P \tag{6.7a}$$

$$(1 - z_j^k) T_{i,j}^{min} \leq \sum_{t \in T_j} (T+t) x_j^{kt} - \sum_{t \in T_i} t x_i^{kt} \leq T_{i,j}^{max}, \forall k \in k_m, \forall (i,j) \in P \tag{6.7b}$$

$$\sum_{i \in M_{arr}} \sum_{t}^{t+L_c-1} x_i^{kt} \leq C_{dt}^{arr}, \forall d \in D, \forall k \in k_m, t \in T | t < T - L_c + 1 \tag{6.8}$$

$$\sum_{j \in M_{dep}} \sum_{t}^{t+L_c-1} x_j^{kt} \leq C_{dt}^{dep}, \forall d \in D, \forall k \in k_m, t \in T | t < T - L_c + 1 \tag{6.9}$$

$$\sum_{m \in M} \sum_{t}^{t+L_c-1} x_m^{kt} \leq C_{dt}^{total}, \forall d \in D, \forall k \in k_m, t \in T | t < T - L_c + 1 \quad (6.10)$$

$$| \sum_{t \in T_m} t x_m^{k't} - \sum_{t \in T_m} t x_m^{k''t} | \leq \gamma + M(2 - \sum_{t \in T_m} x_m^{k't} - \sum_{t \in T_m} x_m^{k''t}), \forall m \in M, \forall k', k'' \in k_m \quad (6.11)$$

Constraints 6.2 through 6.10 are similar to the constraints used in the non-flexible model. For simplicity, we do not repeat the description of each constraint here. The main difference is that the flexible scheduler creates constraints for each series of slots on the same day of the week, whereas the non-flexible model creates constraints for each request, which may include multiple series of slots on different days of the week. The flexibility range constraint 6.11 is new. It ensures that the difference of slot times allocated to a request m is no greater than a pre-defined flexibility range γ. Suppose k' and k'' are two arbitrary days of the week that request m is requested for, the two corresponding series of slots are denoted by $m^{k'}$ and $m^{k''}$. There are three possible outcomes of $m^{k'}$ and $m^{k''}$ in a flexible schedule and constraints 6.11 need to hold true in all cases. The first outcome is that both series of slots are scheduled, then the left-hand side of 6.11 equals the absolute time difference of the slots allocated to the two series, and the right-hand side is γ. Therefore, the range of changes in slot times can be restricted by the flexibility range parameter. Note that this constraint applies to any two series of slots of the same request. The second possible outcome is that $m^{k'}$ and $m^{k''}$ are both rejected. In this case, 6.11 becomes $0 \leq \gamma + 2M$, where M is a positive number. Since the right-hand side is non-negative, this inequality holds true. The third outcome is that one series of slots is

scheduled and the other one is rejected. In this case, no flexibility range constraints need to be applied. To achieve this, M needs to be sufficiently large to satisfy 6.11. Since in this case, the left-hand side is the slot time for the scheduled series of slots, M is therefore set to the last time interval of the day.

6.3 Experiments

6.3.1 Data and set-up

All integer linear programs in this section are solved using Gurobi 9.0.2 on a computer with an Intel Core i5-8365U processor. Models are implemented using Python and Gurobi's Python API (gurobipy). A branch-and-cut method proposed by Fairbrother et al. (2020) was used to speed up the solution time for the flexible scheduler. Since the capacity constraints are only active during certain congested time periods of the season, they are added as lazy constraints through a callback to the solver (the Gurobi LazyConstraints parameter is set to value 1). The rest of the Gurobi solver parameters have been left at their default settings, such as the TimeLimit parameter is Infinity and the Relative MIP optimality gap is 1e-4. In addition, the greedy constructive heuristic proposed in Section 5.2 can still be used to generate initial feasible solutions to the flexible scheduler. However, the initial solution is non-flexible.

The proposed flexible scheduler is solved for Airport 2 by the optimisation solver and for Airport 3 by using the two-stage approach introduced in Chapter 5. The tests are designed to evaluate the effectiveness of the flexible scheduler, assess its computational tractability, and identify its sensitivity to input parameters, in particular, the *series threshold* and the flexibility

range parameter. We solved the model both non-hierarchically and hierarchically, which incorporates slot priorities. In order to promote schedule flexibility, we set the minimum turnaround time parameter to the minimum of 1 hour and the initial requested turnaround time, as was done in Zografos et al. (2012); Fairbrother et al. (2020). The maximum turnaround time parameter is set to 15 minutes more than the requested turnaround time. According to our previous studies, this can lead to a 15% of reduction in the total schedule displacement as demonstrated in Table 4.2 (scenario D).

6.3.2 Non-hierarchical results

In this section, we analyse the results of solving the flexible scheduler non-hierarchically, which means that all requests are scheduled at the same time regardless of their priorities. The problem then becomes larger to solve. The greedy constructive heuristic developed for the non-flexible model is modified to generate feasible but non-flexible solutions. These solutions can be used as the warm-start solution to speed up the solving process of the exact solver. The maximum allowable displacement is not set at this stage but will be incorporated later.

We combine changing the *series threshold* and the flexibility range in the tests. The *series threshold* is set to 5 weeks (the current practice) and 10 weeks. The tested flexibility ranges are 0, 15, 30, 45, and 60 minutes. We also solved the model without considering the flexibility range constraints, the results correspond to γ=none.

In Table 6.4 we present the results in terms of schedule displacement (SchedDisp) in the number of time intervals, maximum slot displacement (MaxDisp) in minutes, number of displaced slots (DisSlots), and the com-

putation time required by the exact solver with a warm-start solution. In addition, we compare solutions by two schedule flexibility metrics. The maximum range metric Eq. (6.12) measures the biggest time difference between any two series of slots of the same request, on different days of the week. The second metric Eq. (6.13) is the average difference in slot times. Note that this metric is calculated only based on requests with multiple operating days in a week, as they are the only requests scheduled with flexibility (the divisor of Eq. (6.13)). The dividend is the sum of the slot time difference between any two series of slots of such requests.

$$\max_{m \in \mathcal{M}} | \sum_{t \in T_m} tx_m^{k't} - \sum_{t \in T_m} tx_m^{k''t} |, \forall k', k'' \in k_m \tag{6.12}$$

$$\sum_{m \in \mathcal{M}} \sum_{k', k'' \in k_m} | \sum_{t \in T_m} tx_m^{k't} - \sum_{t \in T_m} tx_m^{k''t} | / |\mathcal{M}|, \forall m \in \mathcal{M} || k_m | > 1 \tag{6.13}$$

We first study the impact of changing the flexibility range on schedule efficiency and regularity. As the flexibility range increases, the optimal schedule displacement decreased by up to 3.6% when the maximum degree of flexibility is allowed. With a flexibility range of 15 minutes, the schedule displacement can be reduced by 2%. In terms of the maximum range, it is always equal to the value of the flexibility range parameter indicating that the schedule flexibility constraints are tight. When the flexibility range is not restricted, the maximum range can be 150 minutes. For the average range, we observe that it is significantly less than the maximum range, showing that only a small percentage of requests received slots at different times on different days of the week. This also demonstrates that the schedule regularity only slightly deteriorated by using the flexible scheduler, even when the

maximum degree of flexibility is permitted (average range at 13.18 minutes). The maximum slot displacement and the number of displaced slots are not sensitive to the flexibility ranges.

Next, we study the effect of changing the *series threshold*. Moving from 5 to 10 weeks means that more requests are not assigned slots in series. As a result, the optimal schedule displacement can be reduced slightly by about 2.5% under the same level of flexibility. This small reduction makes sense in the light of Table 6.2 which shows that only three pairs of requests moved from series to ad hoc group when the *series threshold* changed from 5 to 10 weeks. The maximum range without flexibility range constraints is 30 minutes less compared to when *series threshold* is 5. Interestingly, both the number of displaced slots and the average range decreased as the *series threshold* increased. These results suggest that longer requests (operation duration above 10 weeks) tend to benefit from the increase of the *series threshold* because shorter requests (operation duration less than 10 weeks) are scheduled flexibly so the slot blocking issue caused by shorter requests are mitigated to some extent. Finally, we observe that the computation time for solving the flexible scheduler is very sensitive to the *series threshold*. We expect this model to be very difficult to solve if the series threshold is high and slots on every single day of the season are allocated individually.

Overall, the benefit of using the flexible scheduler appears to be more significant than increasing the *series threshold*. We believe the benefit of using the flexible scheduler will be more pronounced at airports with a larger number of requests consisting of multiple series of slots on different days of the week. As the flexible scheduler does not greatly increase the problem size and is, therefore, more efficient than increasing the *series threshold* to improve schedule flexibility.

Table 6.4: Non-hierarchical results for flexible scheduler with different series thresholds and flexibility ranges γ. $\gamma = 0$ indicates that the schedule is non-flexible. γ=None indicates that the flexibility range constraints are not considered. Values in parentheses are the relative reduction rate to the non-flexible schedule in the first row

series threshold	γ (minutes)	SchedDisp (intervals)	MaxDisp	DispSlots	Max.range (min)	Ave.range (min)	Time (sec)
5	0	11,307	3h45m	5,019	-	-	740
5	15	11,084 (-2.0%)	3h45m	5,144 (2.5%)	15	2.95	861
5	30	11,049 (-2.3%)	3h45m	5,144 (2.5%)	30	5.00	876
5	45	11,012 (-2.6%)	3h45m	5,053 (0.7%)	45	7.27	653
5	60	10,983 (-2.9%)	3h45m	5,055 (0.7%)	60	7.73	707
5	none	10,897 (-3.6%)	3h45m	5,046 (0.5%)	150	13.18	997
10	0	10,926 (-3.4%)	3h30m	4,736 (-5.6%)	-	-	4,844
10	15	10,747 (-4.9%)	5h15m	4,165 (-17.0%)	15	2.83	6,780
10	30	10,705 (-5.3%)	4h	4,752 (-5.3%)	30	3.00	3,724
10	45	10,688 (-5.5%)	4h	4,736 (-5.6%)	45	3.50	4,288
10	60	10,669 (-5.6%)	4h	4,731 (-5.7%)	60	3.67	6,488
10	none	10,635 (-5.9%)	3h45m	4,867 (-3.0%)	120	4.33	4,189

6.3.3 Hierarchical results

In this section, we solve the flexible scheduler hierarchically for the four primary priority classes in the order of *Historics* (H), *Changes-to-historic* (CH), *New entrants* (NE) and *Others* (OT). The results are presented in Table 6.5. The schedule of the *Historics* is unchanged across all tests since there is no displacement of historical requests. When a 15-minute flexibility range is permitted, the schedule displacement of *Changes-to-historic* is reduced by 3.5% compared to the non-flexible schedule. For *New entrants*, the schedule displacement and displaced slots both decreased by more than 70% compared to the non-flexible solution. This indicates that the flexible schedule of *Changes-to-historic* requests had a significant impact on the lower priority groups, especially on the *New entrants*. Further increase in the flexibility range does not affect the schedule of *Changes-to-historic* and *New entrants*, so the results for flexibility ranges above 15 minutes are not reported. The schedule displacement of *Others* can be reduced by 9.2% to 14.5% as the flexibility range further increases but the reduction becomes

less significant. The maximum slot displacement remains unchanged until the flexibility range is increased to 1 hour, and it is reduced by 30 minutes. The average range shows that very few *Changes-to-historic* and *New entrants* requests received various slots, even though they are allowed to be scheduled with flexibility. The *Others* have seen much more flexibility but the average range is still less than 30 minutes. The conclusion is that the impact of the proposed flexible scheduling approach is more significant when solving the model hierarchically. Even allowing a small level of flexibility for the higher priority class will greatly reduce the schedule displacement of requests with lower priorities. Therefore, by using the proposed flexible scheduler, *New entrants* and *Others* requests can benefit the most and the schedule regularity can also be maintained to the desired level by changing the values of the flexibility range parameter.

6.3.4 Effects on slot rejections

In this experiment, we examine whether the number of rejected slots can be reduced by using the flexible scheduler. The model is solved hierarchically with the maximum allowable slot displacement at 45 minutes and 1 hour. Results are shown in Table 6.6. Without schedule flexibility, 562 slots and 1,136 slots of the *Others* were rejected with the maximum allowable slot displacement at 1 hour and 45 minutes respectively. By using the flexible scheduler, 409 slots (12.8% less) were rejected and the schedule displacement was also reduced by 35%. Note that the same requests were rejected in the non-flexible and the flexible solution, but the number of rejected slots was reduced. This is because the flexible scheduler allows a series of slots for different days of the week to be rejected independently. However, the reduction in slot rejections as a result of flexible scheduling tends to diminish

as the maximum slot displacement threshold gets tighter (6.5% fewer slots were rejected compared to the non-flexible solution when the maximum allowable slot displacement is 45 minutes). Another observation is that the number of rejected slots is mainly determined by the maximum allowable slot displacement and it is not very sensitive to the flexibility range parameter. Any flexibility range above 45 minutes will not affect the number of rejected slots. This suggests that by using the flexible scheduler, the slot rejections can be reduced with little sacrifice to the schedule regularity.

Table 6.5: Hierarchical results for flexible scheduler with different flexibility ranges. The *series threshold* is fixed at 5. Values in parentheses are the relative reduction rate to the non-flexible schedule when $\gamma = 0$

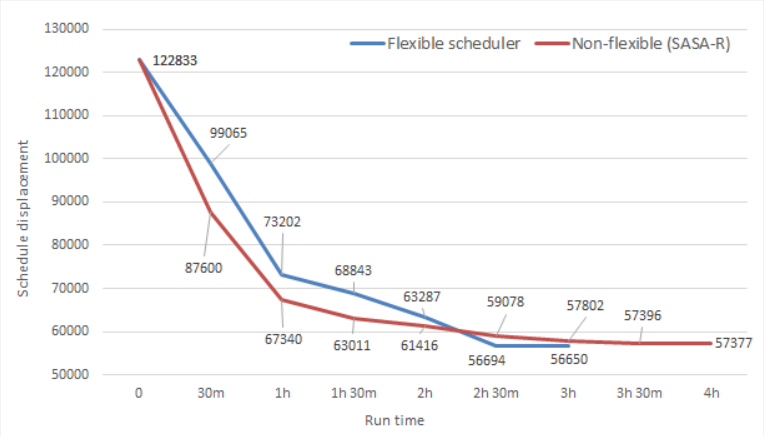
	γ	**SchedDisp**	**MaxDisp**	**DispSlots**	**Max.range(min)**	**Ave.range (min)**	**Time (sec)**
H		0	0	0	-	-	
CH		510	45m	334	-	-	
NE		255	15m	255	-	-	
OT		17,699	4h15m	5,968	-	-	
All	**0**	**18,464**	**4h15m**	**6,557**	-	-	**176**
H		0	0	0	0	0	
CH		492 (-3.5%)	45m	379 (13.5%)	15	1.5	
NE		75 (-70.6%)	15m	75 (-70.6%)	15	1.9	
OT		16,069 (-9.2%)	4h15m	5,203 (-12.8%)	15	7.3	
All	**15**	**16,636 (-9.9%)**	**4h15m**	**5,657 (-13.7%)**	-	-	**311**
OT		15,733 (-11.1%)	4h15m	5,077 (-14.9%)	30	10.5	
All	**30**	**16,300 (-11.7%)**	**4h15m**	**5,531 (-15.7%)**	-	-	**172**
OT		15,374 (-13.1%)	4h15m	4,799 (-19.6%)	45	14.8	
All	**45**	**15,941 (-13.7%)**	**4h15m**	**5,253 (-19.9%)**	-	-	**167**
OT		15,267 (-13.7%)	3h45m	4,808 (-19.4%)	60	19.1	
All	**60**	**15,834 (-14.2%)**	**3h45m**	**5,262 (-19.7%)**	-	-	**165**
OT		15,130 (-14.5%)	3h45m	4,746 (-20.5%)	180	21.8	
All	**None**	**15,697 (-15.0%)**	**3h45m**	**5,200 (-20.7%)**	-	-	**156**

6.3.5 Applying the ALNS heuristic to the flexible scheduler

The experiments are designed to test the capability of the proposed two-stage solution approach proposed in Chapter 5 to solve the flexible slot allocation model. Due to time constraints, the experiment is still in progress. Preliminary results show that the flexible scheduler model can be solved

by using the two-stage approach. Figure 6.1 compares the performance of the solution algorithm for solving the non-flexible model and the flexible scheduler for Airport 3. With regard to solution quality, flexible scheduling obtained solutions with 1.3% (3,635 minutes) displacement than the non-flexible model with a 30-minute flexible range. In terms of computational time, the two-stage algorithm converges in less time (1 to 1.5 hours) when applied to the flexible scheduler than the non-flexible model.

Figure 6.1: The performance of the two-stage solution approach for solving the non-flexible model and the flexible scheduler for Airport 3



6.4 Conclusions

In this chapter, we have investigated two ways to improve schedule flexibility. First, a new slot allocation model was proposed to allow multiple series of slots, in the same request, for different days of the week to be allocated or rejected individually. Meanwhile, the time difference between these series of slots can be limited by the flexibility range parameter to maintain the desired level of schedule regularity. Therefore, this model can provide a trade-off between schedule regularity and flexibility. Since the flexible scheduler

is more computationally complex than the non-flexible model proposed in Chapter 4, we used a modified version of the greedy constructive heuristic to generate initial feasible solutions to the flexible scheduler to improve solution times.

Experiment results showed that the flexible scheduler can reduce the number of rejected slots and the schedule displacement simultaneously. When the problem is solved hierarchically, the benefit of using the flexible scheduler is more significant, especially for requests with lower priorities such as the *New entrants* and *Others* (70% and 10% reduction of schedule displacement with 15 minutes flexibility range). Note that the schedule flexibility only applies to multiple series of slots on different days of the week, and requests with a longer operation duration tend to have multiple series of slots in a week, therefore the flexible scheduler gives more flexibility to requests with a long operation duration. This is different from the method of increasing the *series threshold*, which gives more slot allocation flexibility to requests with a shorter operation duration. These results also indicate that the flexible scheduler would likely be more useful for airports with more full-season requests such as very congested airports. Finally, we observe that even when given the flexibility of slot allocation, a high level of schedule regularity still can be achieved due to the objective to minimise the schedule displacement of all slots.

With regard to the flexibility range, we used a uniform range for all requests. However, the flexible scheduler allows one to specify this parameter for different priority groups, requests and flight types. For example, considering passengers' behaviours, departure flights should be more regularly scheduled than arrival flights, then departure requests should be given smaller flexibility ranges compared to arrival requests.

Table 6.6: Hierarchical results for flexible scheduler with different flexibility ranges and maximum allowable slot displacement ('Max. threshold'). The series threshold is fixed at 5. Values in parentheses are the relative reduction rate to the non-flexible schedule when $\gamma = 0$

Max. threshold		γ	**SchedDisp**	**MaxDisp**	**DispSlots**	**RejSlots**
	H		0	0	0	0
	CH		510	45m	318	
	NE		255	15m	255	
	OT		17,009	1h	7,721	562
1 hour	**All**	**0**	**17,774**	**1h**	**8,294**	562
	H		0	0	0	0
	CH		492 (-3.5%)	45m	379 (19.2%)	0
	NE		75 (-70.6%)	15m	75 (-70.6%)	0
	OT		10,974 (-35.8%)	1h	4,851 (-37.2%)	490 (-12.8%)
1 hour	**All**	**45min**	**11,541 (-35.1%)**	**1h**	**5,305 (-36.0%)**	490 (-12.8%)
	OT		11,405 (-33.3%)	1h	4,928 (-36.2%)	490 (-12.8%)
1 hour	**All**	**1h**	**11,972 (-32.6%)**	**1h**	**5,382 (-35.1%)**	490 (-12.8%)
	OT		10,915 (-36.2%)	1h	4,720 (-38.9%)	490 (-12.8%)
1 hour	**All**	**1h15m**	**11,482 (-35.4%)**	**1h**	**5,174 (-37.6%)**	490 (-12.8%)
	OT		10,718 (-37.2%)	1h	4,664 (-39.6%)	490 (-12.8%)
1 hour	**All**	**None**	**11,285 (-36.5%)**	**1h**	**5,118 (-38.3%)**	490 (-12.8%)
	H		0	0	0	0
	CH		525	45m	316	0
	NE		255	15m	255	0
	OT		10,945	45m	5,688	1,136
45min	**All**	**0**	**11,725**	**45m**	**6,259**	1,136
	H		0	0	0	0
	CH		492 (-3.5%)	45m	379 (19.2%)	0
	NE		75 (-70.6%)	15m	75 (-70.6%)	0
	OT		6,995 (-36.1%)	45m	4,102 (-27.9%)	1,062 (-6.5%)
45min	**All**	**45m**	**7,562(-35.5%)**	**45m**	**4,556 (-27.0%)**	1,062 (-6.5%)
	OT		6,934 (-36.6%)	45m	3,885 (-31.7%)	1,062 (-6.5%)
45min	**All**	**1h15m**	**7,501 (-36.0%)**	**45m**	**4,339(-30.7%)**	1,062 (-6.5%)
	OT		7,105 (-35.1%)	45m	3,991 (-29.8%)	1,062 (-6.5%)
45min	**All**	**None**	**7,672 (-34.6%)**	**45m**	**4,445 (-29.0%)**	1,062 (-6.5%)

Chapter 7

Conclusions

This chapter summarises the research in this thesis and highlights the key results derived from the experiments and analysis. Our final contribution is a discussion of the limitations of this study and suggestions for future directions.

7.1 Overview of this research

The main objective of this research is to enhance the decision support systems for airport slot allocation from an operational research point of view. My research mainly focuses on tackling single airport slot allocation problems with new models and solution algorithms. Three different areas were the focus of this research.

First, chapter 4 proposes a new model (SASA-R), which enhances existing models by considering slot rejections under different maximum displacement thresholds. The model is then used to investigate slot rejections and possible changes to the current slot allocation rules. The research findings help us better understand how these rules affect slot allocation and can be changed in the future.

Second, we introduced a two-stage solution approach for solving the SASA-R model in Chapter 5. A greedy constructive heuristic was developed, which is able to generate feasible solutions in less than one minute for a medium-sized airport. These feasible solutions can also be used to accelerate the solving process of optimisation solvers. We then proposed an adaptive large neighbourhood search heuristic to improve the quality of the initial feasible solution. It iteratively changes a large part of the solution by applying the so-called destroy and repair method. Four destroy operators are used and selected based on past success, and the repair method is an exact method that optimises a partial solution in each iteration.

Third, this thesis contributes to flexible slot allocation research. Chapter 6 investigates two possible ways to provide flexibility to slot allocation while maintaining schedule regularity. A flexible scheduler was developed, which allows a series of slots to be allocated individually on different days of the week. The flexibility range can be limited to maintain the desired level of schedule regularity. Therefore, the model helps us better understand the trade-off between schedule regularity and flexibility.

In addition, we presented a review of the latest slot allocation regulations and related literature and highlighted some future research needs in Chapter 2. Finally, a data analysis report is presented in Chapter 3, which helps us better understand the real-world slot request data and slot demand variations between airports. The results can provide references for developing mathematical models, solution algorithms, and insights on generating artificial request data sets from real-world data due to a lack of data availability. Our research findings will assist authorities in improving the existing slot allocation rules, and airport coordinators in improving the efficiency of airport slot allocation, especially in large airports, and ultimately improve

the utilisation of airport slots and passenger interests.

7.2 Key results

The following summarises the key results in the order of chapters of this thesis.

Better understanding of slot demand pattern at different slot coordinated airports. An in-depth analysis of the slot request data has identified a number of key attributes that can be used to define the underlying relationships between different properties in the data set of the demand for slots present in the data. The results demonstrate the difference in slot demand patterns at different airports, which should be taken into account when developing mathematical models and solution algorithms for the slot allocation problem. Due to the lack of real-world request data, this analysis also provides a basis for algorithm development and the generation of artificial data sets.

Better model for slot allocation considering slot rejections. Using the SASA-R model proposed in Chapter 4, the number of rejected slots can be minimised subject to different maximum allowable displacements. Slots can also be weighted based on the number of seats and operation duration of the aircraft using that slot. Therefore, requests associated with the highest number of passengers and the longest operating period will be more expensive to reject.

Better understanding of the maximum displacement. In contrast to many other models, the maximum slot displacement is modelled as a constraint rather than a minimisation objective in the SASA-R model. As a result, it reduces the number of decision variables and limits the worst case of slot displacement across all slots. Results demonstrated that when

the maximum displacement threshold is set low, the computation time for solving the model is significantly reduced. In addition, the maximum displacement threshold can be set individually for each slot, according to slot priorities, movement types (arrival or departure), flight distance or airlines' preferences.

Better understanding of the turnaround time constraints. Results in 4.3.1 suggest that restricting the allocated turnaround time to be the same as the requested turnaround time may be unnecessary. A small degree of change in turnaround time can improve schedule efficiency and reduce the computation time of finding the optimal solutions. However, it is necessary to limit the extension of the requested turnaround time. It is worth considering different feasible turnaround times for different flight pairs according to factors that affect the aircraft turnaround operations, such as aircraft type, route, airline service level, etc.

Schedule efficiency can be improved with holistic slot allocation, but the impact of the 'equal priorities' change on each priority class needs more investigation. Holistic slot allocation can improve the overall slot allocation efficiency compared to hierarchical allocation, reducing total schedule displacement, maximum displacement and the number of displaced slots. However, the slot allocation of *Change-to-historic* is much more degraded due to having equal priority with *New entrants* and *Others*. Results also show that *New entrants* cannot always benefit from the holistic allocation due to the fact that slot priority distribution varies at different airports. More research is needed to investigate how exactly the holistic allocation regime can be implemented to deliver better competition for airlines.

There is a way to reduce the variance of slot displacement at

the expense of a large number of displaced slots. To penalise the large displacement of individual slots, a squared cost of displacement is employed in the objective function. Results suggested that the variance of slot displacement can be significantly reduced. However, under the same level of maximum displacement, the squared cost of displacement leads to a 40% reduction in average slot displacement, a 30% increase in the schedule displacement, and a 120% increase in the number of displaced slots.

Model capability can be improved by a two-stage solution approach based on an adaptive large neighbourhood search heuristic. Chapter 5 proposes a two-stage solution method based on the large neighbourhood search heuristic with a self-adaptive mechanism. Results suggest that this solution method can produce high-quality solutions within 6.5% to the lower bound in about four hours for a medium-sized airport. The exact method could only return a lower bound of the optimal solution after 3 days of computation with a warm-start solution produced by the greedy constructive heuristic. Preliminary experiment results also demonstrated that the two-stage approach is effective to solve the flexible slot allocation model.

Feasible solutions can be generated quickly by the greedy constructive heuristic. The greedy constructive heuristic can generate feasible solutions for all three airports tested in less than one minute. Furthermore, feasible solutions can significantly speed up the solving time required by the optimisation solver.

Better understanding of the trade-off between solution construction and improvement. Experiment results show that a better quality feasible solution does not guarantee that a better final solution will be achieved by improvement heuristics, as constructive heuristics may lead

to the locally optimal solution.

We believe the adaptive LNS heuristic is able to adapt to various instance characteristics. The effectiveness of the adaptive LNS heuristic benefits from the neighbourhood structure defined by the four complementary destroy operators and the adaptive heuristic. The destroy operator takes advantage of problem features and inter-dependency between slot requests. In addition, the algorithm only involves a small number of parameters, and only the relatedness parameters affect the best solution quality. Therefore, we believe the ALNS heuristic can adapt to the search state and the instance at hand.

Flexible slot allocation model to analyse the trade-off between schedule regularity and flexibility. A flexible slot allocation model was proposed to schedule multiple series of slots, in the same request, individually for different days of the week. The time difference between slots can be restricted to a preferred level. Therefore, the model can provide a trade-off between schedule regularity and flexibility. We believe that the model can make a more significant impact when it is solved for a very congested airport with more full-season requests and when slot priorities are incorporated.

7.3 Future work

Our research has also identified some limitations with existing research, which can be improved in future research or lead to new research directions.

Fairness of slot rejections. An area that was not discussed in this thesis is the fairness of slot allocation. This issue has been the subject of a lot of previous research. However, none of them addressed the fairness of slot rejections. It is vital to have a decision support system for slot rejections, particularly at very congested airports. The issue of fair distribution of

rejections to airlines is a potential research direction that needs attention.

Testing for more airports. Due to a lack of data availability, the proposed model and algorithm have not been applied at some of the largest schedule-coordinated airports in the world. Testing the scalability of the proposed model and solution methods on larger instances represents an important avenue for future research. There is ongoing work to generate artificial slot request data to test the solution algorithm's effectiveness.

Solution algorithms for the flexible slot allocation model. There is ongoing work on solving the flexible slot allocation model using the two-stage method proposed in Chapter 5. A constructive heuristic is needed for generating feasible solutions with schedule flexibility.

Enhance the capability of existing models. There are currently many models proposed for the single airport slot allocation problem. Some of them have multiple objectives or multi-level solution frameworks. However, due to a lack of effective solution algorithms, these models have only been solved for small airports. Therefore, effective solution approaches need to be developed urgently to enhance the capability of existing models.

Improvements of the ALNS heuristic. There is ongoing work to develop heuristic repair operators. For example, in each iteration, the set of removed requests can be ordered by different request ordering heuristics used in the greedy constructive heuristic. Next, selected requests for removal can be reinserted into the schedule by a greedy insertion algorithm or probabilistic allocation algorithm. Several meta-heuristics can be used at the top level of ALNS to help the heuristic escape a local minimum.

Incorporating more slot allocation criteria. The SASA-R model does not formulate the 50/50 rules of slot allocation, which requires that 50% of the slots in the slot pool must be allocated to *New entrants* and the

other 50% to *Others* unless requests in such class are less than 50%. These constraints will affect the slot allocation result, especially slot rejections at very congested airports.

Heuristic search for network-wide slot allocation problems. One potential future research direction is to develop heuristic approaches to solve network-wide slot allocation problems. Although some researchers have been working on this area, more sophisticated approaches are required to cope with the huge problem size, flight duration constraints, flight pairing constraints and coherence, and objectives for network-wide slot allocation.

Appendix A

Results for SASA-R model

A.1 Results for Airport 1

Table A.1: Non-hierarchical results for Airport 1

Threshold	RejSlots	RejReqs	SchedDisp	MaxDisp	DispReqs	DispSlots	Disp/Slot(min)	Time(sec)
None	0	0	6,073	2h45m	259	2748	33	133
2h	0	0	6,083	2h	269	3189	29	92
1h30m	0	0	6,175	1h30m	255	2864	32	26
1h	0	0	6,374	1h	282	3466	28	19
30m	126	66	9,577	30m	304	5903	24	6
0	1,858	82	0	0	0	0	-	0.56

Table A.2: Hierarchical results for Airport 1

	Threshold	SchedDisp	MaxDisp	DispReqs	DispSlots	Disp./slot (min)	Time(sec)
H	–	205	45m	22	141	22	
CH	–	694	1h	16	300	35	
NE	–	676	1h	12	252	40	
OT	–	13,243	3h45m	133	3,518	57	
All		**14,818**	**3h45m**	**183**	**4,211**	**53**	**12**

Table A.3: Holistic results for Airport 1. Values in parentheses is the relative reduction rate to the hierarchical allocation results

	Threshold	**SchedDisp**	**MaxDisp**	**DispReqs**	**DispSlots**	**Disp./slot (min)**	**Time(sec)**
H	45m	205	30m	20	148 (5%)	21 (-4%)	
CH	1h	1,774 (155%)	1h	52	700 (133%)	38(8%)	
NE	1h	500 (-26%)	45m	12	252 (0%)	30 (-25%)	
OT	3h45m	6,728 (-49%)	3h15m	138	2,715 (-23%)	37 (-35%)	
All		**9,207 (-38%)**	**3h15m**	**222**	**3,815 (-9%)**	**36(-32%)**	**30**

Table A.4: Non-hierarchical results with the linear and squared cost of displacement. Values in parentheses are the relative change rate to the results for linear cost of displacement

Displacement cost	**SchedDisp**	**DisReqs**	**DisSlots**	**Disp./slot (min)**
1 Linear	6,374	282	3,466	28
2 Squared	7,228 (13%)	321 (14%)	5,146 (48%)	21 (-25%)

Table A.5: Sensitivity of non-hierarchical results to changes in turnaround times. The second column indicates the allowable reduction and increase in turnaround times

Scenario	**Changes in turnaround times**	**SchedDisp**	**DispReqs**	**DispSlots**	**Disp/Slot (min)**	**Time (sec)**
MaxDisp threshold = 2h, RejSlots=0						
A	0,0	6,352	280	2,980	32	43
B	flexible, 0	6,092 (-4%)	266	3,171 (6%)	29 (-9%)	64
C	0, +15min	5,313 (-16%)	255	2,803 (-6%)	28 (-13%)	43
D	flexible, +15min	5,192 (-18%)	254	2,787 (-6%)	28 (-13%)	32
E	flexible, +30min	4,905 (-23%)	229	2,498 (-16%)	29 (-9%)	26
F	flexible, none	4,674 (-26%)	220	2,402 (-19%)	29 (-9%)	13
MaxDisp threshold = 1h, RejSlots= 0						
A	0,0	6,626	296	3,688	27	41
B	flexible, 0	6,374 (-4%)	282	3,466 (-6%)	28 (4%)	19
C	0, +15min	5,569 (-16%)	235	3,086 (-16%)	27 (0%)	14
D	flexible, +15min	5,392 (-19%)	252	2,919 (-21%)	28 (4%)	9
E	flexible, +30min	5,110 (-23%)	239	2,682 (-27%)	29 (7%)	13
F	flexible, none	4,932 (-26%)	227	2,544 (-31%)	29 (7%)	6

A.2 Results for Airport 3

Table A.6: Non-hierarchical results for Airport 3

Threshold	RejSlots	RejReqs	SchedDisp	MaxDisp	DispReqs	DispSlots	Disp/Slot (min)	Time (sec)
30m	325	42	67,159	30m	920	19,997	3.358	2866
15m	1,243	148	37,889	15m	827	17,930	2.113	353
0	6,963	668	0	0	0	0	-	3.26

Table A.7: Hierarchical results for Airport 3

	Threshold	SchedDisp	MaxDisp	DispReqs	DispSlots	Disp./slot (min)	Time
H	–	23	5m	1	23	5	
CH	–	999	15m	40	667	7	
NE	–	2,650	30m	29	995	13	
OT	–	122,600	4h45m	527	13,039	141	
All		**126,272**	**4h45m**	**597**	**14,724**	**9**	**50 min**

Table A.8: Holistic results for Airport 3

	Threshold	SchedDisp	MaxDisp	DispReqs	DispSlots	Disp./slot (min)	Time
H	5m	23	5min	1	23	5	
CH	15m	8,288 (730%)	15min	230	4,496 (574%)	9 (29%)	
NE	30m	3,105 (17%)	30min	44	1,277 (23%)	12 (-8%)	
OT	4h45m	65,445 (-47%)	4h	496	9,954 (-24%)	33 (-77%)	
All	**-**	**76,861[1](-31%)**	**4h**	**771**	**15,750 (7%)**	**24 (167%)**	**41h**

[1] The Gurobi solver terminated after 41 hours due to out of memory. This is the best known solution which has an optimality gap of 5.96%.

Bibliography

Andersen, K., Cornuéjols, G., and Li, Y. (2005). Reduce-and-split cuts: Improving the performance of mixed-integer gomory cuts. *Management Science*, 51(11):1720–1732.

Androutsopoulos, K. N. and Madas, M. A. (2019). Being fair or efficient? a fairness-driven modeling extension to the strategic airport slot scheduling problem. *Transportation Research Part E: Logistics and Transportation Review*, 130:37–60.

Androutsopoulos, K. N., Manousakis, E. G., and Madas, M. A. (2020). Modeling and solving a bi-objective airport slot scheduling problem. *European Journal of Operational Research*, 284(1):135–151.

Atkin, J. A., Burke, E. K., Greenwood, J. S., and Reeson, D. (2008). A meta-heuristic approach to aircraft departure scheduling at london heathrow airport. In *Computer-aided Systems in Public Transport*, pages 235–252. Springer.

Ball, M., Barnhart, C., Nemhauser, G., and Odoni, A. (2007). Air transportation: Irregular operations and control. *Handbooks in operations research and management science*, 14:1–67.

Barnhart, C., Fearing, D., Odoni, A., and Vaze, V. (2012). Demand and

capacity management in air transportation. *EURO Journal on Transportation and Logistics*, 1(1-2):135–155.

Barnhart, C., Kniker, T. S., and Lohatepanont, M. (2002). Itinerary-based airline fleet assignment. *Transportation Science*, 36(2):199–217.

Benlic, U. (2018). Heuristic search for allocation of slots at network level. *Transportation Research Part C: Emerging Technologies*, 86:488–509.

Bent, R. and Van Hentenryck, P. (2004). A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38(4):515–530.

Boschetti, M. A., Maniezzo, V., Roffilli, M., and Bolufé Röhler, A. (2009). Matheuristics: Optimization, simulation and control. In *International workshop on hybrid metaheuristics*, pages 171–177. Springer.

Burke, E. K., Burke, E. K., Kendall, G., and Kendall, G. (2014). *Search methodologies: introductory tutorials in optimization and decision support techniques*. Springer.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2010a). A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pages 449–468. Springer.

Burke, E. K., Hyde, M., Kendall, G., and Woodward, J. (2010b). A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics. *IEEE Transactions on Evolutionary Computation*, 14(6):942–958.

Burke, E. K., Kendal, G., McCollum, B., and McMullan, P. (2007a). Constructive versus improvement heuristics: an investigation of examination

timetabling. In *3rd Multidisciplinary international scheduling conference: theory and applications*, pages 28–31. Citeseer.

Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., and Qu, R. (2007b). A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192.

Castelli, L., Pellegrini, P., and Pesenti, R. (2011a). Ant colony optimization for allocating airport slots. In *2nd International Conference on Models and Technologies for ITS*. Katolieke Universiteit Leuven.

Castelli, L., Pellegrini, P., Pesenti, R., et al. (2011b). Airport slot allocation in europe: economic efficiency and fairness. *International journal of revenue management*, 6(1-2):28–44.

Chakhlevitch, K. and Cowling, P. (2008). Hyperheuristics: recent developments. In *Adaptive and multilevel metaheuristics*, pages 3–29. Springer.

Chen, S., Chen, R., Wang, G.-G., Gao, J., and Sangaiah, A. K. (2018). An adaptive large neighborhood search heuristic for dynamic vehicle routing problems. *Computers & Electrical Engineering*, 67:596–607.

Commission, E. (1993). Regulation (eec) no 95/93 of the european parliament and of the council of 18 january 1993 on common rules for the allocation of slots at community airports.

Corolli, L., Lulli, G., and Ntaimo, L. (2014). The time slot allocation problem under uncertain capacity. *Transportation Research Part C: Emerging Technologies*, 46:16–29.

Delahaye, D., Chaimatanan, S., and Mongeau, M. (2019). Simulated annealing: From basics to applications. In *Handbook of metaheuristics*, pages 1–35. Springer.

Ding, H., Lim, A., Rodrigues, B., and Zhu, Y. (2005). The over-constrained airport gate assignment problem. *Computers & Operations Research*, 32(7):1867–1880.

Drake, J. H., Hyde, M., Ibrahim, K., and Ozcan, E. (2014). A genetic programming hyper-heuristic for the multidimensional knapsack problem. *Kybernetes*, 43(9/10):1500–1511.

Drexl, A. and Nikulin, Y. (2008). Multicriteria airport gate assignment and pareto simulated annealing. *IIE Transactions*, 40(4):385–397.

EUROCONTROL (2022). Eurocontrol aviation outlook 2050. `https://www.eurocontrol.int/sites/default/files/2022-04/eurocontrol-aviation-outlook-2050-main-report.pdf`.

EUROPE, A. C. I. (2022). Airport slot allocation position paper. `https://www.aci-europe.org/downloads/resources/ACI%20EUROPE%20Slots%20Position%20Paper%20with%20Preface%20March%202022_final%201.pdf`.

Fairbrother, J. and Zografos, K. G. (2021). Optimal scheduling of slots with season segmentation. *European Journal of Operational Research*, 291(3):961–982.

Fairbrother, J., Zografos, K. G., and Glazebrook, K. D. (2020). A slot-scheduling mechanism at congested airports that incorporates efficiency, fairness, and airline preferences. *Transportation Science*, 54(1):115–138.

Fischetti, M., Glover, F., and Lodi, A. (2005). The feasibility pump. *Mathematical Programming*, 104(1):91–104.

Gavish, B. and Pirkul, H. (1991). Algorithms for the multi-resource generalized assignment problem. *Management science*, 37(6):695–713.

Gendreau, M. (2003). An introduction to tabu search. In *Handbook of metaheuristics*, pages 37–54. Springer.

Glover, F. (1989). Tabu search part i. *ORSA Journal on computing*, 1(3):190–206.

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.

Haimes, Y. (1971). On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE transactions on systems, man, and cybernetics*, 1(3):296–297.

Hanafi, R. and Kozan, E. (2014). A hybrid constructive heuristic and simulated annealing for railway crew scheduling. *Computers & Industrial Engineering*, 70:11–19.

Hansen, P. and Mladenović, N. (2001). Variable neighbourhood search: Principles and applications. *European journal of operational research*, 130(3):449–467.

Hemmelmayr, V. C., Cordeau, J.-F., and Crainic, T. G. (2012). An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & operations research*, 39(12):3215–3228.

IATA (2019). Worldwide airport slots. `https://www.iata.org/policy/slots/Pages/index.aspx`.

IATA (2021). Worldwide airport slots fact sheet. `https://www.iata.org/en/iata-repository/pressroom/fact-sheets/fact-sheet---airport-slots/`.

IATA, ACI, and WWACG (2020). Worldwide Airport Slot Guidelines (WAGS). Technical report.

Jacquillat, A. and Odoni, A. R. (2015). An integrated scheduling and operations approach to airport congestion mitigation. *Operations Research*, 63(6):1390–1410.

Jiang, Y. and Zografos, K. G. (2021). A decision making framework for incorporating fairness in allocating slots at capacity-constrained airports. *Transportation Research Part C: Emerging Technologies*, 126:103039.

Johnson, D. S. and Garey, M. R. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. WH Freeman.

Jorge, D., Ribeiro, N. A., and Antunes, A. P. (2021). Towards a decision-support tool for airport slot allocation: Application to guarulhos (sao paulo, brazil). *Journal of Air Transport Management*, 93:102048.

Karsu, Ö., Azizoğlu, M., and Alanlı, K. (2021). Exact and heuristic solution approaches for the airport gate assignment problem. *Omega*, 103:102422.

Kasirzadeh, A., Saddoune, M., and Soumis, F. (2017). Airline crew scheduling: models, algorithms, and data sets. *EURO Journal on Transportation and Logistics*, 6(2):111–137.

Katsigiannis, F. A. and Zografos, K. G. (2021). Optimising airport slot allocation considering flight-scheduling flexibility and total airport capacity constraints. *Transportation Research Part B: Methodological*, 146:50–87.

Katsigiannis, F. A., Zografos, K. G., and Fairbrother, J. (2021). Modelling and solving the airport slot-scheduling problem with multi-objective, multi-level considerations. *Transportation Research Part C: Emerging Technologies*, 124:102914.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

Kohl, N. and Karisch, S. E. (2004). Airline crew rostering: Problem types, modeling, and optimization. *Annals of Operations Research*, 127(1-4):223–257.

Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press.

Lambelho, M., Mitici, M., Pickup, S., and Marsden, A. (2020). Assessing strategic flight schedules at an airport using machine learning-based flight delay and cancellation predictions. *Journal of Air Transport Management*, 82:101737.

Land, A. H. and Doig, A. G. (2010). An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*, pages 105–132. Springer.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2010). Iterated local search: Framework and applications. In *Handbook of metaheuristics*, pages 363–397. Springer.

Lourenço, H. R. and Serra, D. (1998). *Adaptive approach heuristics for the generalized assignment problem*. Universitat Pompeu Fabra.

Lučić, P. and Teodorović, D. (2007). Metaheuristics approach to the aircrew rostering problem. *Annals of Operations Research*, 155(1):311–338.

Marler, R. T. and Arora, J. S. (2010). The weighted sum method for multi-objective optimization: new insights. *Structural and multidisciplinary optimization*, 41(6):853–862.

Mavrotas, G. (2009). Effective implementation of the ε-constraint method in multi-objective mathematical programming problems. *Applied mathematics and computation*, 213(2):455–465.

Odoni, A., Morisset, T., Drotleff, W., and Zock, A. (2011). Benchmarking airport airside performance: Fra vs. ewr. *Proceedings of the 9th USA/Europe Air Traffic Management Research and Development Seminar, ATM 2011*.

Odoni, A. R. (2021). A review of certain aspects of the slot allocation process at level 3 airports under regulation 95/93.

Öncan, T. (2007). A survey of the generalized assignment problem and its applications. *INFOR: Information Systems and Operational Research*, 45(3):123–141.

Osman, I. H. (1995). Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. *Operations-Research-Spektrum*, 17(4):211–225.

Pearl, J. (1984). *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc.

Pellegrini, P., Bolić, T., Castelli, L., and Pesenti, R. (2017). Sosta: An effective model for the simultaneous optimisation of airport slot allocation. *Transportation Research Part E: Logistics and Transportation Review*, 99:34–53.

Pellegrini, P., Castelli, L., and Pesenti, R. (2012). Metaheuristic algorithms for the simultaneous slot allocation problem. *IET Intelligent Transport Systems*, 6(4):453–462.

Pillac, V., Gueret, C., and Medaglia, A. L. (2013). A parallel matheuristic for the technician routing and scheduling problem. *Optimization Letters*, 7(7):1525–1535.

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8):2403–2435.

Pisinger, D. and Ropke, S. (2019). Large neighborhood search. In *Handbook of metaheuristics*, pages 99–127. Springer.

Pour, S. M., Drake, J. H., and Burke, E. K. (2018). A choice function hyper-heuristic framework for the allocation of maintenance tasks in danish railways. *Computers & Operations Research*, 93:15–26.

Praseeratasang, N., Pitakaso, R., Sethanan, K., Kosacka-Olejnik, M., Kaewman, S., and Theeraviriya, C. (2019). Adaptive large neighborhood search to solve multi-level scheduling and assignment problems in broiler farms. *Journal of Open Innovation: Technology, Market, and Complexity*, 5(3):37.

Qu, R. and Burke, E. K. (2009). Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems. *Journal of the Operational Research Society*, 60(9):1273–1285.

Raidl, G. R. and Puchinger, J. (2008). Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In *Hybrid metaheuristics*, pages 31–62. Springer.

Ribeiro, N. A., Jacquillat, A., and Antunes, A. P. (2019a). A large-scale neighborhood search approach to airport slot allocation. *Transportation Science*, 53(6):1772–1797.

Ribeiro, N. A., Jacquillat, A., Antunes, A. P., and Odoni, A. (2019b). Improving slot allocation at level 3 airports. *Transportation Research Part A: Policy and Practice*, 127:32–54.

Ribeiro, N. A., Jacquillat, A., Antunes, A. P., Odoni, A. R., and Pita, J. P. (2018). An optimization approach for airport slot allocation under iata guidelines. *Transportation Research Part B: Methodological*, 112:132–156.

Ross, G. T. and Soland, R. M. (1975). A branch and bound algorithm for the generalized assignment problem. *Mathematical programming*, 8(1):91–103.

Ross, G. T. and Zoltners, A. A. (1979). Weighted assignment models and their application. *Management Science*, 25(7):683–696.

Ross, P., Schulenburg, S., Marín-Bläzquez, J. G., and Hart, E. (2002). Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 942–948. Morgan Kaufmann Publishers Inc.

Sahni, S. and Gonzalez, T. (1976). P-complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565.

Scala, P., Mota, M. M., Wu, C.-L., and Delahaye, D. (2021). An optimization–simulation closed-loop feedback framework for modeling the airport capacity management problem under uncertainty. *Transportation Research Part C: Emerging Technologies*, 124:102937.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer.

Soykan, B. and Rabadi, G. (2016). A tabu search algorithm for the multiple runway aircraft scheduling problem. In *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, pages 165–186. Springer.

Vaz, M. (2015). *Large neighborhood search for the vehicle routing problem with time windows and deterministic and stochastic travel and service times.* PhD thesis, uniwien.

Wang, S., Drake, J. H., Fairbrother, J., and Woodward, J. R. (2019). A constructive heuristic approach for single airport slot allocation problems. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1171–1178. IEEE.

WASG (2020). Wasg edition 1 - english version (pdf) official version. `https://www.iata.org/contentassets/4ede2aabfcc14a55919e468054d714fe/wasg-edition-1-english-version.pdf`.

Xu, J. and Bailey, G. (2001). The airport gate assignment problem: mathematical model and a tabu search algorithm. In *Proceedings of the 34th annual Hawaii international conference on system sciences*, pages 10–pp. IEEE.

Yang, X.-S. (2010). *Engineering optimization: an introduction with metaheuristic applications.* John Wiley & Sons.

Zeng, W., Ren, Y., Wei, W., and Yang, Z. (2021). A data-driven flight schedule optimization model considering the uncertainty of operational displacement. *Computers & Operations Research*, 133:105328.

Zografos, K. and Jiang, Y. (2016). Modelling and solving the airport slot scheduling problem with efficiency, fairness, and accessibility considerations.

Zografos, K. G., Androutsopoulos, K. N., and Madas, M. A. (2018). Minding the gap: Optimizing airport schedule displacement and acceptability. *Transportation Research Part A: Policy and Practice*, 114:203–221.

Zografos, K. G. and Jiang, Y. (2019). A bi-objective efficiency-fairness model for scheduling slots at congested airports. *Transportation Research Part C: Emerging Technologies*, 102:336 – 350.

Zografos, K. G., Salouras, Y., and Madas, M. A. (2012). Dealing with the efficient allocation of scarce resources at congested airports. *Transportation Research Part C*, 21:244–256.