Computers & Operations Research 177 (2025) 106972

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

An efficient solution methodology for the airport slot allocation problem with preprocessing and column-and-row generation

Paula Fermín Cueto *, Sergio García, Miguel F. Anjos

School of Mathematics and Maxwell Institute for Mathematical Sciences, University of Edinburgh, Edinburgh, EH9 3FD, United Kingdom

ARTICLE INFO

Keywords:
Airport slot allocation
Integer programming
Column generation
Preprocessing
Scheduling

ABSTRACT

Airport coordination is a demand control mechanism that maximizes the use of existing infrastructure at congested airports. Aircraft operators submit a list of regular flights that they wish to operate over a five to seven-month period and a designated coordinator is responsible for allocating the available airport slots, which represent the permission to operate a flight at a specific date and time. From an optimization perspective, this problem is a special class of the Resource Constrained Project Scheduling Problem where the objective is to minimize the difference between the allocated and requested slots subject to airport capacity constraints and other operational restrictions. Most studies on the topic focus on developing complex models and fast heuristics. Little attention has been paid to exact methods despite their potential to obtain higher quality solutions with better airline acceptability and fewer slot rejections. In this paper, we present Caracal, an efficient column-and-row generation algorithm to solve the single airport slot allocation problem. We also present a problem-specific preprocessing scheme that can identify more redundant constraints and variables than a commercial solver in a fraction of the time. We propose a novel formulation to model historic overages in Level 3 airports, and we find optimal or near optimal solutions to instances originating from practical slot allocation models and real data from UK airports coordinated by Airport Coordination Limited significantly faster than the best exact method in the literature to date. We also conduct experiments on a set of synthetic, realistic instances that we include in this paper, along with the code to generate them, to facilitate benchmarking of slot allocation software.

1. Introduction

Demand often exceeds capacity at congested airports. For this reason, airlines and other aircraft operators need to be granted permission to operate a flight at a specific date and time. This permission is known as a slot and each airport that is congested enough has a designated slot coordinator who is responsible for allocating slots in a transparent, unbiased manner, considering the interests of all the relevant stakeholders. Outside the United States, coordinators follow the World Airport Slot Guidelines (WASG), set out by the International Air Transport Association (IATA, 2024), as well as local regulations, such as the Council regulation (EEC) no. 95/93 (European Commission, 1993) in Europe.

The allocation process works as follows. The year is divided into a summer and a winter season, which span seven and five months, respectively. Several months before the start of each season, airports declare their capacity limits, the most common of which are maximum runway and terminal throughputs, expressed in numbers of flights and passengers, respectively. Shortly after, airlines submit a list of regular flights that they would like to operate during the season. The coordinator must then make adjustments to the requested services to ensure the capacity is not exceeded. These adjustments may include offering alternative slots, which must be as close as possible to the requested times, or even denying permission to operate in certain cases. The result of this initial coordination is then disclosed to the airlines, which have some flexibility to request changes, return unwanted slots, or even swap slots with other airlines.

Airports require different levels of coordination depending on how congested they are. Level 1 airports are those where the capacity can accommodate all the demand without intervention, and are therefore not relevant for this work. Level 2 airports are airports with potential for congestion during peak periods, and Level 3 airports are those where the demand exceeds the available capacity significantly. The process described above applies to these last two categories. In addition, in Level 3 airports, IATA mandates that coordinators consider priority levels within the requested operations, and distribute the available slots sequentially: historic requests are allocated first, followed by changes

* Corresponding author.
E-mail address: paula.fermin@ed.ac.uk (P. Fermín Cueto).

https://doi.org/10.1016/j.cor.2024.106972
Received 11 May 2024; Received in revised form 26 November 2024; Accepted 31 December 2024
Available online 6 January 2025
0305-0548/© 2025 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

to historic slots, new entrants, and, lastly, other requests.

Slot coordinators may need to consider operational restrictions such as minimum times on the ground between arriving and departing flights operated by the same aircraft (known as turnaround constraints), or secondary objectives, such as balancing different types of markets or encouraging airline competition. For a more detailed description of the slot allocation process we refer the reader to the WASG (IATA, 2024).

Slot coordinators use specialized software to perform the initial coordination, but these support tools have limited optimization capabilities and coordinators need to make adjustments manually, resulting in a time-consuming process. It has been shown that optimization algorithms have great potential for improving current practice. Ribeiro et al. (2018) analyzed several Portuguese airports and reported potential improvements in solution quality with respect to the slot coordinator's solutions between 4.7% and 27%. Zografos et al. (2012) reported even larger improvements of up to 95% in a study that analyzed the case of three regional Greek airports.

The focus of slot allocation papers is usually placed on modeling the problem. Industry guidelines are ambiguous in the way they define the allocation criteria, and different slot coordinators, as well as different researchers, interpret these rules differently. This has resulted in a wide variety of complex optimization models in which the authors propose their own interpretation of the industry guidelines.

A smaller body of work looks at the slot allocation problem from an algorithmic perspective. In those studies, heuristics are clearly favored over exact algorithms. Zografos et al. (2016) stated that the computational results of exact methods are not encouraging and that trade-offs between complexity and solution quality must be investigated. Subsequent studies have not challenged this assumption and various heuristic methods have been proposed to solve real-world instances.

In our view, the search for optimal solutions merits further investigation: better solution quality translates into fewer rejections (requests that are denied permission to operate) and/or schedule displacements (requests that are offered alternative flight times), which in turn lead to a maximized use of airport infrastructure.

In this paper we focus on the algorithmic side of the problem and show that it is possible to find optimal solutions for real-size problems quickly and with modest memory requirements. We present a column-and-row generation algorithm that starts with a reduced set of variables and constraints and increases the problem size dynamically until a solution is obtained that is feasible for the complete problem. We also present a series of fast preprocessing steps that can be used in conjunction with any solution algorithm to reduce the size of the slot allocation model substantially.

One important aspect hindering the research on this topic is the absence of publicly available data. Computational studies are typically based on data sets from a handful of airports and this data is rarely published, which makes identifying the state of the art an impossible task. There is a need to provide researchers with a large collection of test instances to facilitate benchmarking of slot allocation software. A secondary objective of this work is to develop a procedure to generate realistic synthetic data sets and to make this procedure and the resulting data publicly available.

We build our algorithm using a basic formulation with the elements that are included in virtually every model in the literature and present a computational study using synthetic instances. Then, we show the flexibility of this algorithm by applying it to several variants of the slot allocation problem, and we evaluate the performance on practical slot allocation models using real data of 7 UK airports coordinated by Airport Coordination Limited (ACL), the largest slot coordinator in the world. One of these models is a formulation we propose for dealing with historic overages. These are situations where the capacity limits can be violated if only historic slots are allocated during a certain congested period.

The remainder of this paper is structured as follows: a discussion on the studies relating to this work is provided in Section 2. Definitions, notation, and the mathematical formulation are introduced in Section 3. Our preprocessing scheme is presented in Section 4, followed by a description of the column-and-row generation algorithm in Section 5. Details of the synthetic data generator are given in Section 6. Computational results using the basic slot allocation model and synthetic data are presented in Section 7. Modeling extensions and computational experiments using UK airport data are presented in Section 8, followed by conclusions in Section 9.

2. Related work

The complexity of the slot allocation problem is exacerbated by the ever-expanding list of coordination parameters and the lack of clarity around some of the allocation criteria (Odoni, 2020). This has led to the development by academics of increasingly complex integer programs that incorporate new components or propose a different interpretation of an existing one. We refer the reader to Zografos et al. (2016) for a comprehensive review of different slot allocation models until 2016. In recent years, the most popular topics are the inclusion of secondary objectives, such as preferences and fairness measures (Zografos and Jiang, 2019; Fairbrother et al., 2019; Katsigiannis and Zografos, 2023), and the modeling of more complex capacity constraints such as apron constraints (Ribeiro et al., 2019) and terminal allocation (Katsigiannis and Zografos, 2021).

Most of these integer programs build on the time-indexed formulation presented by Zografos et al. (2012). Time-indexed formulations yield tight linear relaxations; the downside is that they contain a large number of variables. In slot allocation, this is aggravated by the sheer number of capacity and turnaround constraints, which makes these problems cumbersome in terms of computational times and memory usage (Zografos et al., 2012). As an example, applying the formulation from Zografos et al. (2012) to Edinburgh Airport, a medium-size airport, in the 2020 summer season results in at least 686,953 decision variables and 277,273 constraints, and the constraint matrix takes up 3.93 Gb of memory.

Computational results obtained with exact methods have been discouraging, and the consensus is that this combinatorial problem is too complex to be solved to optimality for large airports (Zografos et al., 2016; Ribeiro et al., 2018). Heuristics are clearly favored in studies where the solution algorithm plays an important role. Some of these heuristics are based on the idea of local search (Ribeiro et al., 2019; Androutsopoulos and Madas, 2019; Pellegrini et al., 2011) and variable fixing (Zografos et al., 2012). Oftentimes a MIP solver is used on small or medium-sized instances only to illustrate the impact of a new component introduced in the mathematical model, and the authors of these studies leave the development of a more efficient solution approach for future work (Ribeiro et al., 2018; Fairbrother et al., 2019; Katsigiannis and Zografos, 2021).

Efforts to improve the performance of exact algorithms have been limited. Some authors have applied preprocessing techniques aimed at reducing the size of the model. Zografos et al. (2012) remove capacity constraints by identifying calendar days with the same set of requests and representing them by the same constraint. This allowed them to eliminate between 77 and 127 of the 270 days in the schedule, depending on the data set. However, the resulting models were still computational cumbersome. Fairbrother et al. (2019) and Zografos et al. (2012) add capacity constraints as lazy cuts, which is sensible, but does address the size issue completely: identifying violated constraints in each iteration remains an expensive operation. Fairbrother et al. (2019) avoid dividing the days into the standard 5-minute intervals when the declared capacity limits do not require this level of detail, and instead choose a greater interval that allows them to eliminate an important amount of decision variables. However, this comes at the expense of having to reduce the resolution of the displacements in the optimized schedules.

Valid inequalities have been explored to some extent. However, these tend to be either specific to a new formulation that yields a weaker linear relaxation (Ribeiro et al., 2018; Katsigiannis and Zografos, 2021) or expensive to separate (Zografos et al., 2012), and the performance improvement they bring is modest.

In the scheduling literature, column generation has proved to be effective at alleviating the difficulties associated with problem size in time-indexed formulations (van den Akker et al., 2000) and it has been suggested as an interesting research direction to solve slot allocation models with a large number of variables (Fairbrother and Zografos, 2020). van den Akker and Nachtigall (1999) developed a column-generation based heuristic to solve a problem that considers the European network of airports over three days. However, to the best of our knowledge, there are no studies of exact methods based on this technique.

3. Basic slot allocation model

We develop our algorithm using a basic formulation that only includes those elements that are applicable to virtually all Level 2 and Level 3 airports: the minimization of the total displacement and the number of rejected slots, terminal and/or runway capacity constraints, and minimum turnaround constraints. We choose a time-indexed formulation that has been widely used to model a variety of scheduling problems. It was first introduced by Christofides et al. (1987) in the context of the RCPSP. Zografos et al. (2012) adapted it to the slot allocation problem, proposing a "minimal" integer program that has served as a starting point for more complex models since then.

We use the following notation:

Sets:

D	Set of calendar days in the season.
T	Set of 5-minute intervals in a day. $T = \{1, \dots, 288\}$.
N	Set of requested series.
E	Set of pairs of consecutive arriving and departing flights operated by the same aircraft.
C	Set of declared types of capacity restrictions.
$T_c \subseteq T$	Set of start times of capacity constraints of each type $c \in C$.

Parameters:

$\tau_i \in T$	Flight time requested for flights in series $i \in N$.
$t_{ij} \in T$	Minimum time on the ground (or turnaround time) between flights in series $(i, j) \in E$.
$\delta_{id} \in \mathbb{B}$	Binary parameter that takes value 1 if and only if series $i \in N$ operates on day $d \in D$.
$w_c \in \mathbb{N}$	Duration of time windows of capacity constraints of type $c \in C$.
$\lambda_c \in \mathbb{N}$	Rolling period of time windows of capacity constraints $c \in C$.
$a_{ic} \in \mathbb{N}$	Resources consumed by flights in series $i \in N$ with respect to capacity constraints $c \in C$.
$B_{cds} \in \mathbb{N}$	Capacity limit of constraint of type $c \in C$ within the time window $[s,\ s + w_c)$ of day $d \in D$.
$f_{it} \in \mathbb{N}$	Displacement resulting from allocating series $i \in N$ to slot $t \in T$. $f_{it} = \sum_{d \in D} \delta_{id}\ \lvert t - \tau_i \rvert$.

Decision variables:

x_{it}	Binary variable that takes value 1 if series $i \in N$ is allocated to time interval $t \in T$, 0 otherwise.
y_i	Binary variable that takes value 1 if request to operate the series $i \in N$ is rejected, 0 otherwise.

A series is a set of at least 5 flights with the same characteristics operated throughout the season at the same time and on the same day or days of the week. This explains why variables x_{it} are not indexed by the calendar day d: it is standard practice to allocate all flights in the same series to the same slot, as per the recommendation in the WASG (IATA, 2024).

Each type of capacity restriction $c \in C$ represents a set of maximum throughputs allowed in small time windows of duration w_c with start times T_c. λ_c represents the separation in 5-minute periods between subsequent start times in T_c. Although capacity constraints can be fully specified by w_c and T_c alone, λ_c parameters are introduced here because they are convenient for describing some of the preprocessing rules presented later.

These time windows may or may not overlap; this will depend on whether $w_c = \lambda_c$. An example of each case is illustrated in Fig. 1. Both cases are commonly encountered in capacity declarations.

Not all series are affected by all types of capacity constraints. This is the case for an arriving series and a constraint on the number of departures, or a capacity constraint that only considers flights operating in one of multiple terminal buildings. This information is encoded in parameter a_{ic}. If series i is not relevant for capacity constraints c, then $a_{ic} = 0$. Otherwise, a_{ic} is the number of passengers in the flight if c is a terminal constraint or $a_{ic} = 1$ in the case of runway constraints.

The two main differences between the slot allocation model that we use in this paper, which is presented in (1)–(6), and the one introduced by Zografos et al. (2012) are (i) the addition of parameters a_{ic} in capacity constraints, which allow to model both terminal and runway constraints, and (ii) the addition of binary variables y_i that take value 1 if and only if the request to operate series i is rejected. Without this consideration, it is assumed that there is sufficient capacity to accommodate all series, which is unlikely in Level 3 airports. Recent studies include the minimization of the number of slot rejections in the objective function (see Ribeiro et al., 2018; Fairbrother and Zografos, 2020).

$$\min_{x,y} \quad \sum_{i \in N} \sum_{t \in T} f_{it}\, x_{it} + \omega \sum_{i \in N} \sum_{d \in D} \delta_{id}\ y_i \tag{1}$$

$$\text{s.t.} \quad \sum_{t \in T} x_{it} + y_i = 1 \qquad i \in N, \tag{2}$$

$$\sum_{i \in N} \sum_{t=s}^{s+w_c-1} \delta_{id}\, a_{ic}\, x_{it} \le B_{cds} \qquad c \in C,\ d \in D,\ s \in T_c, \tag{3}$$

$$\sum_{t=s-t_{ij}+1}^{|T|} x_{it} + \sum_{t=1}^{s} x_{jt} \le 1 \qquad (i,j) \in E,\ s \in T, \tag{4}$$

$$x_{it} \in \{0,1\} \qquad i \in N,\ t \in T, \tag{5}$$

$$y_i \in \{0,1\} \qquad i \in N. \tag{6}$$

The objective function (1) minimizes two terms: the total displacement across all flights and the number of rejected flights. We use $\omega = |T|$ to ensure the cost of a rejection is greater than the cost of a displacement for any series. Constraints (2) are the assignment constraints, and they ensure that exactly one slot is selected for each series, unless the series is rejected. Capacity constraints are defined in (3) and constraints (4) impose a minimum turnaround time t_{ij} between linked flights.

Precedence relations, such as the turnaround constraints in this model, are common in scheduling problems and they have a strong and a weak version in terms of the strength of their linear relaxation (Artigues, 2017). Constraints (4) use the strong version, which has the downside that they add $|T| \times |E|$ rows to the constraint matrix. A weak version of these constraints was proposed by Pritsker et al. (1969): $\sum_{t \in T} t\, x_{jt} - \sum_{t \in T} t\, x_{it} \ge t_{ij}$. This alternative formulation only requires one constraint per pair of connected activities $(i,j) \in E$. However, this is an aggregation of constraints (4), and it yields weaker lower bounds. Both options have been used in slot allocation models in recent years. We favor obtaining tighter relaxations because problem size is not an issue when we use the column-generation algorithm presented in Section 5.

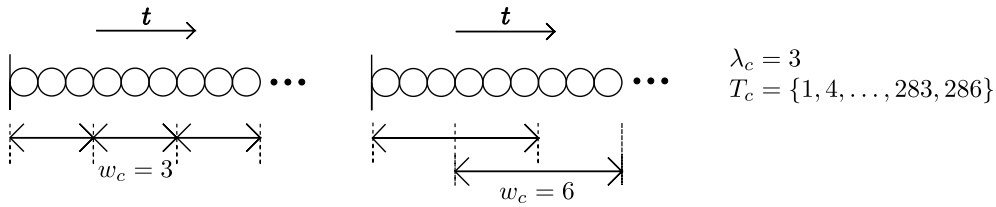

Fig. 1. Example of λ_c, w_c, and T_c in two capacity constraints with the same rolling period but different time windows.

4. Preprocessing

As discussed in Section 3, the main limitation of time-indexed formulations is their size. In this section we present a preprocessing scheme to remove a large fraction of redundant constraints and variables from the formulation in (1)–(6). Many of these rules build on general preprocessing principles for MIPs, most of which require an exponential number of comparisons between pairs of constraints and variables. We exploit the structure of the slot allocation problem to identify dominance relations with minimal computational effort.

This preprocessing pipeline is particularly effective when series can only be displaced within certain limits. This is the case when the slot coordinator knows the maximum displacement that is acceptable to an airline or when a column-generation algorithm such as the one we describe in Section 5 is used.

4.1. Variable fixing

Given a MIP of the form $\min\{c^T x \;/\; Ax \le b,\ \ell \le x \le u,\ x_j \in \mathbb{Z}\}$, we say that variable x_i dominates x_j ($x_i \succ x_j$) if $c_i \le c_j$ and $A_{ki} \le A_{kj}$ for all constraints $A_{k\cdot}^T x \le b_k$ in the problem (Gamrath et al., 2015). A dominated variable can be fixed to 0 as long as at least one of its dominant variables is not. The purpose of this section is to identify dominance relations, such as the one described above, that are valid for every assignment, capacity, and turnaround constraint in the model defined by (1)–(6).

We begin with the assignment constraints. While these constraints are equalities, they can be rewritten as two separate constraints of the form $a^T x \le b$:

$$\sum_{t \in T} x_{it} + y_i \le 1 \qquad i \in N, \tag{7}$$

$$-\sum_{t \in T} x_{it} - y_i \le -1 \qquad i \in N. \tag{8}$$

Since variables of the same series share the same coefficients within constraint (7) and within constraint (8), $x_{it'} \succ x_{it''}$ holds for any pair of slots t', t'' such that $f_{it''} \ge f_{it'}$. Consequently, any dominance relations identified for other constraints — which must also have met the condition $f_{it''} \ge f_{it'}$ — will be valid for assignment constraints.

We now turn our attention to capacity constraints, while assuming for the moment that minimum turnaround constraints are not in the model. When the rolling period λ_c exceeds 5 min for all $c \in C$ — in other words, when no time windows of any capacity constraints start at consecutive 5-minute periods — the constraint matrix contains sets of duplicate variables in the sense that they have the same coefficients in the same rows. Eliminating all but one of the variables in each of these sets can reduce the problem size substantially.

Proposition 1. *Given a series $i \in N$, a set of capacity constraints of type $c \in C$ with rolling period λ_c and duration w_c, multiple of λ_c, variables $x_{it''}$ for all t'' not listed below are dominated in these capacity constraints by some other variable $x_{it'}$ with t' on this list:*

- $t' = \tau_i$,
- $t' = 1 + k\lambda_c,\ \ k \in \mathbb{Z}^+,\ \ t' > \tau_i$,
- $t' = k\lambda_c,\ \ k \in \mathbb{Z}^+,\ \ t' < \tau_i$.

Proof. See Appendix A.1. □

This proposition assumes that w_c is a multiple of λ_c. This assumption is valid for most of the capacity declarations we have seen from the Level 3 airports coordinated by ACL in the summer 2023 season.

Example 1. Consider an airport that only declared one type of capacity limit c in non-overlapping 30-minute windows ($w_c = \lambda_c = 6$, $T_c = \{1, 7, \ldots, 277, 283\}$). A series i has requested the slot 05:10 ($\tau_i = 63$). It is clear that there is at most one slot in each 30-minute window that would be selected for this series in an optimal solution, as shown in the first diagram in Fig. 2. The variables corresponding to these slots $t \in \{6, 12, \ldots, 60, 63, 67, 73, \ldots, 283\}$ dominate the other variables in their 30-minute window, because they have the same coefficient in the corresponding capacity constraint, but the dominated variables would cause a higher displacement, as they are further away from the requested time τ_i.

To generalize this rule to instances with more than one type of capacity constraints, we need to identify dominance relations that hold for all $c \in C$. A dominance relation between a pair of variables identified using Proposition 1 for a given c with rolling period λ_c is also valid for constraints c' with a larger λ'_c as long as this rolling period is a multiple of λ_c (this is easily visualized in the example depicted in Fig. 2). Therefore, when all rolling periods are a multiple of λ_c for some $c \in C$ — this was true for most UK airports coordinated by ACL in Summer 2023 — we can use the dominance relations derived for the index c with the smallest λ_c.

The preprocessing rules presented so far can identify dominance relations in the presence of assignment and capacity constraints. Lastly, we incorporate minimum turnaround constraints and complete the set of rules to identify dominance relations that are valid for all constraints in the slot allocation model.

Proposition 2. *Let $(i, j) \in E$ represent a pair of linked arriving (i) and departing (j) series. Consider two slots for the arriving series $t'' > \tau_i$ and $t' \ge \tau_i$. If $t' < t''$, then $x_{it'} \succ x_{it''}$ with respect to minimum turnaround constraints. An analogous dominance relation exists for the departing series: given slots $t'' < \tau_j$ and $t' \le \tau_j$, if $t' > t''$, then $x_{jt'} \succ x_{jt''}$.*

Proof. See Appendix A.2. □

These dominance relations are valid for minimum turnaround constraints. It is necessary to check whether they are also valid for the remaining constraints in the model. In particular, for capacity constraints. Consider an arriving series $i \in N$. Proposition 1 established a dominance relation $x_{it'} \succ x_{it''}$ for a capacity constraint of index c and a pair of slots t', t'' such that t' is a multiple of λ_c, $t' > \tau_i$ and $t'' \in (t', t' + w_c)$. Any such t' and t'' also meet the validity conditions stated in Proposition 2 for variables of arrivals. Consequently, any dominated arrivals variables x_{it} identified with Proposition 1 can be fixed to 0 if $t > \tau_i$. A similar analysis can be applied to the departure variables to conclude that variables x_{jt} with $t < \tau_j$ can be fixed to 0 if Proposition 1 identified them as dominated.

Proposition 2 identified dominance relations for late slots of arrivals and early slots of departures. So far, no dominance relations for the remaining variables have been found that are valid for the entire model. Proposition 3 addresses this gap by deriving dominance relations for early slots of arrivals and late slots of departures.

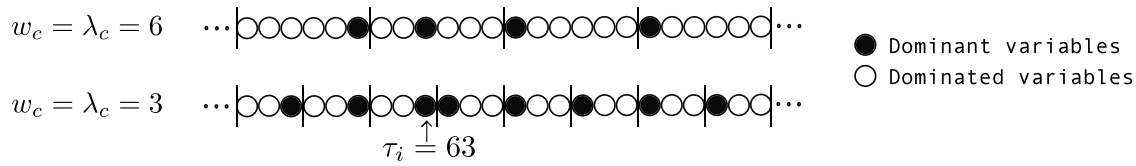

Fig. 2. Dominant and dominated variables for a series i with $\tau_i = 63$ and two types of capacity constraints.

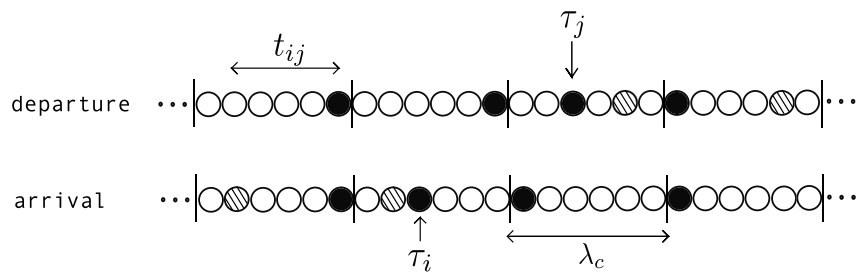

Fig. 3. Variable fixings in a small instance with two linked series and one type of capacity constraints.

Proposition 3. *Let $(i, j) \in E$ represent a pair of linked arriving (i) and departing (j) series. Given a pair of slots t' and t'' that meet the following conditions:*

- $\tau_j \le t' < t''$,
- *there is a value of $k \in \mathbb{Z}^+$ such that $t' = 1 + k\,\lambda_{c'}$, $c' = \arg\min_{c \in C} \lambda_c$ and $t'' \le (k+1)\lambda_{c'}$,*
- *all variables x_{it} with $t \in (t' - t_{ij}, t'' - t_{ij}]$ are dominated by another variable by Propositions 1 and 2,*

then $x_{jt'} \succ x_{jt''}$ and $x_{jt''}$ can be fixed to zero in the model in (1)–(6). An equivalent dominance relation can be found for variables of the arrival series: $x_{it'} \succ x_{it''}$ for any pair of slots t', t'' that meet the following conditions:

- $t'' < t' \le \tau_i$,
- *there is a value of $k \in \mathbb{Z}^+$ such that $t' = k\,\lambda_{c'}$, $c' = \arg\min_{c \in C} \lambda_c$ and $t'' > (k-1)\lambda_{c'}$,*
- *all variables x_{jt} with $t \in [t'' + t_{ij}, t' + t_{ij})$ are dominated another variable by Propositions 1 and 2.*

Proof. See Appendix A.3. □

A way to interpret the first part of this rule is that, if for any feasible solution where $x^*_{jt''} = 1$, we can find an earlier slot t' for this departure such that the variable corresponding to this slot dominates $x_{jt''}$ for all capacity constraints while still resulting in a feasible turnaround time, then the alternative slot is not only feasible, but better from a displacement point of view, thus making $x_{jt''}$ redundant.

Fig. 3 illustrates the variable fixing process in an instance with a pair of linked series $(i, j) \in E$ with requested times τ_i, τ_j, minimum ground time $t_{ij} = 4$, and one type of capacity constraints c with $\lambda_c = w_c = 6$. The solid circles are the variables that cannot be fixed because they represent the least displaced slots within the 30-minute window of a capacity constraint (Proposition 1). The circles with a striped pattern are variables that are dominated for capacity constraints, but no longer redundant when minimum turnaround constraints are considered (Proposition 3). The empty circles are the variables that can be fixed to 0.

4.2. Dominated capacity constraints

Due to the periodic nature of the series of slots, which consist of regular flights, and the capacity restrictions, which follow the same structure throughout the season, slot allocation problems contain many redundant capacity constraints. Here we present a set of rules to identify some of these redundancies.

4.2.1. Redundant scheduling days

This is a generalization of the preprocessing rule proposed by Zografos et al. (2012), which removes identical scheduling days. Given two days $d, d' \in D$, if the following two conditions are met:

- capacity limits on day d never exceed those in day d', namely, $B_{cds} \le B_{cd's} \quad \forall\, c \in C,\ s \in T_c$,
- all the series requested on day d' are also requested on day d, namely, $\delta_{id} \ge \delta_{id'} \quad \forall\, i \in N$,

then, constraints with indices c, d, s dominate constraints with indices c, d', s:

$$\sum_{i \in N} \sum_{t=s}^{s+w_c-1} \delta_{id'}\, a_{ic}\, x_{it} \le \sum_{i \in N} \sum_{t=s}^{s+w_c-1} \delta_{id}\, a_{ic}\, x_{it} \le B_{cds} \le B_{cd's}. \tag{9}$$

From now on, whenever we refer to capacity constraint (c, d, s), this implies the capacity constraint defined for indices c, d, s, where $c \in C, s \in T_c$ and $d \in D$.

4.2.2. Dominated time windows

The previous rule can be extended further by comparing sets of series in smaller time periods, not only whole days. This is relevant when variables x_{it} are not defined for the entire day, but only within a time window $[e_i, \ell_i]$, $e_i \le \tau_i \le \ell_i$. Let U_{cds} denote the set of series that have at least one variable x_{it} included in capacity constraint (c, d, s):

$$U_{cds} = \left\{ i \in N \,/\, \delta_{id} = 1,\ e_i < s + w_c \text{ and } \ell_i \ge s \right\}. \tag{10}$$

Using U_{cds}, a capacity constraint (c, d, s) can be rewritten as $\sum_{i \in U_{cds}} \sum_{t=s}^{s+w_c-1} a_{ic}\, x_{it} \le B_{cds}$. By the same logic applied in (9), if $U_{cd's} \subseteq U_{cds}$ and $B_{cds} \le B_{cd's}$ then constraint (c, d', s) is dominated by (c, d, s), and it can be removed from the model.

4.2.3. Redundant arriving or departing limits

Airports often define restrictions for arriving, departing and total movements with the same T_c and w_c. The limit on total movements should be less than the sum of the equivalent limits on departures and arrivals. Otherwise, this limit would have no impact on the solution. However, two relevant scenarios have been identified where the solution algorithm requires an update to the capacity limits, which may result in limits on departures or arrivals becoming dominated by the equivalent limits on total movements:

- In Level 3 airports, there is a group of historic series that do not request a change of slot. Any efficient algorithm will begin by allocating these series to their historic slots to simplify the problem before any optimization is performed. After these allocations, the capacity limits B_{cds} must be updated to account for the resources used by these historic series.
- The primal heuristic described in Section 5.5 involves fixing the allocation of some series and updating the capacity limits B_{cds} accordingly before solving the resulting sub-MIP.

Consider three capacity constraints (c_A, d, s), (c_D, d, s) and (c_T, d, s), where c_A, c_D and c_T denote capacity limits for arrivals, departures and total movements, respectively. All other characteristics of these constraints are identical. It is clear that $U_{c_A ds} \subseteq U_{c_T ds}$ and $U_{c_D ds} \subseteq U_{c_T ds}$. If $B_{c_A ds} \geq B_{c_T ds}$, then, following the same reasoning used in Section 4.2.2, constraint (c_A, d, s) is dominated by (c_T, d, s) and can be removed. Similarly, if $B_{c_D ds} \geq B_{c_T ds}$, then constraint (c_D, d, s) is redundant.

4.3. Redundant turnaround constraints

Here we present two preprocessing rules to identify dominated minimum turnaround constraints.

4.3.1. Dominance relations between consecutive turnaround constraints

Proposition 4. *Let $i \in N$ be an arrival series, $j \in N$ its linked departure series and t_{ij} the minimum turnaround time between them. If variable x_{js} for some $s \in T$ is fixed to 0, then the turnaround constraint with index s is dominated by the constraint with index $s-1$.*

Proof. See Appendix A.4. □

4.3.2. Minimum and maximum s

Proposition 5. *Let e_i and ℓ_i denote the earliest and latest available slots for series $i \in N$, respectively. Let $(i,j) \in E$ be a pair of linked series with minimum ground time t_{ij}. Turnaround constraints not included in (11) are redundant because they either do not include at least one assignment variable of each series, or they are dominated by another turnaround constraint.*

$$\sum_{t=s-t_{ij}+1}^{\ell_i} x_{it} + \sum_{t=e_j}^{s} x_{jt} \leq 1, \quad (i,j) \in E,\ s \in T \ / \ \max\{e_j,\ e_i + t_{ij} - 1\} \leq s \leq \min\{\ell_j,\ \ell_i + t_{ij} - 1\}. \tag{11}$$

Proof. See Appendix A.5. □

Constraints with $s < e_j$ are redundant because they do not include any variables of the departure, and a constraint with $s = e_i + t_{ij} - 1$ dominates the constraint with $s = e_i + t_{ij} - 2$ because the former has an additional term $x_{j,e_i+t_{ij}-1}$, but the only new term in the latter, x_{ie_i-1}, is equal to 0 because variable x_{ie_i-1} is not defined. A similar interpretation for the upper bound on s is provided in Appendix A.5.

As with other rules, Proposition 5 is only useful if some series can only be allocated within a time window smaller than the day ($[e_i, \ell_i] \neq [1, 288]$).

In addition to the preprocessing rules described in this section, we also check infeasible allocations to fix additional variables and remove capacity constraints where the maximum demand can never exceed the capacity limit B_{cds}. Although these preprocessing steps are built into MIP solvers (see Achterberg et al., 2020), we have implemented them because applying these steps before our preprocessing rules that look for dominance relations can reduce the time required to run the whole preprocessing pipeline substantially.

The variable fixing rules in this section target only assignment variables x. The presence of rejection variables y does not invalidate any of these rules. Variables y only appear in assignment constraints (2), none of which are removed in preprocessing. While some x variables are removed from these constraints, this is due to dominance relations that are unaffected by the presence of the term y_i in (2). There may exist variants of the slot allocation problem where some variables y can be fixed. However, given the small number of these variables, the potential benefit of such fixings seems unlikely to be significant enough to justify the complexity of implementing additional preprocessing rules.

5. The caracal algorithm

In this section we present the Caracal (*Column-And-Row generation branch-And-Cut ALgorithm*) algorithm for single airport slot allocation problems. It is inspired by *Zebra*, an algorithm developed by García et al. (2011) to find exact solutions of very large instances of the p-median problem. The authors of this work take advantage of the problem structure introduced by a radius formulation to solve the linear relaxation in an iterative manner, starting with a reduced model that only contains a small subset of columns and rows, and increasing these subsets in each iteration until an optimal solution is obtained that is feasible for the complete problem.

Despite the evident differences between these two problems, the ideas that gave rise to *Zebra* are also valid in slot allocation. In both problems we can anticipate that a large proportion of the assignment variables are unpromising due to their high cost. Historical data shows that most slot requests are usually granted permission to operate at the requested time. Another property of the p-median formulation exploited by García et al. (2011) is the natural ordering of variables in the sense that the cost coefficients increase monotonically with the variable index. This property is also found in the slot allocation model in (1)–(6), only in this case, the cost coefficients of variables x_{it} increase linearly with $|\tau_i - t|$, regardless of the direction of the displacement.

The remainder of this section describes the steps followed in Caracal to solve the linear relaxation (LP) of the slot allocation model (1)–(6), as well as the branch-and-cut framework into which this process is embedded to find integer solutions. The flow of the algorithm is depicted in Fig. 4.

5.1. Initial reduced model

The starting point of Caracal is a relaxed model (LP_0) with a small subset of variables and constraints from the linear relaxation of the original formulation in (1)–(6), denoted by (LP). The first step to obtain (LP_0) is to fix variables in (LP) using the preprocessing rules in Section 4.1. Next, variables x_{it} corresponding to slots different from the requested slot ($t \neq \tau_i$) are removed from all capacity (3) and turnaround constraints (4). Note that because constraints (3) and (4) take the form $a^T x \leq b$, $a \geq 0$ and this is a minimization problem, this last step gives rise to a relaxation of (LP).

Let e_i and ℓ_i denote the closest slots to τ_i such that $e_i < \tau_i < \ell_i$ and variables x_{ie_i} and $x_{i\ell_i}$ have not been fixed in preprocessing. Variables x_{it} with $t < e_i$ are now dominated by x_{ie_i} since they all have the same coefficients in the constraint matrix (they only appear in assignment constraints) but variables x_{ie_i} have a lower cost (they are closer to the requested time τ_i). Equally, variables x_{it} with $t > \ell_i$ are dominated by $x_{i\ell_i}$. This means all variables x_{it} with $t < e_i$ or $t > \ell_i$ can be fixed to 0.

Only variables $x_{i\tau_i}$ remain included in turnaround and capacity constraints. This makes a large proportion of these constraints redundant. The final step to obtain (LP_0) consists in applying the preprocessing rules described in Sections 4.2 and 4.3 to remove most of these redundant constraints.

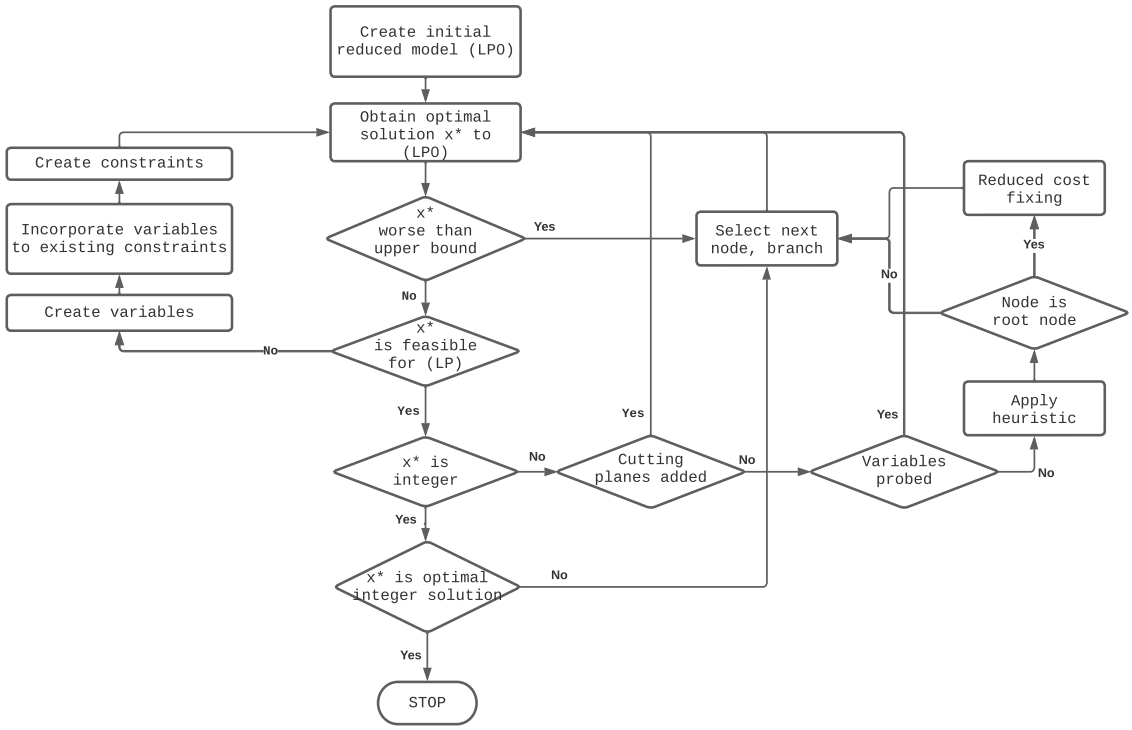

Fig. 4. Flow chart of the Caracal algorithm from an initial relaxed model (LP_0) to the optimal integer solution.

Step 1.	For each $i \in N$:
	Step 1.1. Define $e_i \leftarrow \min\{t \in T \ / \ x_{it}$ is defined in $(LP_0)\}$;
	Step 1.2. Define $\ell_i \leftarrow \max\{t \in T \ / \ x_{it}$ is defined in $(LP_0)\}$;
Step 2.	Construct the sets $\Gamma^- = \{i \in N \ / \ x^*_{ie_i} > 0\}$ and $\Gamma^+ = \{i \in N \ / \ x^*_{i\ell_i} > 0\}$;
Step 3.	For each $i \in \Gamma^-$:
	Step 3.1 Define $e'_i \leftarrow \max\{t \in T \ / \ t < e_i$ and x_{it} not fixed to 0 in $(LP)\}$;
	Step 3.2. Reintroduce variable x_{ie_i} into capacity and turnaround constraints;
	Step 3.3. Generate variable $x_{ie'_i}$;
	Step 3.4. Reintroduce variable $x_{ie'_i}$ into assignment constraints and the objective function;
Step 4.	For each $i \in \Gamma^+$:
	Step 4.1 Define $\ell'_i \leftarrow \min\{t \in T \ / \ t > \ell_i$ and x_{it} not fixed to 0 in $(LP)\}$;
	Step 4.2. Reintroduce variable $x_{i\ell_i}$ into capacity and turnaround constraints;
	Step 4.3. Generate variable $x_{i\ell'_i}$;
	Step 4.4. Reintroduce variable $x_{i\ell'_i}$ into assignment constraints and the objective function;
Step 5.	For each capacity and turnaround constraint not currently in (LP_0), apply preprocessing rules in Sections 4.2 and 4.3 to identify constraints that, due to the changes introduced in Steps 3.2 and 4.2, are no longer redundant, and incorporate them to (LP_0);

Algorithm 1: Naïve column-and-row generation step from solution x^* to relaxed model (LP_0).

5.2. Column-and-row generation

Once the initial reduced model (LP_0) has been created, the next step in Caracal is to apply the dual simplex method to obtain an optimal solution to (LP_0), denoted by x^*. (See Section 5.5 for an explanation of why we choose the dual simplex to solve the linear relaxation). If $x^*_{ie_i} \neq 0$ or $x^*_{i\ell_i} \neq 0$ for some $i \in N$, we cannot conclude that this solution is feasible for (LP), since variables $x^*_{ie_i}$ and $x^*_{i\ell_i}$ were excluded from some constraints in (LP). We need to generate some variables and constraints and repeat this process. To do this we present a naïve approach in Algorithm 1 that is useful to explain the main idea behind Caracal. In Section 5.3 we present an efficient alternative to this algorithm.

In every iteration of Algorithm 1, (LP_0) remains a relaxation of the initial linear relaxation of the slot allocation problem, (LP): the difference between (LP_0) and (LP) is (i) a set of capacity and turnaround constraints that are either removed from (LP_0) or some x are variables removed from the left-hand side, and (ii) a set of variables that can be fixed to 0 as a result of this relaxation.

If $\Gamma^- = \Gamma^+ = \emptyset$ $(x^*_{ie_i} = x^*_{i\ell_i} = 0 \ \forall \ i \in N)$ in some iteration, then this solution is feasible for (LP), as all other x variables are included in all capacity and turnaround constraints from (LP). If x^* is feasible for (LP) then it is also optimal, since (LP_0) is a relaxation of (LP). This proves the exactness of this method for solving the linear relaxation (LP) of the slot allocation problem in (1)–(6). By embedding this column-and-row generation algorithm into a branch-and-cut algorithm we obtain an exact method to solve the integer problem.

These algorithms are unaffected by the presence of y variables. Neither the y variables nor the constraints where they appear are ever removed from the model. They appear in the assignment constraints, where most x variables are removed during the initialization of the (LP_0) model, and some x variables are restored in subsequent iterations. However, it has been established that (LP_0) remains a relaxation of the original (LP) model despite these changes.

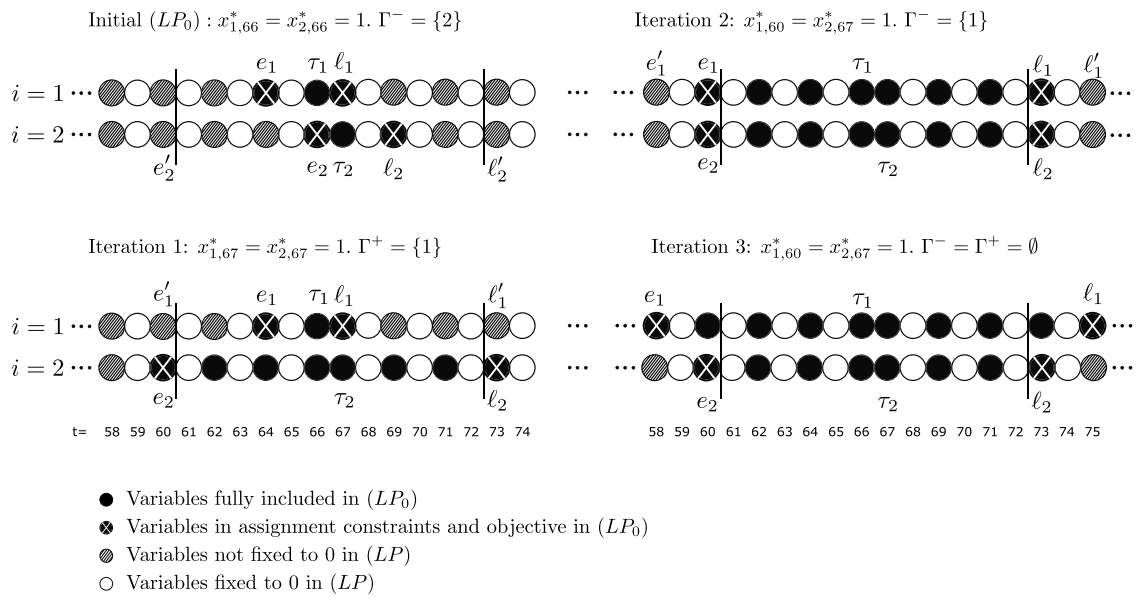

Fig. 5. Modified column-and-row generation process applied to a small instance with two series.

5.3. Improving performance of Algorithm 1

In Algorithm 1 we generate at most two variables per series in $\Gamma^+ \cup \Gamma^-$ in each iteration. We can avoid unnecessary iterations if we can anticipate that the next available variable for some series is not promising. This is best explained with an example.

Example 2. Consider a scenario with two series $i = 1, 2 \in N$ requesting to operate at 5:25 ($t = 66$) and 5:30 ($t = 67$), respectively, in an airport that declared two runway limits: one flight per hour and one flight in each 10-minute period. Fig. 5 shows the x variables that were fixed to 0 in (LP) as well as the x variables included in the initial model (LP_0) and its optimal solution x^*.

We cannot conclude that this solution is feasible and therefore optimal for (LP), since $e_2 = 66$ and $x^*_{2,66} > 0$. If we use the naïve approach in Algorithm 1 we will generate one variable per iteration and it will take 12 iterations to reach optimality.

However, we know that in this particular instance at most one series can be scheduled between 5:00 and 5:55 due to the hourly runway limit, so we can save several steps if we generate enough variables in one iteration to allow series $i = 2$ to "escape" the time window of this constraint. We define $e'_2 = 60$ and $\ell'_2 = 73$ and we generate $x_{2,60}$ and $x_{2,73}$ and include them in the corresponding assignment constraint and in the objective function. Then we generate variables x_{2t} from the preprocessed (LP) with $61 \le t \le 72$ and incorporate them to (LP_0) in all constraints and the objective function.

We follow the same logic in the second iteration, and we finally obtain $x^*_{1,60} = x^*_{2,67}$ in iteration 3, at which point $\Gamma^+ = \Gamma^- = \emptyset$, and we can conclude that this solution is feasible and optimal for (LP). With this modified approach, we reach optimality in just four iterations: three to generate variables outside the interval 5:00–5:55 for each of the series, and one to verify optimality.

The previous example illustrates the rationale behind this modification, which is described for the general case in Algorithm 2:

5.4. Cutting planes

We developed our own branch-and-bound code, as most MIP solvers do not allow the integration of column-generation algorithms. This means we cannot access the large repertoire of cutting planes included in solvers such as CPLEX or Gurobi. Even though the time-indexed formulation (1)–(6) has a tight linear relaxation, cutting planes are required to obtain a satisfactory performance in harder instances. We implemented two types of cutting planes that proved to be effective, simple to implement, and compatible with column generation: lifted cover cuts and lifted Generalized Upper Bound (GUB) cover cuts. We use the ideas of Gu et al. (1998) and Kaparis and Letchford (2008), and exploit the structure of the capacity constraints to speed up the separation of lifted GUB covers. More details of this implementation can be found in Appendix B.

Combining these cover inequalities with column generation is simple. Adding variables to an existing constraint will give rise to stronger cuts if we can lift the new variables into existing cover inequalities. In the case of simple cover cuts, we apply the same up-lifting procedure used when deriving the initial cut to the new variable. In the case of GUB cover cuts, we first check if the series i associated with the new variable x_{it} is in the GUB cover. If it is, x_{it} is added to the inequality with the same lifting coefficient as the other variables of series i. Otherwise, the up-lifting problem is solved to determine the largest possible lifting coefficient.

These cuts are applied in every node of the branch-and-bound tree. Once the linear relaxation has been solved and no more columns can be generated, we attempt to generate cutting planes and repeat this process until no more cuts can be added.

5.5. Additional implementation details

Early stopping: As pointed out by García et al. (2011), the intermediate solutions of (LP_0) yield lower bounds on the optimal solution of (LP). When solving the linear relaxation of a node of the branching tree with column-and-row generation, if the objective value of some intermediate solution is worse than the current upper bound on the optimal integer solution, this node can be discarded immediately, before the linear relaxation is solved to optimality.

Primal heuristic: We propose a heuristic similar to RENS (Berthold, 2014). Starting from the solution to the linear relaxation, x^*, we fix the variables of all series that were allocated to the requested time,

Step 1.	For each $i \in N$:
	Step 1.1. Define $e_i \leftarrow \min\{t \in T \ / \ x_{it}$ is defined in $(LP_0)\}$;
	Step 1.2. Define $\ell_i \leftarrow \max\{t \in T \ / \ x_{it}$ is defined in $(LP_0)\}$;
Step 2.	Construct set $\Gamma = \{i \in N \ / \ x^*_{ie_i} > 0$ or $x^*_{i\ell_i} > 0\}$;
Step 3.	Construct set $\mathcal{A}$ of capacity and turnaround constraints from (LP) that are violated by x^*;
Step 4.	For each $i \in \Gamma$:
	Step 4.1. Update $e'_i \leftarrow \max\{t \in T \ / \ t < e_i$ and variable x_{it} does not appear in any constraint in $\mathcal{A}\}$;
	Step 4.2. Update $\ell'_i \leftarrow \min\{t \in T \ / \ t > \ell_i$ and variable x_{it} does not appear in any constraint in $\mathcal{A}\}$;
Step 5.	For each $i \in \Gamma$:
	Step 5.1. Create variables x_{it} for each $t \in [e'_i, e_i) \cup (\ell_i, \ell'_i]$, excluding variables fixed to 0 in (LP_0);
	Step 5.2. Reintroduce variables created in Step 5.1 into assignment constraints and objective function;
	Step 5.3. Reintroduce variable x_{it} for each $t \in (e'_i, e_i] \cup [\ell_i, \ell'_i)$ into capacity and turnaround constraints;
Step 6.	For each capacity and turnaround constraint not currently in (LP_0), apply preprocessing rules in Sections 4.2 and 4.3 to identify constraints that, due to the changes introduced in Step 5.3, are no longer redundant, and incorporate them to (LP_0);

Algorithm 2: Modified algorithm to perform a column-and-row generation step from solution x^* to (LP_0).

namely, series $i \in N$ with $x^*_{i\tau_i} = 1$, and solve the resulting MIP subproblem. This heuristic is guaranteed to find a feasible solution with the formulation in (1)–(6). In other nodes of the branching tree we use the RENS heuristic. If the number of fractional variables is less than 50 we solve the resulting MIP subproblem to optimality. Otherwise, a greedy algorithm is used.

Reduced cost fixing: Once the root node is solved to optimality and after applying the primal heuristic described above, we eliminate all variables with a reduced cost RC such that $LB + RC > UB$, where LB and UB are the lower and upper bound on the optimal solution, respectively. Then, the preprocessing rules outlined in Section 4.2 are used to eliminate redundant capacity constraints after reduced cost fixing.

Branching and node selection: We use reliability branching with the procedure and default parameters proposed by Achterberg et al. (2005). The node selection strategy is depth-first search exploring the up branch first, to encourage integer solutions early in the search. When depth-first search gets stuck because it encounters either an infeasible node, an integer solution or a fractional solution that is worse than the current best bound, best-bound search is used instead to select the next node.

Probing: Once a fractional solution is obtained we fix each binary variable with a fractional solution to 0 and 1, separately. If either fixing results in an infeasible model, the variable is fixed to the only feasible value. This technique is used to potentially improve the lower bound and reduce the solution space by tightening the bounds of some variables. Probing is only performed in nodes with a depth in the tree less than 6.

LP reoptimization: A dual problem remains feasible when new rows are added to the primal. The dual simplex algorithm can be warm-started from the previous basis by simply treating the slack variables of any new rows as basic variables (Hillier and Lieberman, 2001). However, in Caracal, we also add new columns and modify variable coefficients of existing rows, which can invalidate the basis and cause the dual simplex to restart the optimization after each pricing iteration. This can be avoided following these steps in the column-generation process:

1. In Step 5.1 of Algorithm 2, we add new variables x_{it} as nonbasic at the lower bound. This new basic solution is still feasible (new variables are only incorporated to assignment constraints) and optimal (their cost coefficient is higher than that of other variables of the same slot request in the model).
2. Whenever we add a variable to an existing constraint (Steps 5.2 and 5.3 of Algorithm 2), we look at the basic status of its slack variable. If it is nonbasic, it may be forced to become basic, thus invalidating the previous basis and forcing the solver to either discard it or perform many operations to recover an optimal basis. For this reason, we only add variables to an existing constraint if the slack variable of the constraint is basic. Otherwise, we add a copy of the constraint with the incorporated variable. We leave the old constraint in the model until it becomes basic after some dual simplex reoptimization. At that point it can be safely removed (Thompson et al., 1966).

6. Synthetic data

We generate synthetic data sets for single airport slot allocation problems (Fermín Cueto, 2022). Our approach is similar to that used by Androutsopoulos et al. (2020). Inspired by slot request data sets published by Agência Nacional de Aviação Civil (ANAC) (2021) and capacity declaration reports from Airport Coordination Limited (ACL) (2021), we create distributions for a set of parameters that characterize airport demand and capacity, such as requested arrival and departure times, turnaround times, series length, and aircraft sizes. We then sample from these distributions to generate realistic slot requests and capacity limits.

This synthetic data does not include priority groups; generated instances are treated as slot requests and capacity declarations for Level 2 airports. The instances encompass a wide range of demand volumes: the number of series varies from 117 to 5790, and the total number of movements throughout the season ranges from 10,008 to 199,069. Instances may include terminal and runway constraints that affect arrivals, departures, or all movements. They can target specific terminal buildings or consider only domestic or international flights. Each instance contains between 11 and 24 different types of capacity constraints, with time windows lasting from 5 min to 2 h and rolling every 5 to 60 min. The total number of capacity constraints across these instances ranges from 136,080 to 438,480. The optimal solutions obtained for these instances showed displacements ranging from 5.3 to 22.6 min per displaced movement. It was not necessary to reject any series in these instances.

The code for generating these data sets, along with tables summarizing key metrics for each instance, is available at https://github.com/paulafernalia/slot-allocation-data-gen. The data sets can be found at https://datashare.ed.ac.uk/handle/10283/4374.

7. Computational study

In this section, we evaluate the performance of Caracal and the preprocessing methodology presented in Section 4. We use the model presented in Section 3 and a problem set consisting of 100 synthetic instances generated using the methodology referenced in Section 6. Further experiments are presented in Section 8 using more realistic model variants and data sets from 7 Level 2 and Level 3 UK airports.

Each problem instance is solved using three different algorithms:

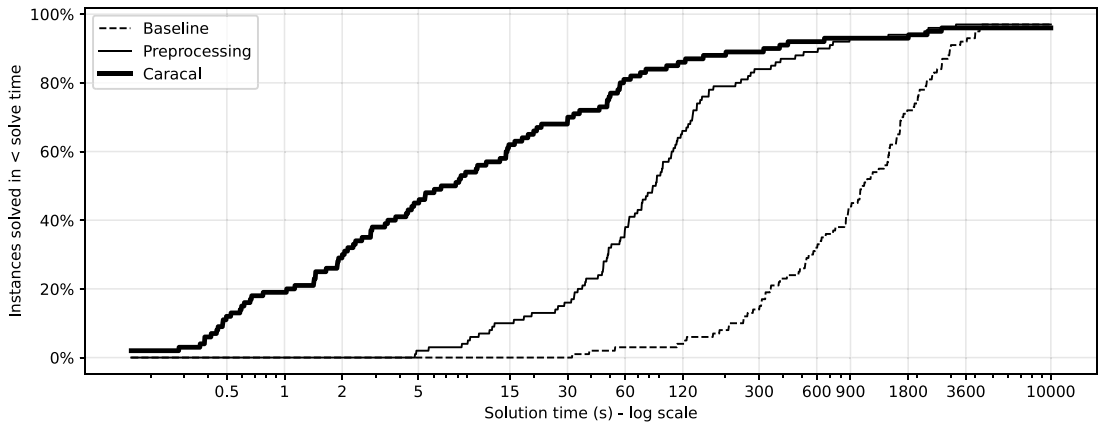

Fig. 6. Performance profile comparing the Baseline, Preprocessing and Caracal algorithms on synthetic instances.

- **Baseline**: Gurobi's MIP solver with the preprocessing step proposed by Zografos et al. (2012) and the initial constructive heuristic proposed by Fairbrother and Zografos (2020). The weak version of the minimum turnaround constraints was used, as the strong version displayed worse performance in our experiments. For the same reason, capacity constraints were not added as lazy cuts, as proposed by Fairbrother et al. (2019).
- **Preprocessing**: same setup as the Baseline algorithm, adding the preprocessing pipeline described in Section 4.
- ***Caracal***: strong turnaround constraints, the algorithm described in Section 5 with Gurobi's LP solver, Gurobi's preprocessing deactivated and our implementation of branch-and-cut.

The experiments are conducted on an Intel® Xeon® 2.2 GHz, 256 GB RAM, setting a maximum of 4 cores. Our algorithms were coded in C++. The LP relaxations in Caracal and the MIPs in other methods are solved with Gurobi 9.1 (Gurobi Optimizer Reference Manual, 2021), using its C++ API with default settings unless otherwise stated. A time limit of 10,000 s was set for each instance and algorithm.

Fig. 6 shows the percentage of instances that were solved within the times marked on the x axis, for each of the three algorithms. Tables 1 and 2 compare the performance of the three algorithms. To create these tables, the test set was divided into six groups. A group denoted "$(k, m]$" includes the instances that the fastest algorithm solved in more than k seconds but at most m. Shifted geometric means[1] are used instead of arithmetic means to limit the effect of individual large values. Following the benchmarking methodology employed by Achterberg (2009), a shift $s = 1$ is used for computational times and $s = 10$ for nodes to decrease the strong influence of trivial instances. A solution time of 10,000 s is assumed for instances that could not be solved within the time limit, which introduces a bias against the algorithms that hit the time limit less often. These results must therefore be read together with the number of timeouts.

In Table 1, "Instances" shows the number of instances from the test set in each sub-group displayed in the "Group" column; "Init NZ" is the number of nonzeros (in millions) in the constraint matrix of the initial model, using weak turnaround constraints; "Timeouts" denotes number of instances that could not be solved to optimality within the time limit; "Time" shows the shifted geometric mean of the solution times in CPU seconds; "NZ" is the geometric mean of the number of nonzeros (in millions) in the constraint matrix after preprocessing, and "Faster" shows the number of instances in which an algorithm was faster than the Baseline. In Table 2, "Nodes" shows the shifted geometric mean of the size of the branch-and-bound tree when the optimal solution was found; "Root" is the number of instances that could be solved in the root node, "Heur (gap)" shows the geometric mean of the optimality gap achieved by our primal heuristic, and "Heur (z)" shows the number of instances in which the primal heuristic found the optimal solution, out of the total number of instances in which the heuristic was applied. The sub-group corresponding to instances that were not solved within 10,000 s by any method is not displayed, but is included in the last row of each table, which aggregates all groups.

It can be seen from Fig. 6 that Caracal outperforms the other two methods, especially the Baseline, in most instances. It solved 53% of the test instances in less than 10 s and 79% in less than one minute. The Baseline could only solve 4% of the instances in less than two minutes. According to the results summarized in Table 1, Caracal was 83 times faster than the Baseline and 8.7 times faster than the Preprocessing algorithm on the entire test set.

The Preprocessing method is also very competitive. It is faster than the Baseline method in 94 of the 100 instances, being 9.5 times faster on average. Our preprocessing rules exploit the underlying problem structure to identify dominance relations more efficiently, resulting in 10% more variables and 41% more constraints eliminated 13.3 times faster.

Fig. 6 also shows that both the Preprocessing and the Baseline algorithms eventually catch up with Caracal at around 800 min and 2 h, respectively. This behavior is observed in instances with a less tight linear relaxation, which require more branching. In those instances, Caracal spends more time solving each node of the tree than the other algorithms, despite the subproblems being significantly smaller and despite Caracal being much faster at solving the root node. Gurobi can reuse the LU factorization from the parent node when performing depth-first search (hot start). In Caracal, we perform warm start, that is, the optimal basis of the parent node is used as a starting point for the LP of the child node. This speeds up reoptimization of the child node to a lesser extent than hot starting.

Table 2 shows that Caracal produces larger trees than the other two methods. This is likely to be attributed to Gurobi's more efficient implementation of branch-and-cut, with a larger repertoire of cutting planes, primal heuristics or faster codes. Table 2 also shows the primal heuristic described in Section 5.5 is highly effective and can find optimal or near-optimal solutions, providing a "safety net" in hard instances.

An important advantage of our two proposed methods, especially Caracal, is their reduced memory usage. To analyze this, we look at the number of nonzeros in the constraint matrix of the root node problem. Each nonzero consumes 12 bytes and, according to Gurobi Support Portal (2021), the model is represented in memory at least three times before attempting to solve the problem. Therefore the

[1] The shifted geometric mean of values $t_1, \ldots, t_n$ is defined as $\gamma_s(t_1, \ldots, t_n) = \left(\prod_{i=1}^{n}(t_i + s)\right)^{1/n} - s$ with a shift $s \geq 0$. It is a variant of the geometric mean that focuses on ratios instead of totals, preventing the hard instances from dominating the results (see Achterberg, 2007).

Table 1
Aggregated computational results of Baseline, Preprocessing and Caracal on synthetic instances (I).

Group	Instances	Init NZ	Baseline			Preprocessing				Caracal			
			Timeouts	Time	NZ	Timeouts	Time	Faster	NZ	Timeouts	Time	Faster	NZ
(0, 1]	19	109.4	0	403.1	47.9	0	27.0	19	26.0	0	0.5	19	0.006
(1, 10]	35	126.0	0	763.6	68.6	0	64.3	35	34.5	0	3.2	35	0.193
(10, 60]	27	174.2	0	1248.0	100.6	0	145.9	27	59.0	0	25.6	27	0.264
(60, 300]	9	192.8	0	1485.6	106.7	0	224.3	8	47.5	0	119.1	9	0.287
(300, 10k]	7	207.7	0	2332.8	59.7	0	985.4	5	93.4	1	1027.8	5	0.183
All	*100*	*147.7*	*3*	*890.8*	*81.0*	*3*	*93.6*	*94*	*44.5*	*4*	*10.1*	*95*	*0.221*

Table 2
Aggregated computational results of Baseline, Preprocessing and Caracal on synthetic instances (II).

Group	Instances	Baseline		Preprocessing		Caracal			
		Nodes	Root	Nodes	Root	Nodes	Root	Heur (gap)	Heur (z)
(0,1]	19	1.0	19	1.0	19	1.0	19	–	0/0
(1,10]	35	1.7	34	2.2	32	3.3	22	0.1%	10/13
(10,60]	27	2.7	26	5.2	23	7.3	16	0.4%	3/11
(60,300]	9	8.1	8	25.6	6	27.5	5	2.4%	0/6
(300,10k]	7	138.2	3	108.5	2	288.5	1	3.8%	1/4
All	*100*	*8.2*	*90*	*10.2*	*82*	*11.2*	*63*	*1.9%*	*14/34*

number of nonzeros times 36 represents a lower bound on the peak memory in bytes required to solve these problems. It can be inferred from the "Init NZ" column in Table 1 that the average Baseline model before preprocessing takes up 5.3 Gb. With the Preprocessing method, all the preprocessing is done before the model is created and therefore the memory required to represent the model comes from the "NZ" column: 1.6 Gb on average. With Caracal, the average initial root node model only takes up 385 kb.

8. Application to real-world slot allocation

The purpose of this section is to demonstrate the flexibility of Caracal to solve practical slot allocation problems and to evaluate its performance on real-world data. For the sake of conciseness we do not exhaustively explore every model variant found in the literature. We explore the incorporation of four elements to the formulation in (1)–(6): priority groups, maximum turnaround constraints, historic overages, and season segmentation. These extensions have been selected because they are widely applicable today and/or they make the slot allocation problem substantially harder.

8.1. Priority groups

In Level 3 airports, slot requests are grouped into categories with different priority levels. These groups must be allocated sequentially, from the highest priority group to the lowest. We define 7 groups:

- Historic — historic series that request to operate the historic slot,
- Historic Retimes — historic series that request a slot different from the historic,
- and 5 groups of non-historic series which, for the purpose of this study, only differ in their priority level.

This sequential allocation process calls for a lexicographic multi-objective approach. Let P denote the set of priority groups, sorted by priority from Historic ($p = 1$) to the lowest priority group ($p = 7$). Let N_p denote the set of series in priority group p, and g_p the objective function of said priority group, namely, $g_p = \sum_{i \in N_p} \sum_{t \in T} f_{it} x_{it} + \omega \sum_{i \in N_p} \sum_{d \in D} \delta_{id} y_i$. Let S denote the set of feasible solutions to the constraints in (2)–(6). With this notation, the lexicographic optimization problem can be written as follows:

For $p = 1, \dots, 7$:

- Solve:

$$z_p = \min \quad g_p(x, y) \tag{12}$$

$$\text{s.t.} \quad (x, y) \in S, \tag{13}$$

$$g_q(x, y) \le z_q \qquad q \in \{1, \dots, p-1\}. \tag{14}$$

- Add constraint $g_p(x, y) \le z_p$ and continue.

For each priority level $p \in P$ we minimize g_p subject to the slot allocation criteria represented by (13), while ensuring through constraints (14) that the solution is selected from the set of solutions that are lexicographically optimal for the higher priority groups. The optimal solution x^* to the problem will be the solution to the last iteration $p = 7$, which includes all priority groups.

This approach is expensive; it involves solving 7 optimization problems, all of which include the entire list of slot requests. This can be simplified substantially when we make the following observation: when the subproblem of a given priority level p is being solved, series with lower priority levels $p' \in \{p+1, \dots, 7\}$ are irrelevant: they do not appear in (12) or (14); they only need to satisfy the slot allocation constraints in (13). Treating these series as rejected ($y_i = 1,\ i \in N_{p'}$) does not violate any constraint in (13) or have any impact on the objective function (12). This allows us to solve smaller subproblems that only include series in groups $N_1, \dots, N_p$ in each iteration p.

Some of these priority groups have special characteristics that must be factored into the formulation in (12)–(14). Let h_i be the historic slot of a series in group $p = 1$ or $p = 2$. Historic requests are automatically allocated to their historic slot, resulting in a trivial allocation problem with $x^*_{i,\tau_i} = x^*_{i,h_i} = 1$ for all series in N_1 and $z_1 = 0$. This allows us to turn subproblem $p = 1$ into a data preparation step where we allocate these slots and update the capacity limits for any subsequent subproblems. For some of the series in the Historic Retimes group, the coordinator must ensure that they are not displaced by more than the difference between the historic slot and the requested slot ($x_{it} = 0\ \forall\ t\ /\ |t - \tau_i| > |h_i - \tau_i|$). For the remaining series in this group, only the requested and the historic slots are acceptable ($x_{it} = 0\ \forall\ t \ne \{\tau_i, h_i\}$). Historic Retimes cannot be rejected ($y_i = 0$).

The addition of constraints (14) does not invalidate any variable fixing rules described in Section 4. Any variables $x_{it''}$ fixed to 0 because there exists a t' such that $x_{it'} \succ x_{it''}$ for all assignment, capacity and turnaround constraints, must have met the condition $f_{it'} \le f_{it''}$. By definition, the coefficients of variables $x_{it'}$ and $x_{it''}$ in constraints (14) with index $q \in \{1, \dots, p-1\}$, are $f_{it'}$ and $f_{it''}$, respectively, if $i \in N_q$, and both are 0 otherwise. In either case, the coefficient of the dominated variable is greater than or equal to that of the dominant variable. Since

these constraints take the form $a^T x \le b$, the dominance relation holds.

Two modifications are required in the column-generation stage of Caracal:

- Constraints (14) must be considered in Steps 3 and 5.3 of Algorithm 1 for correctness.
- Because historic requests cannot be rejected, the primal heuristic described in Section 5.5 can no longer guarantee a feasible solution at the root node in the subproblems defined by (12)–(14). We propose an alternative heuristic that consists of fixing x_{it} and y_i variables of higher priority requests allocated in previous subproblems and solving the resulting MIP subproblem to obtain a feasible initial solution. We continue to use the original primal heuristic in every applicable node of the branching tree.

8.2. Maximum turnaround time

In this section we describe how Caracal can deal with maximum turnaround constraints. We use the strong version of these constraints, using the same rationale presented in Section 3 to opt for the strong version of minimum turnaround constraints. Let $t_{ij}^{\max}$ denote the maximum time on the ground permitted between an arrival i and a departure j, where $(i,j) \in E$. Maximum turnaround constraints can be written as follows:

$$\sum_{t=1}^{s} x_{it} + \sum_{t=s+t_{ij}^{\max}+1}^{|T|} x_{jt} \le 1 \qquad \forall\ (i,j) \in E,\ s \in T. \tag{15}$$

To avoid eliminating non-redundant variables, the variable fixing rules presented in Section 4.1 must now include additional conditions to ensure their validity in the presence of maximum turnaround constraints. Once the set of non-dominated variables is obtained using the criteria laid out in Propositions 1, 2, and 3, we check the following conditions to determine which of these variables can still be fixed with respect to maximum turnaround constraints:

Proposition 6. *Let $(i,j) \in E$ represent a pair of linked arriving (i) and departing (j) series. Consider two slots for the departing series $t'' > \tau_j$ and $t' \ge \tau_j$. If $t' < t''$, then $x_{jt'} \succ x_{jt''}$ with respect to maximum turnaround constraints. An analogous dominance relation exists for the arriving series: given slots $t'' < \tau_i$ and $t' \le \tau_i$, if $t' > t''$, then $x_{it'} \succ x_{it''}$.*

Proposition 7. *Let $(i,j) \in E$ represent a pair of linked arriving (i) and departing (j) series. Given a pair of slots t' and t'' that meet the following conditions:*

- *$\tau_i \le t' < t''$,*
- *there is a value of $k \in \mathbb{Z}^+$ such that $t' = 1 + k\,\lambda_{c'}$, $c' = \arg\min_{c \in C} \lambda_c$ and $t'' \le (k+1)\,\lambda_{c'}$,*
- *all variables x_{jt} with $t \in (t' + t_{ij}^{\max}, t'' + t_{ij}^{\max}]$ are dominated by another variable by Propositions 1 and 6,*

then $x_{it'} \succ x_{it''}$ and $x_{it''}$ can be fixed to zero in the model in (1)–(6). An equivalent dominance relation can be found for variables of the departure: $x_{jt'} \succ x_{jt''}$ for any pair of slots t', t'' that meet the following conditions:

- *$t'' < t' \le \tau_j$,*
- *there is a value of $k \in \mathbb{Z}^+$ such that $t' = k\,\lambda_{c'}$, $c' = \arg\min_{c \in C} \lambda_c$ and $t'' > (k-1)\lambda_{c'}$,*
- *all variables x_{it} with $t \in [t'' - t_{ij}^{\max}, t' - t_{ij}^{\max})$ are dominated another variable by Propositions 1 and 6.*

We also adapt Propositions 4 and 5 to derive equivalent preprocessing rules for identifying redundant maximum turnaround constraints:

Proposition 8. *Let $i \in N$ be an arrival series, $j \in N$ its linked departure series and $t_{ij}^{\max}$ the maximum turnaround time between them. If variable x_{is} for some $s \in T$ is fixed to 0, then the maximum turnaround constraint with index s is dominated by the constraint with index $s-1$.*

Proposition 9. *Let e_i and ℓ_i denote the earliest and latest available slots for series $i \in N$, respectively. Let $(i,j) \in E$ be a pair of linked series with minimum ground time $t_{ij}^{\max}$. Maximum turnaround constraints not included in (16) either do not include at least one assignment variable of each series or they are dominated by another turnaround constraint and, therefore, they are redundant.*

$$\sum_{t=e_i}^{s} x_{it} + \sum_{t=s+t_{ij}^{\max}+1}^{\ell_j} x_{jt} \le 1, \quad (i,j) \in E,\ s \in T \ \ / \ \ \max\{e_i,\ e_j - t_{ij}^{\max} - 1\} \le s \le \min\{\ell_i,\ \ell_j - t_{ij}^{\max} - 1\}. \tag{16}$$

The proofs of the propositions presented in this section are omitted for the sake of brevity because can be derived using the same arguments employed in the analogous rules for minimum turnaround constraints.

The only other change required in Caracal to accommodate the new constraints pertains to Steps 3 and 5.3 of Algorithm 2. These steps now need considering maximum turnaround constraints in addition to their minimum counterparts and capacity constraints.

8.3. Historic overages

A historic slot cannot be withdrawn if the airline operated this slot at least 80% of the time in the same season of the previous year (IATA, 2024). If a capacity limit is reduced with respect to this earlier season, historic series must still be allowed to operate the historic slot, even if doing so means exceeding the new capacity limit. This limit can only be relaxed if historic slots — and no other slots — are allocated in this period; otherwise the new, reduced capacity limit must be enforced.

If these overages only affected series in N_1 (historic series that must be allocated to their historic slot), they could be addressed before any optimization is required: allocate these slots first, adjust capacity limits accordingly, and remove overages. However, because series in N_2 (historic series that request a different slot) are also responsible for overages and there is no trivial solution for this priority group, overages must be included in the formulation. They alter the behavior of capacity constraints, which can now be viewed as knapsack constraints with a right-hand side that can take two possible levels: the declared limit plus the overage if only historic slots are allocated in the time window of this constraint, and the declared limit otherwise.

We need to define some additional sets and parameters to integrate historic overages in our formulation:

- $N^H = N_1 \cup N_2$: set of series in the Historic and Historic Retimes groups,
- h_i: historic slot of series $i \in N^H$,
- O_{cds}: historic overage, caused by historic slots in capacity constraint (c,d,s), namely,

$$O_{cds} = \max\left\{ \sum_{\substack{i \in N^H \\ s \le h_i < s + w_c}} (a_{ic}\,\delta_{id}) - B_{cds},\ 0 \right\}. \tag{17}$$

Example 3. Consider an example where an airport has declared a capacity limit of 1 movement from 12:00 to 12:10 in the summer season of 2023, namely, $B_{cds} = 1$. In summer 2022, this limit was 3 movements and 3 historic series i_1, i_2 and i_3 operated at 12:00. This means there is a historic overage of 2 movements ($O_{cds} = 2$) in 2023. Series i_1, i_2 and i_3 are all Historic Retimes, and they request new slots 11:55, 11:55 and 12:05, respectively. There is a fourth series i_4, a New

Entrant, requesting slot 12:05. In this scenario, the old capacity limit of 3 movements only applies if any or all of i_1, i_2 and i_3 are allocated to their historic slot 12:00 and i_4 is scheduled outside the time window [12:00, 12:10). If i_3 is allocated to 12:05 or i_4 is allocated to 12:00 or 12:05, the capacity limit will remain at 1 movement.

To model these variable capacity limits we make use of a new family of binary variables Ω_{cds}, which take value 1 if at least one non-historic slot is allocated in the time window of a capacity constraint (c, d, s) with a historic overage. In any other case, we do not impose any bounds on Ω_{cds}. This definition is enforced through constraints (18)–(19). Constraints (20) modify the original capacity constraints to allow the demand to exceed the capacity when all the allocated slots are historic slots:

$$\sum_{t=s}^{s+w_c-1} x_{it} \le \Omega_{cds} \qquad i \in N \setminus N^H / \ a_{ic} > 0, \ \delta_{id} = 1, \tag{18}$$

$$\sum_{\substack{t\in[s,s+w_c)\\ t\neq h_i}} x_{it} \le \Omega_{cds} \qquad i \in N^H / \ a_{ic} > 0, \ \delta_{id} = 1, \tag{19}$$

$$\sum_{i\in N} \sum_{t=s}^{s+w_c-1} \delta_{id}\, a_{ic}\, x_{it} \le B_{cds} + O_{cds}(1 - \Omega_{cds}). \tag{20}$$

These constraints are defined for all $c \in C$, $s \in T_c$, $d \in D$ such that $O_{cds} > 0$. For all other c, d, s, only the original capacity constraint (3) is needed. We also add a family of constraints to strengthen the formulation, motivated by the following example:

Example 4. Consider a runway capacity constraint (c, d, s) with a limit $B_{cds} = 1$, duration $w_c = 1$ and historic overage $O_{cds} = 1$. There are two historic series $i = 1$ and $i = 2$ operating on day d with s as their historic slot, and a new entrant $i = 3$. The corresponding capacity constraint (20) for this period is $x_{1s} + x_{2s} + x_{3s} \le 2 - \Omega_{cds}$. Consider a fractional solution where $x^*_{1s} = 1$, $x^*_{2s} = 0$, $x^*_{3s} = \Omega^*_{cds} = 0.5$. Adding the inequality $\Omega_{cds} \ge 1 - x_{2s}$ would force $\Omega_{cds} \ge 1$, thus cutting off this fractional solution.

In the general case, when $O_{cds} = 1$, these inequalities can be written as

$$\Omega_{cds} \ \ge \ 1 - x_{ih_i} \qquad i \in N^H / \ a_{ic} = \delta_{id} = 1, \ h_i \in [s, s + w_c). \tag{21}$$

These are not exactly valid inequalities, but rather a redefinition of Ω_{cds} that forces these variables to take value 1 in some cases where previously no bounds being imposed and this was leading to fractional solutions. This redefinition does not eliminate any feasible schedule because the inequality (21) is always true in an integer solution as per the rules around historic overages described at the beginning of this section.

The reasoning for $O_{cds} = 1$ can be generalized to runway constraints with $O_{cds} \ge 1$, resulting in the following family of inequalities for each of these constraints, which replace (21):

$$\Omega_{cds} \ \ge \ 1 - \sum_{i\in A} x_{ih_i} \qquad A \subseteq N^H \ / \ |A| = O_{cds}, \forall i \in A, \ a_{ic} = \delta_{id} = 1, h_i \in [s, s+w_c). \tag{22}$$

Due to the large number of constraints (22), we implement them as lazy cuts.

The addition of constraints related to historic overages does not invalidate any of the dominance relations between pairs of constraints presented in Section 4. To ensure the variable fixing rules remain valid, it suffices to prohibit the fixing of certain variables: x_{ih_i}, no longer dominated due to the addition of constraints (18), and either $x_{j,h_i+t^{\min}_{ij}}$ if $(i, j) \in E$, or $x_{j,h_i-t^{\min}_{ij}}$, if $(j, i) \in E$. As with previous modeling extensions, the new constraints must be considered in Steps 3 and 5.3 of Algorithm 1 to ensure the correctness of the column-and-row generation algorithm.

We do need to adapt the cover cuts for terminal capacity constraints with historic overages, as these now include an additional variable. Ω_{cds} can be moved to the left-hand side of the capacity constraint with a positive coefficient O_{cds}, and this allows to treat this variable as a candidate to be included in a cover.

In Section 5.4 we had only considered terminal constraints for cover cuts, as runway constraints had only unit coefficients in the left-hand side. When historic overages are included in the formulation, runway constraints with $O_{cds} > 1$ can now be considered, as they can potentially trigger a violated cover cut due to the non-unit coefficient of variable Ω_{cds}.

8.4. Week segmentation

The vast majority of slot allocation models assume that all flights in a series must operate at the same time. Fairbrother and Zografos (2020) propose a formulation where they break down the season into segments and allow flights within the same series to be allocated to a different time in each segment. Constraints are added to ensure these time differences across segments stay within a given range. This makes the resulting MIP more challenging, as it multiplies the number of decision variables of each series by the number of segments in which the series operates.

The WASG mandate that series of slots be allocated at *approximately* the same time on the *same day-of-the-week*. This is in line with ACL's current practice in some of the airports they coordinate. To reflect this, we use the formulation proposed by Fairbrother and Zografos (2020), but we segment the season by days-of-the-week instead of blocks of consecutive days.

Let W denote the set of week days, and let $W_i \subseteq W$ be the set of days-of-the-week in which the flights in series $i \in N$ operate. Variables x_{it} now require an additional index $w \in W_i$. Auxiliary integer variables $\bar{\psi}_i$ and $\underline{\psi}_i$ represent, the earliest and latest slot times, respectively, allocated to flights in series i. All other constraints in the problem, as well as the objective function are now defined for each $i \in N$ and each $w \in W_i$. The following constraints ensure that the range of slot times allocated to a series i does not exceed a fixed amount r. They are based on the formulation proposed by Fairbrother and Zografos (2020), where we have added the term y_{iw} in constraint (23) to ensure feasibility when a series is rejected.

$$\underline{\psi}_i \le \sum_{t\in T} t \ x_{iwt} + |T| \ y_{iw} \qquad i \in N, \ w \in W_i, \tag{23}$$

$$\sum_{t\in T} t x_{iwt} \le \bar{\psi}_i \qquad i \in N, \ w \in W_i, \tag{24}$$

$$\bar{\psi}_i - \underline{\psi}_i \le r \qquad i \in N. \tag{25}$$

We set r to 6 intervals (30 min), which is the average range tested by Fairbrother and Zografos (2020).

These constraints are problematic for the column-generation scheme in Caracal. All constraints considered so far had the form $a^T x \le b$ and fixing variables to 0 in these constraints led to a relaxation of our minimization problem. This condition does not hold for constraints (23). To remedy this, we propose an alternative formulation that is similar to the strong version of maximum turnaround constraints, with two differences: we link pairs of flights of the same series corresponding to two different days of the week and the maximum time between their allocated slots applies in both directions.

$$\sum_{t=1}^{s} x_{ivt} + \sum_{t=s+r+1}^{|T|} x_{iwt} \le 1, \qquad v, w \in W_i \ / u \neq v, \ i \in N, \ s \in T. \tag{26}$$

This formulation results in a tighter relaxation. However, it has the disadvantage that it increases the size of the model considerably, more so than strong turnaround constraints, as there are $|W_i| \times (|W_i| - 1)$ constraints (26) for each $i \in N$ and $s \in T$. To overcome this, we add (26) as lazy cuts. When using Caracal, we generate the lazy cuts with whatever x_{iwt} variables are available at that point and incorporate new variables to these cuts in Step 5.3 of Algorithm 1.

Table 3
Computational results of Baseline, Preprocessing and Caracal applied to the basic model and real airport data.

	z	Baseline			Preprocessing			Caracal				
		Time	Gap (%)	Time RN	Time	Gap (%)	Time RN	Time	Gap (%)	Time RN	LB RN	Heur RN
BFS	4900	87	0	87	11	0	11	**0.2**	0	0.2	4900	–
EMA	2845	66	0	66	34	0	34	**0.1**	0	0.1	2845	–
GLA	8200	458	0	458	212	0	212	**3.2**	0	1.0	8000	8200
EDI	14,450	604	0	604	497	0	497	**1.0**	0	1.0	14,450	–
BHX	304,905	43	0	43	19	0	19	**0.8**	0	0.8	304,905	–
LCY	668,145	602	0	602	193	0	193	**57.0**	0	57.0	668,145	–
LGW	18,461k	–	∞	2021	–	∞	1511	**393.0**	0	393.0	18,461k	18,462k

Table 4
Performance of Baseline, Preprocessing and Caracal applied to a model with maximum turnaround time and real data.

	z	Baseline			Preprocessing			Caracal				
		Time	Gap (%)	Time RN	Time	Gap (%)	Time RN	Time	Gap (%)	Time RN	LB RN	Heur RN
BFS	4900	86	0	87	57	0	57	**0.2**	0.00	0.2	4900	–
EMA	2845	77	0	77	47	0	47	**0.1**	0.00	0.1	2845	–
GLA	8200	474	0	474	228	0	228	**3.5**	0.00	3.5	8200	8200
EDI	14,495	650	0	650	448	0	448	**1.1**	0.00	1.1	14,495	–
BHX	372,850	59	0	59	22	0	22	**1.9**	0.00	1.1	372,850	–
LCY	727,035	–	0.04%	1275	**1746**	0	825	–	0.02	115.3	723,475	727,035
LGW	18,498k	–	∞	2667	–	∞	2364	–	3.81	1102.1	17,993k	18,537k

Table 5
Performance of Baseline, Preprocessing and Caracal applied to a model with historic overages and real data.

	z	Baseline			Preprocessing			Caracal				
		Time	Gap (%)	Time RN	Time	Gap (%)	Time RN	Time	Gap (%)	Time RN	LB RN	Heur RN
BHX	318,835	42	0	42	17	0	17	**2.9**	0	2.9	318,835	–
LCY	689,150	547	0	547	143	0	143	**94.1**	0	94.1	689,150	–
LGW	18,584k	2078	0	1989	1656	0	1366	**436.5**	0	409.5	18,584k	18,586k

8.5. Computational experiments

We evaluate the performance of Caracal and the preprocessing scheme on the models presented in the previous section and real instances from four Level 2 UK airports: Belfast International (BFS), Edinburgh (EDI), Glasgow (GLA) and East Midlands (EMA); and three Level 3 airports: Birmingham (BHX), London City (LCY) and London Gatwick (LGW). In all these instances, the data originates from slot requests and capacity limits declared in the 2020 summer season. We analyze the same three algorithms described in the computational experiments in Section 7: Baseline, Preprocessing, and Caracal. Maximum turnaround time and week segmentation constraints use the weak formulation in Baseline and Preprocessing and the strong formulation in Caracal. Experiments are conducted in the same machine with the same settings, except for the time limit, which was lowered to 3600 s to account for the multiple subproblems that must be solved per instance in Level 3 airports. We conduct four experiments with each algorithm and data set. First, we evaluate the basic formulation described in Section 3 with the addition of priority groups in Level 3 airports. Then we conduct three additional experiments where we take our model with priority groups (when applicable) and separately incorporate maximum turnaround constraints, historic overages, and week segmentation.

For Level 2 airports, the results in Tables 3–6 can be interpreted as follows: "Time" is the time to optimality or "–" when the time limit was reached. "Gap (%)" denotes the optimality gap at the end of the experiment; "Time RN" shows the time required to solve the root node, including cutting planes, heuristics and other algorithms run before branching; "LB RN" is the optimality gap reported after solving the root node, and "Heur RN" displays the objective value of the heuristic solution obtained by Caracal in the root node or "–" if the root node solution was integer. "z" indicates the optimal objective function value.

In Level 3 airports, where a lexicographic optimization approach is adopted, we apply the same definitions with a few special considerations: "Time" and "Time RN" aggregate the times to optimality and to solve the root node, respectively, across all individual subproblems, or "–" when the time limit is reached in any subproblem — in our experiments, this was always the last subproblem with $p = 7$. We define z as $\sum_{p \in P} z_p$ to ensure this value captures the total cost of displacements and rejections across all priority groups. Similarly, the "Gap (%)", "Heur RN" and "LB RN" columns are calculated considering all priority groups in the last iteration of the lexicographic approach.

Caracal outperforms the other methods in all Level 2 instances with all the models tested; it required less than 4 s to solve each of these instances, and it was 109.6 times faster than the preprocessing method and 390 times faster than the baseline, across all Level 2 instances. The model reduction achieved through the preprocessing method translated into a 3.9-fold speedup with respect to the baseline. All Level 2 instances were solved in under 10 min with this method.

The superiority of Caracal over the other methods is less pronounced in Level 3 airports. Not only do Level 3 airports generate larger instances, they are also more congested and the assumption that most of the series get allocated to their requested time in optimal solutions, which is an important factor for Caracal's performance, holds to a lesser extent than it did in Level 2 airports. In 9 of the 12 Level 3 instances tested, Caracal was the fastest method, with a speed-up consistent with that observed in Level 2 instances. This includes one instance that could

Table 6
Performance of Baseline, Preprocessing and Caracal applied to a model with week segmentation and real data.

	z	Baseline			Preprocessing			Caracal				
		Time	Gap (%)	Time RN	Time	Gap (%)	Time RN	Time	Gap (%)	Time RN	LB RN	Heur RN
BFS	4900	90	0	90	42	0	42	**0.2**	0	0.2	0	–
EMA	2615	109	0	109	37	0	37	**0.1**	0	0.1	0	–
GLA	7015	874	0	874	237	0	237	**3.2**	0	3.5	7015	8,200
EDI	12,520	934	0	934	329	0	329	**1.0**	0	1.1	0	–
BHX	259,474	66	0	66	32	0	32	**0.8**	0	1.1	0	–
LCY	633,740	724	0	724	412	0	412	**165.0**	0	158.4	633,740	633,795
LGW	17,363k	–	∞	–	–	∞	–	–	∞	–	∞	–

not be solved by any of the other two methods. Of the remaining three instances, one could only be solved by the preprocessing method and two could not be solved by any algorithm.

With the exception of the week segmentation model applied to LGW, Caracal solved the initial linear relaxation faster than other methods and the optimality gaps reported in the root node are small. The integer solutions obtained in the root node with our primal heuristic were either optimal or within 2.4% from the optimal. However, where extensive branching was required, Caracal struggled to improve the lower bounds and close small optimality gaps within the time limit, as it happened with some challenging instances in the synthetic data set.

Caracal may benefit from techniques to improve the lower bound in the root node, such as additional cutting planes, lower bounds obtained through the Lagrangian relaxation or more aggressive probing, to further reduce the optimality gap in the root node so that less branching is required to solve the problem.

9. Conclusions

In this paper we presented Caracal, an exact column-and-row generation algorithm that can solve real-world instances of the slot allocation problem significantly faster than the best exact methods known in the literature. The effectiveness of this algorithm can be attributed to the fact that, in practice, most of the series are allocated to their requested time, making the great majority of variables in the model "unnecessary", which in turn allows to eliminate a large proportion of capacity and turnaround constraints. The results obtained in this paper suggest that it is a flexible algorithm that can be applied to multiple variants of the problem, and it can obtain optimal or near-optimal solutions to very hard instances.

The primal heuristic in Caracal can be used as a stand-alone heuristic by stopping the algorithm just before branching. This heuristic has been shown to yield optimal or near optimal solutions in most hard instances, and it has the advantage over other heuristic methods in the literature that it provides an optimality gap; this is reassuring for slot coordinators.

Our problem-specific preprocessing pipeline achieved a substantial reduction of solution times by identifying a large number of unnecessary variables and constraints in the model with minimal computational effort. They can be applied to reduce the size of integer programs used in exact methods and MIP-based heuristics, and they have the potential to simplify feasibility checks used in some heuristic approaches.

Lastly, we provide a test set of synthetic instances with the aim of enabling benchmarking of slot allocation algorithms. These test instances cover different levels of complexity and yield computational results consistent with real-world Level 2 airports.

CRediT authorship contribution statement

Paula Fermín Cueto: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Data curation. **Sergio García:** Writing – review & editing, Supervision, Resources. **Miguel F. Anjos:** Writing – review & editing, Supervision, Resources.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors thank Airport Coordination Limited for providing valuable insights and real-world slot coordination data.

Appendix A. Omitted proofs

A.1. Proof of Proposition 1

Proposition 1. Given a series $i \in N$, a set of capacity constraints of type $c \in C$ with rolling period λ_c and duration w_c, multiple of λ_c, variables $x_{it''}$ for all t'' *not* listed below are dominated in these capacity constraints by some other variable $x_{it'}$ with t' on this list:

- $t' = \tau_i$,
- $t' = 1 + k\lambda_c,\ \ k \in \mathbb{Z}^+,\ \ t' > \tau_i$,
- $t' = k\lambda_c,\ \ k \in \mathbb{Z}^+,\ \ t' < \tau_i$.

Proof. If w_c is a multiple of λ_c, then the start times $s \in T_c$ of constraints (3) are also multiples of λ_c and no capacity constraints c start or finish in $(s, s + \lambda_c)$, for any $s \in T_c$. Then, given a series $i \in N$ and a time period $t' = 1 + k\lambda_c$ for some $k \in \mathbb{Z}^+$ such that $t' > \tau_i$, any variable $x_{it''}$ with $t' < t'' \leq (k+1)\lambda_c$ is dominated by $x_{it'}$, because $x_{it'}$ and $x_{it''}$ appear in the same capacity constraints with the same coefficients but variables $x_{it'}$ have a lower displacement. The same reasoning can be used to prove the symmetric case: $x_{it'} \succ x_{it''}$ holds for any t' and t'' such that $t' = k\lambda_c < \tau_i$ and $(k-1)\lambda_c < t'' < t'$.

Lastly, given a constraint with starting period $s = 1 + k\lambda_c$ such that $s \leq \tau_i < s + \lambda_c$, a similar reasoning can show that variables $x_{it''}$ with $s \leq t'' < s + \lambda_c$, $t'' \neq \tau_i$ are dominated by $x_{i\tau_i}$. □

A.2. Proof of Proposition 2

Proposition 2. Let $(i, j) \in E$ represent a pair of linked arriving (i) and departing (j) series. Consider two slots for the arriving series $t'' > \tau_i$ and $t' \geq \tau_i$. If $t' < t''$, then $x_{it'} \succ x_{it''}$ with respect to minimum turnaround constraints. An analogous dominance relation exists for the departing series: given slots $t'' < \tau_j$ and $t' \leq \tau_j$, if $t' > t''$, then $x_{jt'} \succ x_{jt''}$.

Proof. Consider the weak version of minimum turnaround constraints: $\sum_{t\in T} t\, x_{jt} - \sum_{t\in T} t\, x_{it} \geq t_{ij}$. Since these constraints and constraints (4) result in the same set of integer solutions, any variables that are dominated when we consider the weak version are also dominated using the strong version. The terms in this constraint can be rearranged as follows to present it in the form $a^T x \leq b$: $-\sum_{t\in T} t\, x_{jt} + \sum_{t\in T} t\, x_{it} \leq -t_{ij}$. Now the coefficient of variable $x_{it''}$, t'', is higher than that of variable $x_{it'}$ (t'). Since both t'' and t' are late slots for the arriving series and $t'' > t'$, then the slot t'' would incur in a higher displacement than t, namely, $f_{it''} > f_{it'}$. By the principle outlined at the start of Section 4, it can be concluded that $x_{it'} \succ x_{it''}$ and $x_{it''}$ can be fixed to 0. The same logic can be applied to prove the analogous rule for the departing series and is omitted here for brevity. □

A.3. Proof of Proposition 3

Proposition 3. Let $(i, j) \in E$ represent a pair of linked arriving (i) and departing (j) series. Given a pair of slots t' and t'' that meet the following conditions:

- $\tau_j \leq t' < t''$,
- there is a value of $k \in \mathbb{Z}^+$ such that $t' = 1 + k\,\lambda_{c'}$, $c' = \arg\min_{c\in C} \lambda_c$ and $t'' \leq (k+1)\,\lambda_{c'}$,
- all variables x_{it} with $t \in (t' - t_{ij}, t'' - t_{ij}]$ are dominated by another variable by Propositions 1 and 2,

then $x_{jt'} \succ x_{jt''}$ and $x_{jt''}$ can be fixed to zero in the model in (1)–(6). An equivalent dominance relation can be found for variables of the arrival series: $x_{it'} \succ x_{it''}$ for any pair of slots t', t'' that meet the following conditions:

- $t'' < t' \leq \tau_i$,
- there is a value of $k \in \mathbb{Z}^+$ such that $t' = k\,\lambda_{c'}$, $c' = \arg\min_{c\in C} \lambda_c$ and $t'' > (k-1)\lambda_{c'}$,
- all variables x_{jt} with $t \in [t'' + t_{ij}, t' + t_{ij})$ are dominated another variable by Propositions 1 and 2.

Proof. We prove the first part of Proposition 3, concerning dominated variables x_{it}. The same reasoning can be used to prove the symmetric case (variables x_{jt}). Consider an instance of the slot allocation problem described by (1)–(6) where some variables have been fixed to 0 following the preprocessing rules in Propositions 1 and 2. Consider an integer solution x^* to this problem with $x^*_{jt''} = 1$ for some $t'' > \tau_j$. Since this solution is feasible, the turnaround time between series i and j must be at least t_{ij}, and therefore the arriving series cannot have been scheduled later than $t'' - t_{ij}$. Proposition 3 assumes that the arrival variable $x_{i,t''-t_{ij}}$ had been fixed to 0. This assumption also applies to the other variables x_{it} with $t \in (t' - t_{ij}, t'' - t_{ij}]$, for some $t' \geq \tau_j$. This means that the arrival cannot have been scheduled later than $t' - t_{ij}$ under these conditions, so the turnaround time in this feasible solution must be at least $t'' - (t' - t_{ij})$, which is greater than the minimum, t_{ij}, by $t'' - t'$ 5-minute periods. As a result, under the conditions stated in Proposition 3, there exists an alternative solution with $x^*_{jt'} = 1$ where the turnaround time is at least t_{ij}. This solution is also feasible with respect to capacity constraints, as any variables that meet the first and second condition in Proposition 3 have a dominance relation with respect to capacity constraints by Proposition 2. The alternative solution with $x^*_{jt'} = 1$ is also better, as t' creates a lower displacement than t''. Therefore it can be concluded that $x_{jt'} \succ x_{jt''}$ and $x_{jt''}$ can be fixed to 0 in the slot allocation model in (1)–(6). □

A.4. Proof of Proposition 4

Proposition 4. Let $i \in N$ be an arrival series, $j \in N$ its linked departure series and t_{ij} the minimum turnaround time between them. If variable x_{js} for some $s \in T$ is fixed to 0, then the turnaround constraint with index s is dominated by the constraint with index $s-1$.

Proof. Consider two consecutive turnaround constraints with indices $s-1$, $s \in T$ and the pair of linked requests $(i, j) \in E$:

$$s-1: \quad \sum_{t=s-t_{ij}}^{|T|} x_{it} + \sum_{t=1}^{s-1} x_{jt} \leq 1, \tag{A.27}$$

$$s: \quad \sum_{t=s-t_{ij}+1}^{|T|} x_{it} + \sum_{t=1}^{s} x_{jt} \leq 1. \tag{A.28}$$

These two constraints only differ in two terms: the constraint with index s has the additional term x_{js} in the left-hand side (LHS) and is missing the term $x_{i,s-t_{ij}}$ present in constraint with index $s-1$. It is clear that if x_{js} is fixed to 0, then the LHS of constraint (A.27) can only be greater or equal to the LHS of constraint (A.28). Therefore, the latter constraint can be removed without altering the set of solutions. □

A.5. Proof of Proposition 5

Proposition 5. Let e_i and ℓ_i denote the earliest and latest available slots for series $i \in N$, respectively. Let $(i, j) \in E$ be a pair of linked series with minimum ground time t_{ij}. Turnaround constraints not included in (16) either do not include at least one assignment variable of each series, or they are dominated by another turnaround constraint and, therefore, they are redundant.

$$\sum_{t=s-t_{ij}+1}^{\ell_i} x_{it} + \sum_{t=e_j}^{s} x_{jt} \leq 1, \qquad (i,j) \in E,\ s \in T \ /\ \max\{e_j,\ e_i + t_{ij} - 1\} \leq s \leq \min\{\ell_j,\ \ell_i + t_{ij} - 1\}. \tag{16}$$

Proof. Let us first analyze the smallest value of s included, $s_{\min} = \max\{e_j,\ e_i + t_{ij} - 1\}$. There are two possibilities:

1. $e_j \geq e_i + t_{ij} - 1$. In this case, $s_{\min} = e_j$ and the second sum in the corresponding constraint (16) with index $s = s_{\min}$ would only include the term x_{je_j}. For any other $s < e_j$, the sum would be empty, the resulting constraints would not include any variables for series j, and these constraints would be redundant.
2. $e_i + t_{ij} - 1 > e_j$. In this case, $s_{\min} = e_i + t_{ij} - 1$ and the turnaround constraint (16) with $s = s_{\min}$ would be $\sum_{t=e_i}^{\ell_i} x_{it} + \sum_{t=e_j}^{e_i+t_{ij}-1} x_{jt} \leq 1$. Any other $s = s_{\min} - k$ with $k \in \mathbb{Z}^+$ would yield a constraint $\sum_{t=e_i-k}^{\ell_i} x_{it} + \sum_{t=e_j}^{e_i+t_{ij}-1-k} x_{jt} \leq 1$. Considering that e_i is the first slot available for flights i, this inequality is equivalent to $\sum_{t=e_i}^{\ell_i} x_{it} + \sum_{t=e_j}^{e_i+t_{ij}-1-k} x_{jt} \leq 1$, and from this expression it can be seen that this constraint can be safely eliminated as it is dominated by the turnaround constraint with $s = s_{\min}$.

The same logic can be applied to show that any constraints with $s > s_{\max}$ are redundant. □

Appendix B. Lifted GUB covers for capacity constraints

A GUB cover for a knapsack constraint is a cover such that no two elements in the cover belong to the same GUB constraint, given a set of non-overlapping GUB constraints. This additional restriction makes the separation problem much harder to solve than lifting simple knapsack covers (Gu et al., 1998). Here we show how the structure of the capacity constraints can simplify this problem.

For the sake of readability, let T_{cs} denote the set of slots included in the time window of a constraint $c \in C$, namely, $T_{cs} = \{t \in T \ /\ s \leq t \leq s + w_c - 1\}$. The following integer program represents the separation problem to find the most violated GUB cover — or to prove that none exists — introduced by Wolsey (1990), adapted to a terminal capacity constraint (c, d, s) with the set of non-overlapping GUBs derived from

assignment constraints (2):

$$\alpha = \min_{z} \sum_{i\in N}\sum_{t\in T_{cs}} (1 - \sum_{t\in T_{cs}} x^*_{it})\, z_{it}, \tag{B.29}$$

$$\text{s.t.} \sum_{i\in N}\sum_{t\in T_{cs}} \delta_{id}\, a_{ic}\, z_{it} > B_{cds}, \tag{B.30}$$

$$\sum_{t\in T_{cs}} z_{it} \le 1 \qquad i \in N, \tag{B.31}$$

$$z_{it} \in \{0,1\} \qquad i \in N,\, t \in T_{cs}. \tag{B.32}$$

Variables z_{it} with the same index i have the same coefficient in every constraint and the same costs in the objective function. This detail simplifies the separation problem substantially: we need to know whether a variable from a request i is selected in the optimal solution of the separation problem, but we do not need to know which one. This allows us to apply the transformation $\zeta_i = \sum_{t\in T_{cs}} z_{it}$. The separation problem in (B.29)–(B.32) can now be rewritten as

$$\alpha = \min_{\zeta} \sum_{i\in N} (1 - \sum_{t\in T_{cs}} x^*_{it})\, \zeta_i, \tag{B.33}$$

$$\text{s.t.} \sum_{i\in N} \delta_{id}\, a_{ic}\, \zeta_i > B_{cds}, \tag{B.34}$$

$$\zeta_i \in \{0,1\} \qquad i \in N. \tag{B.35}$$

The above problem is a standard knapsack cover separation problem that can be solved using the procedure employed for lifted knapsack covers, adding one step at the end: once a lifted cover based on variables ζ is found, it is converted into a GUB knapsack cover inequality in the original space by simply including all variables x_{it} for each request i in the cover (requests such that $\zeta^*_i = 1$) and each time period $t \in T_{cs}$.

Data availability

A link to part of the data is included in the manuscript. Other data used is confidential.

References

Achterberg, T., 2007. Constraint Integer Programming (Ph.D. thesis). Technische Universität Berlin, Available online: http://opus.kobv.de/tuberlin/volltexte/2007/1611/.

Achterberg, T., 2009. SCIP: solving constraint integer programs. Math. Program. Comput. 1 (1), 1–41. http://dx.doi.org/10.1007/s12532-008-0001-1.

Achterberg, T., Bixby, R.E., Gu, Z., Rothberg, E., Weninger, D., 2020. Presolve reductions in mixed integer programming. INFORMS J. Comput. 32 (2), 473–506. http://dx.doi.org/10.1287/ijoc.2018.0857.

Achterberg, T., Koch, T., Martin, A., 2005. Branching rules revisited. Oper. Res. Lett. 33 (1), 42–54. http://dx.doi.org/10.1016/j.orl.2004.04.002.

Agência Nacional de Aviação Civil (ANAC), 2021. Slot coordination.

Airport Coordination Limited (ACL), 2021. Latest airport info. URL www.acl-uk.org/latest-airport-info/.

Androutsopoulos, K.N., Madas, M.A., 2019. Being fair or efficient? A fairness-driven modeling extension to the strategic airport slot scheduling problem. Transp. Res. E 130, 37–60. http://dx.doi.org/10.1016/j.tre.2019.08.010.

Androutsopoulos, K.N., Manousakis, E.G., Madas, M.A., 2020. Modeling and solving a bi-objective airport slot scheduling problem. European J. Oper. Res. 284 (1), 135–151. http://dx.doi.org/10.1016/j.ejor.2019.12.008.

Artigues, C., 2017. On the strength of time-indexed formulations for the resource-constrained project scheduling problem. Oper. Res. Lett. 45 (2), 154–159. http://dx.doi.org/10.1016/j.orl.2017.02.001.

Berthold, T., 2014. RENS. Math. Program. Comput. 6 (1), 33–54. http://dx.doi.org/10.1007/s12532-013-0060-9.

Christofides, N., Alvarez-Valdes, R., Tamarit, J., 1987. Project scheduling with resource constraints: A branch and bound approach. European J. Oper. Res. 29 (3), 262–273. http://dx.doi.org/10.1016/0377-2217(87)90240-2.

European Commission, 1993. Council regulation (EEC) no. 95/93 of 18 january 1993 on common rules for the allocation of slots at community airports.

Fairbrother, J., Zografos, K.G., 2020. Optimal scheduling of slots with season segmentation. European J. Oper. Res. 291 (3), 961–982. http://dx.doi.org/10.1016/j.ejor.2020.10.003.

Fairbrother, J., Zografos, K.G., Glazebrook, K.D., 2019. A slot-scheduling mechanism at congested airports that incorporates efficiency, fairness, and airline preferences. Transp. Sci. 54 (1), 115–138. http://dx.doi.org/10.1287/trsc.2019.0926.

Fermín Cueto, P., 2022. Slot allocation synthetic data. http://dx.doi.org/10.7488/ds/3416, University of Edinburgh. School of Mathematics.

Gamrath, G., Koch, T., Martin, A., Miltenberger, M., Weninger, D., 2015. Progress in presolving for mixed integer programming. Math. Program. Comput. 7 (4), 367–398. http://dx.doi.org/10.1007/s12532-015-0083-5.

García, S., Labbé, M., Marín, A., 2011. Solving large p-median problems with a radius formulation. INFORMS J. Comput. 23 (4), 546–556. http://dx.doi.org/10.1287/ijoc.1100.0418.

Gu, Z., Nemhauser, G.L., Savelsbergh, M.W.P., 1998. Lifted cover inequalities for 0-1 integer programs: Computation. INFORMS J. Comput. 10 (4), 427–437. http://dx.doi.org/10.1287/ijoc.10.4.427.

Gurobi Optimizer Reference Manual, 2021. Gurobi optimization, LLC. URL www.gurobi.com.

Gurobi Support Portal, 2021. Gurobi optimization, LLC. https://tinyurl.com/gurobisupportportal.

Hillier, F.S., Lieberman, G.J., 2001. Introduction to operations research. In: Introduction to Operations Research. McGraw-Hill, USA, (Chapter 6).

IATA, 2024. Worldwide airport slot guidelines - edition 3. www.iata.org/en/policy/slots/slot-guidelines/.

Kaparis, K., Letchford, A.N., 2008. Local and global lifted cover inequalities for the 0–1 multidimensional knapsack problem. European J. Oper. Res. 186 (1), 91–103. http://dx.doi.org/10.1016/j.ejor.2007.01.032.

Katsigiannis, F.A., Zografos, K.G., 2021. Optimising airport slot allocation considering flight-scheduling flexibility and total airport capacity constraints. Transp. Res. B 146, 50–87. http://dx.doi.org/10.1016/j.trb.2021.02.002.

Katsigiannis, F.A., Zografos, K.G., 2023. Incorporating slot valuation in making airport slot scheduling decisions. European J. Oper. Res. 308 (1), 436–454. http://dx.doi.org/10.1016/j.ejor.2022.11.008.

Odoni, A.R., 2020. A Review of Certain Aspects of the Slot Allocation Process at Level 3 Airports Under Regulation 95/93. Technical Report, Massachusetts Institute of Technology.

Pellegrini, P., Castelli, L., Pesenti, R., 2011. Metaheuristic algorithms for the simultaneous slot allocation problem. Intell. Transp. Syst. IET 6 (4), 453–462. http://dx.doi.org/10.2139/ssrn.2037688.

Pritsker, A.A.B., Waiters, L.J., Wolfe, P.M., 1969. Multiproject scheduling with limited resources: A zero-one programming approach. Manage. Sci. 16 (1), 93–108. http://dx.doi.org/10.1287/mnsc.16.1.93.

Ribeiro, N.A., Jacquillat, A., Antunes, A.P., Odoni, A., 2019. Improving slot allocation at level 3 airports. Transp. Res. A 127, 32–54. http://dx.doi.org/10.1016/j.tra.2019.06.014.

Ribeiro, N.A., Jacquillat, A., Antunes, A.P., Odoni, A.R., Pita, J.P., 2018. An optimization approach for airport slot allocation under IATA guidelines. Transp. Res. B 112, 132–156. http://dx.doi.org/10.1016/j.trb.2018.04.005.

Thompson, G.L., Tonge, F.M., Zionts, S., 1966. Techniques for removing nonbinding constraints and extraneous variables from linear programming problems. Manage. Sci. 12 (7), 588–608. http://dx.doi.org/10.1287/mnsc.12.7.588.

van den Akker, J., Hurkens, C., Savelsbergh, M., 2000. Time-indexed formulations for machine scheduling problems: Column generation. INFORMS J. Comput. 12 (2), 111–124. http://dx.doi.org/10.1287/ijoc.12.2.111.11896.

van den Akker, J.M., Nachtigall, K., 1999. Slot Allocation by Column Generation. Technical Report, National Aerospace Laboratory NLR.

Wolsey, L.A., 1990. Valid inequalities for 0–1 knapsacks and MIPS with generalised upper bound constraints. Discrete Appl. Math. 29 (2–3), 251–261. http://dx.doi.org/10.1016/0166-218x(90)90148-6.

Zografos, K.G., Jiang, Y., 2019. A bi-objective efficiency-fairness model for scheduling slots at congested airports. Transp. Res. C 102, 336–350. http://dx.doi.org/10.1016/j.trc.2019.01.023.

Zografos, K.G., Madas, M.A., Androutsopoulos, K.N., 2016. Increasing airport capacity utilisation through optimum slot scheduling: review of current developments and identification of future needs. J. Sched. 20 (1), 3–24. http://dx.doi.org/10.1007/s10951-016-0496-7.

Zografos, K.G., Salouras, Y., Madas, M.A., 2012. Dealing with the efficient allocation of scarce resources at congested airports. Transp. Res. C 21 (1), 244–256. http://dx.doi.org/10.1016/j.trc.2011.10.008.