FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Heuristische Optimierungsverfahren für die Koordinierung von Flughafenslots

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Simeon J. Kuran
Matrikelnummer 0928921

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao. Univ. Prof. Dipl.-Ing. Dr. techn. Günther Raidl
Mitwirkung: Dr. Andreas Chwatal

Wien, 25. Februar 2020

Simeon J. Kuran	Günther Raidl

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Heuristic optimization methods for seasonal airport slot allocation

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computational Intelligence

by

Simeon J. Kuran
Registration Number 0928921

to the Faculty of Informatics

at the TU Wien

Advisor: Ao. Univ. Prof. Dipl.-Ing. Dr. techn. Günther Raidl
Assistance: Dr. Andreas Chwatal

Vienna, 25th February, 2020

Simeon J. Kuran	Günther Raidl

Erklärung zur Verfassung der Arbeit

Simeon J. Kuran
Brüder-Pez Str. 4
3390 Melk

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 25. Februar 2020

Simeon J. Kuran

Danksagung

Zunächst und zumeist möchte ich große Dankbarkeit gegenüber meinem Betreuer Prof. Raidl ausdrücken. Er hat mich die ganze Zeit über unterstützt und es mir ermöglicht, an diesem Thema zu arbeiten. Seine Ratschläge und Rückmeldungen waren überaus hilfreich.

In gleicher Weise möchte ich mich bei Andreas Chwatal bedanken. Er hat mich durchgehend unsterstützt und mir geholfen aufkommende Schwierigkeiten zu meistern. Viele Ideen entstanden im Zuge unzähliger wertvoller Diskussionen. Ich bin ihm sehr dankbar für die viele Zeit und die großen Aufwände, die er in die Betreuung der vorliegenden Arbeit gesteckt hat. Es war immer eine große Freude und ich verdanke ihm sehr viel.

Des Weiteren gilt meine Dankbarkeit ebenso der Schedule Coordination Austria. Die zahlreichen Ideen, Rücksprachen und die großartige Unterstützung, die für diese Arbeit geleistet wurde, erfüllt mich mit großer Freude. Bei dieser Gelegenheit möchte ich mich bei Wolfgang Gallistl, dem Geschäftsführer der Schedule Coordination Austria bedanken. Darüber hinaus möchte ich Harald Steinmetz meinen größten Dank aussprechen. Er hat mir tiefe Einblicke in die betriebliche Praxis ermöglicht und mein Verständnis für die vielen Restriktionen und Paramater erweitert. Neben den vielen Ratschlägen und Tipps wurden auch alle Testdaten und Referenzlösungen von der Schedule Coordination Austria zur Verfügung gestellt.

Außerdem möchte ich mich ganz herzlich bei meinem Kollegen und langjährigen Freund, Matthias Horn, bedanken. Wir haben viele Jahre lang sehr eng zusammengearbeitet, sowohl bei zahlreichen Aufgaben und Projekten im Laufe unseres Studiums an der Technischen Universität, als auch innerhalb der Firma. In vielen Gesprächen konnte er mir komplizierte Zusammenhänge näher bringen und erklären.

Überdies möchte ich auch meiner Familie überaus große Dankbarkeit aussprechen. Insbesondere meine Eltern, Renate und Wolfgang Kuran, haben mir den Weg bereitet, diese Arbeit erfolgreich abschließen zu können. Sie haben mich immerzu ermutigt und mich umsorgt.

Zuletzt möchte ich mich auch bei Katja Kobler bedanken, die mir immer ein offenes Ohr geschenkt hat, wenn ich beispielsweise im Schlaf von dieser Arbeit gesprochen habe.

Diese Liste ist weder klimaktisch, noch bathetisch und sicherlich nicht vollständig.

Acknowledgements

First and foremost, I would like to express deep gratitude to my advisor Prof. Raidl. He supported me and my work all the time throughout. He gave me the opportunity to work on this topic and guided me from beginning to end. His advice and feedback was very helpful.

I am equally thankful to my co-advisor, Andreas Chwatal. He continually assisted me and helped me to solve arising difficulties. Furthermore, many ideas grew out of fruitful discussions with him. I am very grateful for all the time and efforts he invested into this work. It was always a pleasure and I own him a lot.

Additionally, I am also very delighted and filled with gratitude for all the inputs, as well as feedback and support provided by Schedule Coordination Austria. On this occasion, I would like to thank Wolfgang Gallistl, the CEO of Schedule Coordination Austria. Furthermore, I would also like to express immense gratitude to Harald Steinmetz. He gave me deep insights into operational practice and helped me to strenghten my understanding of lots of constraints and parameters. Besides advice and important hints, Schedule Coordination Austria provided all test instances and reference solutions.

I also want to thank Matthias Horn, my colleague both at Technical University and in the company. We worked closely together for many years. He helped me a lot and many discussions with him helped me to clarify difficult matters.

Moreover, I would like to express my sincere gratitude to my family. In particular, my parents Renate and Wolfgang Kuran paved the way for successful completion of this work. They always encouraged me and cared for me at all times.

Last, I also want to thank Katja Kobler, who always had a sympathetic ear when I talked about this work during my sleep.

Please note, that this list is neither climactic, nor bathetic and certainly not exhaustive.

Kurzfassung

Die vorliegende Arbeit beschäftigt sich mit der *Erstkoordinierung* von *Flughafenslots*. In Europa unterliegt es der Verantwortung des jeweiligen Flughafenkoordinators zu Beginn einer Saison einen initialen Flugplan zu erstellen. Dies ist eine komplexe Aufgabe und bietet hohes Potential für Optimierungsverfahren. Der Fokus dieser Arbeit liegt in der vollständig automatisierten Erstellung eines initialen Flugplans anhand heuristischer Algorithmen.

Diese Arbeit wurde in engem Kontakt mit der *Schedule Coordination Austria* entwickelt und ein großer Schwerpunkt liegt daher in der praktischen Anwendbarkeit. Aufgrund der großen Menge an Flugdaten wurde ein heuristischer Ansatz gewählt. Erstmals wird eine Konstruktionsheuristik vorgestellt, die die Koordinierung von Flughafenslots in relativ kurzer Laufzeit ermöglicht. Zusätzlich werden heuristische Verbesserungsmethoden vorgestellt, um die Ergebnisse weiter zu optimieren.

Die Erstkoordinierung basiert auf initialen Anfragen der Fluglinien für Ankunfts- und Abflugslots an den jeweiligen Flughäfen für die nächste Saison. Im Allgemeinen ist es das Ziel, so viele dieser Anfragen wie möglich zu bestätigen und so wenig wie möglich von der angefragten Zeit abzuweichen. Allerdings unterliegt die Koordinierung zahlreichen Beschränkungen. Zum einen müssen die *IATA Vorgaben*, sowie europäische Bestimmungen, eingehalten werden. Dies beinhaltet under anderem Bestandsrechte (sog. "Großvaterrechte“), sowie Prioritätsregeln. Zum anderen unterliegen die Ressourcen des Flughafens für gewöhnlich zahlreichen Kapazitätsbeschränkungen. In dieser Arbeit wird ein umfangreiches Konzept mit vielen Konfigurationsmöglichkeiten vorgestellt, um Pisten-, Passagier- und Vorfeldbeschränkungen einzuhalten. Des Weiteren muss eine bestimmte *Bodenzeit* zwischen Ankunft und Abflug eingehalten werden.

Überdies erfolgt die Anfrage von Flughafenslots in *Serien*, bestehend aus mehreren Anfragen gleichartiger Flüge im Verlauf einer Saison. Die Slots einer solchen Serie sollten möglichst einheitlich zugewiesen werden. Das entspricht einem weiteren Optimierungsziel. Die Optimierung besteht aus zwei wesentlichen Komponenten: einer Konstruktionsheuristik und darauf folgende Verbesserungsmethoden. Mit dem Konstruktionsalgorithmus wird die Anzahl der bestätigten Anfragen maximiert. Die Verbesserungsmethoden dagegen approximieren die Pareto Front des mehrdimensionalen Optimerungsproblems. Auf diese Weise erzeugt der Algorithmus mehrere Pareto-effiziente Lösungen mit unterschiedlicher

Zeitabweichung und Fragmentierung, einem Maß für die einheitliche Zuweisung von Serien.

Die Algorithmen werden anhand von operativen Daten des Wiener Flughafens sowohl in Bezug auf die Zeitabweichung zur angefragten Wunschzeit, als auch in Bezug auf die Fragmentierung evaluiert und getestet. Außerdem werden die Ergebnisse der Algorithmen mit Daten aus der angewandten Praxis der letzten Saisonen verglichen.

Die Ergebnisse sind hinsichtlich der betrachteten Optimierungsziele mit den manuell erzeugten Flugplänen vergleichbar, bzw. in manchen Situationen sogar besser. Somit können die Algorithmen die Erstkoordinierung von Flughafenslots mit hoher Flexibilität unterstützen und ermöglichen es, den manuellen Aufwand bei der Erstellung eines initialen Flugplans zu reduzieren.

Abstract

This work deals with long-term *airport slot allocation*. In Europe, local coordination authorities are responsible to create an initial flight schedule in advance of a season. This is a sophisticated problem and bears great potential for optimization methods. The focus of this work is to create an initial flight schedule in a fully automated way by heuristic algorithms.

This thesis is developed with high emphasis on practical applicability. It was carried out in close contact with *Schedule Coordination Austria*. Due to the high amount of flight data, a heuristic approach is taken. For the first time, a construction heuristic is proposed to solve the airport slot allocation problem within relatively short running times. Additionally, heuristic improvement methods are presented to further optimize the results.

The coordination process is based on air carriers requesting arrival and departure slots for certain airports for the upcoming season. In general, the aim is to confirm as many such submissions as close as possible to the initially requested times. However, the airport slot allocation process is restricted by several respects. For once, the *IATA guidelines* and European regulations must be met. This involes inter alia compliance to the *grandfather rights* and consideration of priority rules. For another, the resources of the airport are usually constrained by several capacity limitations. Within this thesis an extensive framework is introduced, to respect *runway limitations* as well as *passenger* and *apron* restrictions in a highly configurable way. Furthermore, to allow for refueling and cleaning, interdependencies between arrivals and departures must be taken into account. In particular, a certain *ground time* must be met.

Moreover, the requests of the initial submission are treated as *series* comprising the requests of similar flights over the course of the season. The slots for such series should be assigned as uniformly as possible, which is a further optimization objective. The optimization framework consists of two major components: a construction algorithm and a subsequent improvement process. Whereas the construction algorithm attempts to maximize the number of confirmed requests, the improvement step approximates the pareto frontier of the multi-objective problem. Thus, the algorithm yields multiple pareto efficient solutions with different time deviations and fragmentations, i. e. the degrees of uniformity of the assigned series of slots.

The developed algorithms are evaluated and benchmarked by real world data of the Vienna airport with regard to both objectives, low time deviation regarding the initially requested times as well as good fragmentation. Furthermore, the computational results are compared to applied practice of historic seasons.

Regarding the considered objective function the results are comparable, and in some situations even better than the manually obtained schedules. This allows to solve the assignment problem of the initial submission with higher flexibility and less manual effort.

Contents

CHAPTER 1

Introduction

This thesis deals with algorithmic methods for a complex optimization problem arising in the aviation sector. Particularly, the problem of creating seasonal flight schedules for major European airports is tackled by heuristic optimization algorithms.

The motiviation for this work is presented in detail in Section 1.1 of this Chapter. Next, we explain the aim of this work in Section 1.2, followed by the contribution in Section 1.3. In the last Section 1.4 an outline to the following chapters of this work is given.

1.1 Motivation

In the last years the annual growth of global air transport stabilized between 5 and 10 percent (see [IAT19a]). Possible causes for the steady increase are higher living standards, rise of low-cost airlines and growth of air cargo. The increasing demand inevitably leads to overcrowded airports. Several (European) airports already suffer from congestion, which causes very high expenses for airlines and aircraft operators. According to latest forecasts the situation will further escalate in the next years, since the amount of passengers is expected to grow to approximately 16 million flights in Europe in 2040 (see [EUR18]). This corresponds to a growth of 53% in respect to 2017.

The increasing air traffic demand already exceeds capacity limitations of many airports. For this reason efficient utilization of airport resources is essential and will become even more important in the following years. In order to maximize utilization of available resources airports are *coordinated.* At the time of writing, over 170 aiports all over the world are (Level 3) coordinated airports.

Coordination is a process to ensure fair, neutral and transparent assignment of airport capacities to air carriers. It involves inter alia capacity declarations, initial slot allocation, a worldwide scheduling conference and slot monitoring. All steps are guided by mandatory worldwide regulations (see [IAT19b]).

In terms of optimization the most important step of coordination is the *initial slot allocation*. This is a preliminary step in creating a flight schedule. It is usually done by a local, independent coordination authority (in Austria it is the responsibility of *Schedule Coordination Austria*). The coordinator constructs an initial flight schedule for the whole season (summer or winter term) based upon requests of the air carriers. In a further step, this initial draft schedule will be discussed and adapted during a worldwide slot conference.

The central element of the initial schedule creation is the *airport slot*. An airport slot refers to a certain day and time and permits an air carrier to arrive at or depart from a coordinated airport. Before the start of a season all air carriers *request* their required airport slots. Then the coordinator assigns such slots to the air carriers for a whole season in advance. During the season arrivals or departures without a previously assigned airport slot are not permitted.

However, due to several capacity restrictions it is not possible to assign all slots (i. e. times) as requested by the air carriers. First and foremost, the number of runway movements is restricted within certain time intervals. Further limitations exist for the number of passengers in the terminal and aircraft parking positions on the APRON. In order to avoid capacity exceedance the coordinator might either assign an alternative time slot, or if not possible, reject the request altogheter.

Besides, during the initial slot allocation several worldwide regulations must be obeyed. Firstly, the whole process must be neutral and transparent. Several priorities, as specified in the slot guidelines (see [IAT19b]) must be respected. Moreover, the airport slot assignment process in Europe is largely based on the *use-it-or-lose-it-rule* also called *grandfather rights*. Hence, air carriers that used their airport slots for at least 80% of the allocated time in the previous season, have the right to further use the same airport slots in the next equivalent season. Apart from that, airport slots are generally requested as *series of slots*. Thus, several requests belong together and should preferably be assigned to the same airport slots if possible.

The key focus of this work is the initial schedule creation. Meeting all requirements is quite sophisticated and complex. Because of the aforementioned air traffic increase it gets harder from season to season. Thus, the initial slot allocation is non-trivial and involves lots of efforts. Although this step bears a great potential for optimization methods, it is still widely done manually. In this work we propose heuristic optimization algorithms capable of creating such initial schedules fully automatically.

1.2 Aim of this work

This work is based on previous work of *Destion - IT Consulting & Software Solutions GmbH*, an Austrian operations research company of which I am a 10 % shareholder. Prior to this thesis, an algorithmic prototype for the considered problem existed, but, however, with many shortcomings. The main contribution of this work is a newly developed, refined algorithmic framework with many new functionalities and improved runtime performance. With this framework it is for the first time possible to algorithmically create and improve feasible solutions for all coordinated Austrian airports. The schedules are created in a fully automated way within relatively short running times even for large amounts of data. All data sets have been provided by *Schedule Coordination Austria*.

Primary goal of the optimization strategy is to fullfill the requested times of the air carriers as close as possible without exceeding capacity limitations. However, further optimization goals are to ensure fair assignments (according to the IATA guidelines, see [IAT19b]), and moreover homogenous assignments. Airports typically need to assign particular series of requests to homogeneous times. We call this aspect *fragmentation* and it has been addressed by the first time in [ACK15].

1.3 Contribution

Optimization for airport slot allocation is a relatively new research topic. Recently some exact models and formulations have been proposed. However, because of the complexity these previous approaches are limited to small or medium sized airports at best. For the very first time, we propose a software solution to optimize airport slot allocation for large airports with 200.000 requests and more per season. Furthermore, we introduce the concept of fragmentation which is essential for practical applications.

1.4 Outline of the Thesis

Chapter 1 presents a motivation for the airport slot allocation problem. The growing demand in air traffic calls for optimization methods to minimize congestion.

Chapter 2 gives a brief overview about previous work dealing with airport slot allocation. Although the work in long-term optimization for airport slot allocation is limited, several exact methods have been proposed. According to the literature the results are quite promising for small and medium sized airports. However, due to the complicated constraints and the complexity of the problem it is not clear whether exact methods can be applied to large airports with reasonable running times. We claim that further increasing the test instances sooner or later requires a change from exact methods to heuristic optimization methods.

In Chapter 3 the worldwide slot guidelines as specified in [IAT19b] are introduced. Details and concepts such as the priority model, the grandfather rights and series of slots are explained. Furthermore, a formal definition of the airport slot allocation problem follows. Limiting constraints, problem specific issues and parametrization details are also shown in detail.

Next, the theoretical background of heuristic optimization methods is presented in Chapter 4. The concepts of solution-*construction* and *improvement* methods as well as *pareto-frontiers* for multiple objective optimization methods are discussed.

Chapter 5 deals with the algorithms developed in this work. Both, the construction heuristic as well as the improvement methods are presented.

The problem instances used for evaluation of the algorithms are presented together with the experimental results in Chapter 6. All test data is kindly provided by Schedule Coordination Austria. The outcome of the algorithms are shown and discussed in respect to several criteria such as running time, time deviation, fragmentation and amount of confirmed requests. Furthermore the results are compared to reference solutions provided by Schedule Coordination Austria.

Last, but not least, the most important results are summarized in Chapter 7, where also the final conclusions are drawn.

CHAPTER 2

Related Work

From an economical point of view, demand and capacity analyzation of airport operations is of high interest. Many evaluations have been carried out (e. g. see [EUR18] among many others). A high discrepancy between demand and available resources causes congestion at several airports all over the world. Hence, efficient airport utilization is extremely important to save costs and ensure on-going operation.

To overcome the challenges arising from insufficient capacities with respect to the growing demand different approaches have been proposed. Several research topics can be distinguished. On the one hand, *short-term scheduling* (also called tactical scheduling) deals with dynamic air traffic optimization on the basis of one or several days. Here the *ground holding problem* (GHP) as presented by [AOR93] gained quite some popularity. Aircraft delays lead to high costs. However, delaying an aircraft during its flight results in much higher costs than delaying an aircraft before takeoff. So, the question is when to delay which aircraft. In some situations it is wise to keep an aircraft longer on the ground in order to minimize overall delay costs. The goal is to have a real time decision support system to decide when to delay which aircraft.

However, in order to reduce overall costs it is crucial to consider scheduling optimization early on. *Long-term scheduling* (often called strategic scheduling) intends to balance workload right away from the very first planning steps. It deals with the initial schedule creation and targets a much longer period of time. Usually several months up to a whole scheduling season are considered.

Several strategies concerning long-term scheduling have been investigated. A major research topic are market-driven slot assignment approaches, where assignments are not strictly based on regulations. Among those, congestion based pricing is particularly important. Several different regulatory models have been proposed. The common idea is that air carriers have to pay tolls dependent on the congestion impact (see for example [Dan95], [Bru02], [PV04], [CL19]). Further approaches are slot trading and auction based methods (see for example [Bru09], [Ver10], [BZ10]). But, in Europe congestion pricing as well as auction based methods are currently not applicable at all because of European regulations (i. e. grandfather rights). Those strategies may be of concern in the USA only.

Despite the economic based debates, further contributions can be found in the field of operations research. However, there has been limited work yet towards long-term optimization for airport slot allocation. The first contribution in this topic dates back to 2007. First proposals to model the initial schedule creation process with strategic optimization in mind can be found in [Koe07]. The author suggests a heuristic iterative approach to minimize the time deviation between requested and granted requests. This very first attempt lacks some crucial coordination parameters though. E. g. slot priority classes and the concept of slot series are not considered at all.

In the PhD-thesis [Zho12] the author also deal with a long-term slot assignment problem. The main focus of the work lies on fairness considerations between individual air carriers. In order to achieve a balanced time deviation over all air carriers a multiple objective integer programming model is proposed. But the introduced model is very restricted in terms of several aspects. Firstly, only arrival slots are taken into account. It is assumed, that a paired departure slot is always available. However, the interdependencies between arrival and departure slots play an important role and cannot be ignored in practice for most airports. To allow for maintainance, refueling, cleaning, etc. aircrafts need to stay at ground for a certain time interval (also called turnaround time). Dropping the turnaround restrictions simplifies the scheduling problem a lot.

Moreover, further differences concern the period of time under examination. On the one hand, a slot is viewed as a resource that can be leased for one to ten years by an air carrier. However, according to the Worldwide Slot Guidelines [IAT19b] the coordination process takes place twice a year. (The airport under consideration by the authors of [Zho12] is LaGuardia Airport in the USA. Possibly the regulatory requirements are quite different there.) On the other hand, the scheduling problem is solved exemplary for a single day of operation only. This is a big simplification, since the whole scheduling season covers half a year with different requests, changing peak times and most importantly, many interdependencies between scheduling days.

Furthermore, many requests must be considered as a series of slots which belong together. For example, an air carrier might request a certain time slot (e. g. 9:30) at the same day of the week (e. g. Monday) over the whole scheduling season. Depending on the priority and some regulatory rules it might be mandatory to confirm such requests.

It is worth to mention that the author of [Zho12] also discusses a heuristic strategy, even though it is quite limited in applicability to the concern of our work. Basically the proposal is to assign the time slots in a round-robin fashion to different air carriers. This might or might not help to meet certain fairness criteria, but in any case, it is not compatible with priority classes and the grandfather rights.

In [ZSM12] the authors also present an integer programming model to solve the long-term slot assignment problem and hence create an initial flight schedule. This contribution is already much more related to current practice in Europe. Real data of three regional Greek airports for the summer season of 2009 is provided by the Greek slot coordination authority. For the first time the whole scheduling season is considered. Furthermore, the authors already take different priority classes into account, though only three different classes are considered. In addition, series of slots are used instead of single requests. However, several restrictions still occur. For one thing, fragmentation is not addressed at all, because series are always assigned homogenously. But, at least at big airports this is not always desirable. In some situations, splitting a series and thus assigning the corresponding individual requests to different times of the day might be more attractive than displacing or rejecting the whole series. For another, it is assumed, that each series can be allocated at any time of the day. But in practice this strongly depends on the type of request, e. g. some requests might be very restricted and can not be assigned to an arbitrary time of the day.

Yet another integer programming model has been proposed by the authors of [RJA$^+$18]. In this work the priority model used for solving the slot assignment problem is slightly improved. Historic change requests are now treated more accurately. For experimental evaluation the approach has been applied to the Portuguese airports of Madeira and Porto. The latter airport operates approximately 85.000 flights per year. Notably, for the first time problem instances of this size can be solved in reasonable running time. Furthermore an extension to this work is announced in [RJA]. By the use of a construction heuristic and an improvement method the proposed integer programming model is applied to the airport of Lisbon which operates 200.000 flights per year. With a running time of 30 minutes up to several hours a suitable solution can be found. But still fragmentation is not a matter at all. Furthermore it is not clear at all, whether the model can be applied to even larger problem instances due to the expected high running time.

For the sake of completeness, it is also worth mentioning that some authors promote a radically different strategy. Currently the initial flight schedule is created on a per airport basis, as specified by the worldwide slot guidelines [IAT19b]. However, the efficiency of this procedure is debatable. Some authors promote to revise the flight schedule creation process altogether and consider aiport slot allocation on a network basis (see for example [CLN14], [PBCP17], [ZMA17], [Ben18] and [RJAO19]). But the biggest problem here is the lack of realistic problem instances, because the initial requests of the air carriers are usually highly confidential. Moreover, such a big change might be hard to put into practice, since it depends on international political decisions.

CHAPTER 3

Airport Slot Allocation

This Chapter deals with the details of the initial schedule creation and presents a formalization, as well as a model for the considered airport slot allocation problem. Section 3.1 gives an overview of the whole coordinaton process. The denotation of time is explained in Section 3.2. Next, Section 3.3 presents the initial submissions of air carriers. The concept of slot series is worked out in Section 3.4. Then, a priority model is introduced in Section 3.5. The possible actions of coordinators are shown in Section 3.6. Several resource limitations such as runway constraints, passenger constraints and APRON constraints are tackled in Section 3.7. Constraints regarding the minimal time on ground are formulated in Section 3.8. The concept of fragmentation and different ways of quantification are proposed in Section 3.9. Last, we discuss the objective function and hence the quantification of scheduling solutions in Section 3.10.

3.1 Coordination

To cope with the high air traffic the *International Air Transport Association* (IATA) published worldwide slot guidelines (see [IAT19b]). These also apply to Europe with slight adaptions and extensions published by the European Airport Coordinators Association (EUACA, see [Par93], [Par04]).

These regulations define a process of *coordination* to ensure a fair, neutral and transparent airport utilization at all times. Depending on demand and capacity three different levels can be distinguished.

- **Level 1 airports:** The capacity of the airport is sufficient to manage the demand at all times.
- **Level 2 airports:** There is potential for congestion at certain times which can be resolved by schedule adjustments.

- **Level 3 airports:** The demand exceeds capacity limitations and expansion of airport infrastructure is not possible in short term.

Depending on the level different rules and regulations apply. Congested airports are in general classified as level 3 airports and thus need to be coordinated. As a result air carriers have to allocate a slot by a coordinator before operating at the airport. In total, more than 170 airports are currently level 3 coordinated airports.

Several distinct procedures are involved when creating a flight schedule for a level 3 coordinated airport. A detailed explanation is given by [IAT19b]. Briefly summarized, the key points are

- capacity declarations of the airport,
- initial submissions of requested airport slots by the air carriers,
- **initial coordination** by the coordinator,
- worldwide slot conference (secondary slot trading) and
- slot monitoring.

Firstly, the airport is responsible to submit the amount of available resources. The available airport capacity is limited by several complex constraints which are further discussed in Sections 3.7 and 3.8.

Afterwards, the air carriers can request the desired airport slots. Different aspects such as historic status, request time, etc. play an important role. In particular, depending on the historic status code several actions are possible, which might include to deviate from the requested time. The different parameters of the requests are explained in Section 3.3.

The initial coordination is a central part of the coordination process and includes to solve a challenging scheduling problem. This bears a great potential to use computational optimization techniques to improve current operational practice significantly.

The outcome of this process is then discussed and further adapted at the worldwide slot conference. Although the slot allocation is done at each airport separately, the flight schedule of a single air carrier obviously causes interdependencies between the airport of departure and the airport of arrival. To account for those dependencies primary slot allocation is subsequently followed by secondary slot trading in which the air carriers can trade slots one-by-one to meet their schedules with minimal time discrepancies. Additionally, air carriers must return slots which they do not intend to operate.

Furthermore, the coordination authority is responsible to monitor the usage of the allocated airport slots in order to prematurely determine the historic status codes and accompanying grand father rights for the following season.

3.2 Denotation of Time

The initial schedule contains data for the period of half a year, usually called a *season*. The winter season commences on the last sunday in October, the summer season on the last sunday in March.

An optimization approach needs to tackle the whole season at once. Because of the concept of series of slots (further discussed in Section 3.4) individual days depend on each other and hence creating separate schedules for individual days is not possible.

So, a season consists of approximately 200 scheduling days. We denote the set of days as D. Now, every scheduling day is discretized into intervals of five minutes. Hence, we define the set $T = \{1, \dots, 288\}$ to describe the discretized times $t \in T$ of a scheduling day $d \in D$. Thus, by this denotation, e. g. $t = 150$, $d = 1$ stands for the time 12:30 at the first scheduling day of the season.

3.3 Airport Slots

An *airport slot* is the permission to use airport infrastructure at a certain date and time in order to arrive at or depart from a level 3 coordinated airport (see [IAT19b]). Thus, such a slot is a prerequisite for air carriers to operate on coordinated airports. Moreover, in terms of optimization it is a quantification of the limited airport resources, i. e. the number of possible movements (arrivals or departures) at a certain date and time due to capacities and regulations.

Before the coordinator creates an initial schedule the air carriers need to submit their initial requests. Let R denote the set of initially submitted requests, then a single request $r \in R$ has the following attributes

- **movement type:** either arrival or departure,
- **day of operation:** number decoding the day of the week (1..Monday,. . ., 7..Sunday),
- **request date and time:** $d \in D, t \in T$ referring to the requested airport slot,
- **historic date and time:** $d \in D, t \in T$ referring to the historic airport slot,
- **historic status code:** describes historic precedence privileges,
- **operator code:** identifies the air carrier,
- **flight number:** airline designator (two letters) and a number (one to four digits),
- **service type:** a code to identify the aircraft,
- **seat count:** maximal number of seats and
- **load factor:** an estimation of the fill grade.

3.4 Series of Slots

However, single requests are only of mere interest. As specified by [IAT19b] requests are usually submitted in bundles called *series of slots*. It is preferable to assign all requests of a series to the same time of day. Hence, different scheduling days might depend on each other if they are part of the same series.

In a first step we consider only series with requests on the same day of the week (same day of operation). Such a series must contain at least five requests, otherwise it is not considered at all in the initial schedule creation. Instead, such individual requests are usually set after the worldwide slot conference (so-called *ad hoc requests*). Furthermore, all requests of a series must have the same movement type (be it arrival or departure), the same operator code and either the same flight number or directly consecutive flight numbers. In addition, the request times may not differ by more than 30 minutes.

Hence we can define an equivalence relation $r_1 \sim_{S_r} r_2$, such that $(r_1, r_2) \in S_r$ if and only if the requests $r_1, r_2 \in R$ are scheduled on the same day of the week, have the same movement type, same operator code and equal or directly consecutive flight numbers. Furthermore, the request times of r_1 and r_2 may not differ by more than 30 minutes.

Then, S_r induces equivalence classes $[r]_{S_r}$, such that

$$[r]_{S_r} = \{r' \in R \mid r \sim_{S_r} r'\} . \tag{3.1}$$

Let $\mathcal{P}(R)$ be the powerset of R. Thus, we get a partition $P_{S_r} \subseteq \mathcal{P}(R)$, such that

$$P_{S_r} = \left\{[r]_{S_r} \mid r \in R\right\} . \tag{3.2}$$

Note, that

$$\forall A \in P_{S_r} : A \neq \emptyset \tag{3.3a}$$
$$\forall A, B \in P_{S_r} : A \neq B \Rightarrow A \cap B = \emptyset \tag{3.3b}$$
$$\bigcup_{A \in P_{S_r}} A = R \tag{3.3c}$$

must hold. That is P_{S_r} must be non-empty (3.3a), pairwise disjoint (3.3b) and the union of all subsets equals to R (3.3c).

However, in operational practice series of slots often span over different days of the week. Hence, we also define *multiday series* to group series with different days of the week together. Let s_1 and s_2 be two series. So, we define another equivalence relation $s_1 \sim_{M_S} s_2$, such that $(s_1, s_2) \in M_S$ holds if and only if the series are have the same movement type, same operator code and equal or directly consecutive flight numbers. Thus, in contrast to S_r the day of the week plays no role for the relation M_S.

But, still another restriction needs to be met. Consider a series a containing only a few requests and another series b with requests on e. g. every Monday of the season. It is not desirable to group such series together, because series b might unnecessarily restrict series a too much (in general, assigning a series with a few requests is easier than assigning a series with many requests). So, for all $(s_1, s_2) \in M_S$ we require $|s_1| \geq 0.7 \cdot |s_2|$.

Then, the relation M_S induces equivalence classes $[s]$, such that

$$[s]_{M_S} = \{s' \in P_{S_r} \mid s \sim_{M_S} s'\} . \tag{3.4}$$

Next, we denote the powerset of P_{S_r} as $\mathcal{P}(P_{S_r})$. We get another partition $P_{M_S} \subseteq \mathcal{P}(P_{S_r})$, such that

$$P_{M_S} = \left\{[s]_{M_S} \mid s \in P_{S_r}\right\} . \tag{3.5}$$

to account for multiday series.

Now, to map an individual request r to its series, we define a relation $\text{series}_{S_r}(r) : R \to \mathcal{P}(R)$ as

$$\text{series}_{S_r}(r) = [r]_{S_r} . \tag{3.6}$$

In the same manner, we define a relation $\text{series}_{M_S}(s) : \mathcal{P}(R) \to \mathcal{P}(P_{S_r})$ to map a series to its multiday series,

$$\text{series}_{M_S}(s) = [s]_{M_S} . \tag{3.7}$$

Note, that we get the multiday series s for a request r by composition

$$s = \text{series}_{M_S}(\text{series}_{S_r}(r)). \tag{3.8}$$

3.5 IATA Priority Model

According to the IATA rules [IAT19b] the initially submitted requests must be categorized into several priority classes. Ultimate priority must be given to slots with *historic precedence*. That is, air carriers can keep a series of slots of the preceding equivalent season as long as the series were operated at least 80% of the allocated time period. This is known as *use-it-or-lose-it rule* (also known as the so-called *grandfather rights*). The historic status code of such series is denoted as F.

Next, priority must be given to historic *change requests*. These include extensions and adaptions (change of time, change of aircraft type, etc.) to historic series.

The remaining slots form a so-called slot pool. 50% of the slots in the pool must be allocated to new entrants, that is requests with status code N. However, extensions of existing operations to operate on a year round basis should be prioritized over new requests within each category. Furthermore, some additional aspects such as fairness criteria, curfews, frequency of operation, local guidelines and more should be taken into account (see [IAT19b]).

However, to conform with current practice and allow for flexibility we implement a very fine-grained and highly configurable priority model. For this purpose, we define a priority for every request $r \in R$ depending on the historic status code, the service type, the number of seats and the number of movements in the multiday series to which r belongs to, that is $\text{series}_{M_S}(\text{series}_{S_r}(r))$.

Firstly, depending on the historic status code h in the set of all possible historic status codes H, a priority class is defined as $\text{pr}_{\text{hist}}(h) : H \to [1, 100]$, such that the highest priority is equal to 1 and the lowest priority has a value of 100. The actual priority classes used in this work together with the corresponding priority values are shown in appendixA.

Furthermore, we assign to each priority class a list of possible times (relative to the requested time). For this purpose, we define a mapping $\text{allowed\_times}(h)$ from a historic status code $h \in H$ to a set of allowed times $T \in \mathcal{P}(T)$. Hence, $\text{allowed\_times}(h) : H \to \mathcal{P}(T)$.

However, note that both the request time and the historic time are always allowed for any request. So, e. g. for historic slots with historic status code F the set of allowed times evaluates to $\text{allowed\_times(F)} = \{\text{request time}\}$ (keep in mind, that for requests with status code F the request time is always equal to the historic time), whereas the allowed times for the priority class with historic status code FI evaluates to $\text{allowed\_times(FI)} = \{\text{rt}, \text{ht}, \text{rt} - 1\text{h}, \text{rt} - 55\text{min}, \ldots, \text{rt} + 55\text{min}, \text{rt} + 1\text{h}\}$ where rt is the request time and ht is the historic time.

Next, the priority of a request shall depend on its service type. Hence, we define a set of service types as

$$S = \{J, C, G, F, H, A, MP, W, I, E, X, O, N, D, U, K, T, Y, Z\}\,. \tag{3.9}$$

Now, let $s \in S$ be a service type, then we can define a relation $\text{pr}_{\text{svc\_type}}(s) : S \to [-1, \ldots, 1]$ to map a service type to a priority. Note, that this mapping is in general configurable by the coordinator. The values used in this work are shown in appendix A.

Furthermore, let $\text{mvts}(r)$ be the cardinality of the series containing the request r, hence $\text{mvts}(r) = |\ \text{series}_{M_S}(\text{series}_{S_r}(r))\ |$. Then, we can denote the size of the series with the most requests as mvts_{max} and the size of the series with the fewest requests as mvts_{min}. Now, we define a priority factor $\text{pr}_{\text{mvts}}(r)$ depending on the number of movements of a request r

$$\text{pr}_{\text{mvts}}(r) = \frac{\text{mvts}(r) - \text{mvts}_{\text{min}}}{\text{mvts}_{\text{max}} - \text{mvts}_{\text{min}}}. \tag{3.10}$$

In the same manner, we define a priority factor $\text{pr}_{\text{seats}}(r)$ to account for the seats. Let $\text{seats}(r)$ be the seats of a request r, further, $\text{seats}_{\text{max}}$ be the number of seats of the request with the most seats and $\text{seats}_{\text{min}}$ be the number of seats of the request with the fewest seats. Then, we can denote the priority factor $\text{pr}_{\text{seats}}(r)$ as

$$\text{pr}_{\text{seats}}(r) = \frac{\text{seats}(r) - \text{seats}_{\text{min}}}{\text{seats}_{\text{max}} - \text{seats}_{\text{min}}}. \tag{3.11}$$

Let $\text{hist}(r)$ be the historic status code and in the same manner $\text{svc}(r)$ be the service type of a given request r. Then, we can now define the priority $\text{pr}_r(r)$ for a single request $r \in R$ as

$$\begin{aligned}\text{pr}_r(r) = \text{pr}_{\text{hist}}(\text{hist}(r)) &+ \xi\, c_1\ \text{pr}_{\text{svc\_type}}(\text{svc}(r)) \\ &+ \xi\, c_2 \left(\frac{1}{2} - \text{pr}_{\text{mvts}}(r)\right) \\ &+ \xi\, c_3 \left(\frac{1}{2} - \text{pr}_{\text{seats}}(r)\right),\end{aligned} \tag{3.12}$$

where c_1, c_2 and c_3 are weighting coefficients, and ξ is a gain factor. Note, that $c_1 + c_2 + c_3 = 1$ holds. The actual values used in this thesis are shown in appendix A.

3.5.1 Priorities of Slot Series

Furthermore, we define the priority of a series $\text{pr}_{S_r}(s)$ as the average priority of the individual requests. That is,

$$\text{pr}_S(s) = \frac{1}{|s|} \sum_{r \in s} \text{pr}_r(r). \tag{3.13}$$

In the same manner we define the priority of a multiday series $\text{pr}_{M_S}(m)$ as the average priority of the individual series. So, let

$$\text{pr}_{M_s}(m) = \frac{1}{|m|} \sum_{s \in m} \text{pr}_{S_r}(s). \tag{3.14}$$

3.6 Action Codes

To every request $r \in R$ the coordinator can assign one of the action codes $\{K, T, U\}$. Action code K means that the request is confirmed and that the corresponding multiday series $\text{series}_{M_S}(\text{series}_{S_r}(r))$ is eligible for historic precedence in the next saison, whereas action code T indicates a temporary confirmation for the current season only. Denied requests on the other hand obtain the action code U, standing for “unable”.

Now, let $\text{action}(r)$ map every request $r \in R$ to its assigned action code $a \in \{K, T, U\}$. Then, for every request with $\text{action}(r) \in \{K, T\}$ the relation $\text{conf\_time}(r)$ maps the request r to the allocated time slot $t \in T$. In the same manner, let relation $\text{conf\_day}(r)$ denote the day $d \in D$ on which the request r is confirmed. Note, that $\text{conf\_day}(r)$ is usually equal to the request date of r (discrepancies might only occur around midnight).

3.7 Capacity Constraints

The flight schedule is restricted by several capacity limitations. Section 3.7.1 deals with limitations of the runway. At a certain time of the day only a limited amount of arrivals or departures can be accepted. In the same manner also the amount of passengers guided through the terminal might be restricted. Details are explained in Section 3.7.2. In addition, the number of grounded aircrafts may not exceed the available parking positions. This is known as APRON limits and is further discussed in Section 3.7.3.

3.7.1 Runway Limits

In general, the number of movements during a certain time interval needs to be restricted, since airport resources are limited. Such runway limitations can be defined separately for arrivals, departures or total (both). Furthermore, the maximal number of movements might also depend on the time of the day, e. g. in the morning the available resources might be tighter than in the afternoon.

Hence, to support separate limits depending on the time of the day, we define six different time ranges $\mathrm{Tr}_{\mathrm{day}}, \mathrm{Tr}_{\mathrm{evening}}, \ldots, \mathrm{Tr}_{\mathrm{morning}}$ as shown in Table 3.1. Column one shows the designator, columns two and four show the actual times for the winter and summer periods, whereas columns three and five show the same times in minutes.

Table 3.1: Time ranges used to define time dependent limits.

time range	winter period	[min]	summer period	[min]
$\mathrm{Tr}_{\mathrm{day}}$	72 - 239	06:00 - 19:55	60 - 227	05:00 - 18:55
$\mathrm{Tr}_{\mathrm{evening}}$	240 - 251	20:00 - 20:55	228 - 251	19:00 - 20:55
$\mathrm{Tr}_{\mathrm{evening\text{-}shoulder}}$	264 - 269	22:00 - 22:25	252 - 257	21:00 - 21:25
$\mathrm{Tr}_{\mathrm{night}}$	270 - 288, 1 - 53	22:30 - 04:25	258 - 288, 1 - 41	21:30 - 03:25
$\mathrm{Tr}_{\mathrm{morning\text{-}shoulder}}$	54 - 59	04:30 - 04:55	42 - 47	03:30 - 03:55
$\mathrm{Tr}_{\mathrm{morning}}$	60 - 71	05:00 - 05:55	48 - 59	04:00 - 04:55

Furthermore, the limits are defined for a certain time interval δ, e. g. in the interval $\delta = 12$ (60 min) at most x arrivals are allowed. However, in order to ensure an uniform distribution, usually several limits with different time intervals $(\delta_1, \ldots, \delta_n)$ are defined at once, e. g. in the interval $\delta_1 = 12$ (60 min) at most x arrivals are allowed and additionally in the interval $\delta_2 = 2$ (10 min) at most y arrivals are allowed. Note, that the actual values used for evaluation are listed in appendix A.

Let μ describe the movement type in $M = \{\mathrm{arrival}, \mathrm{departure}, \mathrm{total}\}$. Then, for certain time intervals $\delta_1, \ldots, \delta_n$ we define the maximal number of movements as $\mathrm{limit}_{\mathrm{def}}(t, \mu, \delta)$, such that

$$\text{limit}_{\text{def}}(t, \mu, \delta) = \begin{cases} \lambda_{1,\mu,\delta} & t \in \text{Tr}_{\text{day}}, \\ \lambda_{2,\mu,\delta} & t \in \text{Tr}_{\text{evening}}, \\ \vdots & \\ \lambda_{6,\mu,\delta} & t \in \text{Tr}_{\text{morning}}. \end{cases} \tag{3.15}$$

To clarify the considerations and introduce some further details regarding limit transitions, we present a small example for the purpose of illustration only. For this purpose, we consider the three different time ranges $\text{Tr}_{\text{morning-shoulder}}$,$\text{Tr}_{\text{morning}}$ and Tr_{day} as shown in Table 3.1. The runway movements shall be restricted for several time intervals $\delta_1 = 2\,(10\text{ min})$, $\delta_2 = 4\,(20\text{ min})$ and $\delta_3 = 12\,(60\text{ min})$. The limits are separately defined for the movement type $\mu \in M$, that is either for arrivals only, departures only or both (total). The limits are shown exemplarily in Table 3.2. The first column contains the time range, the second column shows the different time intervals δ and the remaining columns show the defined maximal number of movements for arrivals, departures and total ($\lambda_{5,\mu,\delta}$, $\lambda_{6,\mu,\delta}$ and $\lambda_{1,\mu,\delta}$).

Table 3.2: Examplary limits for several time ranges depending on the time interval δ and the movement type μ.

time range	interval	arrivals	departures	total
	δ_1	5	-	-
$\text{Tr}_{\text{morning-shoulder}}$	δ_2	10	-	15
	δ_3	20	20	-
	δ_1	7	12	16
$\text{Tr}_{\text{morning}}$	δ_2	15	20	25
	δ_3	40	-	-
	δ_1	-	-	10
Tr_{day}	δ_2	5	15	-
	δ_3	10	35	-

So, e. g. for the movement type μ_1 = arrival, the interval δ_2 and the time $t \in \text{Tr}_{\text{morning}}$, the term $\text{limit}_{\text{def}}(t, \mu_1, \delta_2)$ evaluates to $\lambda_{6,\mu_1,\delta_2} = 15$ meaning that in any time interval of length 20 minutes no more than 15 arrivals are allowed in the morning (between 05:00 and 05:55 in the winter period).

Next, Figure 3.1 shows a minimalistic flight plan for an arbitrary day of the season. Note, that we only consider a few arrival movements in this example to keep the details as clear as possible. Furthermore, we only focus on the daytimes 04:30 to 06:30. In addition, the limit for the time interval of 20 minutes (δ_2) as defined in Table 3.2 is shown for the movement type μ_1 = arrival.

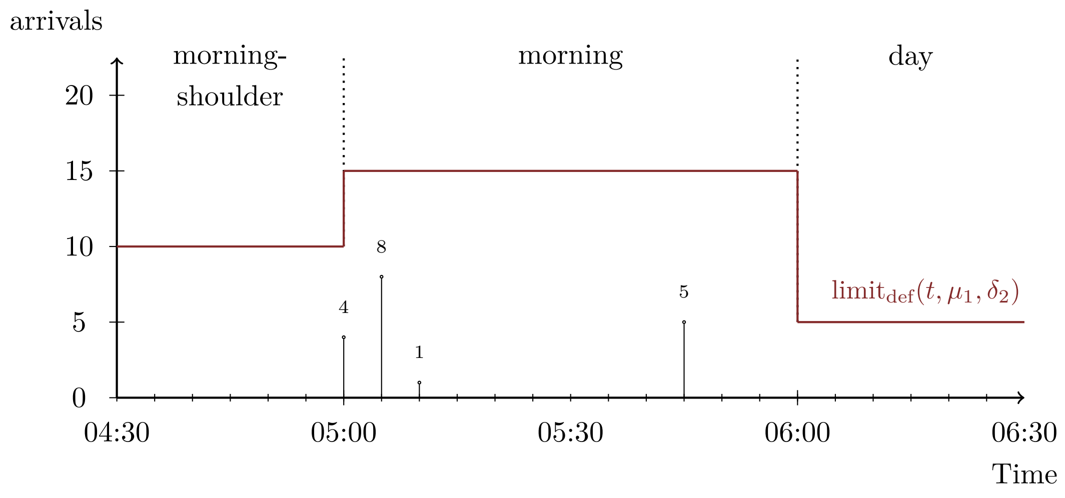


Figure 3.1: Examplary flight plan for an arbitrary day of the season; Defined limit for the movement type arrival and a time interval of 20 minutes.

However, to compare the arrivals against the defined limits, we need to *aggregate* the counts. Aggregation denotes the accumulation of the counts in the corresponding time interval. In this manner, we define the *aggregated counts* depending on the discrete time $t \in T$, the time interval δ and the movement type $\mu \in M$ as

$$\text{agg}(t, \mu, \delta) = \sum_{i=t}^{t+\delta-1} |\{ r \in R \mid \text{conf\_time}(r) = i \wedge (\mu = \text{total} \vee \text{mvt\_type}(r) = \mu) \}|, \tag{3.16}$$

where $\text{mvt\_type}(r)$ denotes the movement type of r. Hence, for a given time t we sum the number of requests with the movement type μ which are confirmed in the interval t to $t + \delta - 1$.

Figure 3.2 shows the aggregated arrivals for the example described above. Note, that the aggregated counts at 05:50 and 05:55 exceed the limit defined by $\text{limit}_{\text{def}}(t, \mu_1, \delta_2)$. This is because of the aggregation which spans over a time period of δ. Hence, for some points in time $t < t_1$ the aggregation also includes counts at $t \geq t_1$ where already another limit is defined. This problem is addressed next.

Propagated Maxima

Suppose, we have two different limits defined, $l_1 = \lambda_{1,\mu,\delta}$ for the time range Tr_{day} and $l_2 = \lambda_{2,\mu,\delta}$ for the time range $\text{Tr}_{\text{evening}}$ complying with the example above. Then special care must be taken at the points of intersection. Since the aggregation is defined over the time interval $(t, t + \delta - 1)$ it spans over both time ranges Tr_{day} and $\text{Tr}_{\text{evening}}$ for certain values of t. But then both of the limits l_1 and l_2 apply.

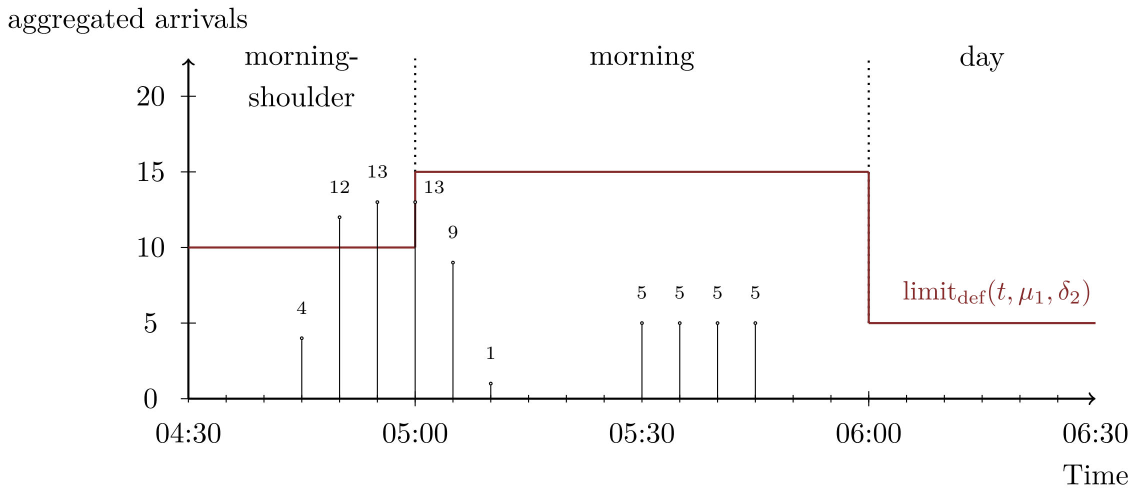


Figure 3.2: Aggregated arrival counts for the example above.

Hence, to compare the aggregated counts directly against the limits, we need to slightly extend the limits and propagate the maxima. Hence, we define $\text{limit}_{\text{prop}}(t, \mu, \delta)$ as

$$\text{limit}_{\text{prop}}(t, \mu, \delta) = \max_{t' \in [t, t+\delta-1]} \text{limit}_{\text{def}}(t', \mu, \delta) \tag{3.17}$$

to take the transitions into account.

Figure 3.3 shows the aggregated arrivals with the propagated maxima limits defined by $\text{limit}_{\text{prop}}(t, \mu_1, \delta_2)$. Now the transitions are adequatly taken into account.

Implicit Limits

The example limits shown in Table 3.2 reveal another important fact. In the time range ($\text{Tr}_{\text{morning}}$) seven arrivals are allowed in the interval δ_1 (10 minutes), whereas 15 arrivals are allowed in the interval δ_2 (20 minutes). However, the limits of interval δ_1 implicitly restrict the interval δ_2. Both, $\text{limit}_{\text{prop}}(t, \mu, \delta_1)$ and $\text{limit}_{\text{prop}}(t, \mu, \delta_2)$ must be satisfied at the same time and hence in the concrete example only 14 arrivals are allowed in the interval of δ_2(20 minutes).

This fact is further illustrated in Figure 3.4. Suppose $\text{limit}_{\text{prop}}(t, \mu, \delta_1)$ and $\text{limit}_{\text{prop}}(t, \mu, \delta_2)$ are given. Then, the implicit limit $\text{limit}_{\text{impl}}(t, \mu, \delta_2)$ results by minimization.

So, we define $\text{limit}_{\text{impl}}(t, \mu, \delta)$, such that

$$\text{limit}_{\text{impl}}(t, \mu, \delta) = \min_{\substack{\delta' < \delta \wedge (\mu' = \mu \vee \\ (\mu = \text{total} \wedge \mu' \in \{\text{arrival}, \text{departure}\}))}} \left(\text{limit}_{\text{prop}}(t, \mu, \delta), \frac{\delta}{\delta'} \text{limit}_{\text{prop}}(t, \mu', \delta')\right). \tag{3.18}$$

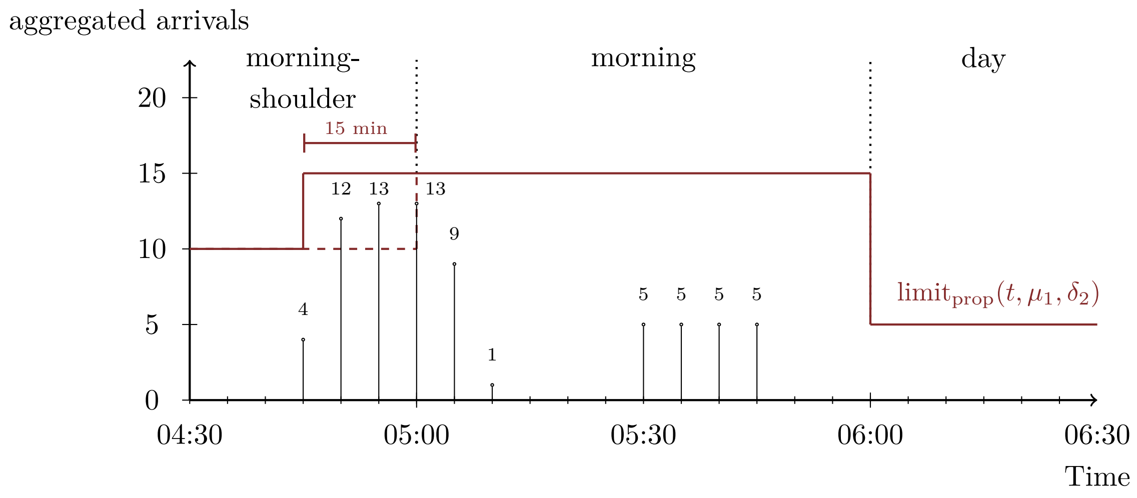


Figure 3.3: Exemplary propagated maxima limits used to respect the limit transitions.

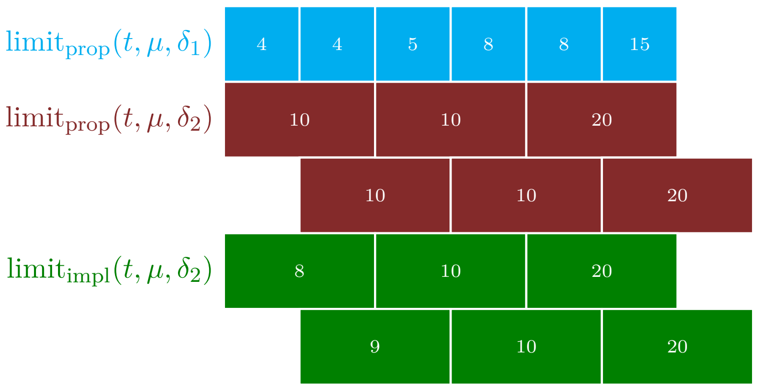

$\text{limit}_{\text{prop}}(t, \mu, \delta_1)$
4
4
5
8
8
15
$\text{limit}_{\text{prop}}(t, \mu, \delta_2)$
10
10
20
10
10
20
$\text{limit}_{\text{impl}}(t, \mu, \delta_2)$
8
10
20
9
10
20

Figure 3.4: Exemplary implicit limits determined by propagated limits $\text{limit}_{\text{prop}}(t, \mu, \delta_1)$ and $\text{limit}_{\text{prop}}(t, \mu, \delta_2)$.

Rolling Counts

Next, in addition to the aggregated counts we define the *rolling counts*, such that

$$\operatorname{roll}(t, \mu, \delta) = \max_{t' \in [t-(\delta-1),\, t]} \operatorname{agg}(t', \mu, \delta). \tag{3.19}$$

Despite being directly related to the aggregated counts, the rolling counts are not necessary for the sole purpose of optimization. However, they are usually used for analyzation by managers and stakeholders, because they represent the actual utilization most appropriately. Hence, they shall not be absent.

Advanced Limits

In the same manner, we define *advanced limits*. Just as the rolling counts they are only used for analyzation and demonstration. In some sense, they can be seen as extension to the implicit limits defined above.

For example, the limit defined for the time interval δ_2 (20 minutes) can be implicitly restricted by the limit defined for 10 minutes as described above, since $10 + 10 = 20$. However, further implicit restrictions might occur when considering several time intervals of different length, i. e. the limit of δ_2 might be even more restricted by the combination of limits for the intervals of 5 and 15 minutes $(5 + 15 = 20)$. The situation further escalates when taking transition points into account. So, the limit for the interval of 20 minutes at time t might be implicitly restricted by the limit for the interval of 10 minutes defined for the time range Tr_{day} plus the limit for the interval of 10 minutes defined for the time range $\mathrm{Tr}_{\text{evening}}$.

Let $C(t, \mu, \delta)$ be a set of all possible combinations of implicit limits for a timespan δ even taking transition points into account. I. e., for $\delta_x = 2$ (10 minutes) and $\delta_y = 1$ (5 minutes) this could be $C(t, \mu, \delta_x) = \{\operatorname{limit}_{\text{prop}}(t, \mu, \delta_x), \operatorname{limit}_{\text{prop}}(t, \mu, \delta_y) + \operatorname{limit}_{\text{prop}}(t+1, \mu, \delta_y)\}$. Then, we define the advanced limits as

$$\operatorname{limit}_{\text{adv}}(t, \mu, \delta) = \min_{c \in C(t,\mu,\delta)} c. \tag{3.20}$$

Note, that the number of combinations $|C(t, \mu, \delta)|$ grows significantly with increasing δ and can get quite high. Figure 3.5 visualizes the situation for a timespan of 30 minutes $(\delta = 6)$. Every path in the tree represents one combination $c \in C(t, \mu, \delta)$.

Night Regulations and Seasonal Limits

Furthermore, some quota constraints such as restrictions regarding nightly operations, noise reduction, etc. might have to be considered. In particular, also some seasonal limits might be defined (e. g. in the morning the number of movements over the whole season might not exceed a defined maximum).

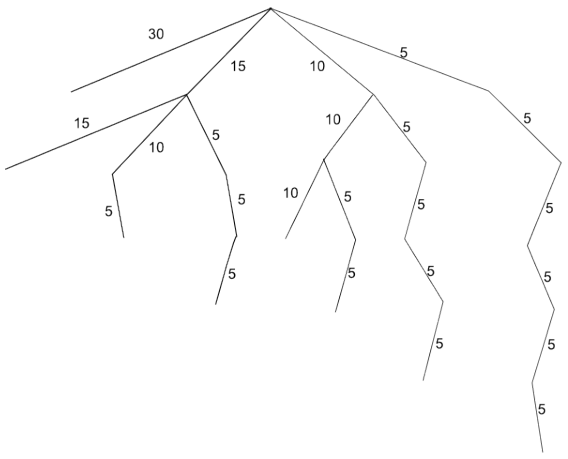


Figure 3.5: All possible combinations $C(t, \mu, \delta)$ for implicit limit with timespan $\delta = 6$ (30 minutes).

Let Tr be a time range $\text{Tr} \in \{\text{Tr}_{\text{day}}, \text{Tr}_{\text{evening}}, \ldots, \text{Tr}_{\text{morning}}\}$. Furthermore, let seasonal(Tr) denote the sum of all confirmed requests in the time range Tr over a whole season. Hence,

$$\text{seasonal}(\text{Tr}) = |\ \{r \mid r \in R \ \wedge\ \text{action}(r) \in \{K, T\} \ \wedge\ \text{conf\_time}(r) \in \text{Tr}\}\ |. \qquad (3.21)$$

So, for example, the number of all confirmed requests of a whole season during the night is given by $\text{seasonal}(\text{Tr}_{\text{night}})$.

Then, a seasonal night quota constraint can be formulated as

$$\text{seasonal}(\text{Tr}_{\text{night}}) \leq \lambda_{\text{night}}, \qquad (3.22)$$

where λ_{night} is the defined maximum number of movements during the night over a whole season. In the case of Vienna, such a seasonal night constraint is active. The actual values are shown in appendix A.

3.7.2 Passenger Limits

Also the number of passengers can be restricted. Let seats(r) be the total number of seats available in the aircraft corresponding to request r. Furthermore, let load(r) be an estimated load factor that approximates the utilization of request r. Thus, we get the estimated number of passengers pax(r) delivered by request r as

$$\text{pax}(r) = \text{seats}(r) \cdot \text{load}(r). \qquad (3.23)$$

Let π^- denote the assumed time for the passengers to stay in the terminal before departure. In the same manner π^+ denotes the assumed time for passengers to stay in the terminal after arrival. Then the number of passengers in the terminal $\text{cnt}_{\text{pax}}(t)$ at a given time $t \in T$ can be defined, such that

$$\mathrm{cnt}_{\mathrm{pax}}(t) = \sum_{\substack{r \in R\,\wedge \\ \mathrm{mvt\_type}(r)=\mathrm{arrival}\,\wedge \\ \mathrm{conf\_time}(r)\in\left[t,t+\pi^{+}\right]}} \mathrm{pax}(r) + \sum_{\substack{r \in R\,\wedge \\ \mathrm{mvt\_type}(r)=\mathrm{departure}\,\wedge \\ \mathrm{conf\_time}(r)\in\left[t-\pi^{-},t\right]}} \mathrm{pax}(r). \tag{3.24}$$

In the same way as for the runway constraints described in Section 3.7.1, limits for the number of passengers in the terminal can be configured for different time intervals and depending on the time of the day.

3.7.3 APRON Limits

Another limiting factor might be the number of available parking positions for aircrafts. This is commonly known as *APRON constraints*. After arrival an aircraft occupies a parking lot until its next departure. Different types of parking lots with varying sizes are plausible. A big aircraft might need a big parking lot, or might perhaps occupy two or even more smaller parking lots.

Let $A = [\alpha_1, \alpha_2, \ldots, \alpha_n]^T$ denote the number of available parking positions for APRON position types 1 to n. Furthermore, let apron(r) assign one ore more necessary parking positions to request $r \in R$. Hence, apron(r) is a vector, such that the element e at index i states, that the request r needs e parking positions of type i. For example, suppose we have three types of parking positions (i. e. small, medium, big) with available capacities $A = [2, 1, 3]$. Then the aircraft belonging to the request r with apron$(r) = [2, 0, 0]^T$ occupies two small parking positions. Note, that a departure never occupies any parking position, but instead releases some.

Hence, we can define the number of parking positions pos(t) in use at time $t \in T$ for every APRON type with index $i \in \{1, \ldots, n\}$ as

$$\mathrm{pos}(t,i) = \sum_{\substack{r \in R\,\wedge \\ \mathrm{mvt\_type}(r)=\mathrm{arrival}\,\wedge \\ t\in[\mathrm{conf\_time}(r),\mathrm{conf\_time}(r')]}} \mathrm{apron}_i(r), \tag{3.25}$$

where r' is the corresponding departure request, also called *turnaround request*, which is explained in more detail in the next section.

Now, for all times $t \in T$ and all $i \in \{1, \ldots, n\}$

$$\mathrm{pos}(t,i) \leq A_i \tag{3.26}$$

must hold.

3.8 Turnaround Constraints

Another constraint deals with the interdependencies between arrivals and departurers. Obviously a departure must be scheduled after the corresponding arrival. Furthermore, in order to clean, refuel, etc. the aircraft a certain ground time must be met.

Hence, we define a relation turnaround$(r) : R \rightarrow R$ to link every request $r \in R$ with its corresponding turnaround request, that is every arrival with its corresponding departure and vice versa.

However, depending on the quality of the input data the linking information might not be available or be incomplete. Hence, the turnaround information must be extracted and approximated based on the movement types, the operator names and the flight numbers in case it is not available.

Besides, the turnaround times also affect the APRON constraints, since every aircraft staying at ground needs to occupy (one or more) parking positions. Hence, it is even more desirable to get linking information of high quality.

So, a brief overview of the matching algorithm used to extract the turnaround information from the input data follows. The basic idea is for every arrival to search for a corresponding departure. Hence, we iterate over the requests $r \in R$ and try to find a matching departure. Obviously, inappropriate requests can be skipped (e. g. arrivals, requests on the wrong day of operation, etc.). For the remaining candidates, the following matching criteria apply (ordered from high to low quality):

1. Full (consistent) linking information available for both, the arrival and the departure
2. Complete linking information available, either for the arrival or the departure
3. Linking information available partly for the arrival and partly for the departure
4. Incomplete linking information, but flight number differs by less than 2, the requested times differ by less than 25 minutes and the operator names are equal
5. Requested times differ by less than 25 minutes and the operator names are equal

However, for *home carriers* the situation is slightly different. In such cases, the turnaround constraints are slightly relaxed, because usually for home carriers enough aircrafts are available at all times. Appendix A shows a list of such home carriers for the special case of Vienna.

For a given request r and its turnaround request r' the requested ground time gnd_time(r, r') equals to

$$\text{gnd\_time}(r, r') = \begin{cases} \text{rt}(r) - \text{rt}(r') & \text{in case r is a departure,} \\ \text{rt}(r') - \text{rt}(r) & \text{in case r is an arrival,} \end{cases} \tag{3.27}$$

where $rt(r)$ denotes the request time of request r.

However, in many cases it is not possible to meet the ground time exactly as requested by the initial submissions. Hence, to comply with current practice, different deviations depending on the requested ground time gnd_time(r, r') are allowed. The exact values used in this work are shown in appendix A.

3.9 Fragmentation

In general, it is desirable to assign all requests of a (multiday) series to the same time of the day $t \in T$. We call such an assignment *homogenous assignment*. However, this is not always possible (or useful). In some situations it is better to split a (multiday) series and assign the requests to different times of the day. Two or even more times of the day might be used. Otherwise the deviation between the requested time and the confirmed time might increase significantly or even worse, some requests of the series must be set to the action code U (unable).

The concept of fragmentation was mentioned first by Destion [ACK15]. Although a homogenous assignment is prefered, fragmentation can not be avoided in every case. Hence, a way to estimate its impact is needed. However, several ways to quantify the fragmentation are possible.

3.9.1 Basic Fragmentation

Probably, the most trivial way is to count the assigned times. So, for a series s we define the *basic fragmentation* $\text{bas\_frag}_{S_r}(s)$, such that

$$\text{bas\_frag}_{S_r}(s) = |\ \{\text{conf\_time}(r) \mid r \in s \wedge \text{action}(r) \in \{K, T\}\}\ | \,. \tag{3.28}$$

In the same manner, we define the basic fragmentation for a multiday series m as $\text{bas\_frag}_{M_S}(m)$, such that

$$\text{bas\_frag}_{M_S}(m) = |\ \{\text{conf\_time}(r) \mid r \in m \wedge \text{action}(r) \in \{K, T\}\}\ | \,. \tag{3.29}$$

3.9.2 Scaled Fragmentation

However, the basic fragmentation as defined above has several drawbacks. For one thing, the values of bas_frag_{S_r} and bas_frag_{M_S} are not (reasonably) limited. For another, the distribution of confirmed times over the season is not respected at all. Table 3.3 shows an example with three (multiday) series labelled by the numbers one to three. Each column represents a day of the season, each line a certain time. For illustration, we consider ten days only. In terms of basic fragmentation, all three series would be equally weighted. However, the fragmentation of series one is clearly more desirable than the fragmentation of series two and three, which are intertwined.

Table 3.3: Fragmentation example 1 – three series with the same number of confirmed times, but very different fragmentation.

t_1	1	1	1	1	1					
t_2	2	3	2	3	2	3	2	3	2	3

Hence, we present another way to quantify the fragmentation of (multiday) series over a whole season. The *scaled fragmentation* was outlined for the first time by Destion [ACK15].

Up to now, fragmentation has been addressed in terms of a series. However, that is not the only way to move forward. The scaled fragmentation is based on time. The goal is to define scal_frag(t) to quantify the fragmentation of a certain time slot $t \in T$ over a whole season.

Now, the central idea is "block counting". Lets consider another example, as shown in Table 3.4. At time t_1, we have a series consisting of three continuous blocks with lengths 3, 1 and 3, whereas series two at time t_2 contains three continuous blocks with lengths 2, 2 and 4.

Table 3.4: Fragmentation example 2 – two series with blocks of different length.

t_1	1	1	1			1		1	1	1
t_2	2	2		2	2		2	2	2	2

Next, let $|D|$ denote the number of days of the whole scheduling season (in our examples $|D| = 10$). Furthermore, let l_i be the length of block i (equal to zero if no such block exists) and let $l_i, i = 1, \ldots, |D|$ be sorted by the length in descending order ($l_1 \geq l_2 \geq l_3 \geq \ldots$). Hence, for the example above, we get $l_1 = 3$, $l_2 = 3$ and $l_3 = 1$ at the time t_1.

Then, to quantify the different blocks of a time slot $t \in T$, let

$$\text{blk}_t = \sum_{i=1}^{|D|} l_i \, (|D| + 1 - i). \tag{3.30}$$

In order to scale the fragmentation between zero and one, we will analyze the extreme cases. For the best-case, let $l_1 = |D|$ and $l_i = 0$ for $i \geq 2$. Then, blk_t evaluates to $|D|^2$. On the contrary, in the worst-case we have $l_i = 1$ for all $i = 1, \ldots, |D|$. Hence, blk_t evaluates to $\frac{1}{2}\, |D| \, (|D| + 1)$.

So, to scale the fragmentation between zero (for the worst-case) and one (for the best-case), we define

$$\text{scal\_frag}_1(t) = 1 - \frac{\text{blk}_t - \text{blk}_{t_{\min}}}{\text{blk}_{t_{\max}} - \text{blk}_{t_{\min}}} = 1 - \frac{\text{blk}_t - \frac{1}{2}\, |D| \, (|D| + 1)}{|D| \, \left(|D| - \frac{1}{2}\, (|D| + 1)\right)}. \tag{3.31}$$

Note, that in general a time slot t_i contains more than one assignment at the same time. However, this is not yet respected in scal_frag$_1(t)$ and shall be addressed now. As shown in the next example in Table 3.5, it is not always possible to compute the scaled fragmentation for every series separately. Hence, we need to compute the scaled fragmentation for all series in a single step.

So, let μ_t denote the maximum number of movements at the same time, that is

$$\mu_t = \max_{d \in D} |\, \{ r \in R \,|\, \text{conf\_time}(r) = t \,\wedge\, \text{conf\_day}(r) = d \,\wedge\, \text{action}(r) \in \{K, T\} \} \,| \,. \tag{3.32}$$

Table 3.5: Fragmentation example 3 – three series with blocks of different length.

t_1	1	1	1		3	3	3	2	2	2
t_1	2	2	2	2	2		1	1	1	1

Thus, equation (3.30) becomes

$$\text{blk}_{t,\mu_t} = \sum_{i=1}^{\mu_t\,|D|} l_i\,(\mu_t\;|D| + 1 - i). \tag{3.33}$$

Once again, we analyze the extreme cases. In the best-case, we have μ_t blocks of length $|D|$ (and $l_i = 0$ for $i > \mu_t$). In that case, we get

$$\begin{aligned}\text{blk}_{t,\mu_{t_{\max}}} &= \sum_{i=1}^{\mu_t\,|D|} l_i\,(\mu_t\;|D| + 1 - i) = \sum_{i=1}^{\mu_t} |D|\;(\mu_t\;|D| + 1 - i) + \sum_{i=\mu_t+1}^{\mu_t\,|D|} 0 \\ &= \mu_t^2\;|D|^2 + \mu_t\;|D| - \sum_{i=1}^{\mu_t} |D|\;i = \mu_t^2\;|D|^2 + \mu_t\;|D| - |D|\,\frac{1}{2}\mu_t(\mu_t+1) \\ &= \mu_t^2\,|D|^2 + \frac{1}{2}(\mu_t\,|D| - \mu_t^2\,|D|).\end{aligned} \tag{3.34}$$

In the worst case, we have $l_i = 1$ for all $i = 1, \ldots, \mu_t\;|D|$. Hence,

$$\begin{aligned}\text{blk}_{t,\mu_{t_{\min}}} &= \sum_{i=1}^{\mu_t\,|D|} l_i\,(\mu_t\;|D| + 1 - i) = \mu_t^2\,|D|^2 + \mu_t\,|D| - \sum_{i=1}^{\mu_t\,|D|} i \\ &= \mu_t^2\,|D|^2 + \mu_t\,|D| - \frac{1}{2}(\mu_t\,|D|\,(\mu_t\,|D| + 1)) \\ &= \frac{1}{2}\mu_t^2\,|D|^2 + \frac{1}{2}\mu_t\,|D| = \frac{1}{2}\mu_t\,|D|\,(\mu_t\,|D| + 1) = \sum_{i=1}^{\mu_t|D|} i.\end{aligned} \tag{3.35}$$

Now, to respect the maximum number of assignments μ_t we define the scaled fragmentation $\text{scal\_frag}_{\mu_t}(t)$ as

$$\text{scal\_frag}_{\mu_t}(t) = 1 - \frac{\text{blk}_{t,\mu_t} - \text{blk}_{t,\mu_{t_{\min}}}}{\text{blk}_{t,\mu_{t_{\max}}} - \text{blk}_{t,\mu_{t_{\min}}}} = 1 - \frac{\text{blk}_{t,\mu_t} - \frac{1}{2}\mu_t\,|D|\,(\mu_t\,|D| + 1)}{\frac{1}{2}\mu_t^2\,|D|\,(|D| - 1)}. \tag{3.36}$$

Note, that for $\mu_t = 1$ equation (3.36) is equivalent to equation (3.31).

Apart from that, we still have to respect multiday series with different days of operation. Consider the examples three and four shown in Tables 3.6 and 3.7. In both cases, series one is spread over three different days of the week, namely Monday, Tuesday and

Thursday. Moreover, in both cases the number of movements is equal. But still, example three is significantly more appealing than example four. Whereas example three can be seen as somehow continuous, example four is clearly interrupted in several columns ($d \in D$). In particular, this might lead to a scenario, where one series fills the gaps of another series. However, this is not desired at all, because it potentially prevents the series to be continuous in the following season. Thus, we will evaluate the blocks of example three as continuous ($l_1 = 17$ and $l_i = 0, i > 1$) and the blocks of example four as disconnected during the days (columns) 6,7 and 8 ($l_1 = 12$, $l_2 = 5$ and $l_i = 0, i > 2$).

Table 3.6: Fragmentation example 4 – multiday series, Monday and Tuesday are somehow connected by Thursday.

t_1, Mo:	1	1	1				1	1	1	1
t_1, Tu:		1	1			1	1	1	1	
t_1, We:										
t_1, Th:			1	1	1	1				

Table 3.7: Fragmentation example 5 – multiday series, all days are disconnected at some $d \in D$.

t_1, Mo:	1	1	1						1	1
t_1, Tu:	1	1	1	1	1				1	1
t_1, We:										
t_1, Th:	1		1	1	1				1	

Formally, for the computation of the block lengths l_i, we define $|D|$ graphs with super-nodes 1 to 7 representing the days of operation (Monday to Sunday). The super-nodes contain in turn nodes representing individual movements of a series. Now, two nodes belonging to the same series are connected by an edge if and only if both nodes are in adjacent super-nodes. That is, either they are on different days in the same week (same plane) or they are on the same day in adjacent weeks (planes). For illustration, the graph is shown in Figure 3.6.

Then, the block lengths l_i result from the sizes of connected components of the fragmentation graph. They can for example be computed by the algorithm of Tarjan (see [Tar71]) with linear time. Hence, for the computation of $\text{scal\_frag}_{\mu_t}(t)$ for multiday series with respect to different days of operation, equation (3.36) can be used by computing the block lengths with the algorithm of Tarjan. Obviously the value of $|D|$ must be equal to the number of planes (weeks) in the fragmentation graph multiplied by seven.

For further clarification, another example is shown in Figure 3.7. Here, the planes contain only three days of the week to keep things simple. In each plane, all nodes of a series are connected by an edge. Furthermore, nodes are also connected across adjacent planes, if the series operates on the same day of the week (e. g. a request belonging to series one

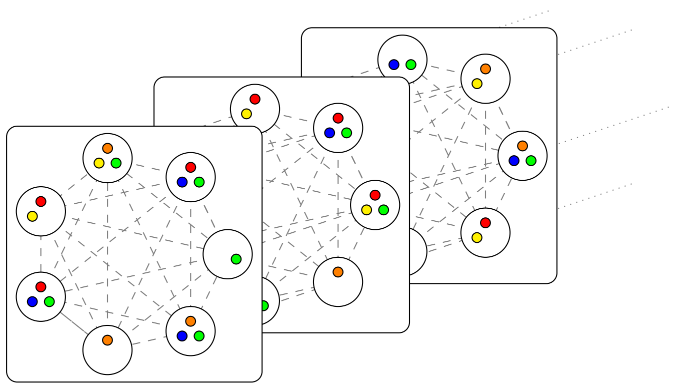


Figure 3.6: Fragmentation graph – Each plane corresponds to a week of the season, each super-node represents a day of the week and contains nodes for individual movements of a series, see [ACK15].

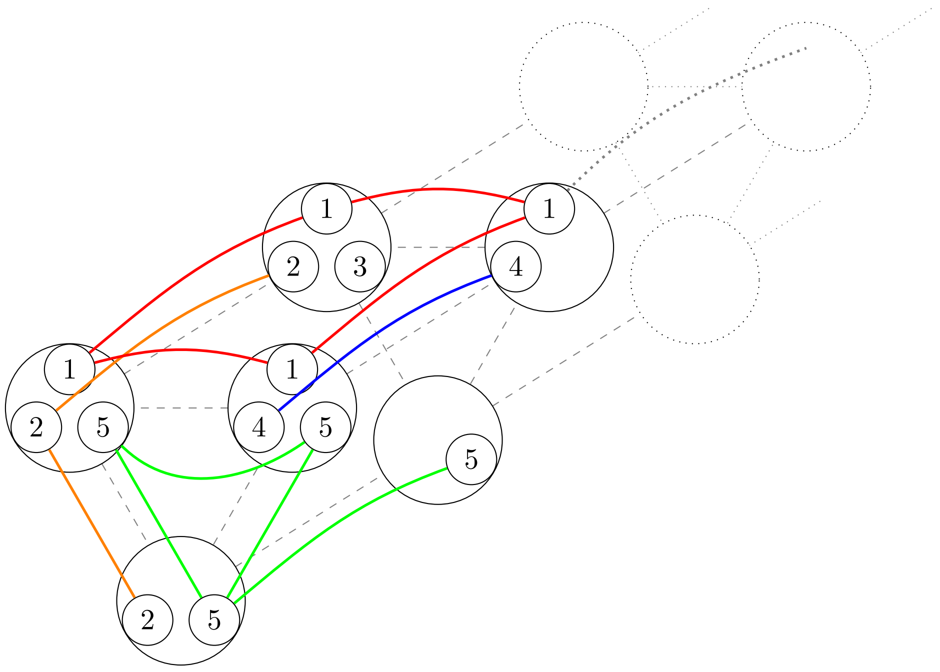


Figure 3.7: Fragmentation graph – Example showing two adjacent planes (weeks) with super-nodes for three days of the week, see [ACK15].

on Monday of week one would be connected by an edge to another request of the same series on Monday of week two).

3.9.3 Relative Fragmentation

Furthermore, we define a *relative fragmentation* on the basis of a reference solution. Such a referene solution could for example be the actual airport slot assignment used by the coordination authority. So, the relative fragmentation suits best to compare the solution computed by optimization algorithms against current practice.

Let $\text{conf\_time}_{\text{ref}}(r)$ be the confirmed time and $\text{action}_{\text{ref}}(r)$ the action code of a request in the reference solution. Now, for a given multiday series m, we define the number of confirmed times as $\text{cnt}(m)$ and the number of confirmed times in the reference solution as $\text{cnt}_{\text{ref}}(m)$, such that

$$\text{cnt}(m) = \left| \{\text{conf\_time}(r) \,|\, r \in m \wedge \text{action}(r) \in \{K,T\} \wedge \text{action}_{\text{ref}}(r) \in \{K,T\}\} \right| \tag{3.37a}$$

$$\text{cnt}_{\text{ref}}(m) = \left| \{\text{conf\_time}_{\text{ref}}(r) \,|\, r \in m \wedge \text{action}(r) \in \{K,T\} \wedge \text{action}_{\text{ref}}(r) \in \{K,T\}\} \right|. \tag{3.37b}$$

Then, we define the relative fragmentation rel_frag, such that

$$\text{rel\_frag} = \sum_{m \in P_{M_S}} (\text{cnt}(m) - \text{cnt}_{\text{ref}}(m)). \tag{3.38}$$

However, it is desirable to take the priorities of the series into account. Hence, we define a relative fragmentation weighted on priority $\text{rel\_frag}_{\text{prio}}$, such that

$$\text{rel\_frag}_{\text{prio}} = \frac{\sum\limits_{m \in P_{M_S}} (\text{cnt}(m) - \text{cnt}_{\text{ref}}(m))\, \text{pr}_{M_S}(m)}{\sum\limits_{\substack{m \in P_{M_S} \\ \text{cnt}(m) \neq \text{cnt}_{\text{ref}}(m)}} \text{pr}_{M_S}(m)}. \tag{3.39}$$

3.10 Objective Function

In general, any assignment of requests $r \in R$ to certain times $t \in T$ is a solution to the scheduling problem. Hence, a solution sol can be denoted by the assigned action codes and the confirmed times, such that

$$\begin{aligned} \text{sol} = \{\, & \text{action}(r) && \forall r \in R, \\ & \text{conf\_time}(r) && \forall r \in \{x \in R \,|\, \text{action}(x) \in \{K,T\}\}\}. \end{aligned} \tag{3.40}$$

However, a solution is only *feasible* if all of the constraints (runway constraints, passenger constraints, APRON constraints and turnaround constraints) are fulfilled. Furthermore, the priorities as described in Section 3.5 must be respected.

Note, that by this definition even the empty set $\emptyset$ can be regarded as a valid solution. But, obviously we are not interested in any solution, but a “good” one. Hence, some quality criteria for a good solution shall be worked out next.

Quantity

Firstly, the quantity of accepted requests can be used to indicate good solutions. So, we define the quantity of accepted requests as Γ, such that

$$\Gamma = |\,\{\ r \in R \mid \text{action}(r) \in \{K, T\}\ \}\,|\,. \tag{3.41}$$

Time Deviation

Next, another important indicator for the quality of solutions is the time deviation. Preferably a request is assigned to the requested time with little or no deviation. Hence, we define the time deviation $\text{dev}(r)$ of a single (confirmed) request $r \in R$ as

$$\text{dev}(r) = |\,\text{rt}(r) - \text{conf\_time}(r)\,|\,, \tag{3.42}$$

where $\text{rt}(r)$ denotes the requested time of r.

In the same manner, we define the overall time deviation Δ, such that

$$\Delta = \sum_{\substack{r \in R\,\wedge \\ \text{action}(r) \in \{K,T\}}} \text{dev}(r). \tag{3.43}$$

Fragmentation

Furthermore, fragmentation also plays a crucial role for the quality of a solution. As worked out in Section 3.9 different ways to measure the fragmentation are possible. For reasons of comparability, we use the relative fragmentation weighted on priority for the objective function. Hence, we define

$$\Theta = \text{rel\_frag}_{\text{prio}}. \tag{3.44}$$

Obviously, the value Θ shall be minimized.

Objective Function

Several different criteria such as quantity, time deviation and fragmentation impact the quality of a solution. Hence, we deal with a *multi-objective optimization* problem. To balance the impact of the different optimization goals, we introduce the weighting factors ω_1, ω_2 and ω_3 such that $\omega_1 + \omega_2 + \omega_3 = 1$. Then, we define the objective function $\text{obj}(\text{sol})$ as

$$\text{obj}(\text{sol}) = -\omega_1 \cdot \Gamma + \omega_2 \cdot \Delta + \omega_3 \cdot \Theta \tag{3.45}$$

to measure the quality of a given solution sol. The objective function 3.45 should be minimized, since lower values represent better solutions.

CHAPTER 4

Optimization Methods

In this Chapter a formal definition of optimization problems is presented. Section 4.1 covers continuous optimization problems. Several methods and solution strategies to solve such problems are discussed. Next, Section 4.2 deals with discrete optimization problems, which are in general harder to solve. A brief overview of exact methods and heuristic solution methods follows. Furthermore, several heuristc improvement strategies are examined in Section 4.3. Last, in Section 4.4 multi-objective optimization problems are considered.

4.1 Continuous Optimization

Optimization theory is a broad topic belonging to applied mathematics with lots of applications in different disciplines such as economics, engineering, physics and many more. In general, the goal is to find an optimum of a set of possible candidates according to a certain criterion. Formally, this can be described as follows. Let X be a set of candidates and f be a function mapping such candidates $\mathbf{x} \in X$ to real numbers $\mathbb{R}$, that is $f : X \to \mathbb{R}$. Then the goal is to minimize f,

$$\min_{\mathbf{x} \in X} f(\mathbf{x}). \tag{4.1}$$

Hence, for a global optimum $\mathbf{x}^*$

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \ \forall \mathbf{x} \in X \tag{4.2}$$

holds.

Note, that the set of candidates X can be constrained by several *equality restrictions* $g_i(\mathbf{x}) = 0$ and *inequality restrictions* $h_j(\mathbf{x}) \leq 0$. In such a case, the set of *feasible* *candidates* X can be denoted as

$$\begin{aligned} X = \{\, \mathbf{x} \in \mathbb{R}^n : {} & g_i(\mathbf{x}) = 0,\ i = 1, \ldots p, \\ & h_j(\mathbf{x}) \leq 0,\ j = 1, \ldots q \,\}, \end{aligned} \tag{4.3}$$

where p describes the number of equality restrictions and q the number of inequality restrictions. Obviously $p \leq n$ must hold, or else X is the empty set and no feasible solution exists.

The objective function $f(\mathbf{x})$ is also called *cost function*, because it assigns every value $\mathbf{x} \in X$ a cost. In physical applications it often refers to the energy of a system and thus $f(\mathbf{x})$ is called *energy function* in such situations.

In the literature, it is common to formalize optimization problems as minimization problems. However, it is always possible to transform a maximization problem into a minimization problem, by applying

$$\max_{\mathbf{x} \in X} f(\mathbf{x}) = \min_{\mathbf{x} \in X} -f(\mathbf{x}). \tag{4.4}$$

Usually the optimal value can not be found analytically. Thus, numerical solution methods are used. An overview of several different solution approaches is given by [NW06]. For unrestricted minimization problems, common solution strategies are for example *line search* and *trust region methods*. Both methods rely on the availability of the derivation of the objective function. Otherwise, if the derivation is not available (or does not exist) *direct methods*, such as the *simplex algorithm*, also called *Nelder-Mead method* and presented for the first time in [NM65], or *particle swarm optimization* as described in [KE95] might be used.

For the restricted case further solution algorithms such as the *active set method* and the *gradient projection method* (see [Kel99]) exist. Another aproach is to transform the restricted optimization problem into an equivalent unrestricted one. In this case, *penalty methods* or *barrier methods* like the *interior-point method* (see [Kar84]) might be used.

4.2 Discrete Optimization

However, some applications require one or more variables to be discrete. Common applications for such scenarios include *vehicle routing*, *scheduling*, *knapsack problems*, *constraint satisfaction problems*, *traveling salesman problems* and many more. Unfortunately, such a restriction makes the problem significantly harder.

In case of airport slot allocation, we also have to deal with discrete variables. For example, an individual request can only be accepted or denied. Accepting it to e. g. 50% is meaningless. Hence, the goal is to minimize $f(\mathbf{x})$, such that some or all variables $\mathbf{x} \in X$ are discrete. In general, the individual values of the candidates $\mathbf{x} \in X$ could be restricted to integral values or in a similar way restricted to countably infinite (or finite) domains. In the concrete case of airport slot allocation, both, the action codes and the confirmed times are discrete variables. The domain of action(r) is restricted to $\{K, T, U\}$ and conf_time(r) denotes a discrete time interval and hence is restricted to an integral value between 1 and 288.

Often, combinatorial optimization problems are hard to solve. Principally, finding the optimum conforms to a search in a discrete set, the *search space*. However, the search space can be very large and thus, the search can be very tedious. Indeed, some problems such as the traveling salesman problem are NP-complete.

In general, two solution strategies are possible. *Exact methods* on the one hand and *heuristic methods* on the other hand. We will take a closer look on each of those.

4.2.1 Exact Methods

Exact methods guarantee to find the optimum if the problem is solveable. However, it can take a verly long time. In general, finding a global optimum is equivalent to a search in a discrete set. So, an exhaustive search can be considered as a (naive) exact method. It will always find the optimum, though the performance might be very bad when the search space is big. Hence, if possible it should be attempted to reduce the search space and eliminate improper candidate solutions early on.

Often the search space can be viewed as a tree or similarly be structured into several distinct partitions. Sometimes it is possible to cut off a whole branch or partition at once. For example, the *branch and bound* method, mentioned for the first time in [LD60], tries to identify such branches or partitions by computing bounds. Hence, a full enumeration of all solution candidates is only necessary in the worst case.

In particular, *LP-based* branch and bound is a popular method to solve discrete optimization problems. Good bounds can be computed by *LP-relaxations* (see [Agm]), where the integrality constraint is dropped. Then, *cutting planes* (see [DFJ54]) can be separated in order to to find better approximations of the integer polyhedron. Based on the obtained bounds the size of the search tree of the underlying branch and bound algorithm can be reduced effectively.

Today, many professional solvers for (mixed) integer programming problems (*MIP solvers*) are available. For example, *CPLEX* is a well-known proprietary solver developed by IBM. An opensource alternative is for example *COIN-OR linear programming (CLP)*. Usually they use branch and bound methods with cutting plane algorithms, as well as several further state of the art methods.

Although MIP solvers often achieve amazing results, they are no magic whizz kids. When the complexity and the size of the problem exceeds certain boundaries, exact methods are not applicable within reasonable running times.

4.2.2 Heuristic Methods

In almost every use case the goal is not to find the very best solution, but instead it suffices to find a "good" solution comparable to the global optimum. Heuristic methods do not guarantee to find the global optimum. Instead, they provide ways to find good solutions in reasonable running times. This is in particular helpful when the size of the search space is very big and exact methods fail due to the bad performance.

Depending on certain characteristics different types of heuristics can be distinguished. In contrast to *problem specific heuristics*, so-called *meta heuristics* can principially be applied to any optimization problem. The overall sequence of abstract steps does not depend on the problem itself, although the individual steps are most probably still implemented in a problem specific way. Typical representatives for such meta heuristics are *local search* and *simulated annealing* among many others.

Another important way to categorize heuristics is the distinction between *constructive* and *improvement* heuristics. Whereas the first type starts out of nowhere and constructs a solution step by step, the improvement method uses an existing solution candidate and tries to futher improve it.

4.3 Improvement Heuristics

Next, we will discuss several improvement (meta) heuristics. As described above the goal is to improve an existing solution candidate $\mathbf{x}$ step by step. A single step, which transforms a solution candidate $\mathbf{x}$ into a new (feasible) solution candidate $\mathbf{x}'$ is called *improvement step*. Furthermore, the *neighborhood* $\mathcal{N}(\mathbf{x})$ denotes all those solution candidates, which can be reached from $\mathbf{x}$ by applying one single improvement step.

A simple seach strategy is the above mentioned *local search*. Algorithm 4.1 shows the corresponding pseudo code.

Algorithm 4.1: Local Search

```
  Data: existing solution candidate $\mathbf{x}$
  Result: improved solution candidate $\mathbf{x}$
1 repeat
2     choose $\mathbf{x}' \in \mathcal{N}(\mathbf{x})$;
3     if $f(\mathbf{x}') \leq f(\mathbf{x})$ then
4         $\mathbf{x} \leftarrow \mathbf{x}'$;
5     end
6 until abort criterion met;
```

The most important step happens in line 2. A new solution candidate $\mathbf{x}'$ is chosen out of the set of neighbors of the current solution candidate $\mathbf{x}$. However, different strategies are possible. A *best improvement strategy* might consider all neighbors of $\mathbf{x}$ and choose the best one, that is $f(\mathbf{x}') \leq f(\mathbf{x}^*)\ \forall \mathbf{x}^* \in \mathcal{N}(\mathbf{x})$. On the contrary, a *first improvement* *strategy* might be content with any neighbor satisfying $f(\mathbf{x}') < f(\mathbf{x})$. This is particularly interesting, when the evaluation of $f(\mathbf{x})$ is expensive. Still another strategy is the *random improvement strategy*. Here, a random neighbor $\mathbf{x}' \in \mathcal{N}(\mathbf{x})$ is chosen.

Another improvement (meta) heuristic is *simulated annealing*. In 1983, it was applied to the Traveling Salesman Problem (see [KGV83]) and since then to lots of optimization problems. Often, a local search algorithm as presented above might not be able to escape local optima. Simulated annealing was developed in analogy to the physical process of heating and cooling some metallic material. Hereby, the particles move less and less with decreasing temperature. Algorithm 4.2 shows the basic algorithm.

Algorithm 4.2: Simulated Annealing

```
Data: existing solution candidate $\mathbf{x}$, time $t$, current temperature $T$, initial
      temperature $T_{\text{init}}$, random variable $Z$
Result: improved solution candidate $\mathbf{x}$
1  $t \leftarrow 0$;
2  $T \leftarrow T_{\text{init}}$;
3  repeat
4      randomly choose $\mathbf{x}' \in \mathcal{N}(\mathbf{x})$;
5      if $f(\mathbf{x}') < f(\mathbf{x})$ then
6          $\mathbf{x} \leftarrow \mathbf{x}'$;
7      else
8          $Z \leftarrow \text{random} \in [0,1)$;
9          if $Z < e^{\frac{|f(\mathbf{x}')-f(\mathbf{x})|}{T}}$ then
10             $\mathbf{x} \leftarrow \mathbf{x}'$;
11         end
12     end
13     $T \leftarrow g(T,t)$;
14     $t \leftarrow t+1$;
15 until abort criterion met;
```

In the beginning, the algorithm also accepts worse solution candidates by a certain probability and thus, we can escape from local minima. However, with decreasing temperature the chance to accept worse solution candidates becomes more and more unlikely and in the end the algorithm converges to local search. Hereby, the "cooling process“ is determined by the function $g(T,t)$. A typical implementation is given by $g(T,t) = T \cdot \alpha$ with $\alpha < 1$.

Furthermore, many other (meta) heuristic methods exist, such as *tabu search*, *ant colony optimization* and *evolutionary algorithms* to name just a few. The latter one memorizes not only the current solution candidate, but a whole set of solution candidates, the so-called *population*. Among others, further meta heuristics are covered in detail in [GP19].

4.4 Multi-Objective Optimization

In practice, quantifying a solution candidate is not always straight forward. Often, several distinct optimization goals can be formulated, e. g. time, monetary costs, distance, etc.. Frequently the individual goals are in conflict with each other. Minimizing goal A might lead to higher costs regarding goal B and vice versa. Although often desired, optimizing both (or even more) goals separately usually yields no sufficient solution at all. As a result, tradeoffs have to be balanced.

So, multi-objective optimization deals with several objective functions simultaneously. In principal, the goal is to minimize the whole vector of objective functions f_1 to f_n at once. Although it might sound simple at first glance, it is hard in practice. In fact, for an optimization problem with conflicting objectives usually no single optimal solution exists. Instead, we need to deal with a whole set of optimal solutions reflecting the possible tradeoffs between different objectives.

Formally, such a set is called *pareto frontier*. Let $f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})$ be n objective functions to quantify n different optimization goals. Then, a solution candidate $\mathbf{x}$ is called *pareto optimal* if there exists no other feasible solution candidate $\mathbf{x}'$, such that $f_i(\mathbf{x}') < f_i(\mathbf{x}) \wedge \forall j \in \{1, \ldots, n\}\ f_j(\mathbf{x}') \leq f_j(\mathbf{x})$ for some i. Hence, the pareto solution describes a state, in which no further improvement is possible, without worsening at least one other objective. In the literature the terms pareto optimal, pareto efficient and nondominated solution are used interchangably. Now, the set of all pareto optimal solutions is called pareto frontier. Note, that its size can possibly be infinite.

Besides, the domains of the objectives might vary significantly. This makes comparison of different solutions even harder. Suppose, the objective functions f_1 and f_2 yield the values 0.2 and 100 for solution A. Furthermore, suppose solution B results in the values 0.1 and 500. Now, which solution should be preferred? This question has no unique answer. Both solutions might be pareto optimal and intuitively solution A might seem desirable at first. But, for one thing, the values might not be directly comparable, because of different domains. Suppose, f_1 yields values in the range $[0, 1]$ and f_2 in the range $[0, 100000]$. Then, solution B seems suddenly much more attractive. For another, besides the possibly differing domains, the objectives just might not be equally important. It strongly depends on the meaning of the objective functions. So, at the end of the day it is in general incumbent upon the humans to assess the tradeoffs between conflicting objective functions. Hence, a common approach is to visualize the pareto frontier and leave the ultimate decision to humans in charge.

CHAPTER 5

Solution Algorithms

The focus of this Chapter are the algorithms developed to solve the airport slot allocation problem. First, we will consider a construction heuristic. In Section 5.1 the basic algorithm is presented and discussed in detail. Next, Section 5.2 shows further improvement heuristics used to refine the solution.

5.1 Construction Heuristic

As described in Chapter 4, for many combinatorial optimization problems finding a global optimum is hard. This is also true for the airport slot allocation problem. Due to the problem size (200.000 requests and more) exact solution approaches are somewhat limited. Hence, we propose heuristic solution strategies to find a "good" solution in relatively short running times.

In a first step, a construction heuristic is used to create a feasible solution. As a hard requirement, the turnaround constraints, the capacity limitations (runway, apron and passenger constraints) as described in Sections 3.7.1, 3.7.2, 3.7.3 and 3.8 must hold. Goal is to confirm as many requests as possible (action_code(r) $\in \{K, T\}$ - see Section 3.6), while minimizing the total time deviation Δ and the fragmentation Θ (see Section 3.10).

The construction heuristic can be decomposed into several parts. First, in algorithm 5.1 we define a basic procedure used to assign a whole set of requests $S \subseteq R$ to a certain time t. Note, that the algorithm ensures feasibility of the current partial solution. Hence, it must also take the aggregated counts of runway, passengers and apron into account. Furthermore, some of the requests $s \in S$ might have already been assigned in advance. Those requests shall be skipped and not be changed at all. Hence, in line 1 only those requests $s \in S$, that are not yet assigned (action(s) $= U$), are set to the desired time t.

After updating the confirmed times of all requests $s \in S$, we need to check whether the partial solution is still feasible. Hence, the aggregated counts of the current partial solution must be updated accordingly and then the satisfaction of the constraints must be evaluated. If the constraints are still fulfilled, the partial solution is still feasible and all $s \in S$ are successfully assigned. Hence, the algorithm returns a flag indicating success. Otherwise, some constraints are violated due to the assignments in line 1. All changes must be reverted to the previous state as if the algorithm was not called at all. In that case, the returned flag indicates that not all demanded assignments are possible, because some of the constraints would be violated.

Algorithm 5.1: set_assignments

```
Data: set $S$ of requests to be assigned at a certain time $t$, aggregated counts of
      current partial solution
Result: flag indicating success or failure
1 assign all requests in $\{s \in S \mid s$ not yet assigned$\}$ to time $t$;
2 update aggregated counts;
3 if feasible (capacity and turnaround constraints satisfied) then
4     return success;
5 else
6     revert assigned requests and aggregated counts to previous state;
7     return failure;
8 end
```

Next, algorithm 5.2 gives an abstract overview of the overall construction procedure. First, all requests $r \in R$ are sorted by their priority $\mathrm{pr}_r(r)$. Then, all mandatory requests are assigned. Mandatory requests are those, for which only one possible time exists (e. g. requests with historic status code F). Last, we iterate over the remaining requests and try to find a feasible assignment.

This involves several strategies, each of which might be successful. In the simplest case, only the current request itself is considered and the algorithm tries to find a feasible assignment for this single request. This can be considered as a fallback case and corresponds to line 10 in algorithm 5.2.

However, in general it is desired to find a homogenous assignment for all requests belonging to the same (multiday) series. Assigning all requests of a (multiday) series to the same time slot keeps the fragmentation low. This is all the more important, since low fragmentation is one of the key objectives. Hence, before using the fallback strategy, the algorithm 5.2 tries to find a feasible assignment for the whole series of the current request. This strategy is applied in line 8.

But, the assignment of a request also depends on its turnaround request. Often, it is essential to assign both at once. Thus, we introduce still another strategy. In line 6, the algorithm takes the (multiday) series of the current request as well as the (multiday) series of its turnaround request into account and tries to assign both series homogenously. This is the preferred case and hence it is tried first.

However, in some situations all three strategies fail. As a last resort, we try to resolve conflicts. That is, the algorithm tries to move already assigned requests to other time slots in order to find a feasible assignment for the current request r. If this keeps to fail, the current request remains unable (action code U) and the iteration moves on.

Algorithm 5.2: construction

```
Result: solution consisting of action codes and confirmed times
1  initially for all $r \in R$, set action code to $U$ (unable);
2  sort requests $r \in R$ by priority $pr_r(r)$;
3  assign all mandatory requests;
4  forall $r \in R$ do
5      skip if $r$ is already assigned;
6      try_homogenous_assignment($r$, respect_turnaround $\leftarrow$ true);
7      if not possible then
8          try_homogenous_assignment($r$, respect_turnaround $\leftarrow$ false);
9          if not possible then
10             try_single_assignment($r$);
11             depth $\leftarrow 0$;
12             while not possible and depth $\leq 2$ do
13                 try_conflict_resolvement($r$, depth);
14                 depth $\leftarrow$ depth $+ 1$;
15             end
16         end
17     end
18 end
```

The different strategies will be explained in detail next.

5.1.1 Homogenous Assignments

When assigning a request r, it makes sense to take the whole (multiday) series S of r into account. Confirming r at time t does not help a lot, if the remaining requests of the same series can not be assigned to the same time slot. Homogenous allocation is almost always the better choice. Hence, we introduce algorithm 5.3, which gets a request r as input and tries to find a time slot t for all requests belonging to the whole (multiday) series of r.

Furthermore, depending on the flag *respect_turnaround*, the algorithm might consider the turnaround requests too. In this case, it tries to find a feasible, homogenous assignment for both, the series S of r and the series of the turnaround request $r' = \text{turnaround}(r)$.

However, note that all requests $s \in S$ with $pr_r(s) < pr_r(r) + \eta$ are skipped. The same is valid for the turnaround series T. This ensures, that the priority model as described in

Algorithm 5.3: try_homogenous_assignment

```
Data: request $r$, respect_turnaround flag
Result: flag indicating success or failure
1  $S \leftarrow \text{series}_{M_S}(\text{series}_{S_r}(r))$;
2  remove requests $s \in S$ with $pr_r(s) < pr_r(r) + \eta$ or differing requested time;
3  forall $t \in get\_priorized\_times(r)$ do
4      set_assignments($S$, $t$);
5      if possible then
6          if not respect_turnaround then
7              return success;
8          end
9          $T \leftarrow \text{series}_{M_S}(\text{series}_{S_r}(\text{turnaround}(r)))$;
10         remove requests $t \in T$ with $pr_r(t) < pr_r(r) + \eta$ or differing requested time;
11         forall $t' \in get\_priorized\_times(turnaround(r))$ do
12             set_assignments($T$, $t'$)
13             if possible then
14                 return success;
15             end
16             revert assignments of $T$ to previous state;
17         end
18     end
19     revert assignments of $S$ to previous state;
20 end
21 return failure;
```

Section 3.5 is respected. The constant η is a configuration parameter and can be flexibly adjusted by the coordinator. Throughout this work, a value of $\eta = 5.0$ is used.

Furthermore, requests which belong to the same series, but have another requested time are also skipped. Because the requested time differs, assigning those requests as well might lead to very high time deviation.

This algorithm iterates over all possible times and tries to assign all requests of the (multiday) series S of r to the same time slot t. If this succeeds and *respect_turnaround* is set to false, we are done. However, if *respect_turnaround* is set to true, it takes more. Since the assignment of a request always affects the assignment of its turnaround request too, it is desirable to assign both at once. Assigning r to the "wrong" time slot might prohibit a feasible assignment for its turnaround request $r' = \text{turnaround}(r)$. Hence, the algorithm also deals with the series of the turnaround request r'. Only, if both series can be assigned homogenously, the algorithm returns success.

Suppose, the series of the turnaround request can not be assigned in a feasible way, because either the turnaround constraints or the capacity constraints are violated. Then, all temporary assignments are reverted to the previous state and the iteration moves on.

Probably, the most important step in this algorithm is the deduction of priorized times. The order, in which we iterate over the possible time slots matters a lot. Hence, we will take a closer look on this part now. Algorithm 5.4 determines a priorized list of possible times for the request r. For that purpose, we once again take the (multiday) series of r into account. Suppose, that some requests $s \in S$ are already assigned to some time slot t in advance. Then, for the sake of low fragmentation, it is desirable to assign r to the same time slot t (provided, $t \in$ allowed_times(r)). On the other hand, the requested time of r is also very desirable, because it minimizes time deviation. This is sort of a tradeoff, which needs to be sorted out. Another factor, which might play a role, is whether the already assigned requests are on the same day of operation. Assigning r_1 to the same time slot as r_2 is more important if r_1 and r_2 are on the same day of operation.

Hence, to sort the allowed times, we use the priority rules as implemented in algorithm 5.4. First, we consider all requests $s \in S$ which are already assigned and take place on the same day of operation. Those confirmed times are most desirable for the request r. Next, we consider the requested time of r. Then, the confirmed times of requests $s \in S$, which take place on another day of operation, are considered. Last, we add the remaining allowed_times(r) sorted by the deviation to the requested time.

However, it must be ensured, that all those priorized times are allowed for every request $s \in S$ (and for r in particular). Hence, in line 6 all potential times, that are not allowed for every request $s \in S$, are removed from P.

Algorithm 5.4: get_priorized_times

```
  Data: request $r$
  Result: sorted array $P$ containing potential times
1 $S \leftarrow \text{series}_{M_S}(\text{series}_{S_r}(r))$;
2 append confirmed times of $s \in S$ with same day of operation as $r$ to $P$;
3 append requested time of $r$ to $P$;
4 append confirmed times of $s \in S$ on another day of operation than $r$ to $P$;
5 append allowed_times($r$) to $P$, sorted by deviation to requested time of $r$;
6 remove all $p \in P : p \notin \bigcap_{s \in S}$ allowed_times($s$) from $P$;
7 return P;
```

5.1.2 Single assignment

Next, we deal with single assignments. When a homogenous assignment is not possible, algorithm 5.5 is used. It iterates over the allowed_times of r, sorted by the deviation to the requested time. If all constraints are satisfied, we return success. Otherwise, the assignment is reverted and the iteration moves on. If no assignment is possible, failure is returned.

Algorithm 5.5: try_single_assignment

```
Data: request $r$
Result: flag indicating success or failure
1  $T \leftarrow$ allowed_times($r$);
2  sort $T$ by deviation to requested time of $r$;
3  forall $t \in T$ do
4      set_assignments($\{r\}$, $t$);
5      if possible then
6          return success;
7      end
8      revert assignment to previous state;
9  end
10 return failure;
```

5.1.3 Conflict Resolvement

If all other strategies fail, we can still try to move an already confirmed request to another time slot in favor of the current request. For this purpose, we first introduce algorithm 5.6 used to identify possible candidates for conflict resolving.

Algorithm 5.6: get_potential_conflict_assignments

```
Data: request $r$
Result: array of conflicting requests $C$ sorted by priority
1  $T \leftarrow$ allowed_times($r$);
2  $t_{\min} \leftarrow \min T - (\kappa - 1)$;
3  $t_{\max} \leftarrow \max T + (\kappa - 1)$;
4  forall $r' \in R \mid r' \neq r \ \wedge \ conf\_day(r) = conf\_day(r') \wedge$
     $t_{min} \leq conf\_time(r') \leq t_{max}$ do
5      if $r'$ not mandatory $\wedge \ pr_r(r') \geq (pr_r(r) - \eta)$ then
6          append $r'$ to $C$;
7      end
8  end
9  sort $C$ by priority $pr_r(r)$;
10 return $C$;
```

In lines 2 and 3, the algorithm defines a time interval. All requests confirmed in this interval are potential candidates for conflict resolvement. The parameter κ expands this interval to account for the aggregated counts as defined in Section 3.7.1, since they span over a time interval δ.

The potential candidates are further restricted in line 5. Since mandatory requests can not be moved to another time, they are never selected for conflict resolvement. Furthermore, in order to respect the priority model, a conflicting request must not have a lower priority than $pr_r(r) - \eta$.

Next, algorithm 5.7 shows the conflict resolvement method. In line 1, a set of possible candidates for conflict resolvement is determined. Then, the algorithm tries to confirm the current request r by moving one of the conflicting requests to some other time slot. In the best case, the whole (multiday) series of the conflicting request can be moved homogenously to another time slot. Otherwise, only the conflicting request itself is moved to another time slot.

Algorithm 5.7: try_conflict_resolvement

```
Data: request $r$, iteration depth $d$
Result: flag indicating success or failure
1  $C \leftarrow$ get_potential_conflict_assignments($r$);
2  forall $c \in C$ do
3      $T \leftarrow$ allowed_times($r$) sorted by deviation to requested time;
4      forall $t \in T$ do
5          set $c$ to unable;
6          set_assignments($\{r\}$, $t$);
7          if possible then
8              try_to_move_series_to_other_time($c$);
9              if possible then
10                 return success;
11             end
12             try_to_move_request_to_other_time($c$, $d$);
13             if possible then
14                 return success;
15             end
16         end
17         revert assignments to previous state;
18     end
19 end
20 return failure;
```

Note, that the conflict resolvement method is performance critical. Hence, a total time limit θ is used and checked throughout the conflict resolvement. If the running time of this algorithm exceeds this time limit, the conflict resolvement is stopped, all temporary assignments are reverted to the previous state, failure is returned, and the current request remains unable.

Furthermore, the whole conflict resolvement strategy is never called for requests with historic status code N due to performance reasons. The set of allowed times for such requests is usually quite big and hence they have high chances to be assigned during the single assignment strategy anyways. The conflict resolvement strategy on the other hand takes quite a long time for such requests with low chances of success. Hence, they are skipped altogether.

The next algorithm 5.8, which tries to move a whole (multiday) series to another time, is straightforward. It iterates over the priorized times of the series and tries to assign the whole series homogenously to another time. However, in order to respect the priority model, all unassigned requests belonging to the (multiday) series are skipped and remain unassigned.

Algorithm 5.8: try_to_move_series_to_other_time

```
Data: request $r$
Result: flag indicating success or failure
$S \leftarrow \text{series}_{M_S}(\text{series}_{S_r}(r))$;
remove unassigned requests from $S$;
$T \leftarrow get\_priorized\_times(r)$;
forall $t' \in T \ \wedge \ t' \neq conf\_time(r)$ do
    set all $s \in S$ to unable;
    set_assignments($S$, $t'$);
    if possible then
        return success;
    end
    revert assignments to previous state;
end
return failure;
```

Now, algorithm 5.9 is slightly more advanced. Here, we respect the iteration depth d. If d is equal or less than zero, we just try the allowed times of the conflicting request c and try to find a new timeslot. However, if d is greater than zero, we restart the whole conflict assingment method for c. That is, we try to move some other confirmed request to a new time slot in favor of c.

Algorithm 5.9: try_to_move_request_to_other_time

```
Data: conflicting request $c$, iteration depth $d$
Result: flag indicating success or failure
1  $T \leftarrow$ allowed_times($c$);
2  sort $T$ by deviation to requested time;
3  forall $t' \in T$ do
4      if $d \leq 0$ then
5          set_assignments($c$, $t'$);
6          if possible then
7              return success;
8          end
9      else
10         try_conflict_resolvement($c$, $d - 1$);
11         if possible then
12             return success;
13         end
14     end
15     revert assignments to previous state;
16 end
```

5.2 Improvement Heuristics

Moreover, we propose some heuristic methods to further improve the results. Experimental evaluation reveals, that in general the construction heuristic as presented in Section 5.1 leads to low time deviation. However, to achieve low fragmentation too, usually further refinement is necessary.

In this manner, we developed some heuristic improvement methods focusing on fragmentation. Since, we have to deal with multiple objectives, a good tradeoff between time deviation and fragmentation needs to be found. Hence, the following algorithm 5.10 works with a set of partial solutions and approximates the pareto frontier in order to find a good balance between deviation and fragmentation.

In a first precomputing step, candidates for further improvement are identified. Regarding fragmentation, two distinct cases need to be distinguished. On the one hand, we need to deal with whole multiday series at once. In the best case, we can assign all requests of a multiday series to the same time slot. However, this is not always possible. Hence, we also need to deal with single days of operation.

Lets consider an example. Suppose, a series contains several requests on Monday and on Thursday throughout the season. Quite often, it is not feasible to assign all requests of both days to the same time slot. However, it might be possible to assign all requests on Monday to time slot t_1 and all requests on Thursday to time slot t_2. Hence, we need to find homogenous assignments for each individual day separately.

In line 1 all multiday series with inhomogenous assignments are identified as possible candidates for further improvement. Every element $a \in A$ contains all requests of such a multiday series. Next, in line 2 individual days of operation with inhomogenous assignments are detected. An element $b \in B$ contains only requests belonging to the same day of operation. Hence, it is only a subset of a multiday series.

The algorithm keeps a set of best solutions found so far. At the beginning, we start with the solution of the construction heuristic described in Section 5.1. While the maximum number of iterations is not yet reached, a random partial solution is picked and the algorithm randomly tries to improve the fragmentation either regarding a whole multiday series or a single day of operation.

Whenever this yields a better solution in terms of pareto efficiency, the new solution is added to the set of best solutions. This might also imply, that a former partial solution needs to be removed from this vector, because it is no longer pareto efficient.

Algorithm 5.10: improvement

```
Result: randomly approximated pareto frontier
1  $A \leftarrow$ inhomogenously assigned multiday (series$_{M_S}$);
2  $B \leftarrow$ inhomogenously assigned series (series$_{S_r}$);
3  best_solutions $\leftarrow$ construction_solution;
4  while max_iterations not reached do
5      $c \leftarrow$ pick random solution from best_solutions;
6      $i \leftarrow$ 0..1 randomly;
7      while max_tries not reached do
8          if $i = 0$ then
9              $S \leftarrow$ pick random candidate from $A$;
10         else
11             $S \leftarrow$ pick random candidate from $B$;
12         end
13         $c \leftarrow$ improve_fragmentation($c$, $S$);
14         if $c$ is pareto efficient regarding best_solutions then
15             append $c$ to best_solutions;
16             remove all pareto inefficient solutions from best_solutions;
17             break;
18         end
19     end
20 end
21 return best_solutions;
```

Next, algorithm 5.11 shows the actual improvement step. It receives a solution cadidate c and a set S of inhomogenously assigned requests of a multiday series (or a subset). First, all possible times for S are determined and sorted by deviation to the requested time. Then, all requests S are reset (set to unable) and the algorithm tries to find a homogenous assignment for all requests in S. After all, the resulting new solution candidate c is returned.

Algorithm 5.11: improve_fragmentation

```
   Data: current solution $c$, set $S$ of requests
   Result: current solution $c$
 1 $T \leftarrow \bigcap_{s \in S}$ allowed_times($s$)
 2 sort $T$ by deviation to requested time;
 3 forall $t \in T$ do
 4     set all $s \in S$ to unable;
 5     set_assignments($S$, $t$) in $c$;
 6     if possible then
 7         return $c$;
 8     end
 9     revert temporary assignments to previous state;
10 end
11 return $c$;
```

CHAPTER 6

Experimental Results

This Chapter presents the computational results of the algorithms developed in the scope of this work. To evaluate the results, the solutions of the optimization algorithms are compared to current operational practice at Austrian airports. Section 6.1 introduces the data sets used in this work. Section 6.2 covers a description of the hardware used to run the tests. The computational results of the construction algorithms are shown in Section 6.3. Then, in Section 6.4 the results of the improvement algorithms are shown, together with an approximated pareto analysis. Section 6.5 summarizes the best solutions for all data sets and last, Section 6.6 shows further results regarding passenger and apron limits.

6.1 Description of the Data Sets

All data sets have been provided by Schedule Coordination Austria. The input data consists of real (historic) data used for the initial schedule creation at the Vienna airport. The test cases contain all initial requests of airport slots as submitted by the air carriers before the start of the season.

Now, to take a closer look on the input data and briefly discuss the format, we show an example with anonymized values. Table 6.1 shows three lines of input data, each corresponding to a single request. All enries of such a request are delimited by a semicolon. A short description of all individual entries is shown in Table 6.2.

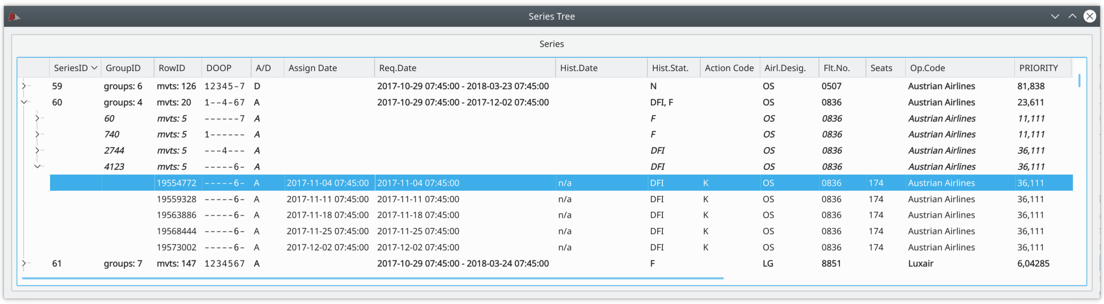
Usually, the data sets span a whole season. Regarding the Vienna airport, such data sets contain between 100,000 and 200,000 lines. Taking other international airports into account, the size of such data sets can increase up to half a million lines and even more. Note, that all requests contain the attributes presented in Section 3.3. Furthermore, for every request some turnaround information might be available. However, the quality of that turnaround information is not always reliable and hence, a preprocessing step as

Table 6.1: Anonymized input data.

Season;Airport;Date;DOOP;ArrDep;Time;Req;Hist;HistStat;ActionCode;AirlDesig;Fltno; OpeSuffix;Seats;IATA Aircraft Type;ICAO Aircraft Type;LastNext;ICAO LastNext; OrigDest;ICAO OrigDest;LastNextCountry;OrigDestCountry;ServType;TurnOpe; TurnServNo;TD; EditDate;ACReg;GA/BA;OpeName;RC
`W16;VIE;2016-10-26;1000000;A;0545;0540;;N;K;EA;1819;;150;J6; VCR;FRA;DUS;FRA;DUS;HG;HG;J;;;;01MAY2016 0001;;0;Ex Air;OK`
`W16;VIE;2016-10-26;0200000;D;0620;0620;;Y;K;AN;1234;;163;SS; CRL;VIE;ZRHF;VIE;ZRH;AT;AT;C;;;;01MAY2016 1131;;0;Anonymous;OK`
`W16;VIE;2016-10-26;1000000;D;0540;0530;;N;K;EA;3546;;192;NG; LDA;MUC;EDDF;MUC;EDDF;CH;CH;J;;;;01MAY2016 0001;;0;Ex Air;OK`

described in Section 3.8 is applied to match as many turnaround requests as possible. Additionally, also the series information is computed in advance. The conditions for this step are explained in detail in Section 3.4.

The algorithms presented in this work are part of a big software framework. Inter alia, it consists of a graphical user interface used for configuration and visualization of the results. In the course of this software framework, we also developed a view to visualize the series information. To get an impression, Figure 6.1 shows a screenshot of such a series window. The series are visualized as a tree. Each root node represents a multiday series. The next level in this hierarchy corresponds to a single day of operation. Last, each line contains a single request.



SeriesID	GroupID	RowID	DOOP	A/D	Assign Date	Req.Date	Hist.Date	Hist.Stat.	Action Code	Airl.Desig.	Flt.No.	Seats	Op.Code	PRIORITY
59	groups: 6	mvts: 126	12345-7	D		2017-10-29 07:45:00 - 2018-03-23 07:45:00		N		OS	0507		Austrian Airlines	81,838
60	groups: 4	mvts: 20	1--4-67	A		2017-10-29 07:45:00 - 2017-12-02 07:45:00		DFI, F		OS	0836		Austrian Airlines	23,611
	60	*mvts: 5*	------7	*A*				*F*		*OS*	*0836*		*Austrian Airlines*	*11,111*
	740	*mvts: 5*	1------	*A*				*F*		*OS*	*0836*		*Austrian Airlines*	*11,111*
	2744	*mvts: 5*	---4---	*A*				*DFI*		*OS*	*0836*		*Austrian Airlines*	*36,111*
	4123	*mvts: 5*	-----6-	*A*				*DFI*		*OS*	*0836*		*Austrian Airlines*	*36,111*
		19554772	-----6-	A	2017-11-04 07:45:00	2017-11-04 07:45:00	n/a	DFI	K	OS	0836	174	Austrian Airlines	36,111
		19559328	-----6-	A	2017-11-11 07:45:00	2017-11-11 07:45:00	n/a	DFI	K	OS	0836	174	Austrian Airlines	36,111
		19563886	-----6-	A	2017-11-18 07:45:00	2017-11-18 07:45:00	n/a	DFI	K	OS	0836	174	Austrian Airlines	36,111
		19568444	-----6-	A	2017-11-25 07:45:00	2017-11-25 07:45:00	n/a	DFI	K	OS	0836	174	Austrian Airlines	36,111
		19573002	-----6-	A	2017-12-02 07:45:00	2017-12-02 07:45:00	n/a	DFI	K	OS	0836	174	Austrian Airlines	36,111
61	groups: 7	mvts: 147	1234567	A		2017-10-29 07:45:00 - 2018-03-24 07:45:00		F		LG	8851		Luxair	6,04285

Figure 6.1: Tree view, used to visualize the series.

Next, Table 6.3 presents an overview of the data sets used in this work and lists the key points. Columns one and two describe the data sets by listing the airport, the year and whether it contains data for the winter or summer season. Column three shows the overall number of requests present in that season and column four the number of slot series. In general, the summer seasons contain significantly more requests than the winter seasons. Furthermore, there is a noticeable increase of data over the years.

Table 6.2: Brief explanation of fields in the input data.

Field	Description
Season	season of the data set
Airport	coordinated airport of the data set
Date	requested date
DOOP	day of operation (1=Monday, 7=Sunday)
ArrDep	movement type (A=arrival, D=departure)
Time	assigned time (in operational practice)
Req	requested time
Hist	historic time
HistStat	historic status code
ActionCode	action code (in operational practice)
AirlDesig	unique designator for every air carrier
Fltno	flight number
OpeSuffix	suffix of flight number (redundant, not used)
Seats	number of available seats
IATA / ICAO Aircraft Type	international identifier of the aircraft
LastNext / ICAO LastNext	last airport of arrivals, next airport of departures
OrigDest / ICAO OrigDest	original airport of arrivals, final airport of departures
LastNextCountry	last country of arrivals, next country of departures
OrigDestCountry	original country of arrivals, final destination of departures
ServType	service type
TurnOpe	turnaround operator code
TurnServNo	turnaround service number
TD	turnaround days, overnight indicator
EditDate	last modified date
ACReg	aircraft registration, not used
GA/BA	general aviation or business aviation
OpeName	descriptive name of aircraft, not used
RC	reason code

Table 6.3: Data sets used to evaluate the optimization algorithms.

Airport	Season	Nr of requests	Nr of series
Vienna	W17	95,138	1,113
Vienna	S18	178,105	1,691
Vienna	W18	121,271	1,433
Vienna	S19	200,353	1,886
Vienna	W19	125,419	1,500

The optimization algorithms need to consider lots of airport and season dependent parameters and configuration values. For example, the capacity limitations depend not only on the size and resources of the airport, but might also vary from winter to summer season. Furthermore, the priority of requests as well as the turnaround constraints (home carriers) depend on several configuration parameters. All settings and paramters used for the evaluation of the algorithms as presented in the following sections, are provided in appendix A.

6.2 Benchmark System Environment

All tests are performed on a Desktop Computer running Ubuntu Linux 18.04 (KDE neon User Edition 5.16). The system runs on an Intel(R) Core(TM) i7-8550U CPU, 8th generation consisting of 4 cores. Furthermore, the system contains 32GB Ram and a SSD hard disc.

The program was written in C++, conforming to the C++11 standard. For compilation, the gnu GCC compiler 7.4.0 was used. Furthermore, the following additional libraries were used.

- C++ Standard Template Library (STL),
- Boost 1.65 (portable C++ library to extend the standard library),
- Qt 5.12.3 (used for graphical user interface),
- Qwt 6.1.3 (used for plotting),
- and Xerces 3.2 (xml parser, used for logging and debugging).

6.3 Evaluation of the Construction Algorithm

Next, we will analyze the computational results of the construction algorithm described in Section 5.1. The objective function, as defined in Section 3.10 consists of three components, the quantity Γ, the time deviation Δ and the fragmentation Θ.

In order to compare the results of the optimization algorithms, Schedule Coordination Austria provided historical slot assignment data used in operational practice. We will refer to those manually created solutions by the term *reference solution*.

Table 6.4 shows the results for all data sets. Column one shows the season of the data set. Column two contains the running time of the construction heuristic. Columns three shows the number of requests which are not confirmed, but remain unable (equivalent to $|R| - \Gamma$). Column four shows the number of unconfirmed requests in the reference solution (operational practice). Column five contains the time deviation Δ. Again, column six contains the time deviation Δ of the reference solution. Last, column seven shows the fragmentation Θ. Note, that for reasons of comparability a relative fragmentation is used and hence the value of this column depends on both, the reference solution and the solution of the construction algorithm. A negative value means, that the fragmentation is clearly better in the optimized solution, whereas a positive value indicates that the fragmentation of the reference solution is better.

Table 6.4: Computational results of the construction algorithm.

season	run time [s]	$\neg$ pos	$\neg$ pos$_{\text{ref}}$	Δ [min]	Δ_{ref} [min]	Θ
W17	120	380	379	**85,680**	101,595	0.302
S18	300	1,316	1312	**346,730**	398,800	0.675
W18	178	447	442	**188,425**	249,080	0.019
S19	530	35	0	**849,555**	1,002,740	0.373
W19	208	7	0	**211,485**	285,605	0.171

To some extent, the number of unconfirmed requests is equivalent. Little discrepancies occur, because the construction algorithm skips requests which do not belong to any series. Such *ad hoc requests* are usually set after the worldwide slot conference. However, the reference solution contains a few ad hoc requests assigned to a certain time slot. Because of the small amount and the low impact, those discrepancies can be neglected.

Note, that the construction algorithm yields very good results regarding time deviation Δ. For every data set, Δ is lower than in the reference solution. However, in terms of relative fragmentation Θ, tables are turned. In this respect, the reference solution yields better results for every data set.

For the sake of completeness, it should also be mentioned, that we encountered a few assignments in the reference solution not conforming to the constraints described in Chapter 3. In consultation with the coordination authority, it turned out, that almost all those cases can be regarded as highly exceptional situations in which some further considerations, not manageable by software, affects the decision making. Note, that this affects only very few requests, approximately 0.5%. Hence, for reasons of comparability, in those rare situations we allowed the optimization algorithms to use the very same decisions. Furthermore, we even allow the algorithms to skip requests, which are not

Table 6.5: Exceptional assignments in reference solution not conforming to the model.

Season	Nr of requests	Percentage [%]
W17	577	0.6064
S18	799	0.4486
W18	563	0.4642
S19	1,002	0.5001
W19	519	0.4138

assigned in the reference solution. Table 6.5 shows the overall impact for every data set. Column one shows the season, column two the number of affected requests not conforming to the model and column three shows the affected requests as percentage on the whole number of requests.

Comparison to Previous Work

As mentioned in Chapter 2, others already investigated similar problems. Of course, it would be highly desirable to compare the results of different approaches with each other. Unfortunately, a comparison to existing approaches was not possible, though it was intended. For one thing, the data sets are usually highly confidential. Usually they are protected by local coordination authorities. For another, several details regarding the limits, series and turnarounds could not be sorted out entirely.

Runway Utilization

Next, we will take a closer look on the aggregated counts and analyze the runway capacity limitations. For this purpose, we focus on the data set W17. First, Figure 6.2 shows the aggregated, total counts for a time interval of 10 minutes. For reasons of clarity, we show the maxima of the season for every time slot. Additionally, we also show the advanced limit as described in Section 3.7.1. Apparently, the limit is reached several times, but never exceeded.

Furthermore, Figures 6.3 and 6.4 show the aggregated total counts for the time intervals of 20 and 60 minutes. Again, we show the maximal values of the whole season. Of course, the aggregated counts shown here are also implicitely restricted by limits of lower time intervals (e. g. interval of 10 minutes). In particular, the limit of 60 minutes is restricted by many limits of lower intervals. This explains, why it can not be reached in Figure 6.4.

The total counts consist of both, arrivals and departures. Now, Figure 6.5 shows only the departures for the time interval of 60 minutes. On the contrary, Figure 6.6 shows solely the arrivals. One can see, that the peek times of high loads differ slightly. At all times, the aggregated counts clearly comply with the limits.

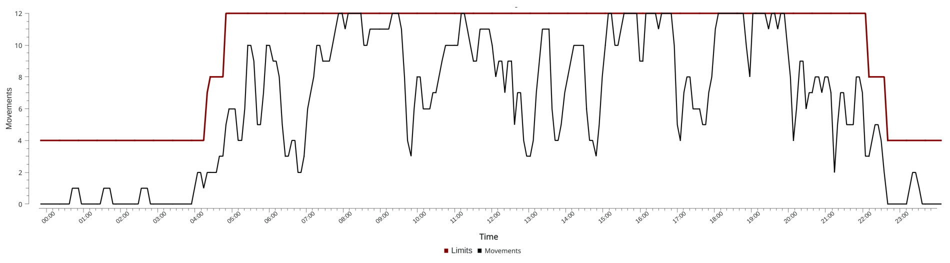


Figure 6.2: W17 – Aggregated counts, 10min, Seasonal Maxima, Total.

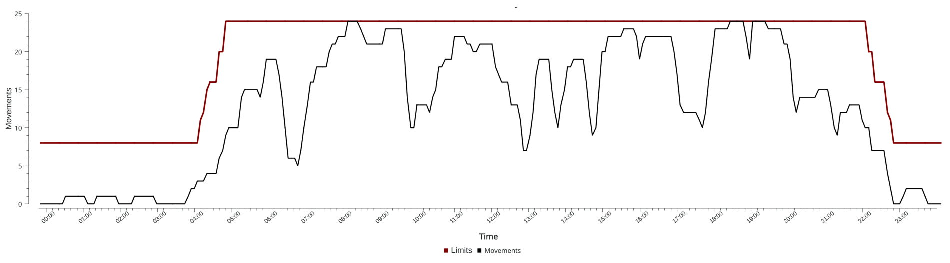


Figure 6.3: W17 – Aggregated counts, 20min, Seasonal Maxima, Total.

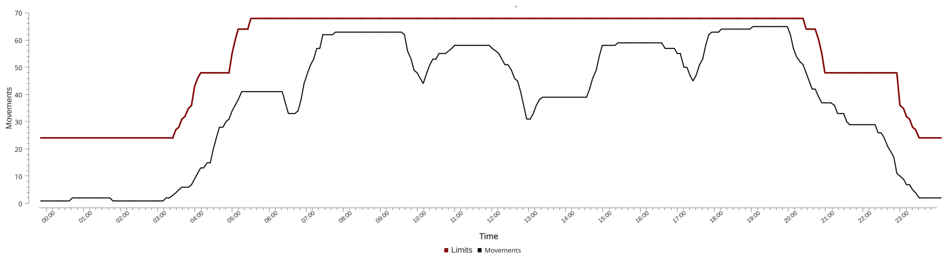


Figure 6.4: W17 – Aggregated counts, 60min, Seasonal Maxima, Total.

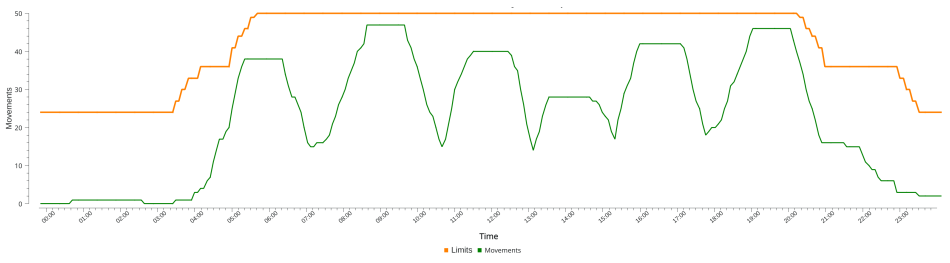


Figure 6.5: W17 – Aggregated counts, 60min, Seasonal Maxima, Departures.

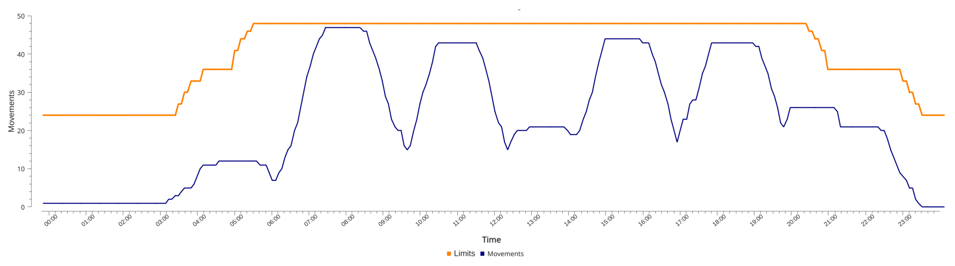


Figure 6.6: W17 – Aggregated counts, 60min, Seasonal Maxima, Arrivals.

6.4 Improvement

Next, the computational results of the improvement algorithms will be discussed. As described in Section 5.2, we deal with multiple objectives. To find a good tradeoff between time deviation and fragmentation, the improvement algorithm approximates the pareto frontier. Figures 6.7, 6.8, 6.9, 6.10 and 6.11 visualize the pareto frontier for each data set. The running time of the improvement algorithms is approximately 1,000 seconds and the maximum number of iterations is set to 1,000. The x axis shows the relative fragmentation and the y axis shows the time deviation. Furthermore, the result of the construction heuristic is also plotted.

In all cases, the improvement algorithm is able find significantly better solutions. As we can see in the plots, the fragmentation can not be improved independently of the time deviation. As expected, better fragmentation leads to higher time deviation. Hence, a tradeoff between both objectives is necessary.

Notice, that in almost every case (except S19), the algorithm finds a solution, which is better than the reference solution in both respects, time deviation and fragmentation. The data set S19, on the other hand, is challenging. Even though the improvement algorithm is not able to find a solution with negative relative fragmentation, a good solution with low time deviation and low (positive) relative fragmentation is found.

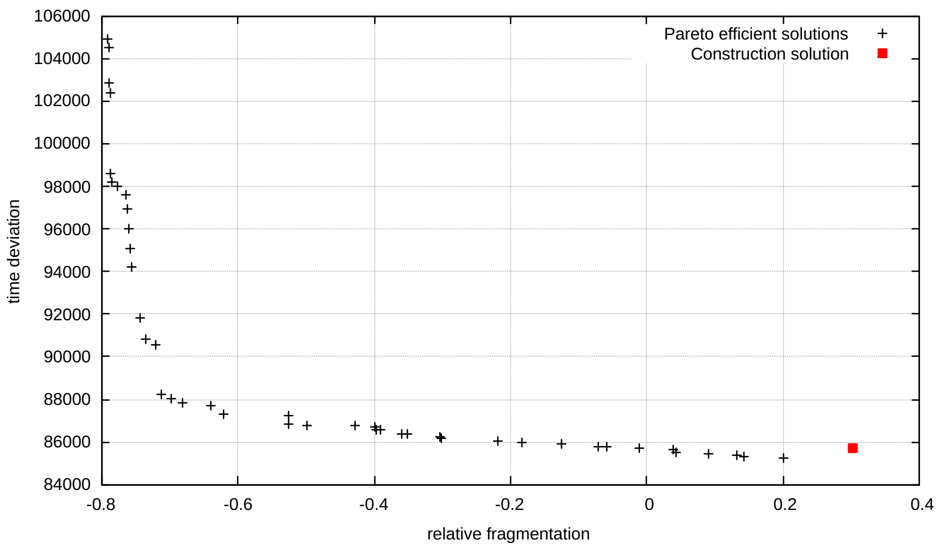


Figure 6.7: Pareto efficient solutions regarding time deviation and fragmentation obtained by the improvement algorithms – W17.

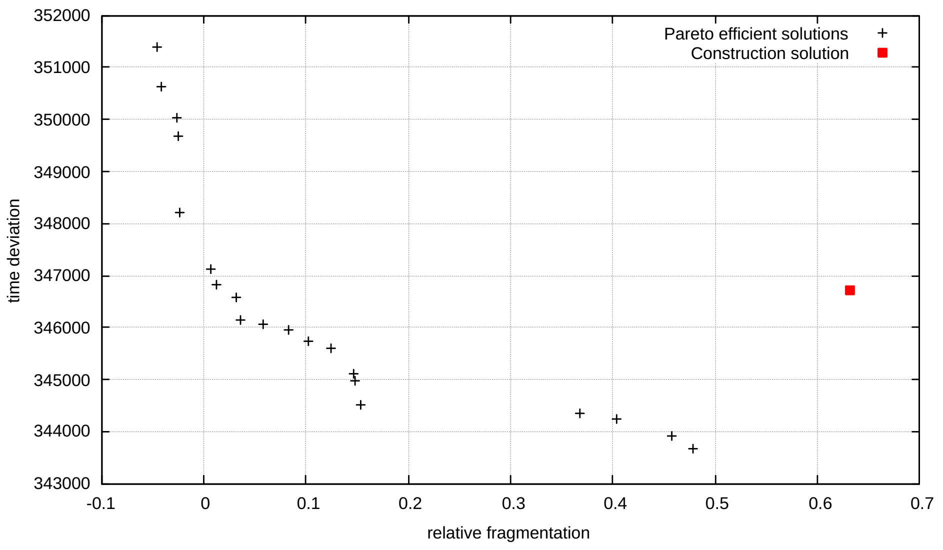


Figure 6.8: Pareto efficient solutions regarding time deviation and fragmentation obtained by the improvement algorithms – S18.

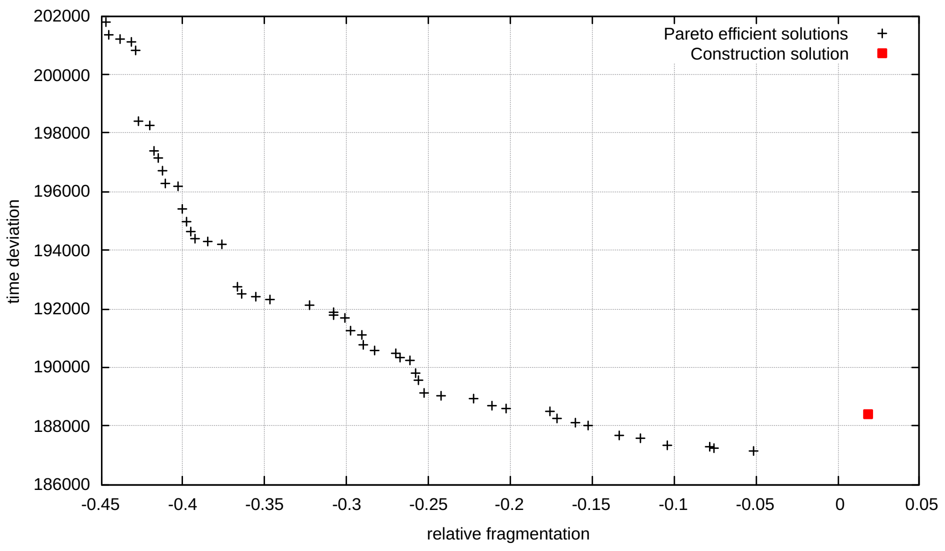


Figure 6.9: Pareto efficient solutions regarding time deviation and fragmentation obtained by the improvement algorithms – W18.

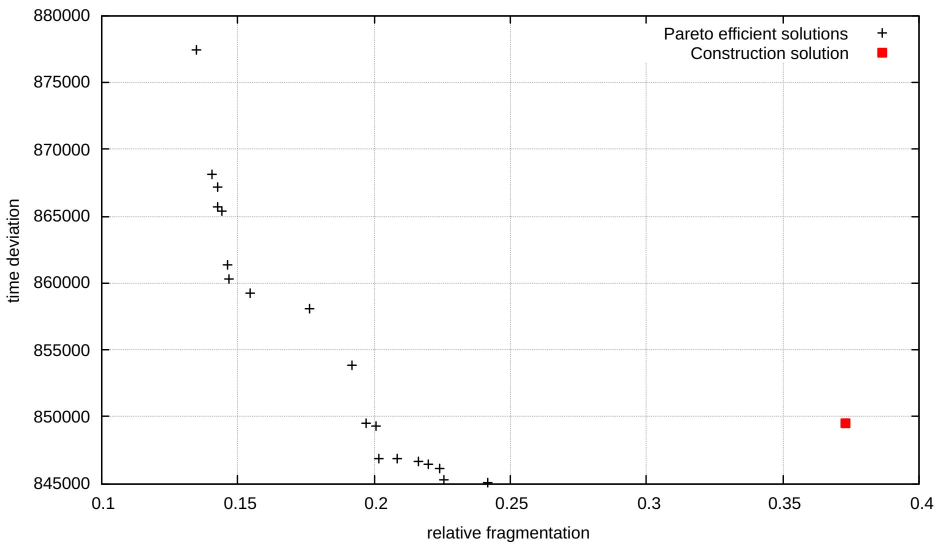


Figure 6.10: Pareto efficient solutions regarding time deviation and fragmentation obtained by the improvement algorithms – S19.

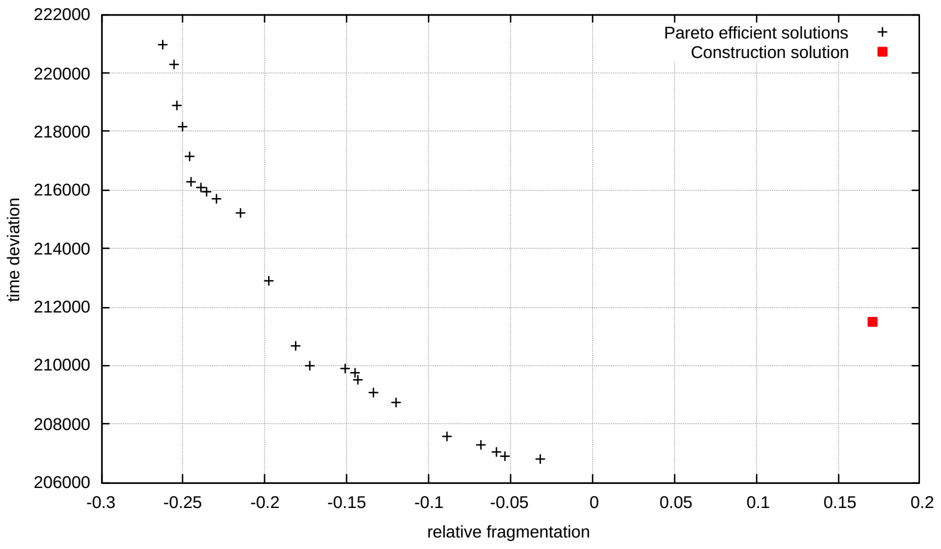


Figure 6.11: Pareto efficient solutions regarding time deviation and fragmentation obtained by the improvement algorithms – W19.

6.5 Best Solutions

The pareto frontier consists of a set of solutions. However, at the end of the day it is necessary to elect a single solution. This step involves human interaction to find a good balance between the different objectives.

Table 6.6 summarizes the best solution found for each data set. Columns one and two show the number of unconfirmed requests of both, the optimized solution and the reference solution. Columns three and four show the time deviation and column five shows the relative fragmentation.

Table 6.6: Best solutions found by the improvement algorithm.

season	$\neg$ pos	$\neg$ pos$_{\text{ref}}$	Δ [min]	Δ_{ref} [min]	Θ
W17	380	379	**86,765**	101,595	**-0.599**
S18	1,316	1,312	**348,220**	398,800	**-0.023**
W18	447	442	**187,695**	249,080	**-0.133**
S19	35	0	**849,555**	1,002,740	0.197
W19	7	0	**209,925**	285,605	**-0.151**

Overall, the algorithms are capable to find good solutions in all respects. In terms of quantity Γ, the results are comparable to existing practice. As shown in Table 6.6, the time deviation Δ is significantly lower than in the reference solution in all test cases. Furthermore, the algorithms yield negative values for the fragmentation Θ for almost all data sets. In case of the summer season S19, the fragmentation Θ is positive, but still very low.

In effect, this means that the found solutions are comparable or even better than the reference solution in all objectives. Hence, the approximation of the pareto frontier permits to find good tradeoffs. Furthermore, the running times of all presented algorithms is remarkably low.

Moreover, the presented algorithms can also be used in operational practice. As further stated in the cooperation statement of Schedule Coordination Austria shown in appendix D, the software framework developed in this work is a valuable innovation and bears great potential to support the coordination authority in future.

6.6 Further Results

Next, we show some more results regarding passenger and apron constraints. In Austria, the Vienna airport is by far the biggest airport with the highest utilization and the most challenging capacity limitations. However, in contrast to runway limitations, passenger and apron restrictions only play a minor role at Vienna.

Hence, to demonstrate compliance with the passenger and apron limitations, we consider some further Austrian airports. Note, however, that we only show a brief overview with the intention to present a proof of concept and not an exhaustive evaluation. In-depth investigations would require much more efforts and are beyond the scope of this work. Moreover, all configuration settings and parameters are highly optimized for the case of Vienna and would require thorough examination.

Now, we consider the Austrian airports of Salzburg and Linz. The configuration settings, as well as the specified capacity limitations are shown in the appendices B and C. Table 6.7 shows the results of the construction heuristic. Columns one and two show the airport and the season, column three shows the running time, columns four and five show the number of unconfirmed requests, columns and six and seven show the time deviation of the solution and of operational practice and column eight shows the relative fragmentation.

Table 6.7: Further results for Austrian airports.

airport	season	run time[s]	$\neg$ pos	$\neg$ pos$_{\text{ref}}$	Δ [min]	Δ_{ref} [min]	Θ
Salzburg	W18	356	17	0	36,620	36,260	**-0.152**
Linz	W18	15	0	0	**0**	110	0

Note, that both airports are very small, notably much smaller than Vienna. In case of Salzburg, the number of requests is 10,832 and in case of Linz it is 2,420. Hence, both data sets are in principal not hard to solve. In particular, the airport of Linz is not utilized to capacity and thus the data set can be solved in several seconds. Consequently, the time deviation Δ is zero meaning that all requests can be confirmed as requested.

Also the relative fragmentation Θ is zero. For the data set of Salzburg, the situation is slightly different. This time, the time deviation Δ is a little bit higher than in the reference solution. Regarding fragmentation Θ on the other hand, the construction heuristic yields a better result.

However, now our intention is not to further analyze the construction heuristic, but to present the results regarding passenger and apron restrictions. For this purpose, we show some selected figures. The number of parking positions is only relevant at the airport of Salzburg. Figure 6.12 shows the total apron counts. As in the following figures, we show the maximum values of the whole season for every time slot. Clearly, the apron limit does not restrict the solution of this data set. At all times, the counts comply with the limit with a big margin of flexibility.

Next, we analyze the passenger counts. Figure 6.13 shows the aggregated counts of arrivals for a time interval of 60 minutes. The departures of the same time interval are shown in Figure 6.14 and the total counts are shown in Figure 6.15. As always, we show the maximal values of the whole season for every time slot. One can see, that the limits are reached several times of the day, but never exceeded. Hence, they are restrictive.

Last, we show the passenger counts for the airport of Linz. Figure 6.16 shows the aggregated total counts for a time interval of 60 minutes. Again, the limits are cleary met and there is lots of margin for further increase.

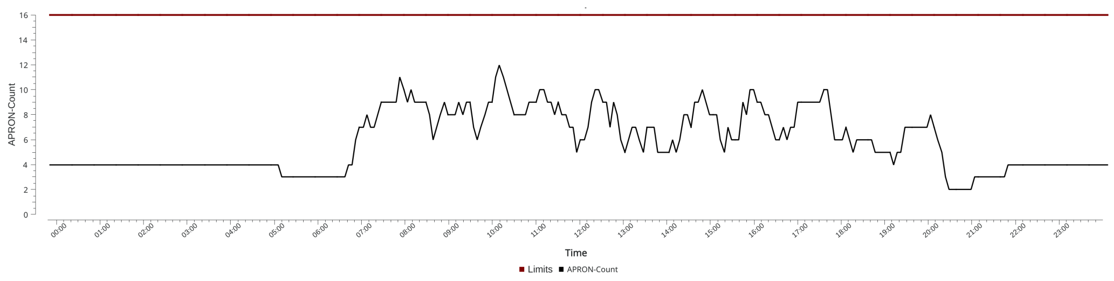


Figure 6.12: SZG18 – Apron counts, Seasonal Maxima, Total.

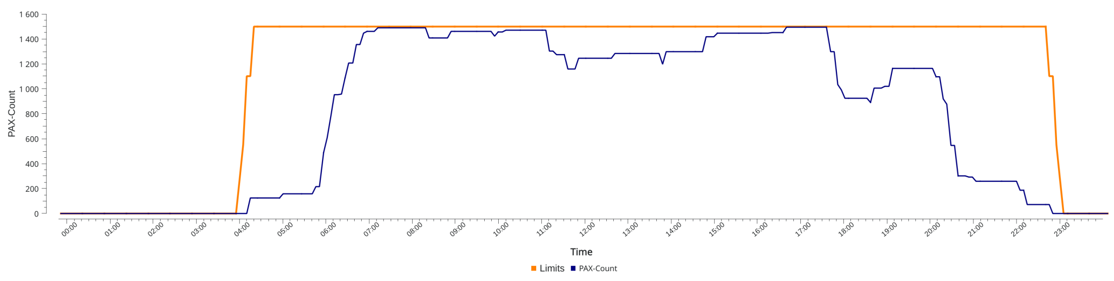


Figure 6.13: SZG18 – Passenger counts, 60min, Seasonal Maxima, Arrivals.

Note, that we encountered some shortcomings regarding the conflict resolvement strategies (see Section 5.1.3), when evaluating the data sets of Salzburg. If necessary, the algorithm

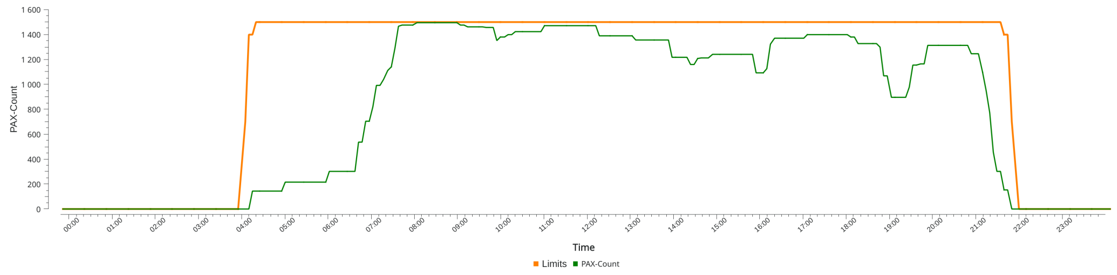


Figure 6.14: SZG18 – Passenger counts, 60min, Seasonal Maxima, Departures.

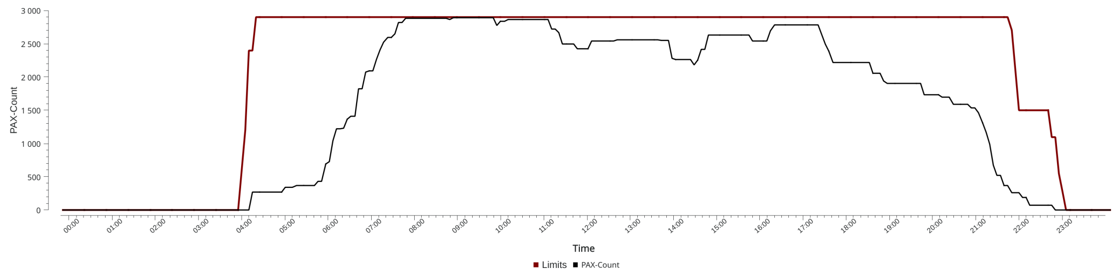


Figure 6.15: SZG18 – Passenger counts, 60min, Seasonal Maxima, Total.

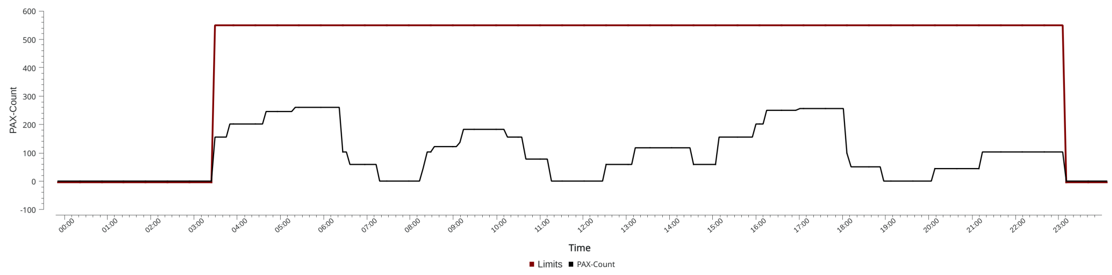


Figure 6.16: LNZ18 – Passenger counts, 60min, Seasonal Maxima, Total.

tries to move a single request or even a whole series to another time. However, it does not respect the turnaround requests. In some situations, it seems worthwhile to move an already assigned request *and its turnaround request* to a new time slot. This allows for even more flexibility. Preliminary implementation and first tests seem promising. However, a stable implementation complying with previous results requires much more work.

CHAPTER 7

Conclusion

The main focus of this work was to solve the airport slot allocation problem and create initial flight schedules in a fully automated way. In previous work, exact integer programming models covering small and medium airports have been presented. In contrast, we proposed heuristic optimization methods capable to solve even large data sets in reasonable running times.

A major part of this thesis deals with the constraints and regulations determining the airport slot allocation. A highly configurable framework is presented to support runway capacity limitations as well as passenger and apron limitations. Furthermore, a flexible priority model is introduced, which also conforms to the IATA guidelines. Additionally, turnaround constraints are worked out to ensure certain ground times of the aircrafts. Furthermore, we presented the concept of fragmentation to respect and promote homogenous assignments of slot series as far as possible and at the same time support inhomogenous assignments as well.

Within this thesis, we developed heuristic optimization algorithms respecting all constraints and regulations. The proposed algorithms consist of a construction heuristic and subsequent improvement methods. They are capable of creating initial flight schedules in remarkable running times. Furthermore, to cope with multiple objectives simultaneously, the algorithms approximate pareto efficient solutions.

To evaluate and benchmark the algorithms, Schedule Coordination Austria provided several data sets with real historic data together with reference solutions of operational practice. Based on these data sets, we showed, that the proposed algorithms are able to solve the airport slot allocation problem within low running times. Furthermore, the automatically created solutions can compete with current operational practice. The improvement methods achieve good results regarding low time deviation, as well as good fragmentation. For almost all data sets, the algorithms can even find a solution, which is better than the reference solution in both objectives.

According to *Schedule Coordination Austria*, the presented algorithms are a valuable tool and can improve current operational practice significantly. Furthermore, the presented algorithms are very configurable and highly extensible to future needs.

However, the downside of this flexible approach is the high effort needed to maintain parameter settings, as well as to adapt configuration settings to future changes. In particular, when integrating new data sets with possibly new historic status codes, operator codes, and so on, error analysis can become tough and tiring.

In the last few years, airport slot allocation rised to a promising field of research and further work is to be expected. There is already a strong tendency to solve airport slot allocation problems of increasing size. Thus, it seems very likely, that more heuristic approaches will show up.

Even though, deep insights and remarkable computational results have been gained by this work, there are also lots of possibilities for further improvement. A very important aspect of the presented approach is fragmentation, a concept which has not been addressed yet in the way shown in this work. Several ways to measure the impact of inhomogenous assignments are introduced. Since this work has a strong focus on practical applicability, relative fragmentation was used to evaluate the algorithms. However, this is not the only way to go. Further efforts to formulate an estimator for the fragmentation seem promising. At the end of the day, optimization methods heavily depend on good performance indicators.

Furthermore, another key issue of airport slot allocation are the (multiday) series of slots. The algorithms presented in this thesis work on basis of single requests. Of course, (multiday) series are still respected and homogenous assignments are preferred. However, a bottom-up method seems also plausible. In this manner, one could write an algorithm working solely on (multiday) series of slots and still allowing for inhomogenous assignments. For example, a possible way to implement such a strategy could be to split a (multiday) series on demand into several sub-series in case a homogenous assignment is not possible. Then the algorithm can possibly find homogenous assignments for the sub-series.

Last, the algorithms described in this work can also be further improved. For one thing, the improvement methods could be extended. The proposed methods in this work focus on fragmentation. However, further neighborhoods targeting time deviation or even quantity of confirmed requests pledge to be fruitful. Ultimately, a variable neighborhood search would be desirable. However, the biggest challenge here is to find good improvement steps.

For another, the conflict resolvement strategy can be futher extended. If all other strategies fail, the algorithms within this work try to move an already assigned request to another time slot. However, such a request also depends on its turnaround request. Hence, it would be worth a try to even move the turnaround request to a new time slot. Preliminary work showed, that such an extension of the conflict resolvement could be lucrative. However, further work is needed to implement and extensively evaluate the working draft.

Despite those potential improvements, the algorithms have shown to be applicable and helpful for practical applications. Due to the short running times the algorithms are not only useful for the initial creation of flight schedules, but also for uses cases in analytics like studies of the impact of changes of available airport and runway capacities.

APPENDIX A

Model Parameters of Vienna Airport

The software framework developed in this work is highly configurable. This Chapter shows the settings and model parameters used throughout the work to develop and evaluate the algorithms.

A.1 Priorities

First, configuration settings regarding the priority model as described in Chapter 3 are shown. Table A.1 shows the priority $\mathrm{pr}_{\mathrm{hist}}(h)$ depending on the historic status code h of a request. Column one lists the historic status code, column two shows the assigned priority (high priority is expressed by low numbers) for the summer periods, column three shows the priority for the winter periods and the remaining columns state the allowed times - a tick in column three indicates that all time slots between the historic time and the request time are possible and column four states an allowed timespan relative to the request time.

Next, we show the mapping of service types to priorities. Table A.2 lists the priority values for every service type $s \in S$ used in this work.

Table A.3 shows the weighting coefficients c_1, c_2, c_3 and the gain factor ξ used to mix the different priority factors.

A.2 Turnaround Parameters

As described in Section 3.8 of Chapter 3 the arrivals and departures are related to each other by turnaround constraints. However, those rules do not apply for home carriers.

Table A.1: Priority classes and configuration values.

historic status code	priority		in-between	timespan [min]
	summer	winter		
B	60	60		180
BCL	20	20		-
BCLT	20	20		-
BCIT	30	30	✓	-
BCLX	20	20		-
BCR	25	25	✓	-
BCRT	20	25	✓	-
BF	10	10		-
CI	30	20	✓	-
CIT	30	30	✓	-
CIX	30	30		60
CL	15	15		-
CLX	15	15		-
CLT	20	20		-
CR	25	25	✓	-
CRT	20	25	✓	-
DFI	35	35		100
F	10	10	✓	-
FI	25	25		60
N	85	85		240
V	50	50		180
Y	80	80		180

Table A.4 shows a list of home carriers used in Vienna. For those operators no turnaround information is used.

For refueling, cleaning, etc. it is necessary to respect a certain ground time. However, in many cases the ground times as requested by the initial submissions can not be met. At least slight adaptions are inevitable in practice. To account for current practice, we allow slight deviations depending on the initially requested ground time as shown in Table A.5. Column one shows the requested ground time gnd_time(r, r') in minutes and column two shows the allowed deviations in minutes.

A.3 Seasonal Night Limit

In Vienna, the number of movements during the night is restricted by a seasonal maximum value. Hence,

$$\text{seasonal}(\text{Tr}_{\text{night}}) \leq \lambda_{\text{night}} \tag{A.1}$$

Table A.2: Priorities of the service types.

service type	priority
J	-0.40
C	-0.35
G	-0.28
F	-0.19
H	-0.18
A	-0.09
M	-0.08
P	0
W	0.11
I	0.12
E	0.13
X	0.21
O	0.22
N	0.31
D	0.32
U	0.33
K	0.41
T	0.42
Y	0.50
Z	0.50

Table A.3: Priority weighting coefficients and gain factor.

coefficient	value
c_1	0.3
c_2	0.6
c_3	0.1
ξ	10

.

For the winter period, λ_{night} is equal to 967 and for the summer period a value of 2600 is used.

A.4 Runway Limits

Next, we show the runway limits of the Vienna airport. Table A.6 shows the time intervals in use. Column one lists the designator of the time interval, column two the timespan and column three the very same timespan in minutes. Furthermore, Table A.7 shows

Table A.4: Vienna home carriers, which are not considered for turnaround constraints.

operator name	IATA code
Austrian Airlines	OS
Air Berlin	AB
Niki Luftfahrt	HG
Lauda Motion	OE
Wizz Air	W6
Eurowings	EW

Table A.5: Allowed ground time deviations.

gnd_time(r, r')	deviation
$0 \ldots 35$	$0 \ldots 10$
$35 \ldots 45$	$-5 \ldots 20$
$45 \ldots 55$	$-5 \ldots 25$
$55 \ldots 95$	$-10 \ldots 25$
$95 \ldots 120$	$-30 \ldots 35$
$120 \ldots 235$	$-30 \ldots 30$
else	0.15 gnd_time(r, r')

the capacity limits of the runway for the different timeranges and intervals. Column one contains the time range, column two the interval and columns three, four and five the maximum limit for every movement type.

Table A.6: Interval lengths.

interval	timespan	timespan [min]
δ_1	1	5
δ_2	2	10
δ_3	6	30
δ_4	12	60

Table A.7: Runway limits at Vienna airport.

time range	interval	arrivals	departures	total
Tr_{day}	δ_1	5	5	7
	δ_2	9	9	12
	δ_3	24	25	34
	δ_4	48	50	68
$\mathrm{Tr}_{\text{evening}}$	δ_1	5	5	7
	δ_2	7	7	12
	δ_3	18	18	26
	δ_4	36	36	48
$\mathrm{Tr}_{\text{evening-shoulder}}$	δ_1	5	5	5
	δ_2	7	7	8
	δ_3	18	18	24
	δ_4	36	36	24
$\mathrm{Tr}_{\text{night}}$	δ_1	2	2	2
	δ_2	4	4	4
	δ_3	12	12	12
	δ_4	24	24	24
$\mathrm{Tr}_{\text{morning-shoulder}}$	δ_1	5	5	5
	δ_2	7	7	8
	δ_3	18	18	24
	δ_4	36	36	24
$\mathrm{Tr}_{\text{morning}}$	δ_1	5	5	7
	δ_2	7	7	12
	δ_3	18	18	26
	δ_4	36	36	48

APPENDIX B

Model Parameters of Salzburg Airport

Further results and discussions deal with the Salzburg airport. In particular apron and passenger limits are demonstrated. Hence, we consider now the configurations and parameter settings used for those evaluations.

First, Table B.1 shows the priority settings for every status code. Column one lists the historic status code, column two shows the priority and columns three and four provide insight about the allowed times.

Table B.1: Priorty configurations for Salzburg.

historic status code	priority	in-between	timespan [min]
CLT	20		-
CLS	20		-
CLTS	20		-
CLX	15		-
CR	25	✓	-
CRT	25	✓	-
CRTS	30	✓	-
DFI	35		100
F	10		-
FI	25		60
N	85		240

Next, we show the runway limits in Table B.2. Column one shows the time range, column two the time interval and columns three, four and five show the maximum values for arrivals, departures and total (both).

Table B.2: Runway limits for Salzburg.

Timerange	interval [min]	arrivals	departures	total
05:00 - 20:55	5	3	3	3
	10	5	5	5
	20	8	8	8
	60	10	10	20
21:00 - 22:00	5	2	0	2
	60	10	0	10

Table B.3 shows the limits for the passenger counts. Column one shows the time range, column two the time interval and columns three, four and five show the maximum values for arrivals, departures and total (both).

Table B.3: Passenger limits for Salzburg.

Timerange	interval [min]	arrivals	departures	total
05:00 - 20:55	10	550	700	1,200
	60	1,500	1,500	2,900
21:00 - 22:00	10	550	0	550
	60	1,500	0	1,500

Last, Table B.4 shows the apron limits. Column one shows the maximum number of small parking positions, columns two and three show the maximum number of medium and large parking positions and column four shows the total number of available parking positions.

Table B.4: Apron limits for Salzburg.

Small	Medium	Large	Total
2	10	4	16

APPENDIX C

Model Parameters of Linz Airport

Furthermore, we show the configuration values used for the airport of Linz. In the same manner as before, this airport is regarded in further discussions in order to provide a proof of concept regarding fulfilment of passenger limitations.

In this case, all requests have the same historic status code, that is N. The allowed time is a timespan of 120 minutes around the requested time.

Next, we show the runway limits in Table C.1. Column one contains the timerange, column two the interval and columns three, four and five contain the maximum values for arrivals, departures and total (both).

Table C.1: Runway limits for Linz.

Timerange	interval [min]	arrivals	departures	total
During opening hours	5	2	2	2
	15	4	4	6

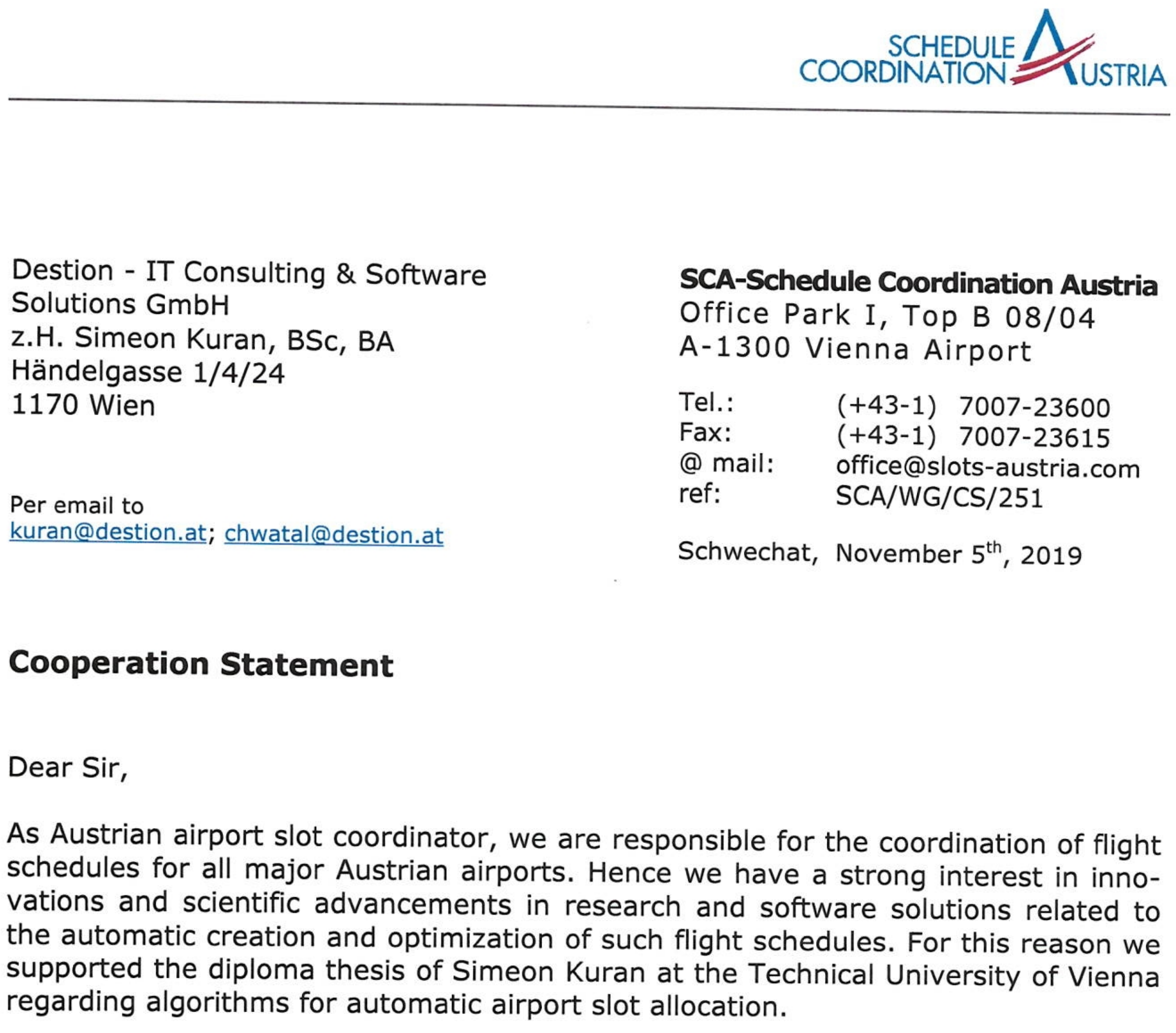
Last, Table C.2 shows the passenger limits. Column one shows the timerange, column two the time interval and column three the maximum values for departures. Note, that for arrivals and totals no limit is defined.

Table C.2: Passenger limits for Linz.

Timerange	interval [min]	departures
During opening hours	20	350
	60	550

APPENDIX D

Cooperation Statement



SCHEDULE COORDINATION AUSTRIA

Destion - IT Consulting & Software Solutions GmbH
z.H. Simeon Kuran, BSc, BA
Händelgasse 1/4/24
1170 Wien

Per email to
kuran@destion.at; chwatal@destion.at

SCA-Schedule Coordination Austria
Office Park I, Top B 08/04
A-1300 Vienna Airport

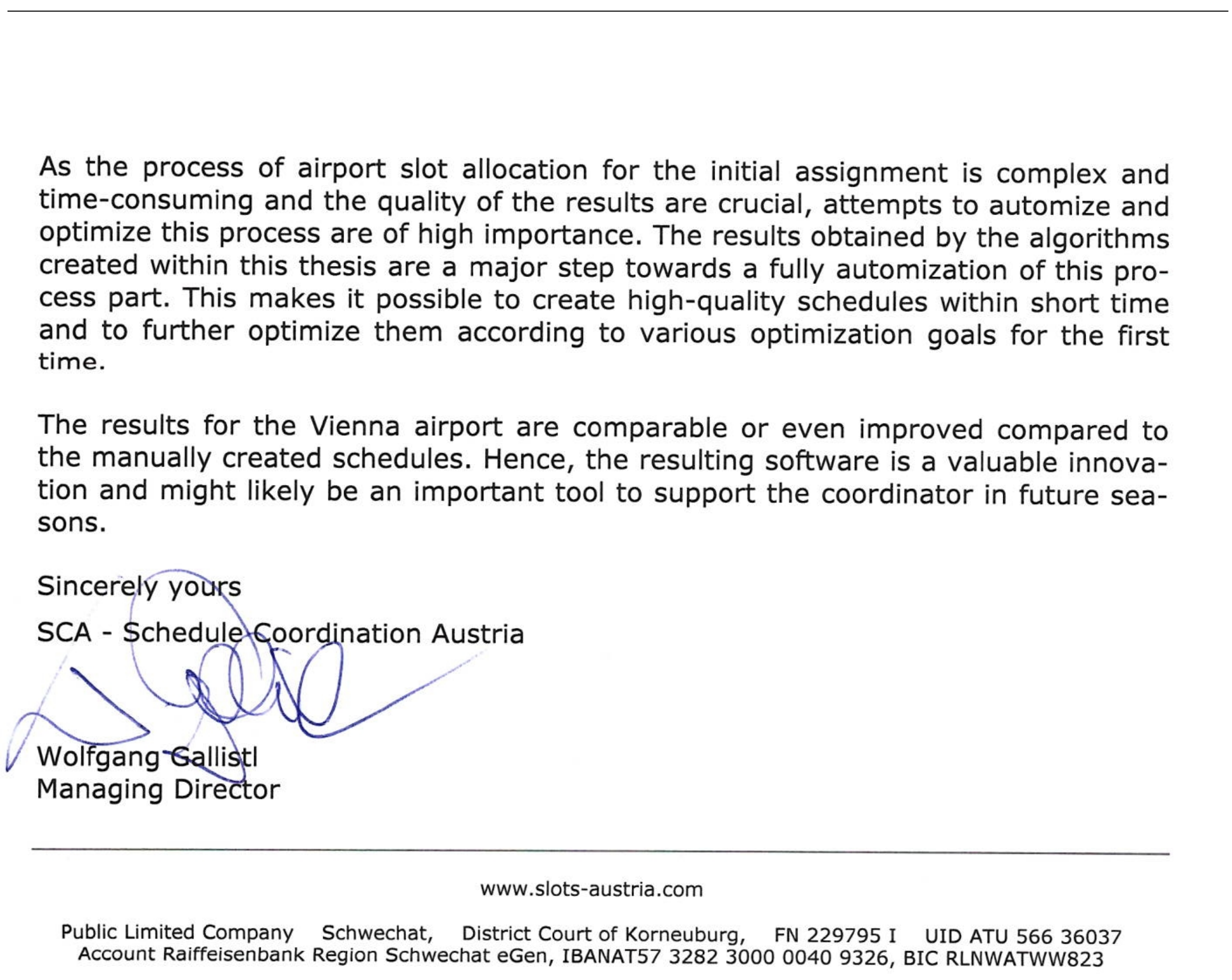
Tel.:	(+43-1) 7007-23600
Fax:	(+43-1) 7007-23615
@ mail:	office@slots-austria.com
ref:	SCA/WG/CS/251

Schwechat, November 5th, 2019

Cooperation Statement

Dear Sir,

As Austrian airport slot coordinator, we are responsible for the coordination of flight schedules for all major Austrian airports. Hence we have a strong interest in innovations and scientific advancements in research and software solutions related to the automatic creation and optimization of such flight schedules. For this reason we supported the diploma thesis of Simeon Kuran at the Technical University of Vienna regarding algorithms for automatic airport slot allocation.



As the process of airport slot allocation for the initial assignment is complex and time-consuming and the quality of the results are crucial, attempts to automize and optimize this process are of high importance. The results obtained by the algorithms created within this thesis are a major step towards a fully automization of this process part. This makes it possible to create high-quality schedules within short time and to further optimize them according to various optimization goals for the first time.

The results for the Vienna airport are comparable or even improved compared to the manually created schedules. Hence, the resulting software is a valuable innovation and might likely be an important tool to support the coordinator in future seasons.

Sincerely yours

SCA - Schedule Coordination Austria

[illegible]

Wolfgang Gallistl
Managing Director

www.slots-austria.com

Public Limited Company Schwechat, District Court of Korneuburg, FN 229795 I UID ATU 566 36037
Account Raiffeisenbank Region Schwechat eGen, IBANAT57 3282 3000 0040 9326, BIC RLNWATWW823

List of Figures

List of Tables

List of Algorithms

Index

Bibliography

[ACK15] S. Pirkwieser A. Chwatal and S. Kuran. Modellbeschreibung: automatisierte und optimierte Slot-Vergabe mittels des Destion Airport Slot Planners. Technical report, Destion IT Consulting & Software Solutions GmbH, 03 2015.

[Agm] Shmuel Agmon. The relaxation method for linear inequalities. *Canadian Journal of Mathematics, 1954, Vol.6, pp.382-392.*

[AOR93] Giovanni Andreatta, Amedeo R. Odoni, and Octavio Richetta. Models for the ground holding problem. 1993.

[Ben18] Una Benlic. Heuristic search for allocation of slots at network level. *Transportation Research Part C: Emerging Technologies*, 86:488 – 509, 2018.

[Bru02] Jan K. Brueckner. Airport congestion when carriers have market power. *American Economic Review*, 92(5):1357–1375, December 2002.

[Bru09] Jan K. Brueckner. Price vs. quantity-based approaches to airport congestion management. *Journal of Public Economics*, 93(5):681 – 690, 2009.

[BZ10] Leonardo J. Basso and Anming Zhang. Pricing vs. slot policies when airport profits matter. *Transportation Research Part B: Methodological*, 44(3):381 – 391, 2010. Economic Analysis of Airport Congestion.

[CL19] Achim I. Czerny and Hao Lang. A pricing versus slots game in airport networks. *Transportation Research Part B: Methodological*, 125:151 – 174, 2019.

[CLN14] L. Corolli, G. Lulli, and L. Ntaimo. The time slot allocation problem under uncertain capacity. *Transportation Research Part C: Emerging Technologies*, 46:16 – 29, 2014.

[Dan95] Joseph I. Daniel. Congestion pricing and capacity of large hub airports: A bottleneck model with stochastic queues. *Econometrica*, 63(2):327–370, 1995.

[DFJ54] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

[EUR18] EUROCONTROL. European Aviation in 2040; Challenges of Growth. Brussels, Belgium, September 2018. `http://www.eurocontrol.int/articles/challenges-growth`.

[GP19] Michel Gendreau and Jean-Yves Potvin. *Handbook of metaheuristics.* International series in operations research & management science ; 272. Springer, Cham, third edition edition, 2019.

[IAT19a] IATA. Iata industry statistics. 2019. `https://www.iata.org/publications/economics/Reports/Industry-Econ-Performance/Airline-industry-economic-performance-Jun19-data-tables.pdf`.

[IAT19b] IATA. Worldwide Slot Guidelines. 2019. `https://www.iata.org/policy/slots/Documents/wsg-edition-9-english-version.pdf`.

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming-ii. *Combinatorica*, 4:373–395, 12 1984.

[KE95] J Kennedy and R Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4. IEEE, 1995.

[Kel99] C. T. Kelley. *Iterative Methods for Optimization.* Society for Industrial and Applied Mathematics, 1999.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[Koe07] D. Koesters. Airport scheduling performance – an approach to evaluate the airport scheduling process by using scheduled delays as quality criterion. volume Proceedings of the Air Transport Research Society (ATRS) Annual World Conference, June 21–23, Berkeley, US, 2007.

[LD60] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[NM65] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical optimization.* Springer series in operations research and financial engineering. Springer, New York, NY, 2. ed. edition, 2006.

[Par93] European Parliament. Council regulation (eec) no 95/93 of 18 january 1993 on common rules for the allocation of slots at community airports. volume L 014, pages pp. 0001 – 0006. Official Journal of the European Union, 1993.

[Par04] European Parliament. Council regulation (eec) no. 793/2004 of april 2004 amending council regulation (eec) no. 95/93 on common rules for the allocation of slots at community airports. volume L138, page pp. 50–60. Official Journal of the European Union, 2004.

[PBCP17] Paola Pellegrini, Tatjana Bolić, Lorenzo Castelli, and Raffaele Pesenti. Sosta: An effective model for the simultaneous optimisation of airport slot allocation. *Transportation Research Part E: Logistics and Transportation Review*, 99:34 – 53, 2017.

[PV04] Eric Pels and Erik T. Verhoef. The economics of airport congestion pricing. *Journal of Urban Economics*, 55(2):257 – 277, 2004.

[RJA] Nuno Antunes Ribeiro, Alexandre Jacquillat, and António Pais Antunes. A large-scale neighborhood search approach to airport slot allocation. Technical report, CITTA, Department of Civil Engineering, University of Coimbra, 3030-788 Coimbra, Portugal.

[RJA$^+$18] Nuno Antunes Ribeiro, Alexandre Jacquillat, António Pais Antunes, Amedeo R. Odoni, and João P. Pita. An optimization approach for airport slot allocation under IATA guidelines. *Transportation Research Part B: Methodological*, 112(C):132–156, 2018.

[RJAO19] Nuno Antunes Ribeiro, Alexandre Jacquillat, António Pais Antunes, and Amedeo Odoni. Improving slot allocation at level 3 airports. *Transportation Research Part A: Policy and Practice*, 127:32 – 54, 2019.

[Tar71] R. Tarjan. Depth-first search and linear graph algorithms. In *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, pages 114–121, Oct 1971.

[Ver10] Erik T. Verhoef. Congestion pricing, slot sales and slot trading in aviation. *Transportation Research Part B: Methodological*, 44(3):320 – 329, 2010. Economic Analysis of Airport Congestion.

[Zho12] M. Zhong. *Models and solution algorithms for equitable resource allocation in air traffic flow management.* dissertation, University of Maryland, 2012.

[ZMA17] Konstantinos G. Zografos, Michael A. Madas, and Konstantinos N. Androutsopoulos. Increasing airport capacity utilisation through optimum slot scheduling: review of current developments and identification of future needs. *Journal of Scheduling*, 20(1):3–24, Feb 2017.

[ZSM12] Konstantinos G. Zografos, Y. Salouras, and Michael A. Madas. Dealing with the efficient allocation of scarce resources at congested airports. *Transportation Research Part C: Emerging Technologies*, 21(1):244 – 256, 2012.