Computers and Operations Research 127 (2021) 105159

Contents lists available at ScienceDirect

Computers and Operations Research

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

Air traffic flow management with layered workload constraints ☆

Carlo Mannino [a,b], Andreas Nakkerud [a,b,*], Giorgio Sartor [b]

[a] Department of Mathematics, P.O box 1053 Blindern, 0316 Oslo, Norway
[b] SINTEF, P.O box 124 Blindern, 0314 Oslo, Norway

ARTICLE INFO

Article history:
Received 4 May 2020
Revised 17 November 2020
Accepted 20 November 2020
Available online 28 November 2020

2010 MSC:
90B06
90B20
90C06
90C08
90C11
90C90

Keywords:
Hotspot problem
Air traffic management
Air traffic flow management
Scheduling
Job-shop scheduling
Optimization
Mixed integer programming
Linear programming

ABSTRACT

Many regions of the world are currently struggling with congested airspace, and Europe is no exception. Motivated by our collaboration with relevant European authorities and companies in the Single European Sky ATM Research (SESAR) initiative, we investigate novel mathematical models and algorithms for supporting the Air Traffic Flow Management in Europe. In particular, we consider the problem of optimally choosing new (delayed) departure times for a set of scheduled flights to prevent en-route congestion and high workload for air traffic controllers while minimizing the total delay. This congestion is a function of the number of flights in a certain sector of the airspace, which in turn determines the workload of the air traffic controller(s) assigned to that sector. We present a MIP model that accurately captures the current definition of workload, and extend it to overcome some of the drawbacks of the current definition. The resulting scheduling problem makes use of a novel formulation, Path&Cycle, which is alternative to the classic big-*M* or time-indexed formulations. We describe a solution algorithm based on delayed variable and constraint generation to substantially speed up the computation. We conclude by showing the great potential of this approach on randomly generated, realistic instances.

1. Introduction

Air Traffic Flow Management consists in regulating air traffic to prevent congestion while optimizing the usage of the available capacity. This is a very important topic in SESAR (SESAR, 2020), an ongoing extensive European collaborative project with the objective of improving and modernizing every aspect of the air traffic management in Europe. In this paper, we present recent results related to our work on SESAR sub-projects.

The European airspace is divided into control sectors, each assigned to one or more air traffic controllers who are in charge of guaranteeing the safety of the air traffic. The tasks of a controller (e.g., communicate with pilots, handover flights to the controller of an adjacent sector, prevent potential conflicts, etc.) are heavily regulated, each one requiring a certain amount of time and effort. Within each sector, controllers are solving the Air Traffic Control problem of finding a feasible combination of flight paths, subject to regulations about the temporal and spatial separation of airplanes. In Europe, these flight paths are also subject to the schedule determined by the central control authority. The workload of a controller is usually a function of these tasks which, in turn, is a function of the flights traversing the sector. Therefore, capacity constraints are imposed on each sector to regulate the number of flights and consequently to keep the workload of controllers within the set boundaries. In Europe, this capacity is computed through involved simulation procedures, standardized by the European control authority. Briefly, depending on the traffic scenario, each task performed by a controller is decomposed into atomic controlling actions, and each such action has an expected time required for execution. For a given traffic scenario, the expected overall time use necessary for a controller in a time window of one hour must not exceed 42 min (70% workload). Then, many historical traffic

☆ This work was funded by The Research Council of Norway [grant numbers 237718, 267554].

* Corresponding author at: Department of Mathematics, P.O box 1053 Blindern, 0316 Oslo, Norway.
E-mail address: andreana@math.uio.no (A. Nakkerud).

https://doi.org/10.1016/j.cor.2020.105159
0305-0548/

scenarios are assessed and, for each such scenario, the maximum number of flights which can be controlled within the time threshold is determined. The capacity of the sector is then simply the minimum of such numbers overall traffic scenarios (Flynn et al., 2003). Whenever a set of flights violates these capacity regulations in a certain sector, we say there is a *hotspot* (Allignol et al., 2012; Dubot et al., 2016) (see Fig. 1 for an illustrated example). In this paper, we present techniques that allow more precise modelling of controller workload, which in turn could be used to allow higher capacities without the risk of overloading a sector.

Typically, the working day of a controller is divided into fixed time windows, e.g., from 10 a.m. to 11 a.m., from 11 a.m. to 12 p.m., and so on. Then the regulations usually impose a limit on the number of aircraft entering the sector during each time window. Since flight plans are submitted to the control authorities by each airline only a few hours before departure (sometimes even half an hour in the case of private jets), it is not uncommon that they will ultimately produce a hotspot. Control authorities have many ways to deal with this issue, such as asking the airline to submit an altered flight plan. When a satisfactory solution cannot be agreed upon in due time, control authorities can simply choose to delay the departure. This is the scenario considered in this paper. In particular, given the schedule for a set of flights and given the capacity constraints for all sectors, the Hotspot Problem (HP) consists in finding new (possibly delayed) departure times for all flights such that the corresponding schedule is hotspot-free and the sum of delays is minimized.

The exact definition of HP depends on the exact definition of *hotspot*. As mentioned above, the current definition agreed in Europe (Flynn et al., 2003) considers fixed intervals of time and, for each sector and for each interval of time, it compares the aircraft entry counts with the predefined sector capacity. There are several drawbacks with this definition. The most evident is perhaps a phenomenon called bunching, where the excess flights of a certain interval of time are simply moved and amassed at the beginning of the next interval. We propose to solve this by considering sliding intervals of time, where the sector capacity must be respected in any interval of time of a certain length. A recent study (Guibert et al., 2019), which also involves the European control authority, considers the possibility to jointly take into account entry counts and occupancy counts. The first measures the number of aircraft entering a sector within a certain interval of time, whereas the latter measures the number of aircraft simultaneously traversing a sector within a certain interval of time (see Fig. 2). In this paper, we describe a model that considers both sliding intervals of time and occupancy counts, and that can be immediately extended to support fixed interval of times and entry counts.

The literature on the hotspot problem is small, with only a few papers dealing with somewhat related problems (of which an overview can be found in Zhong (2018)). None of these papers matches exactly with the Hotspot Problem. In Schefers et al. (2017), the airspace is subdivided into micro-cells of unit capacity, and airplanes can be delayed at the departure, but only within the assigned time slot. The question of delaying flights to reduce congestion is discussed in de Jonge and Seljée (2011) along with a greedy-like prioritization-based iterative algorithm to compute delays. In contrast, a MILP model for the same problem is discussed in Damhuis et al. (2015), with some interesting experimental results on some simulation scenarios. In Vaaben and Larsen (2015) the problem is studied from the side of the airlines. When control authorities issue flying restrictions, airlines need to modify flight trajectories to meet such restrictions. Both schedules and trajectories can be modified in this study, but the feasible trajectories are chosen from a predefined, finite set. In Sailauov and Zhong (2016) the standpoint is again from the control authority side. The model factors in many details, including, for instance, stops at intermediate airports and fairness of the solutions. The resulting, overarching time-indexed MILP model is probably too complex to be solved for the size of practical instances, and indeed the experiments reported in the paper involves only two flights.

One of the most closely related problems in the literature is presented in Kim et al. (2009), where a combination of greedy and randomized rounding heuristics is used to minimize the maximum occupancy of any sector. Our problem mainly differs from the one in Kim et al. (2009) in that we minimize delays while respecting capacity restrictions rather than minimize occupancy, and that we apply an exact method. Our computational experiments (Section 7) are on instances of the single-sector problem, which is already NP-hard in general Kim et al. (2009). Our ability to solve these instances using our exact method rests on the fact that we have an available schedule which is already close to feasible. If no schedule is available, a heuristic method like the one presented in Kim et al. (2009) could be applied as a preprocessing step.

Most of the above-mentioned papers focus on modelling issues, using either constraint (CP) or mathematical programming—mainly Mixed Integer Programming (MIP). The resulting formulations are then solved by directly invoking a MIP solver (with the exception of Damhuis et al. (2015) and Kim et al. (2009) where delayed constraint generation is applied). In our experience, however, this approach typically does not suffice to find exact solutions to instances of a practical size. The main reason is that the classical formulations for this class of problems are either too weak or too large to work well in practice without additional algorithmic enhancements. More specifically, the Hotspot Problem is a variant of the job-shop scheduling problem with blocking and no-wait constraints (see Mascis and Pacciarelli, 2002), where the sectors can be seen as machines and the flights as jobs. The main issue of this class of problems is that we have to introduce disjunctive constraints to represent and solve conflicts in the use of shared resources. Basically, two MIP models are competing in the literature: the big-*M* and the time-indexed formulations (see Queyranne and Schulz, 1994). The former usually provides weak bounds, and thus large search trees; the latter produces better bounds but at the cost of increasing time to solve the relaxations. While time-indexed formulations work well for some scheduling problems, it is a well-known problem that they struggle when the number of time slots becomes large, which is often the case in modern transportation scheduling (Mannino and Mascis, 2009). So, the reduction in the number of branching nodes due to a stronger bound is typically not enough to compensate for the increase in running times.

Finally, while the Hotspot Problem takes into account en-route conflicts, it only allows delaying airplanes at the airports. The Hotspot Problem, therefore, belongs to a larger class of air traffic scheduling problems in terminal control areas, such as Avella et al. (2017), Kim et al. (2009), Bianco et al. (2006), D'Ariano et al. (2015), Samà et al. (2017).

In this paper, we experiment with a different MIP formulation for job-shop scheduling in transportation, recently introduced by Lamorgese and Mannino (2019) for rail traffic management, and then extended to cope with air traffic in Mannino and Sartor (2018). The *Path&Cycle formulation* is a MIP formulation for job-shop scheduling problems. As the classical big-*M* formulation, it uses a set of binary variables per disjunction, but without resorting to the notorious big-*M* coefficients and constraints. This allows for stronger relaxations, without the excessive increase in running times typically associated with time-indexed formulations. For the Hotspot Problem, the Path&Cycle formulation was indeed proven to be more effective than the big-*M* formulation (Mannino and Sartor, 2018). In this paper, we develop a Path&Cycle formulation for the Hotspot Problem, where delayed constraint generation is applied to cope with the potentially large number of constraints.

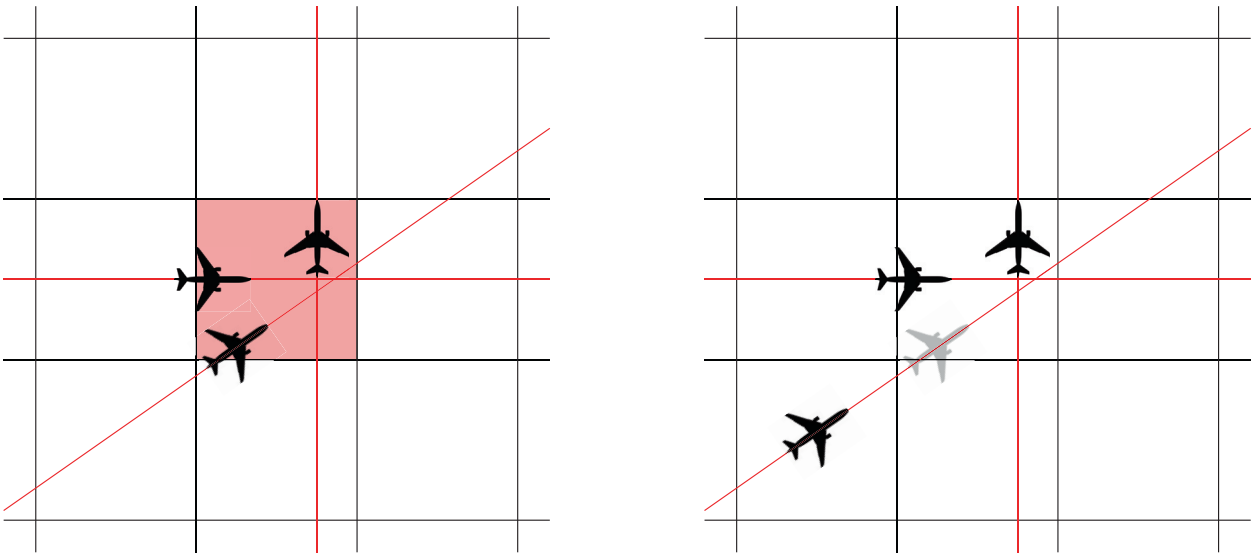

Fig. 1. In this illustration a sector is considered to be a hotspot if it is occupied by more than two flights at any point in time. On the left, the highlighted sector is a hotspot. On the right, we have delayed one flight to resolve the hotspot.

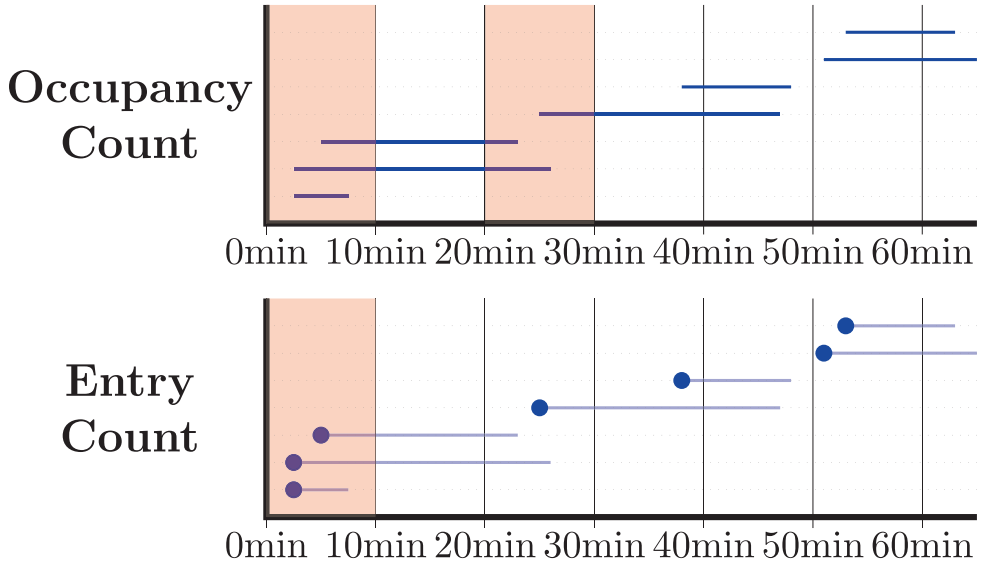

Fig. 2. Occupancy and entry counts with fixed windows of capacity 2. Aircraft entries are shown as dots to illustrate that only the first 10-minute window contains more than 2 entries.

To speed up the separation process, we also describe a projection-based, preprocessing technique that has the potential to significantly reduce the size of the *alternative graph* which is the input to our solution algorithm. This projection technique can also be applied to other problems that can be modelled using an alternative graph, for example, train scheduling. Finally, since real-life instances were not available, we tested our approach on randomly generated instances of sizes comparable with the real ones.

A preliminary to this work (Mannino et al., 2018) was presented at SESAR Innovation Days 2018, organized by the Single European Sky ATM Research (SESAR) Joint Undertaking under the Horizon 2020 framework.

Our contribution is twofold: modelling and algorithmic. From the modelling standpoint, we introduce and model the concepts of sliding window capacity constraints (Section 3.1) and and fixed window capacity constraints (Section 5.1), counting either occupancy (Section 3.2 and Mannino and Sartor (2018)) or entries (Section 5.2) within the specific window towards capacity. Finally, we discuss layering of capacity constraints (Section 5.3). These extensions allow for a better representation of the actual controller workload with respect to previous works. From the algorithmic standpoint, we have extended the Path&Cycle solution algorithm to tackle these model extensions.

The rest of the paper is organized as follows. In Section 2, we describe in detail the Hotspot Problem, and in Section 3 we introduce a mathematical model for the Hotspot Problem. We introduce the Path&Cycle model for the Hotspot Problem in Section 4, and in Section 5 we introduce extensions to this model. We discuss our solution algorithm and its implementation in Section 6, and our computational results in Section 7.

2. The hotspot problem

We are given a set $\mathcal{F}$ of flights through an airspace divided into predefined control sectors $\mathcal{S}$. Conventionally, we also include in $\mathcal{S}$ a fictitious "arrival sector" a_f for every $f \in \mathcal{F}$, to represent the final

sector of flight f. Note that a_f may be an airport or the space outside the control region. Also, we conventionally assume $a_f \neq a_g$ for any pair of distinct flights $f, g \in \mathscr{F}$. The route of each flight $f \in \mathscr{F}$ is given as an ordered set $\mathscr{S}_f = \{s_1^f, s_2^f, \ldots, s_{n_f}^f\}$ of sectors, with $a_f = s_{n_f}^f$. Also, we let $d_f = s_1^f$ be the departure sector (departure sectors of different flights may coincide). Note that the first sector s_1^f may be the one immediately after the flight takes off from a controlled airport, or the first sector in the controlled airspace when the flight enters the control space already airborne. The *travel time* Λ_f^s of flight f through sector $s \in \mathscr{S}_f$ is the time f uses to travel through s (we let $\Lambda_f^{a_f} = 0$). The *release time* Γ_f of f is either (i) the (expected) earliest departure time of f (if the departure airport is within the controlled airspace), or (ii) the time f enters the controlled airspace. Release times are given relative to an arbitrary *reference time* t_o. Because the task is to schedule flights sometime in the future, normally (but not necessarily) t_o coincides with the time when such planning is carried out. We may assume t_o precedes all release times, so we have $\Gamma_f \geqslant 0$ for all f.

The planning horizon H is subdivided into a set of contiguous intervals, the *time windows*. The subdivision is completely defined by the starting time and the size of each time window. Typically, one-hour intervals are considered, starting at 12 a.m., 1 a.m., ..., but other window sizes may be possible. Each control sector $s \in \mathscr{S}$ is assigned a capacity $c_s \in \mathbb{Z}$. There are two alternative ways to interpret such capacity.

2.1. Entry count

In this interpretation, the capacity c_s of a sector represents the maximum number of flights which can enter the sector in the time window. Indeed, if things go according to plan, most of the (time-consuming) activities carried out by controllers occur when a flight enters the sector. This is the definition used by the European control authorities.

2.2. Occupancy count

The capacity c_s of a sector represents the maximum number of flights which can be in the sector during the time window. Note that, depending on the travel time and window size, entry and occupancy counts may differ significantly (see Fig. 2).

We finally define the schedule of a flight $f \in \mathscr{F}$ as the time $t_s^f \in \mathrm{IR}$ the flight enters sector $s \in \mathscr{S}_f$.

The Hotspot Problem is then the problem of finding a feasible schedule $\mathbf{t}^f$ for each flight such that, for each sector $s \in \mathscr{S}$, the number of flights in s never exceeds the capacity c_s of the sector in any predefined time window.

Note that since the travel time in each sector is fixed, that is, it is subject to a no-wait constraint, the schedule of a flight is completely determined once the departure time is established. As a consequence, flights entering the controlled space from "outside" will have fixed release times, and their schedules cannot be modified.

2.3. Fixed and sliding windows

The official definition of hotspot (Flynn et al., 2003) is based on the above described *fixed window* approach. This approach has some major drawbacks. The first is that it may lead to unwanted violations of capacity. Suppose we ensure that in given fixed windows, say from 10 a.m. to 11 a.m. and from 11 a.m. to 12 a.m., the number of aircraft in a given sector does not exceed the capacity, say 10. This does not prevent that in "intermediate" windows, say from 10:30 a.m. to 11:30 a.m., we have up to 20 flights in the sector. The second is mainly an operational problem, namely *bunching*. When the capacity of a sector is violated during a time window, one possibility is to hold some of the involved flights at the airport. Obviously, one would like to generate the least possible delay, so a natural algorithm is to hold the flights precisely the time necessary to skip the overloaded time window. However, this may produce a schedule with many flights bunching up at the beginning of the next time window.

An alternative approach that solves both the "intermediate" capacity violations and bunching is the *sliding window*. It simply states that for *any* time interval of a given size, the number of flights in a sector should not exceed the capacity of the sector. In other words, the starting time of the control window is not fixed and can be anywhere in the time horizon. Depending on the adopted time window, we have different definitions of a hotspot.

2.4. Fixed window hotspot

A *fixed window* hotspot occurs when too many flights occupy the sector during one of the predefined fixed windows. In Fig. 3 the windows have width 10 min, and start at time $0, 10, 20, \ldots$. In this example, the windows do not overlap, but in principle, they could. The red shaded areas of the fixed window portion of Fig. 3 shows when the sector is a fixed window hotspot.

2.5. Sliding window hotspot

A *sliding window* hotspot occurs when too many flights occupy the sector during any period of time at most as long as the window width Δ. It may be viewed as a family of overlapping fixed windows, where the next window starts one unit of time after the previous. In Fig. 3, the sliding window has width 10 min. The red shaded areas of the sliding window portion of Fig. 3 shows when the sector is a sliding window hotspot. The grey crosshatched area around the last hotspot shows how far that window can slide with the sector remaining a hotspot.

We summarize the above with the following formal definition.

Definition 1. Let $n_s(t)$ be some count of flights in sector s at time t based on the current planned departure times. The sector s is a *hotspot* at time t if $n_s(t) > c_s$, where c_s is the capacity of s.

The form of $n_s(t)$ in the above definition depends on the type of the corresponding capacity constraint. We will use Δ to denote window width, and we will subscript Δ to denote widths of specific windows. We let $W(t)$ denote the window corresponding to time t, that is either $W(t) = W_{\text{fixed}}(t) = [a, a + \Delta]$ is the fixed window interval such that $t \in [a, a + \Delta]$, or $W(t) = W_{\text{sliding}}(t) = [t, t + \Delta]$ is the sliding window starting at t. Let $I_f^s = [t_f^0, t_f^1]$ be the time interval flight f spends in sector s. Then

$$n_s(t) = \begin{cases} |\{f \in \mathscr{F} : I_f^s \cap W(t) \neq \varnothing\}|, & \text{for occupancy counts} \\ |\{f \in \mathscr{F} : t_f^0 \in W(t)\}|, & \text{for entry counts.} \end{cases} \quad (1)$$

In order to model controller workload more realistically, we also propose layering different hotspot definitions. We propose using a long-term window to manage sustained workload, and a simultaneous short-term window to manage peak workload (Section 5.3). In this case, the capacity may not depend only on the sector, but also on the specific capacity constraint.

3. Modelling the sliding window hotspot problem

In this section, we present our MILP model for the sliding window hotspot problem. This model is an extension of the novel Path&Cycle formulation, which is a special version of the job-

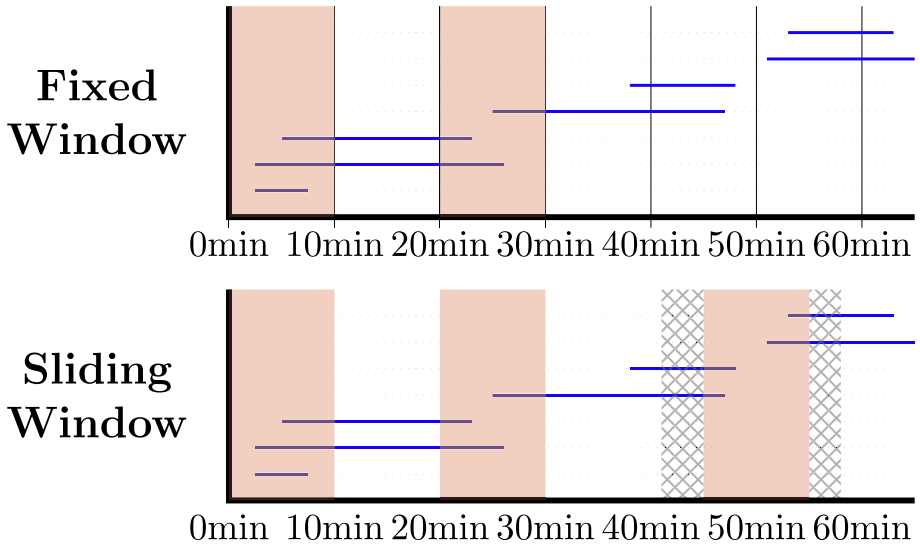

Fig. 3. Fixed time windows versus sliding time windows. The red boxes (and line) illustrate times when the section becomes a hotspot. In each case, we have set the capacity to 2. In the sliding window case, the windows can move slightly to the left and right with the sector remaining a hotspot, as illustrated by the grey crosshatched areas on the right side of the figure. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

shop scheduling problem with blocking and no-wait constraints (Mascis and Pacciarelli, 2002). The formulation was recently introduced by Lamorgese and Mannino in Lamorgese and Mannino (2019) for train scheduling and then extended by Mannino and Sartor in Mannino and Sartor (2018) to flight scheduling.

Let $f \in \mathscr{F}$ be a flight and $s \in \mathscr{S}_f$ be a sector on the flight plan $\mathscr{S}_f = \{d_f = s_1^f, s_2^f, \ldots, s_{n_f}^f = a_f\}$ of f. With some abuse of notation, if $s \in \mathscr{S}_f \setminus \{s_{n_f}^f\}$, we denote by t_f^{s+1} the time f enters the sector following s in the flight plan.

Using this notation, since t_f^{s+1} is also the time when f leaves sector s. We get the *schedule constraints*

$$t_f^{s+1} - t_f^s = \Lambda_f^s \tag{2}$$

that is the travel time of f through s is precisely Λ_f^s.

Next, recalling that the release time $\Gamma_f \geqslant 0$ is given relative to the *reference time* t_o, we have

$$t_f^{d_f} - t_o \geqslant \Gamma_f \tag{3}$$

for all flight f, with equality holding if f enters the controlled region already airborne.

Any feasible schedule $\mathbf{t}$ must satisfy (2) and (3). Linear constraints of the form $t_v - t_u \geqslant l_{uv}$ are called *time precedence* constraints. We note that (2) can be written as a pair of time precedence constraints.

3.1. Modelling sliding window constraints

Let Δ be the size of the sliding window and $\epsilon > 0$ be a small constant. We say that two distinct flights $f, g \in \mathscr{F}$ *do not meet* in a sector $s \in \mathscr{S}_f \cap \mathscr{S}_g$ if either (i) f leaves sector s at least Δ units of time before g enters it, that is $t_g^s - t_f^{s+1} \geqslant \Delta + \epsilon$; or (ii) g leaves sector s at least Δ units of time before f enters it, that is $t_f^s - t_g^{s+1} \geqslant \Delta + \epsilon$. In contrast, we say that f, g *meet* in s if none of the two conditions (i) and (ii) is satisfied, that is,

$$t_g^s - t_f^{s+1} \leqslant \Delta \ \wedge \ t_f^s - t_g^{s+1} \leqslant \Delta.$$

Because f, g either meet or do not meet in sector s, we have that any feasible schedule must satisfy the following constraint:

$$t_g^s - t_f^{s+1} \geqslant \Delta + \epsilon \quad \bigvee \quad t_f^s - t_g^{s+1} \geqslant \Delta + \epsilon \quad \bigvee \quad t_g^s - t_f^{s+1} \leqslant \Delta \ \wedge \ t_f^s - t_g^{s+1} \leqslant \Delta. \tag{4}$$

The above constraint is called a *disjunctive constraint*, and it is the disjunction of three terms. The first two terms are time precedence constraints, whereas the third term is a conjunction of two time-precedence constraints. Any feasible schedule $\mathbf{t}$ will satisfy exactly one of the three terms in disjunction (4).

In order to represent this disjunctive constraint in a MILP, we introduce the *selection variables*. For any ordered pair (f, g) of flights f, g both flying through a common sector $s \in \mathscr{S}_f \cap S_g$, we define the *precedence variable*

$$y_{fg}^s = \begin{cases} 1, & \text{if } f \text{precedes } g \text{ in } s \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

and for any unordered pair $\{f, g\}$ of distinct flights f, g both flying through a common sector s, we define the *meeting variable*

$$z_{fg}^s = \begin{cases} 1, & \text{if } f \text{ and } g \text{ mhboxmeetin } s \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where we have preferred the notation z_{fg}^s for $z_{\{fg\}}^s$, with the convention that z_{fg}^s is the same as z_{gf}^s.

So, associated with a pair of distinct flights f, g with a shared sector s, we have three binary variables: two precedence variables and one meeting variable. Each of these variables corresponds to one of the terms of the disjunction (4) associated with f, g, and s. Since exactly one of the three terms must be satisfied by any feasible schedule, the binary variables must satisfy the following *selection constraints*:

$$y_{fg}^s + y_{gf}^s + z_{fg}^s = 1. \tag{7}$$

We use $(\mathbf{y}, \mathbf{z})$ to denote the vector of selection variables, that is, the *selection vector*. A schedule $\bar{t}$ associated with a selection vector $(\bar{y}, \bar{z})$ must satisfy, for all pair of distinct flights $f, g \in \mathscr{F}$, and all $s \in \mathscr{S}_f \cap \mathscr{S}_g$

$$\begin{array}{llll} & (i) & t_g^s - t_f^{s+1} \geqslant \Delta + \epsilon, & \text{if } \bar{y}_{fg}^s = 1 \\ \text{or} & (ii) & t_f^s - t_g^{s+1} \geqslant \Delta + \epsilon, & \text{if } \bar{y}_{gf}^s = 1 \\ \text{or} & (iii) & t_g^s - t_f^{s+1} \leqslant \Delta \ \wedge \ t_f^s - t_g^{s+1} \leqslant \Delta, & \text{if } \bar{z}_{fg}^s = 1. \end{array} \tag{8}$$

In other words, the selection variables decide which time precedence constraints in each disjunction must be satisfied by the schedule **t**. Associated with any selection vector (y,z) we have thus a system of precedence constraints. This motivates the next:

Definition 2. We let $A(y,z)$ be the system of precedence constraints (2) and (3), and the constraints of the disjunctions (8) associated with the selection vector (y,z).

The system of precedence constraints $A(y,z)$ ensures that each flight's schedule is feasible in isolation and that the precedence decisions coded by **y** and **z** are respected. However, this system of constraints does not take into account capacity.

3.2. Capacity constraints

The vector **z** determines which pairs of flights meet in a given sector. We say that a set of flights $F \subseteq \mathscr{F}$ *meet* is a shared sector s if there is a time window of size Δ when all flights are in s. Now, a set $F \subseteq \mathscr{F}$ of flights meet in a sector s if and only if every pair $f,g \in F$ of distinct flights meet in s. In this case we have that $z^s_{fg} = 1$ for all distinct $\{f,g\} \subseteq F$ and $\sum_{\{f,g\} \subseteq F} z^s_{fg} = \binom{|F|}{2}$.

We denote by $\mathscr{F}_s \subseteq \mathscr{F}$ the set of flights going through s and let c_s be the capacity of s, i.e. no set of $c_s + 1$ flights can meet in s. Then, for every set of flight $F \subseteq \mathscr{F}_s$ where $|F| = c_s + 1$, **z** must satisfy the following constraint:

$$\sum_{\{f,g\} \subseteq F} z^s_{fg} \leqslant \binom{c_s+1}{2} - 1. \tag{9}$$

A meeting (selection) vector is *hotspot free* if it satisfies (9).

3.3. A disjunctive formulation for the Hotspot problem

We can now state more formally our basic version of the Hotspot problem.

3.3.1. The sliding window hotspot problem

The Sliding Window Hotspot problem amounts to finding a hotspot free selection vector $(\mathbf{y},\mathbf{z})$ satisfying (7) and (9), and a schedule **t** satisfying $A(y,z)$, such that the cost $w(\mathbf{t})$ of the schedule is minimized.

The function $w : \mathbf{t} \to \mathbb{R}$ determines the cost of the schedule and typically increases monotonically with the delays (that is with **t**)). Note that constraints (2), (3), (8) and (9), provide a *disjunctive formulation* for the feasible solutions $(\mathbf{y},\mathbf{z},\mathbf{t})$ to the Hotspot problem. A standard way to linearize (8) is by means of the so called big-*M* trick (see, for instance, Queyranne and Schulz, 1994). However, as discussed in the introduction, this leads to very weak relaxations. Following (Lamorgese and Mannino, 2019), we prefer a different model.

4. The Path&Cycle model

4.1. The route node graph

At the heart of the Path&Cycle formulation is an event graph $G = (V,E)$ called the *route node graph*. The nodes are associated with the time variables of the disjunctive formulation for the hotspot problem. In particular, V contains a special node o, the *origin*, associated with the reference time variable t_o and a *route node* $\langle f,s\rangle$ associated with the variable t^s_f, for each $f \in \mathscr{F}, s \in \mathscr{S}_f$. The node $\langle f,s\rangle$ represents the event "flight f enters sector s". The directed arcs of the graph are associated with time precedence constraints between the schedule variables. So, if $u = \langle f,s\rangle \in V$ and $v = \langle g,r\rangle \in V$, then $(u,v) \in E$ with length l_{uv} represents the constraint $t^g_r - t^f_s \geqslant l_{uv}$. Note that the inequality $t_u - t_v \leqslant l$ is equivalent to $t_v - t_u \geqslant -l$, which in turn is associated with an arc (u,v) with length $-l$. Furthermore, an equality precedence constraint $t_v - t_u = l$ can be transformed into two inequalities, which in turn correspond to two anti-parallel arcs (u,v) and (v,u) of length l and $-l$, respectively.

Release time constraints (3) are thus represented by arcs from the origin to the node associated with the first sector of every flight. Fig. 4 shows an example of a route node graph for two flights.

When two flights f and g both fly through a common sector s, then either they meet in s or one precedes the other as represented by (4). Since each term in the disjunction is either a time precedence constraint or the conjunction of two time-precedence constraints, it can be represented in the route node graph by arcs called *alternative arcs*, as shown in Fig. 5. The alternative arcs, which are drawn in the figure as either dashed or dotted arrows, have length equal Δ or $-\Delta$, according to (8). That is, the arc lengths are determined by the window width Δ. Note that we have two arcs associated with the "meet" case of the disjunction. Now, each alternative arc (or pair of arcs) is associated with one term in the disjunctive constraint (4), which in turn is associated with a selection variable. Consequently, each alternative arc is associated with exactly one selection variable. For instance, in Fig. 5, the arc $(\langle g,s_2\rangle, \langle f,s_3\rangle)$ is associated with the meeting variable $z^{s_2}_{fg}$.

We now summarize the above construction by giving a formal definition of the route graph.

Definition 3. Let $\mathscr{F}$ be a set of flights and, for $f \in \mathscr{F}$, let Γ_f be the departure time, and let Λ^s_f be the travel time for each $s \in \mathscr{S}_f$. Let $\mathscr{F}^* \subseteq \mathscr{F}$ be the set of flights with fixed departure times. Let $\mathscr{U}_{\mathscr{F}} = \{\langle f,s\rangle : f \in \mathscr{F}, s \in \mathscr{S}_f\}$ be the set of route nodes. The *route node graph* is a directed graph $G = (V, E_R \cup E_T \cup E_A)$, where E_R are the *release arcs*, E_T are the *fixed precedence arcs*, and $E_A = E^y_A \cup E^z_A$ are the *alternative arcs*. We have

$$\begin{aligned}
V &= \mathscr{U}_{\mathscr{F}} \cup \{o\}\\
E_R &= \{(o, \langle f, d_f\rangle) : f \in \mathscr{F}\}\\
&\quad \cup \{(\langle f, d_f\rangle, o) : f \in \mathscr{F}^*\}\\
E_T &= \{(\langle f,s\rangle, \langle f, s+1\rangle) : f \in \mathscr{F}, s \in \mathscr{S}_f\}\\
&\quad \cup \{(\langle f, s+1\rangle, \langle f,s\rangle) : f \in \mathscr{F}, s \in \mathscr{S}_f\}\\
E^y_A &= \{(\langle f, s+1\rangle, \langle g,s\rangle) : s \in \mathscr{S}, (f,g) \in \mathscr{F}_s \times \mathscr{F}_s, f \neq g\}\\
E^z_A &= \{(\langle f,s\rangle, \langle g, s+1\rangle) : s \in \mathscr{S}, (f,g) \in \mathscr{F}_s \times \mathscr{F}_s, f \neq g\}.
\end{aligned}$$

For each arc $e \in E$, its length l_e is the constant term in the corresponding time precedence constraints. So, for $e = (o, \langle f, d_f\rangle) \in E_R$ we have $l_e = \Gamma_f$; for $e = (\langle f, d_f\rangle, o) \in E_R$ we have $l_e = -\Gamma_f$; for $e = \{(\langle f,s\rangle, \langle f, s+1\rangle) \in E_T$ we have $l_e = \Lambda_s$; for $e = \{(\langle f, s+1\rangle, \langle f,s\rangle) \in E_T$ we have $l_e = -\Lambda_s$; for $e \in E^y_A$ with have $l_e = \Delta$; and finally, for $e \in E^z_A$ we have $l_e = -\Delta$. If e is an alternative arc, we let $\mathrm{Var}(e)$ be the variable associated with e. Since every alternative arc is associated with a selection variable in the binary vector $(\mathbf{y},\mathbf{z})$, we may interpret $(\mathbf{y},\mathbf{z})$ as the incidence vector of a subset $E_A(\mathbf{y},\mathbf{z}) \subseteq E_A$. We use $G(\mathbf{y},\mathbf{z}) = (V, E(\mathbf{y},\mathbf{z}))$ to denote the subgraph of G induced by the set of arcs $E(\mathbf{y},\mathbf{z}) = E_R \cup E_T \cup E_A(\mathbf{y},\mathbf{z})$, that is, the graph we obtain from G by removing all alternative arcs not selected in $(\mathbf{y},\mathbf{z})$.

We extend the definition of Var to a set of arcs $E = \{e_1, \ldots, e_n\}$ so that $\mathrm{Var}(E) = \{\mathrm{Var}(e_i) : e_i \in E\}$.

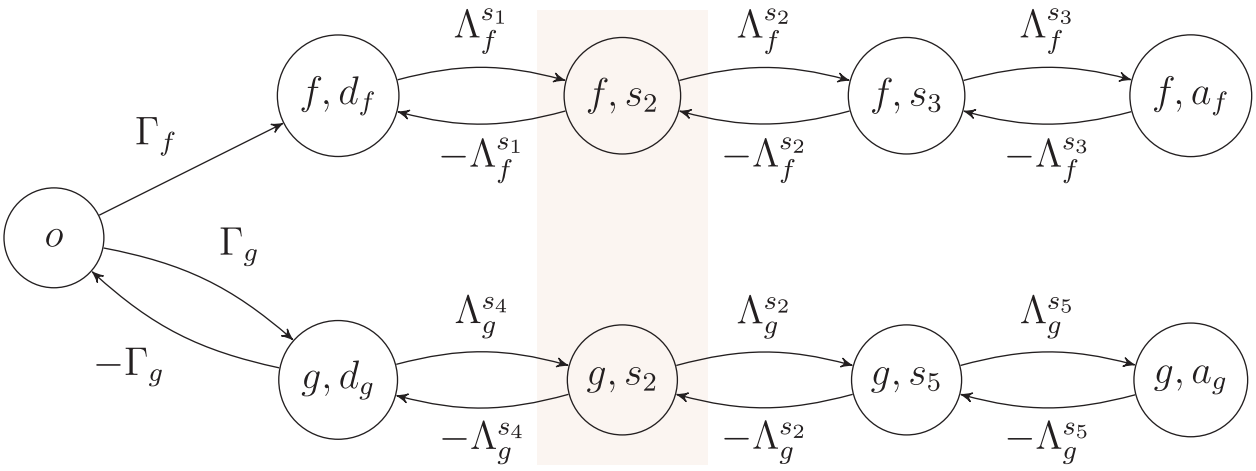

Fig. 4. Route node graph for two flights f and g. g has a fixed departure time $t_o + \Gamma_g$, while f has an earliest departure time $t_o + \Gamma_f$. Both flights fly through sector s_2. The arcs represent time precedence constraints.

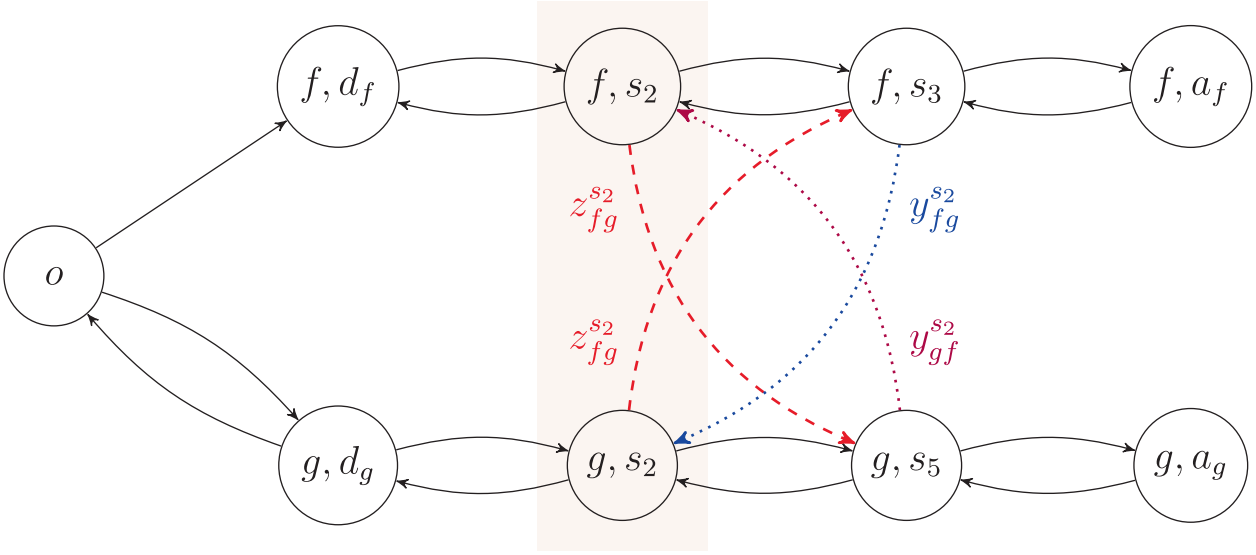

Fig. 5. Since both flights f and g fly through s_2, they must either meet there, or one must precede the other. The dashed and dotted *alternative arcs* between nodes of different flights represent each of these options. All the arc lengths have been omitted, and the alternative arcs have instead been labelled with their associated binary variables.

4.2. Positive cycles and longest paths

By construction, the route node graph contains a directed path from the origin o to any other node of the graph. Indeed, there is a directed edge from o to the node $\langle f, d_f \rangle$ associated with the first sector in the route of flight $f \in \mathscr{F}$, and a directed path from $\langle f, d_f \rangle$ to every node $\langle f, s_f \rangle$ associated with any sector $s_f \in \mathscr{S}_f$ on the route of f. Note that this path does not make use of alternative edges, and so for any selection vector $(\mathbf{y}, \mathbf{z})$, the graph $G(\mathbf{y}, \mathbf{z}) = (V, E(\mathbf{y}, \mathbf{z}))$ also contains a path P_u from o to any other node $u \in V$. By construction, even if G contains negative length arcs, the length $l(P_u)$ of this path is always non-negative.

It is well known that, if $G(\mathbf{y}, \mathbf{z})$ does not contain a strictly positive directed cycle (corresponding to an infeasible subset of time precedence constraints), then it contains a maximum length path from o to any other node u. For all $u \in V$, let us denote by $L^*(\mathbf{y}, \mathbf{z}, u)$ such length. The following Lemma 4 follows from well-known results (see, for instance, Bertsimas and Tsitsiklis, 1997).

Lemma 4. *Let $(\bar{y}, \bar{z})$ be a selection vector. There exists a feasible solution to the set of inequalities $A(\bar{y}, \bar{z})$ if and only if the graph $G(\bar{y}, \bar{z})$ does not contain strictly positive length directed cycles. Then, a feasible solution is given by $t_u^* = L^*(\bar{y}, \bar{z}, u)$, for $u \in V$. Finally, if the cost function $w(\mathbf{t})$ is non-decreasing, then $\mathbf{t}^*$ is a feasible solution which minimizes $w(\mathbf{t})$.*

We let $\mathscr{C}^+$ be the set of all strictly positive directed cycles of the route node graph G and let $C \in \mathscr{C}^+$ be one such cycle. We denote by $\mathrm{Alt}(C) = C \cap E_a$ the set of alternative arcs in C. We divide the alternative arcs into precedence arcs $\mathrm{Alt}_{\mathbf{y}}(C) = C \cap E_a^y$, and meeting arcs $\mathrm{Alt}_{\mathbf{z}}(C) = C \cap E_a^z$. It follows from Lemma 4 that, for any feasible solution $(\mathbf{y}, \mathbf{z}, \mathbf{t})$ to the hotspot problem, the graph $G(\mathbf{y}, \mathbf{z})$ does not contain a strictly positive directed cycle. So, for any $C \in \mathscr{C}^+$, at least one alternative arc in C must be excluded in any feasible solution, and the selection vector $(\mathbf{y}, \mathbf{z})$ satisfies the following set of (no good) constraints:

$$\sum_{e \in \mathrm{Alt}_{\mathbf{y}}(C)} y_e + \sum_{e \in \mathrm{Alt}_{\mathbf{z}}(C)} z_e \leqslant |\mathrm{Alt}(C)| - 1, \qquad C \in \mathscr{C}^+. \tag{10}$$

4.3. Path&Cycle MILP formulation

To simplify the following discussion, we assume now that the objective function amounts to minimizing the sum of the delays of all flights at their destinations; the extension to any function non-decreasing with time is straightforward. With this assumption, it follows by Lemma 4 that the Hotspot problem can be modelled as follows. We let $\mathscr{A} = \{\langle f, a_f\rangle : f \in \mathscr{F}\}$ be the set of arrival route nodes, and get the mathematical model

$$
\begin{aligned}
&\min \quad \sum_{u\in\mathscr{A}} L^*(\mathbf{y},\mathbf{z},u)\\
&s.t.\\
&(i) \quad y^s_{fg} + y^s_{gf} + z^s_{fg} = 1 && s\in\mathscr{S}, \{f,g\}\subseteq\mathscr{F}_s\\
&(ii) \quad \sum_{e\in \mathrm{Alt}_{\mathbf{y}}(C)} y_e + \sum_{e\in \mathrm{Alt}_{\mathbf{z}}(C)} z_e \leqslant |\mathrm{Alt}(C)| - 1 && C\in\mathscr{C}^+\\
&(iii) \quad \sum_{\{f,g\}\subseteq F} z^s_{fg} \leqslant \binom{|F|}{2} - 1 && s\in\mathscr{S}, F\subseteq\mathscr{F}_s, |F| = c_s + 1\\
&\qquad y^s_{fg}, y^s_{gf}, z^s_{fg} \in \{0,1\} && s\in\mathscr{S}, \{f,g\}\subseteq\mathscr{F}_s.
\end{aligned}
\tag{11}
$$

Note that this is a non-compact formulation since the size of $\mathscr{C}^+$ in (ii) is potentially exponential in G. To turn (11) into a MILP, we need to deal with the objective function.

To this end, for an arrival node $u\in\mathscr{A}$, we introduce a non-negative, continuous variable $\eta_u \in \mathrm{IR}_+$ such that, for any $(\mathbf{y},\mathbf{z})$, we have $\eta_u \geqslant L^*(\mathbf{y},\mathbf{z},u)$. Equivalently, we want $\eta_u \geqslant l(P)$ for any path P from o to u in $G(\mathbf{y},\mathbf{z})$.

Now, let $\mathscr{P}_u$ be the set of all *simple* paths from o to u in $G=(V,E)$, let $P\in\mathscr{P}_u$, let $l(P)$ be its length, and let $\mathrm{Alt}(P)$ be the set of alternative arcs of P. If all such arcs are selected in a solution $(\mathbf{y},\mathbf{z})$ then P belongs to $G(\mathbf{y},\mathbf{z})$ and $\eta_u \geqslant l(P)$. This can be expressed by the following linear expression:

$$
\eta_u \geqslant l(P)\left(\sum_{e\in \mathrm{Alt}_{\mathbf{y}}(P)} y_e + \sum_{e\in \mathrm{Alt}_{\mathbf{z}}(P)} z_e - |\mathrm{Alt}(P)| + 1\right) \tag{12}
$$

where Alt is defined in the same way as for cycles. Indeed, if all selection variables in (12) are 1, then the r.h.s reduces to $l(P)$, otherwise the constraint is redundant.

Combining (11) and (12) we get the following MILP formulation

$$
\begin{aligned}
&\min \quad \sum_{u\in A}\eta_u\\
&s.t.\\
&(i) \quad y^s_{fg} + y^s_{gf} + z^s_{fg} = 1 && s\in\mathscr{S}, \{f,g\}\subseteq\mathscr{F}_s\\
&(ii) \quad \sum_{e\in \mathrm{Alt}_{\mathbf{y}}(C)} y_e + \sum_{e\in \mathrm{Alt}_{\mathbf{z}}(C)} z_e \leqslant |\mathrm{Alt}(C)| - 1 && C\in\mathscr{C}^+\\
&(iii) \quad \sum_{\{f,g\}\subseteq F} z^s_{fg} \leqslant \binom{|F|}{2} - 1 && s\in\mathscr{S}, F\subseteq\mathscr{F}_s, |F| = c_s + 1\\
&(iv) \quad \eta_u \geqslant l(P)\left(\sum_{e\in \mathrm{Alt}_{\mathbf{y}}(P)} y_e + \sum_{e\in \mathrm{Alt}_{\mathbf{z}}(P)} z_e - |\mathrm{Alt}(P)| + 1\right) && u\in A, P\in\mathscr{P}_u\\
&\qquad y^s_{fg}, y^s_{gf}, z^s_{fg} \in \{0,1\} && s\in\mathscr{S}, \{f,g\}\subseteq\mathscr{F}_s\\
&\qquad \eta_u \geqslant 0 && u\in A.
\end{aligned}
\tag{13}
$$

Now, since $\eta_u \geqslant L^*(\mathbf{y},\mathbf{z},u)$ for $u\in A$, the optimal value of (13) is an upper bound on the optimal value of (11). Actually, one can show that the two formulations are equivalent, (see Lamorgese and Mannino, 2019), and the following result holds

Lemma 5. *Let $\eta^*,\mathbf{y}^*,\mathbf{z}^*$ be an optimal solution to (13) and let $t^*_u = L^*(\mathbf{y}^*,\mathbf{z}^*,u)$ for $u\in V$. Then $\mathbf{t}^*$ is an optimal solution for the Hotspot Problem, and $w(\mathbf{t}^*) = \sum_{u\in A}\eta^*_u$.*

In the next section, we show how to adapt this basic formulation to fixed windows and entry counts, respectively.

5. Model extensions

The mathematical program so far introduced is based on the sliding windows model. We now show how to extend it to fixed windows.

5.1. Modelling fixed windows

In the fixed-window hotspot problem, for each sector $s\in\mathscr{S}$ the planning time horizon is subdivided into windows $\mathscr{W}_s$. For each $w\in\mathscr{W}_s$ we let Δ_w be the size of window w (in the current air traffic control practice, Δ_w is 60 min or 25 min) and T_w be the start time. The capacity constraint now requires that, in any $w\in\mathscr{W}_s$, the number of flights is bounded by c_w. We model this by introducing new selection variables to represent the fact that a flight traverses a sector before, after, or during a given time window. We introduce variables similar to (5) and (6), namely

$$
\begin{aligned}
y^s_{fw} &= \begin{cases} 1, & \text{if } f \text{ flies through } s \text{ before } w\\ 0, & \text{otherwise}\end{cases}\\
y^s_{wf} &= \begin{cases} 1, & \text{if } f \text{ flies through } s \text{ after } w\\ 0, & \text{otherwise}\end{cases}\\
z^s_{fw} &= \begin{cases} 1, & \text{if } f \text{ flies through } s \text{ during } w\\ 0, & \text{otherwise}\end{cases}
\end{aligned}
\tag{14}
$$

and associated terms in the corresponding disjunctive constraint

$$
\begin{aligned}
&(i) \quad t^{s+1}_f \leqslant T_w - \epsilon && \text{if} \quad y^s_{fw} = 1\\
\text{or} \quad &(ii) \quad t^s_f \geqslant T_w + \Delta_w + \epsilon && \text{if} \quad y^s_{wf} = 1\\
\text{or} \quad &(iii) \quad t^{s+1}_f \geqslant T_w \wedge t^s_f \leqslant T_w + \Delta_w && \text{if} \quad z^s_{fw} = 1
\end{aligned}
\tag{15}
$$

where $\epsilon > 0$ is a suitably small constant. Note that each term in the above disjunction is a standard time precedence constraint and can be represented by an arc with suitable length in the route node graph. Fig. 6 (corresponding to Fig. 5 for the sliding window capacity constraint) shows how we represent (15) in the route node graph.

The capacity constraint for the fixed windows can now be written as follows:

$$
\sum_{f\in\mathscr{F}_s} z^s_{fw} \leqslant c_w, \quad s\in\mathscr{S},\ w\in\mathscr{W}_s. \tag{16}
$$

Note that, for a given sector s, we may have fixed windows of different sizes and overlapping windows. Also, fixed window and sliding window constraints can easily be used together in a combined formulation.

5.2. Modelling entry counts

It is very easy to amend our model to count entry rather than occupancy. Without going into details, this is obtained by simply replacing, in the disjunctive constraints (4), (8) and (15), variables t^{s+1}_f and t^{s+1}_g with t^s_f and t^s_g, respectively, leaving sign and constants unchanged. Accordingly, the alternative arcs in the node-route graph will be "re-directed", so that all tails and heads will be either $\langle f,s\rangle$ or $\langle g,s\rangle$, but the lengths will stay unchanged.

The result of this is that we replace the occupancy time interval of each flight with the single moment in time when the flight enters the sector. That is, from the point of view of the algorithm, the flight exists only at the moment it enters the sector. It is therefore irrelevant when the flight leaves the sector, or, equivalently, when it enters the next sector.

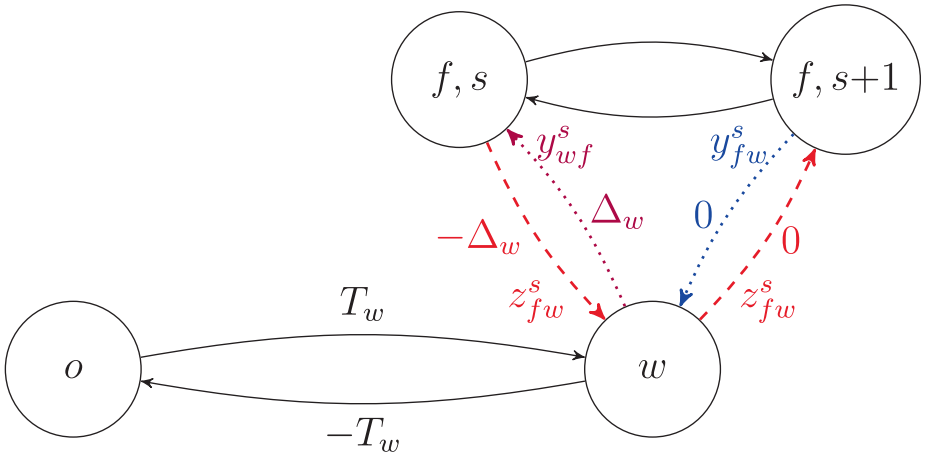

Fig. 6. Route node and arcs for a fixed window. The solid arcs between o and w represent the starting time of the window. The dashed and dotted arcs to and from w are disjunctive arcs. The disjunctive arcs are labelled with both their weights and their associated binary variables.

5.3. Modelling layered workload constraints

We are now ready to state our general MILP. The alternative arcs of the route node graph $G = (V, E)$ will be associated with each of the terms in the disjunctions (8) and (15). Then $\mathscr{C}^+$ will be the set of strictly positive directed cycles of G, whereas, for $u \in V, \mathscr{P}_u$ is the set of directed simple paths from o to u in G.

$$
\begin{aligned}
&\min \quad \sum_{u \in A} \eta_u \\
&s.t. \\
&(ia) \quad y^s_{fg} + y^s_{gf} + z^s_{fg} = 1 && s \in \mathscr{S}, \{f,g\} \subseteq \mathscr{F}_s \\
&(ib) \quad y^s_{fw} + y^s_{wf} + z^s_{fw} = 1 && s \in \mathscr{S}, w \in \mathscr{W}_s, f \in \mathscr{F}_s \\
&(iia) \quad \sum_{\{f,g\} \subseteq F} z^s_{fg} \leqslant \binom{|F|}{2} - 1 && s \in \mathscr{S}, F \subseteq \mathscr{F}_s, |F| = c_s + 1 \\
&(iib) \quad \textstyle\sum_{f \in F_s} z^s_{fw} \leqslant c_w && s \in \mathscr{S}, w \in \mathscr{W}_s \qquad \text{(HP)} \\
&(iii) \quad \textstyle\sum_{e \in \mathrm{Alt}_{\mathbf{y}}(C)} y_e + \sum_{e \in \mathrm{Alt}_{\mathbf{z}}(C)} z_e \leqslant |\mathrm{Alt}(C)| - 1 && C \in \mathscr{C}^+ \\
&(iv) \quad \eta_u \geqslant l(P) \left(\sum_{e \in \mathrm{Alt}_{\mathbf{y}}(P)} y_e + \sum_{e \in \mathrm{Alt}_{\mathbf{z}}(P)} z_e - |\mathrm{Alt}(P)| + 1 \right) && u \in A, P \in \mathscr{P}_u \\
&\quad y^s_{fg}, y^s_{gf}, z^s_{fg} \in \{0,1\} && s \in \mathscr{S}, \{f,g\} \subseteq \mathscr{F}_s \\
&\quad \eta_u \geqslant 0 && u \in A.
\end{aligned}
$$

In order to model multiple simultaneous fixed windows, we simply add more windows to the sets $\mathscr{W}_s$. We can also model multiple simultaneous sliding windows, but then we must add indexed copies of the selection variables (5) and (6) with corresponding copies of (ia) and (iia).*Summary of notation* The schedule related variables η_u are introduced in Section 4.3, while the decision variables $y^s_{fg}, z^s_{fg}, y_{fw}, y_{wf}$, and z_{fw} are introduced in Eqs. (5), (6) and (14). $\mathscr{S}$ is the set of all sectors, $\mathscr{F}_s$ is the set of all flights using sector s, and $\mathscr{W}_s$ is the set of all fixed windows of sector s. c_s is the sliding window capacity of sector s, and c_w is the capacity of the fixed window w. The sets $\mathscr{C}^+$ (of strictly positive cycles) and $\mathscr{P}_u$ (of longest paths) are introduced in Section 4.2, along with the notation $\mathrm{Alt}, \mathrm{Alt}_{\mathbf{y}}$, and $\mathrm{Alt}_{\mathbf{z}}$ (alternative arcs in a set of arcs).

6. Solution algorithm and implementation

In principle, HP could be solved by simply feeding it to an off-the-shelf MILP solver. However, we would quickly find out that even for very small instances both the number of variables and constraints would grow prohibitively large.

In order to deal with the size of the model, we apply delayed variable and constraint generation. We present the details of our custom row and column generation algorithm (Section 6.1) and and show how we separate violated constraints (Section 6.2). Finally, we show how to significantly reduce the size of the route node graph in order to speed up the constraint generation process (Section 6.3).

6.1. Delayed variable and constraint generation

Following the typical framework of a row and column generation algorithm, we build a sequence $\mathrm{HP}^0, \mathrm{HP}^1, \ldots$ of subproblems of HP, where each MILP is obtained from the previous one by adding some constraints and/or some variables. The initial subProblem HP^0 is obtained from HP by removing all variables but η and all the constraints but the non-negativity constraints on η.

$$
\begin{aligned}
&\min \quad \textstyle\sum_{u \in A} \eta_u \\
&s.t. \quad \eta_u \geqslant 0 \quad u \in A.
\end{aligned}
\qquad \mathrm{HP}^0
$$

At iteration $k = 0, 1, \ldots$ we solve Problem HP^k to optimality. Since HP^k is a relaxation of HP, if HP^k is infeasible, so is HP, and we are done.

Otherwise, let $(\eta^k, \mathbf{y}^k, \mathbf{z}^k)$ be the optimal solution to HP^k at iteration k and let $G(\mathbf{y}^k, \mathbf{z}^k)$ be the corresponding route node graph as described throughout Section 4. Then, we exploit $G(\mathbf{y}^k, \mathbf{z}^k)$ to look

for violated inequalities (the details are described in Section 6.2) by performing the following steps:

1. We start by searching for strictly positive directed cycles in the graph $G(\mathbf{y}^k, \mathbf{z}^k)$, which is equivalent to searching for violated cycle inequalities. If any such cycle is found, we add the corresponding inequality to HP^k and iterate.
2. Otherwise, $G(\mathbf{y}^k, \mathbf{z}^k)$ does not contain strictly positive directed cycles and we can construct a feasible schedule $\mathbf{t}^k$ by Lemma 4, which can then be used to look for violated path inequalities. If any is found, we add it to HP^k and iterate.
3. Finally, if no cycle and path inequalities are violated, we look for capacity constraints violations. If no capacity constraints are violated, then $(\eta^k, \mathbf{y}^k, \mathbf{z}^k)$ is feasible and thus optimal for the overall Problem HP, and $\mathbf{t}^k$ is the optimal schedule by Lemma 5. Otherwise, we add the violated capacity constraint and iterate.

Note that during step 3 it might happen that HP^k does not yet contain all the variables z that are needed for separating a particular violated capacity constraint. For example, at the very first iteration, HP^0 does not contain any variable y, z, which means that no positive cycle can be generated, and the length of a path associated to a certain airplane will be simply equal to the corresponding minimum travel time. This is usually called free-running and it represents the situation in which all airplanes travel as if there were no other airplanes (see Fig. 4). The schedule $\mathbf{t}^0$ that arises from this situation might violate some of the capacity constraints, which means that some decisions must be made to prevent it. As we have seen in Section 4, these decisions are represented by the variables y, z. In general, if one of the violated capacity inequalities contains a certain variable z^s_{fg} and that variable is not present in HP^k, then we need to add to HP^k the set of variables $y^s_{fg}, y^s_{gf}, z^s_{fg}$ that is associated with z^s_{fg} together with its corresponding constraint $y^s_{fg} + y^s_{gf} + z^s_{fg} = 1$. In other words, we need to give the model the ability to prevent a certain capacity violation by adding the appropriate decision variables and constraints. However, the newly added decision variables, which are associated to newly added arcs in the route node graph (see Fig. 5), will now contribute to possibly generating more violated cycle and path inequalities.

6.2. Separating violated inequalities

The effectiveness of the solution algorithm described in the previous section is clearly conditional to the effectiveness of the procedure for separating violated inequalities. Here we describe the details of such a procedure for each class of inequality (capacity HP.*ii*, cycle HP.*iii*, and path HP.*iv*), and we show that it can be performed efficiently.

It is important to recall that the different classes of inequalities are separated sequentially. Given a vector $(\bar\eta, \bar y, \bar z)$, with $\bar y, \bar z$ 0,1-vectors, we generate first all the violated cycle inequalities. Therefore, when we separate new classes of inequalities for the current point $(\bar\eta, \bar y, \bar z)$, the associated graph $G(\bar y, \bar z)$ contains no strictly positive directed cycles. As stated in Lemma 4, this is indeed a necessary condition for generating a feasible schedule $\bar t$. Also, we do not look for capacity violations until we have found a feasible schedule $\bar t$ that is optimal for the current subproblem, or in other words until there are no more path inequalities violated. Finding an optimal schedule is not a necessary condition for the separation of capacity inequalities, but we still separate capacity inequalities last, since this separation may require the introduction of binary variables.

- Cycle inequalities. This problem is reduced to that of finding a strictly positive directed cycle in graph $G(\bar y, \bar z)$. In turn, this can be performed in $O(|V| \cdot |E|)$ time by applying a specialized label correcting algorithm (see Ahuja et al., 1993). If the algorithm returns a strictly positive directed cycle C of $G(\bar y, \bar z)$, then we add the corresponding cycle inequality to the current MILP.
- Path inequalities. Let $\mathscr{P}_u(\bar y, \bar z) \subseteq \mathscr{P}_u$ be the set of *ou*-paths in $G(\bar y, \bar z)$. Recall that when we separate path inequalities, $G(\bar y, \bar z)$ does not contain strictly positive directed cycles. Assume that, for some $u \in A$, there exists a path inequality associated with $\bar P \in \mathscr{P}_u$, which is violated by $(\bar\eta, \bar y, \bar z)$, that is

$$\bar\eta_u < l(\bar P)\left(\sum_{e \in \text{Alt}_{\mathbf{y}}(\bar P)} \bar y_e + \sum_{e \in \text{Alt}_{\mathbf{z}}(\bar P)} \bar z_e - |\text{Alt}(\bar P)| + 1\right) \tag{17}$$

  Note that $\bar P$ is contained in $G(\bar y, \bar z)$, otherwise $\sum_{e \in \text{Alt}_{\mathbf{y}}(\bar P)} \bar y_e + \sum_{e \in \text{Alt}_{\mathbf{z}}(\bar P)} \bar z_e < |Alt(\bar P)|$, and the r.h.s. of (17) is non-positive, a contradiction since $\bar\eta_u \geqslant 0$. So, $\bar P \in P_u(\bar y, \bar z)$, and we denote by $P^*_u \in P_u(\bar y, \bar z)$ the path of maximum length in $G(\bar y, \bar z)$ (such path exists because the graph does not contain strictly positive directed cycles). By definition, $l(\bar P) \leqslant l(P^*_u)$ and thus there exists a violated path inequality if and only if

$$\bar\eta_u < l(\bar P^*_u) \tag{18}$$

  So, the separation of a path inequality violated by $(\bar\eta, \bar y, \bar z)$ amounts to computing the maximum *ou*-path in $G(\bar y, \bar z)$, for $u \in A$. As for cycles, this can be performed in $O(|V| \cdot |E|)$ time. If, for some u, we have $\bar\eta_u < l(P^*_u)$, then we add the constraints associated with P^*_u to the current MILP. Otherwise, no violated path inequalities exist.
- Capacity constraints. When separating violated capacity constraints for point $(\bar\eta, \bar y, \bar z)$, the associated graph $G(\bar y, \bar z)$ contains no strictly positive directed cycles, and we can compute a tentative schedule $\bar t$ by letting $\bar t_u = L^*(\bar y, \bar z, u)$ for $u \in V$. Note that, since $\bar t$ is constructed on $G(\bar y, \bar z)$, then $\bar t$ will respect all meetings imposed by the (arcs associated with the) meeting variables $\bar z$, and we can use $\bar t$ to check whether any of the capacity constraints are violated.

  Fixed window. Checking if $\bar t$ violates a fixed window constraint can be done by inspection, and thus in linear time in the number of flights and the number of windows.

  Sliding window. This case a bit more tricky. First, recall from (8) that if two flights f, g are in the same sliding window (with width Δ) in sector s, then we have

$$t^s_f \leqslant t^{s+1}_g + \Delta \ \wedge \ t^s_g \leqslant t^{s+1}_f + \Delta. \tag{19}$$

  The above condition is satisfied if and only if the intervals $I^s_f = [t^s_f, t^{s+1}_f + \Delta]$ and $I^s_g = [t^s_g, t^{s+1}_g + \Delta]$ overlap. Consider the largest set $F^*_s \subseteq \mathscr{F}_s$ of flights mutually satisfying condition (19) in s. If $|F^*_s| > c_s$ then we have identified a violated sliding window capacity constraint (associated with F^*_s). This reduces to the problem of finding a maximum cardinality clique in an interval graph, that is the undirected graph obtained by associating a node with each interval and an edge with every pair of overlapping intervals. Finding a maximum clique in an interval graph $H = (F, W)$ can be performed in $O(|W| \log |W|)$ (see Gupta et al., 1982).

6.3. Reduction of the route node graph

The time spent separating violated path and cycle inequalities depends on the size of the route node graph. Since we use Bellman-Ford for both separations, the worst-case complexity is $O(nm)$, where n is the number of nodes and m the number of arcs.

In this section, we will show that we can look for cycles and paths in a reduced graph with only $|F|+1$ nodes and potentially far fewer arcs than in the original route node graph. The reduced graph is quickly computed as a preprocessing step and has the potential to save significant time in each iteration of path and cycle constraint separation process. The effectiveness of the reduction has been confirmed by a set of experiments carried out during the development of our implementation. The greatest effect is expected when the number of sectors on each flight path is large.

Recall that the nodes of the route node graph correspond to the continuous schedule variables of a disjunctive formulation of the hotspot problem (see Section 2). The basic idea for our reduction is to project out all continuous-time variables except one for each flight (the arrival time $t_f^{a_f}$) and the reference time variable t_o. To this end, we use an analogue of the Fourier-Motzkin elimination scheme applied to the original disjunctive formulation. Indeed, if the formulation only contains precedence constraints and no disjunctions, then the projection operation has an immediate interpretation on the route-node graph. Namely, projecting out one variable t_v corresponds to removing the corresponding node v from the route node graph and replace each pair of arcs $(u,v),(v,w)$ (of length l_{uv}, l_{vw}, respectively), with a single arc of length $l_{uv}+l_{vw}$ (see, for instance, Mannino and Mascis, 2009). The new arc corresponds to the time precedence constraint obtained by Fourier-Motzkin combining the precedence constraints associated with (u,v) and (v,w) (see Wolsey and Nemhauser, 1999). This operation can in principle create an exponential number of constraints, and thus exponentially many arcs in the reduced graph. In case only time precedence constraints are involved in the projection, however, this does not happen. We can see this on the reduced graph, where the proliferation is prevented by the fact that when parallel arcs are created, we can remove all but the one with the largest length.

Unfortunately, when dealing with disjunctive precedence constraints, things become complicated and Fourier-Motzkin cannot be applied straightforwardly. In the following, we show one way to adapt the procedure to cope with disjunctive precedence constraints and avoid that exponentially many arcs appear. The reduced graph will however contain parallel edges. Before introducing the reduced graph $G'=(V',E')$, we need a few definitions.

Definition 6. Let f be a flight flying through sectors $s=s_1,\ldots,s_k=s'$ in sequence, where $k\geqslant 2$. Then the *flight time of f from s to s'* is

$$\delta_f(s,s')=\Lambda_f^{s_1}+\ldots+\Lambda_f^{s_{k-1}}$$

We also define $\delta_f(s',s)=-\delta_f(s,s')$.

In other words, if s precedes s' on the route of f, then $\delta_f(s,s')$ is the travel time of f from s to s', otherwise it is the negative of the travel time. In both cases, $\delta_f(s,s')$ is the length $l(P^f_{ss'})$ of the path $P^f_{ss'}$ from s to s' on the route of f in the route node graph.

Note that since $\delta_f(s,s')$ includes the flight time through sector s, but not through s', we have

$$\delta_f(s,s'')=\delta_f(s,s')+\delta_f(s',s'') \tag{20}$$

In fact, this holds regardless of the order in which f flies through s,s', and s''.

We will also use the notation $\delta(\langle f,s\rangle,\langle f,s'\rangle)=\delta_f(s,s')$ and $\delta(o,\langle f,d_f\rangle)=\Gamma_f$. We observe that $\delta(u,v)=-\delta(v,u)$ and that $\delta(u,w)=\delta(u,v)+\delta(v,w)$ if $u,v,w\in\mathscr{U}_f$ are route nodes of the same flight f, and that if G contains the edge $(\langle f,d_f\rangle,o)$, then $\delta(\langle f,d_f\rangle,o)=-\Gamma_f$.

We now go back to define the reduced graph G'. The nodes of the reduced graph are the origin o' and a node n'_f for each flight $f\in F$, corresponding to, respectively, the reference time t_o and the arrival times t_{n_f}. Note that these nodes correspond to the origin o and the arrival nodes $n_f=\langle f,a_f\rangle$ in the route node graph. The arcs of the reduced graph are of two types:

- Fixed arcs, all incident in o' (either incoming or outgoing). In particular, for every $f\in\mathscr{F}$ we have that $(o',n'_f)\in E'$. The length $\lambda(o',n'_f)=\Gamma_f+\delta_f(d_f,a_f)$ is the minimum arrival time of flight f at destination, and equals the length $l(P^f_{o,n_f})$ of the path from o to n_f in the route of f on the route node graph. When the flight f has fixed departure time Γ_f, then we also have the backward arc $(n'_f,o')\in E'$. The length $\lambda(n'_f,o')=-\Gamma_f-\delta_f(d_f,a_f)=-\lambda(o',n'_f)$ is the length of the path from n_f to o on the route of f in the route node graph.
- Alternative arcs. The alternative arcs are "copies" of the original alternative arcs, and each is associated with the same decision variable as is associated with its original version. In particular, if $e=(u,v)=(\langle f,s\rangle,\langle g,s'\rangle)\in E$ is an alternative arc of G with length l_e, then $e'=(n'_f,n'_g)\in E'$ is an alternative arc of G' and we have $\mathrm{Var}(e')=\mathrm{Var}(e)$. The length of e' is defined as

  $$\lambda_{e'}=\lambda(e')=l_e+\delta_g(s',a_g)-\delta_f(s,a_f).$$

  Note that the reduced graph may contain parallel arcs between pairs of nodes.

Definition 7 (*Reduced route node graph*). Let $G=(V,E)$ be a route node graph. The *reduced route node graph* is the graph $G'=(V',E')$, where $V'=\{o'\}\cup\{n'_f:f\in\mathscr{F}\}, E'=E'_R\cup E'_A$, and

(i) for each $f\in\mathscr{F}$, we have $e'=(o',n'_f)\in E'_R$ and $\lambda_{e'}=\Gamma_f+\delta_f(d_f,a_f)$;
(ii) for each $f\in\mathscr{F}$ with fixed departure time we have $e'=(n'_f,o')\in E'_R$, and $\lambda_{e'}=-\Gamma_f-\delta_f(d_f,a_f)$;
(iii) for each alternative arc $e=(u,v)=(\langle f,s\rangle,\langle g,s'\rangle)\in E_A$, we have $e'=(n'_f,n'_g)\in E'_A, \lambda_{e'}=l_e+\delta_g(v,a_f)-\delta_f(u,a_f)$, and $\mathrm{Var}(e')=\mathrm{Var}(e)$.

Fig. 7 shows an example of a reduced route node graph. If two flights f and g share more than one sector, then there will be multiple sets of alternative arcs between (f,a_f) and (g,a_g) in the reduced route node graph, each labelled according to the corresponding shared sector.

Lemma 8. *There is a simple path P from o to n_g in $G=(V,E)$ of length $l(P)$ if and only if there is a simple path P' from o' to n'_g in G', with $l(P)=\lambda(P')$. Moreover, the alternative arcs of P' corresponds one-to-one to the alternative arcs of P so that* $\mathrm{Var}(P')=\mathrm{Var}(P)$.

Proof. *If.* Let $e_1=(u_1,v_1),e_2=(u_2,v_2),\ldots,e_{q-1}=(u_{q-1},v_{q-1})$ be the sequence of alternative arcs encountered on P, with $f_q=g$. Path P is the concatenation of $P_1\circ e_1\circ P_2\circ\ldots\circ P_{q-1}\circ e_{q-1}\circ P_q$, where P_1 is the path from the origin $o=v_0$ to u_1 on the route of flight f_1, P_q is the path from v_q to $u_{q+1}=n_g$ on the route of flight $f_q=g$ and, for $1<i<q$, P_i is the path from route node v_{i-1} to route node u_i on the route of flight f_i (note that v_{i-1} does not necessarily precedes u_i on the route of f_i). Now, we have that $l(P)=l(P_1)+l_{e_1}+\ldots+l_{e_{q-1}}+l(P_q)$. Note that $l(P_i)$ can be negative (i.e. u_i precedes v_{i-1} on the route of f_i). By (20), for $i=1,\ldots,q-1$ we have $l(P_i)=\delta(v_{i-1},u_i)=\delta(v_{i-1},n_{f_i})-\delta(u_i,n_{f_i})$, whereas $l(P_q)=\delta(v_q,n_{f_q})$. So, we have:

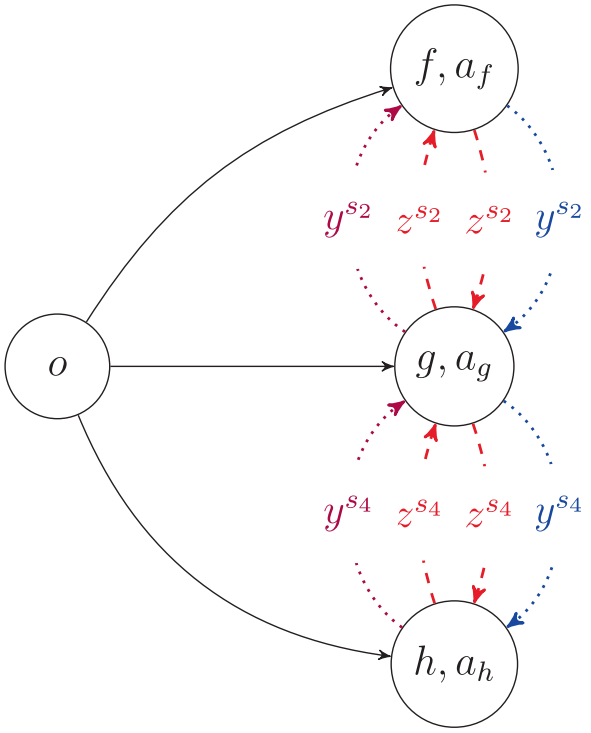

Fig. 7. Reduced route node graph with three flight f, g, and h. Flights f and g are as shown in Figure 5 and both fly through sector s_2, while flights g and h both fly through sector s_4. Flights f and h have no common sectors, so there are no alternative arcs between them. The reduced route node graph may have parallel alternative arcs, so we label the alternative arcs to distinguish them.

$$l(P) = \sum_{i=1}^{q-1}\left[l_{e_i} + \delta(v_{i-1}, n_{f_i}) - \delta(u_i, n_{f_i})\right] + \delta(v_{q-1}, n_{f_q})$$
$$= \delta(v_0, n_{f_1}) + \sum_{i=1}^{q-1} l_{e_i} + \delta(v_i, n_{f_{i+1}}) - \delta(u_i, n_{f_i}) \quad (21)$$

where the last equality is obtained by simply rearranging the terms (i.e. taking the first term of the summation out and bringing the term outside the summation in). The above steps are illustrated in Fig. 8.

Now, the path P' on G' associated with P will be $P' = (o', n'_{f_1}) \circ (n'_{f_1}, n'_{f_2}) \circ \ldots \circ (n'_{f_{q-1}}, n'_{f_q})$, as shown in Fig. 9, and we have

$$\lambda(P') = \lambda(o', n'_{f_1}) + \sum_{i=1}^{q-1} \lambda(n'_{f_i}, n'_{i+1})$$
$$= \delta(o, n_{f_1}) + \sum_{i=1}^{q-1} l_{e_i} + \delta(v_i, n_{f_{i+1}}) - \delta(u_i, n_{f_i}). \quad (22)$$

Only if. The proof goes like in the if case, only by reversing the construction (i.e. from P' to P).

Lemma 9. *There is a simple cycle C in $G = (V, E)$ of length $l(C)$ if and only if there is a simple cycle C' from in G', with $l(C) = \lambda(C')$. Moreover, the alternative arcs of C' corresponds one-to-one to the alternative arcs of C and $Var(C') = Var(C)$.*

The proof of the above lemma is very similar to the one of Lemma 8 and we omit it. The main idea is to prove two cases separately: one case for cycles that go through the origin, and one case for cycles that do not go through the origin.

7. Computational results

In order to test the performance of our model and our algorithm in the presence of layered capacity constraints, we run two different numerical experiments. In the first, we compare different capacity constraint combinations on a collection of small instances in order to highlight the differences in solutions and performance. In the second, we test a particular capacity constraint combination on a series of larger instances to test performance on a larger scale.

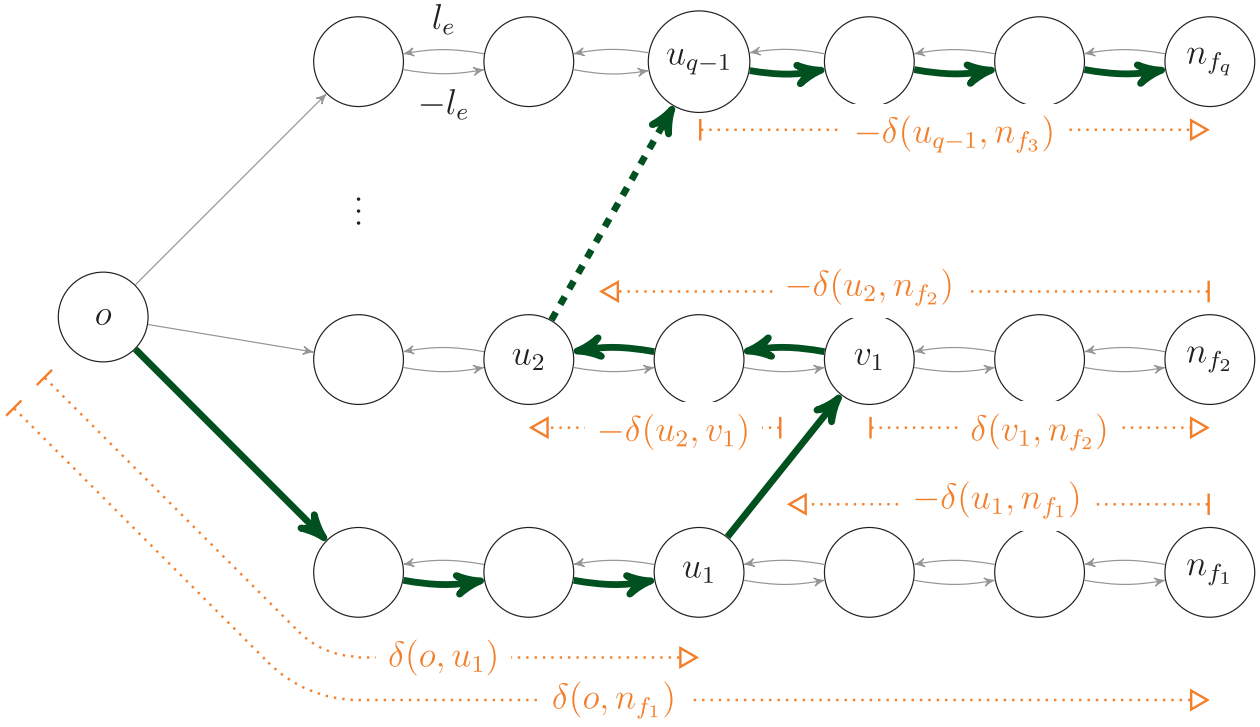

Fig. 8. A simple on_g-path P in the route node graph G. The dotted arcs represent path lengths.

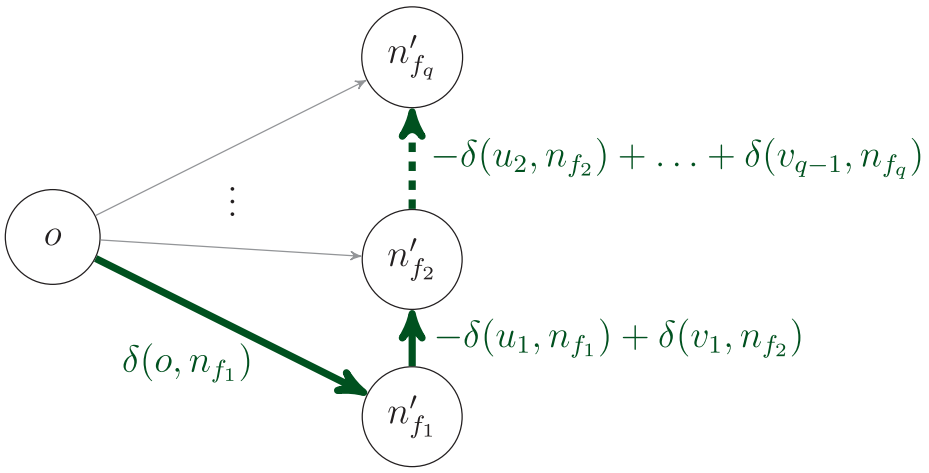

Fig. 9. The $on_{g'}$-path $P_{I'}$ in the reduced route node graph $G_{I'}$.

Table 1
The table shows 6 different capacity constraint setups. *Baseline* represents a common existing solution, and *Goal* represents our suggested ideal solution. The alternative solutions represent various simplifications of *Goal*.

Baseline	(B)	**FW** (6, 60 min); **FW** (3, 15 min)
Alternative 1	(A1)	**FW** (6, 60 min)
Alternative 2	(A2)	**SW** (6, 60 min)
Alternative 3	(A3)	**FW** (3, 15 min)
Alternative 4	(A4)	**SW** (3, 15 min)
Alternative 5	(A5)	**FW** (6, 60 min); **SW** (3, 15 min)
Goal	(G)	**SW** (6, 60 min); **SW** (3, 15 min)

Our experiments were run on a MacBook Pro (15-inch, 2016) with a 2.9 GHz Quad-Core Intel Core i7 processor and 16 GB 2133 MHz LPDDR3 memory, running macOS Catalina version 10.15, Python 3.7.3, and CPLEX 12.9 running on a single thread with default settings.

7.1. Comparison of capacity constraint combinations

For our first experiment, we simulate traffic in a small sector. The simulated traffic follows a realistic pattern, for example for a smaller airport, and highlights the differences in performance between different capacity constraints we study. With only a few notable exceptions, most Norwegian airports have no more than 6 aircraft movements per hour on average across the year, according to 2019 data from Avinor (2020).

We simulate 7 different capacity constraint setups. The setups, summarized in Table 1, are as follows

Baseline (B). Maximum 6 flights in fixed windows of 1 h, and maximum 3 flights in fixed windows of 15 min. This solution can easily be found by conventional time-indexed formulations, so we use it as the baseline for comparison.
Alternative 1 (A1). Maximum 6 flights in fixed windows of 1 h. This is a relaxation of the baseline (B).
Alternative 2 (A2). Maximum 6 flights in sliding windows of 1 h. This is similar to (A1), but the sliding window should reduce bunching. (A2) is a strengthening of (A1), in the sense that (A1) is a relaxation of (A2).
Alternative 3 (A3). Maximum 3 flights in fixed windows of 15 min. This is a relaxation of the baseline (B).
Alternative 4 (A4). Maximum 3 flights in sliding windows of 15 min. This is similar to (A3), but the sliding window should reduce bunching. (A4) is a strengthening of (A3), in the sense that (A3) is a relaxation of (A4).
Alternative 5 (A5). Maximum 6 flights in fixed windows of 1 h, and maximum 3 flights in sliding windows of 15 min. This combination of (A1) and (A4) is a strengthening of (B). The conversion of the shorter fixed windows to sliding windows is expected to reduce any bunching introduced by the longer fixed windows.
Goal (G). Maximum 6 flights in sliding windows of 1 h, and maximum 3 flights in sliding windows of 15 min. This combination of (A2) and (A4) is a further strengthening of (A5). The use of sliding windows for both constraints is expected to further reduce bunching. However, this setup is expected to be computationally heavy.

Our main interest is to compare the performances of (B), (A5), and (G). We have also included (A1), (A2), (A3), and (A4) to show the benefit of combining capacity constraints. In each of our setups using fixed windows, the fixed windows partition the timeline, with a window starting at the beginning of every hour. We use occupancy counts in each capacity constraint. The bar graphs labelled *I* in Fig. 10 show the initial traffic density in each simulated schedule.

The results of our experiments are shown in Fig. 10 and Tables 2 and 3. We have used a time cut-off of 1800 seconds[1]. The figure shows the number of flights entering the sector every 10 min. It shows that the solution for (A5) is very similar to that of (G) for the schedules where both are computed.

For all of the capacity constraint setups, the processing time starts growing very rapidly when the schedule becomes too crowded and too many conflicts must be resolved. Most cases that were solved before the cut-off took only a few seconds to solve, and in many cases were solved in the presolve, even after a few conflicts were added.

Looking at Fig. 10, we see that (A5) generates schedules with similar flight distributions to those generated using (G) in the cases where both schedules could be computed. That is, (A5) does not significantly increase bunching when compared to (G) in our experiments. (A4) also compares favourably to (A5) and (G) in most cases, but allows a larger sustained load since it only limits peak capacity. In our simulations, (A5) comes out as a strong com-

[1] Each successive MILP was solved without a time cut-off. In one case (S3, A4) an optimal solution was found after the global time cut-off. This solution has been included for comparison.

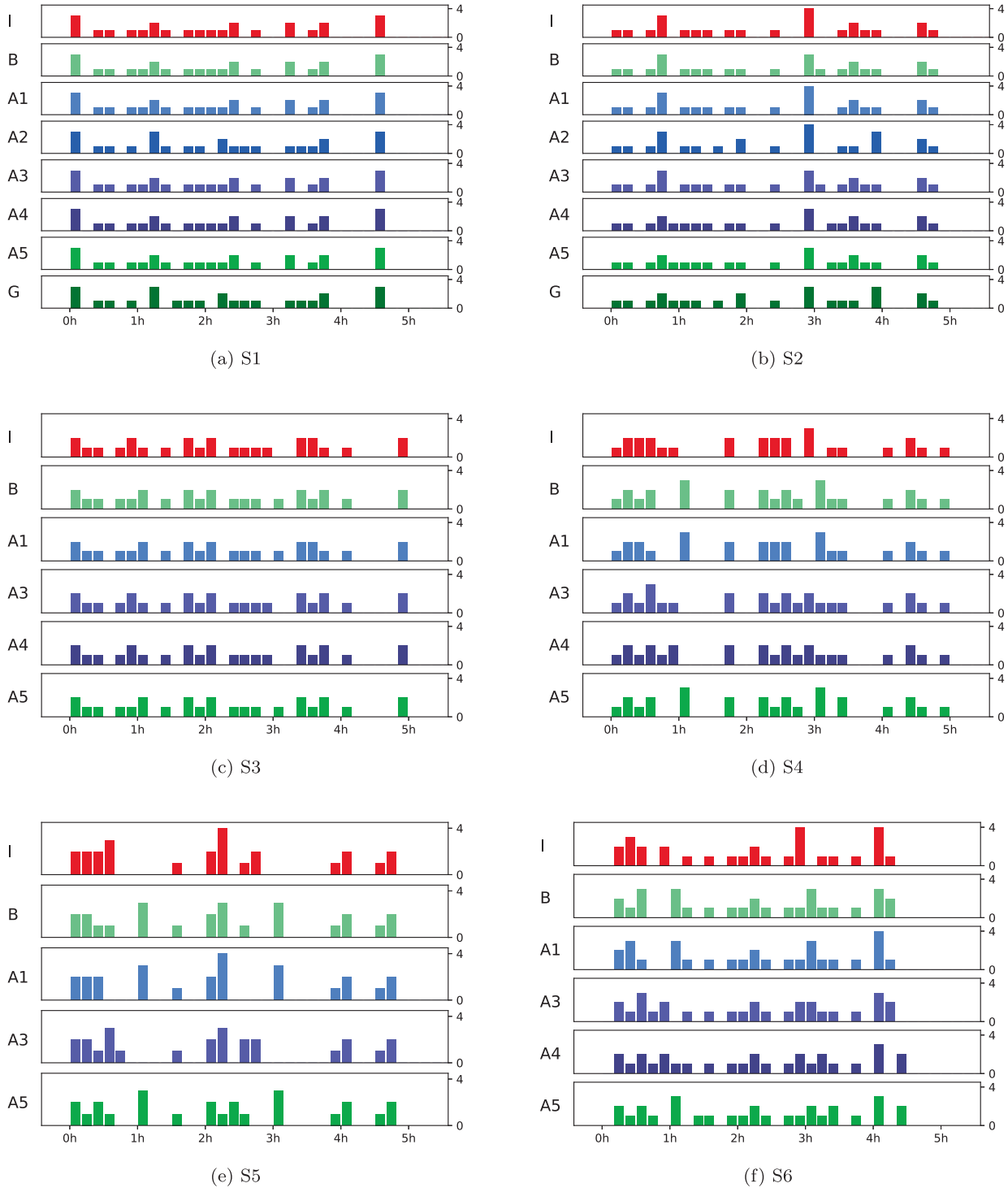

Fig. 10. An illustration of bunching. The bars show number of arrivals in 10-minute windows. The initial schedule *I* is shown in red; the baseline *B*, alternative *A5*, and the goal *G*, all combinations of two capacity constraints, are shown in shades of green; and the alternatives *A1*, *A2*, *A3*, and *A4*, all single capacity constraints, are shown in shades of blue. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

promise between lower sustained loads, less bunching, and lower processing times.

Table 3 shows the total and maximum delays in each solution. We have included maximum delays to show that our solutions do not tend to heavily delay a small number of flights, even though the maximum delay is not taken into account in our models. That is, our solutions tend to retain some measure of fairness between flights.

Tables 2 and 3 and Fig. 10 together show a fuller picture of how (A5), which is a combination of two capacity constraints, compares to a set of different single capacity constraints. The comparisons to (A1), (A2) and (G) are of particular interest. (A1) is the 1 h fixed window that is part of (A5). Since (A1) is a relaxation of (A5), it is expected to be easier to compute. However, (A5) gives significantly less bunching. (A2) is the sliding window version of (A1). (A2) is slower to compute than (A1), but does reduce bunching

Table 2
Processing time t, number of nodes processed n, and number of conflicts resolved **c**. For each schedule, we list the number of flights $|F|$. For the computations that did not complete before the processing time cut-off, we still show the number of nodes processed and conflicts resolved before the cut-off was reached. When the number of processed nodes for an instance is 0, each successive MILP for that instance was solved in the presolve. The instances are sorted according to the processing time of A5.

		B			**A1**			**A2**			**A3**			**A4**			**A5**			**G**		
	$\|F\|$	t	n	**c**	t	n	**c**	t	n	**c**	t	n	**c**	t	n	**c**	t	n	**c**	t	n	**c**
S1	25	0.0	0	**0**	0.0	0	**0**	2.1	548	**8**	0.0	0	**0**	0.0	0	**2**	0.0	0	**2**	3.4	1324	**9**
S2	24	0.0	0	**2**	0.0	0	**0**	3.1	2694	**15**	0.0	0	**2**	0.1	3	**4**	0.1	3	**4**	6.0	7083	**17**
S3	26	0.0	0	**3**	0.0	0	**2**	–	1.3e6	**63**	0.0	0	**1**	0.1	8	**6**	0.1	14	**8**	–	1.3e6	**63**
S4	27	0.1	0	**6**	0.0	0	**2**	–	1.8e6	**53**	0.0	0	**4**	21.0	4.9e4	**30**	4.0	1.4e4	**16**	–	1.8e6	**53**
S5	25	0.1	1	**6**	0.0	0	**2**	–	2.1e6	**73**	0.1	4	**6**	–	4.2e6	**59**	5.6	2.0e4	**20**	–	2.1e6	**73**
S6	29	0.1	0	**5**	0.0	0	**2**	–	1.8e6	**103**	0.0	0	**4**	2143.1	3.2e6	**54**	1159.9	2.1e6	**60**	–	1.8e6	**103**

Table 3
Total delay Σ and maximum delay m. The maximum delay was not taken into account by the model, but is included as a measure of how delays are distributed.

	B		**A1**		**A2**		**A3**		**A4**		**A5**		**G**	
	Σ	m	Σ	m	Σ	m	Σ	m	Σ	m	Σ	m	Σ	m
S1	**0**	0	**0**	0	**39**	13	**0**	0	**7**	5	**7**	5	**45**	13
S2	**3**	2	**0**	0	**64**	22	**3**	2	**20**	12	**20**	12	**81**	22
S3	**18**	8	**7**	6	–	–	**11**	8	**25**	12	**32**	12	–	–
S4	**75**	21	**62**	21	–	–	**15**	8	**65**	10	**107**	21	–	–
S5	**144**	25	**120**	25	–	–	**30**	11	–	–	**180**	25	–	–
S6	**59**	24	**44**	24	–	–	**29**	9	**114**	22	**133**	26	–	–

in some cases. (A5) is faster than (A2), but still appears to reduce bunching even more. In fact, (A5) seems to reduce bunching almost as much as the very much slower (G), where (G) is the standard we are aiming for.

7.2. Larger-scale performance test

In order to test the performance of our algorithm and explore its limits, we compute optimal schedules for 40 randomly generated instances with a higher traffic load. These instances all cover 6 h, with between 10 and 15 flights per hour. This traffic density range is representative for the average hourly traffic density of larger airports in Norway, like Stavanger or Bergen, according to data from Avinor (2020). The capacity constraint combination used is similar to A5, but with a fixed window capacity constraint allowing 12 entries per hour, and a sliding window constraint allowing 4 occupants in any 10-minute window. Table 4 shows the performance data.

As seen in Table 4, the number of flights in the smallest instance M1 is 63. With a sliding window capacity limit of 4, the total number of capacity constraints in the full formulation, HP, would be $\binom{63}{5}$, which is just over 7 million constraints in almost 2 thousand binary variables. Had the hourly limit of 12 flight also been a sliding window, the total number of capacity constraints in the full HP would be over 10^{13}. As shown in the table, the number of capacity constraints generated is only a tiny fraction of the total number of such constraints. Furthermore, since every permutation (down to rotational symmetry) of every subset of the flights corresponds to a cycle, the number of cycle constraints in the full HP would be $62! + 61! + \ldots$, which is about 10^{85}. Computing the number of path inequalities would require an in-depth analysis of the individual instance. A comparison of t_{total} and t_{last} shows that a significant amount of the total processing time is spent solving the final formulation $\text{HP}^{k_{\text{last}}}$. Since the final formulation can be very small compared with the total formulation, the overhead in processing time from delayed variable and constraint generation is comparatively small.

Figs. 11 and 12 show how the total processing time varies with other solution statistics. The size of each dot represents the number of flights in the related instance, but not in direct proportion. Differing sizes are used to show that total processing time is affected by more than just the number of flights in the instance.

Fig. 11 shows how the total processing time varies with the number of time the MIP solver is invoked, and the total number of MIP nodes processed. We use a logarithmic scale for total processing time since the time tends to increase exponentially with some measure of instance size. In Fig. 11 we use log-log axes since the quantities being compared appear to have a near-linear relationship.

Fig. 12 shows how the total processing time varies with the number of binary variables and the number of generated constraints. In this case, we use log-linear axes, since the total processing time appears to increase exponentially with the number of variables, and near exponentially in the number of constraints.

As we can see in Table 4, our model solves most instances very quickly and only breaks down when the traffic becomes very dense and the number of separated capacity conflicts exceeds a few dozen. Since the schedules we are targeting are ones initially created to be handled by controllers, we can reasonably expect that real-world instances will have a relatively small number of capacity conflicts to be resolved.

8. Conclusions and future work

We have shown that the Path&Cycle approach can be used to model the Hotspot Problem with occupancy counts and layered capacity constraints. We have also discussed how we can speed up the solution process so that we can use our model to solve instances with novel capacity constraint setups that are of interest to air-traffic control authorities.

It is clear from the model HP that sliding window capacity constraints can generate a very large number of variables, should the number of conflicts grows too large. This has led us to propose (A5), where fixed windows are used to spread the flights out, and a sliding window is used to smooth away bunching. With access to real-world instances, we could further validate our approach and test its scalability under a variety of conditions.

Table 4 shows how our model performs on a collection of randomly generated instances with a larger number of flights. We

Table 4
Computational performance on 40 randomly generates instances. In each case, the traffic is divided across 6 h, with between 10 and 15 flight per hour. The capacity constraint is similar to A5, with at most 12 entries per hour, and at most 4 occupants in any 10-minute window. The instances are sorted according to the total number of flights and total processing time. Each instance was run with a cut-off where no new calls were made to the MIP solver after 1 h. Instance M22 has a node count of 0 because it was solved in the initial presolve.

	$\|F\|$	MIP nodes	MIP solves	t_{total}	t_{last}	t_{callback}	n_{vars}	n_{capacity}	n_{path}	n_{cycle}
M1	63	4805	12	1.29	0.30	0.05	90	11	190	6
M2	64	59	6	0.22	0.05	0.03	57	5	129	5
M3	65	3340	7	1.33	0.43	0.05	117	6	252	0
M4	65	39849	23	10.30	3.98	0.24	153	22	432	55
M5	66	88	4	0.15	0.08	0.03	72	3	142	1
M6	66	138	4	0.22	0.13	0.03	90	3	159	0
M7	66	488582	139	165.82	4.05	1.18	240	138	576	147
M8	67	2	2	0.02	0.02	0.01	30	1	54	0
M9	67	1858	9	1.13	0.57	0.09	111	8	324	20
M10	67	5429	12	1.55	0.76	0.09	120	11	257	12
M11	68	14	4	0.05	0.02	0.02	45	3	100	0
M12	68	10	3	0.07	0.05	0.02	60	2	109	0
M13	68	266	4	0.25	0.16	0.03	90	3	163	0
M14	68	3663575	163	2829.23	329.72	2.71	285	162	1312	427
M15	68	6178644	45	–	–	–	210	45	1149	410
M16	68	8196053	38	–	–	–	219	38	815	82
M17	69	2	2	0.02	0.02	0.01	30	1	54	0
M18	69	19	3	0.11	0.08	0.03	60	2	108	0
M19	69	1606	9	0.83	0.51	0.08	132	8	353	4
M20	69	6118	8	2.33	1.45	0.15	219	7	401	13
M21	69	31692	29	14.57	1.12	0.89	162	28	373	222
M22	70	0	1	0.00	0.00	0.00	0	0	0	0
M23	70	145229	20	52.90	32.43	0.46	303	19	735	74
M24	70	443948	20	247.86	38.50	1.62	213	19	1132	343
M25	71	57	20	0.69	0.10	0.40	240	19	279	80
M26	71	23364	11	7.00	5.10	0.19	204	10	415	23
M27	71	35420	71	13.72	3.43	0.71	309	70	419	101
M28	71	120844	27	60.08	20.77	2.44	321	26	603	450
M29	71	221104	27	78.66	41.22	0.52	177	26	695	125
M30	71	196980	24	97.81	62.80	0.79	348	23	807	135
M31	71	4296321	125	–	–	–	468	125	1251	526
M32	72	98	9	0.45	0.08	0.19	153	8	171	48
M33	72	425	15	0.82	0.32	0.54	243	14	242	108
M34	72	143	16	1.03	0.21	0.52	252	15	297	100
M35	72	1457	17	1.33	0.26	0.29	219	16	309	50
M36	72	6649	19	3.75	1.22	0.57	300	18	422	95
M37	72	19707	13	6.64	1.96	0.26	216	12	657	41
M38	72	6212160	55	5222.85	1955.66	1.26	282	54	1158	229
M39	73	9541090	295	–	–	–	339	295	873	186
M40	73	7633255	420	–	–	–	423	420	665	272

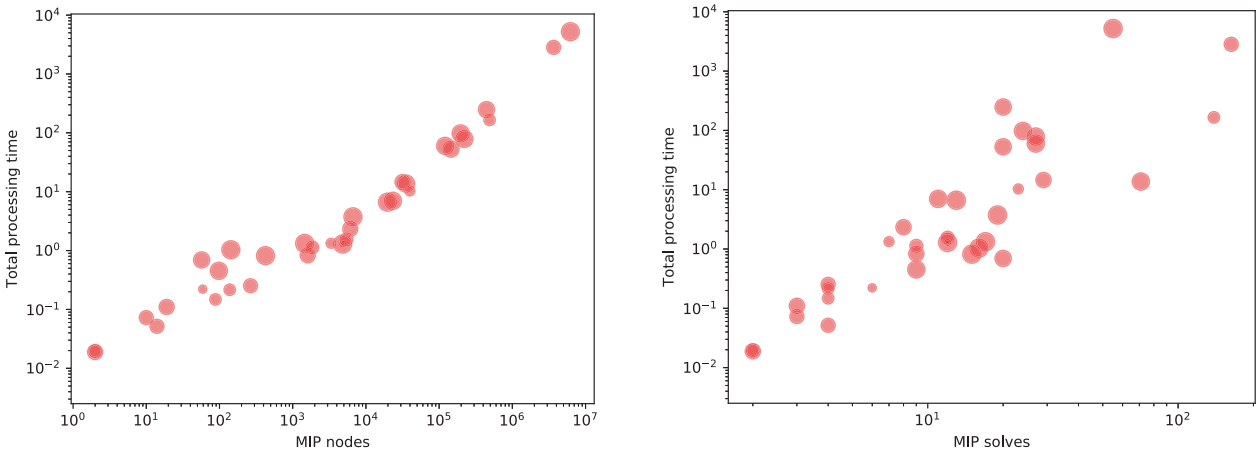

Fig. 11. Total processing time against, respectively, the total number of MIP nodes and the number of calls to the MIP solver. Since the running time appears to increase exponentially with some measure of the difficulty of the instance, the processing time is shown on a logarithmic scale. Since the processing time is approximately linear in the number of MIP nodes and solves respectively, these are also shown on a logarithmic scale. The size of each dot represents the number of flights in the instance, but not in direct proportion.

see that even for quite dense traffic patterns, a relatively small number of variables and constraints are generated before an optimal solution is found. Figs. 11 and 12 show that the total processing time of an instance depends more on the number of conflicts (variables and constraints) detected than on the number of flights in the instance. This is in line with the findings in

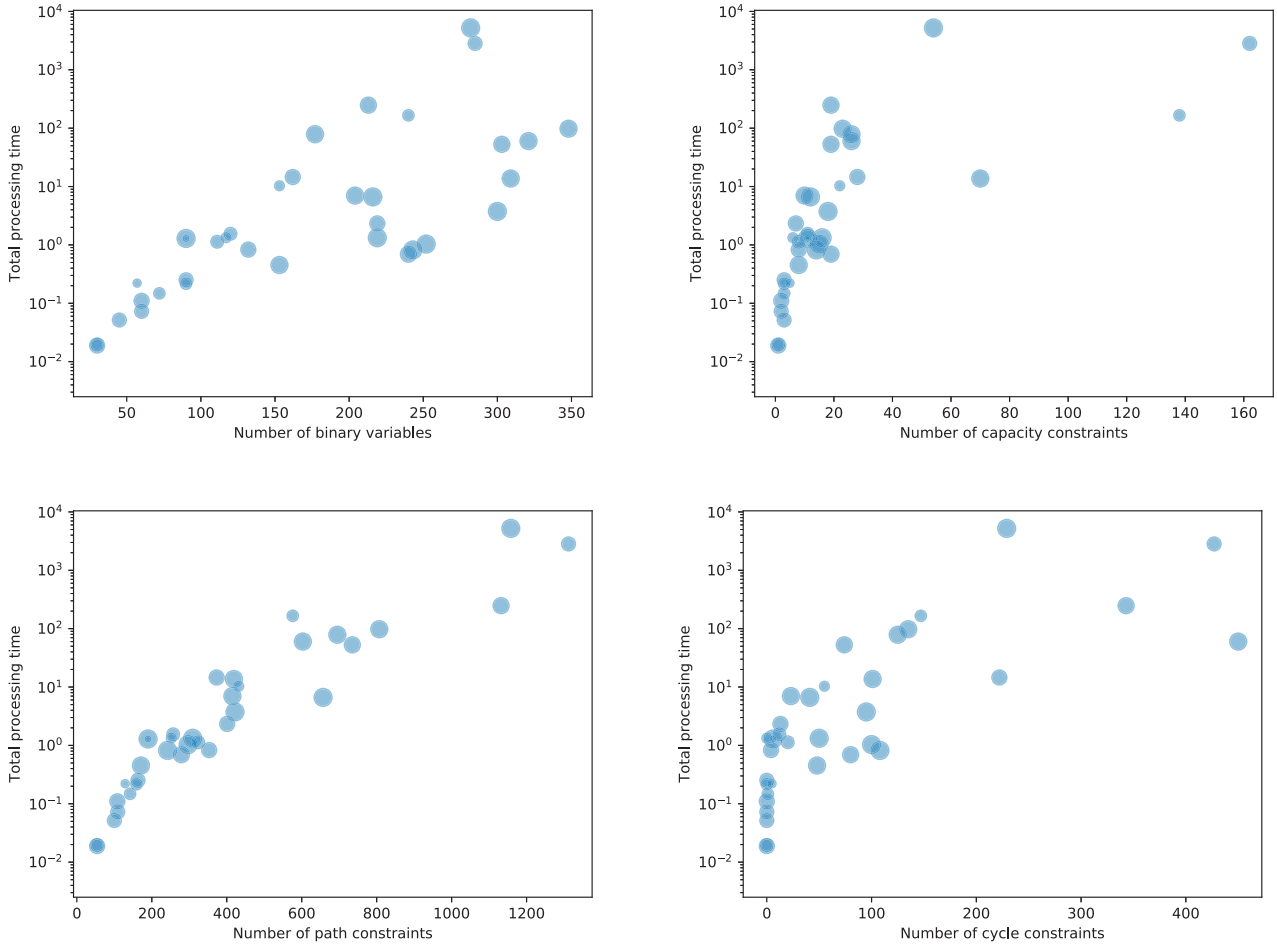

Fig. 12. Total processing time against respectively, the number of variables, capacity constraints, path constraints, and cycle constraints. Since the running time appears to increase exponentially with some measure of the difficulty of the instance, the processing time is shown on a logarithmic scale, while the number of variables and constraints are shown on a linear scale. The size of each dot represents the number of flights in the instance, but not in direct proportion.

Mannino and Sartor (2018), where the Path&Cycle approach is used to solve a version of the Hotspot problem for multiple, low-capacity sectors. This scaling property makes the Path&Cycle approach well suited to last-minute Air Traffic Flow Management, where the current schedule in nearly feasible.

In the future, we would like to build a more precise understanding of the problems surrounding bunching. In order to show bunching effects, we use Fig. 10 to illustrate the density of flights in 10-minute intervals. This gives a good visual indication of the bunching effect, and studying the figure can give us an indication about how to tweak our capacity constraints. Working with controllers and control authorities we could further our understanding of how bunching affects the workload on controllers, which in turn would allow us to further improve our analysis of different capacity constraint setups.

The Hotspot Problem as we define it falls under last-minute Air Traffic Flow Management (ATFM), which does not take into account aircraft separation and other local feasibility constraints. Instead, capacities are set low enough that local Air Traffic Control (ATC) can handle feasibility in each sector of the airspace. The separation of ATFM and ATC requires ATFM to be very conservative. By expanding our model into the domain of ATC, we could potentially allow higher capacities in the ATFM problem by enforcing local feasibility already at the ATFM level. Our model can already handle temporal aircraft separation constraints, so the major challenge would be to expand the model to route and flight level selection. In this case, we may have to resort to heuristic approaches, like the ones presented in Kim et al. (2009), Samà et al. (2017), or other tried approaches for job-shop scheduling (Allahverdi, 2016).

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.

CRediT authorship contribution statement

Carlo Mannino: Conceptualization, Methodolody, Writing - review & editing, Supervision, Funding acquisition. **Andreas Nakkerud:** Methodology, Software, Writing - original draft, Writing - review & editing, Visualization. **Giorgio Sartor:** Conceptualization, Methodolody, Writing - review & editing.

References

Ahuja, Ravindra K., Magnanti, Thomas L., Orlin, James B., 1993. Network Flows: Theory, Algorithms, and Applications. Prentice-Hall Inc., USA.
Allahverdi, Ali, 2016. A survey of scheduling problems with no-wait in process. Eur. J. Oper. Res. 255 (3), 665–686.
Allignol, Cyril, Barnier, Nicolas, Flener, Pierre, Pearson, Justin, 2012. Constraint programming for air traffic management: a survey: in memory of pascal brisset. Knowl. Eng. Rev. 27 (3), 361–392.

Avella, Pasquale, Boccia, Maurizio, Mannino, Carlo, Vasilyev, Igor, 2017. Time-indexed formulations for the runway scheduling problem. Transp. Sci. 51 (4), 1196–1209.

Avinor, 2020. Avinor Statistics Archive. https://avinor.no/en/corporate/about-us/statistics/archive, (visited September 2020)..

Bertsimas, Dimitris, Tsitsiklis, John N., 1997. Introduction to Linear Optimization. Athena Scientific Belmont, MA.

Bianco, Lucio, Dell'Olmo, Paolo, Giordani, Stefano, 2006. Scheduling models for air traffic control in terminal areas. J. Schedul. 9, 223–253.

Damhuis, E.J.H., Visser, H.G., de Jonge, Hugo.W.G., Seljée, Ron.R., 2015. Optimising air traffic flow management. Technical Report NLR-TP-2015-180, National Aerospace Laboratory NLR, 2015..

D'Ariano, Andrea, Pacciarelli, Dario, Pistelli, Marco, Pranzo, Marco, 2015. Real-time scheduling of aircraft arrivals and departures in a terminal maneuvering area. Networks 65 (3), 212–227.

de Jonge, Hugo.W.G., Seljée, Ron.R., 2011. Optimisation and Prioritisation of Flows of Air Traffic through an ATM Network. Technical Report NLR-TP-2011-567, National Aerospace Laboratory NLR..

Dubot, Thomas, Bedouet, Judicaël, Degrémont, Stéphane, 2016. Modelling, generating and evaluating sector configuration plans. In: 30th Congress of the International Council of the Aeronautical Sciences (ICAS 2016).

Geraldine Flynn, Benkouar, A., Christien, R., 2003. Pessimistic sector capacity estimation. Technical Report EEC Note No. 21/03, Eurocontrol Experimental Centre..

Guibert, S., Fitzpatrick, M., Criscuolo, P., Dohy, D., Iliev, B., Allard, E., Fabio, A., Puntero, E., Iglesias, E., Carrera, T., Valle, N., Neyns, V., Karahasanovic, A., 2019. SESAR Solution 08.01 Validation Report (VALR) for V2. Technical Report D2.1.050, SESAR Joint Undertaking..

Gupta, Udaiprakash I., Lee, Der-Tsai, Leung, Joseph Y.-T., 1982. Efficient algorithms for interval graphs and circular-arc graphs. Networks 12 (4), 459–467.

Joondong Kim, Alexander Kroeller, Mitchell, Joseph S.B., 2009. Scheduling aircraft to reduce controller workload. In: Jens Clausen, Gabriele Di Stefano (Eds.), ATMOS 2009 – 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, IT University of Copenhagen, Denmark, September 10, 2009, Volume 12 of OASICS. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

Lamorgese, Leonardo, Mannino, Carlo, 2019. A noncompact formulation for job-shop scheduling problems in traffic management. Oper. Res. 67 (6), 1586–1609.

Mannino, Carlo, Mascis, Alessandro, 2009. Optimal real-time traffic control in metro stations. Oper. Res. 57 (4), 1026–1039.

Mannino, Carlo, Nakkerud, Andreas, Sartor, Giorgio, Schittekat, Patrick, 2018. Hotspot resolution with sliding window capacity constraints using the Path&Cycle algorithm. SESAR Innov. Days.

Carlo Mannino, Giorgio Sartor, 2018. The Path&Cycle formulation for the hotspot problem in air traffic management. In: Ralf Borndörfer and Sabine Storandt, editors, 18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018), volume 65 of OpenAccess Series in Informatics (OASIcs), pp. 14:1–14:11, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik..

Mascis, Alessandro, Pacciarelli, Dario, 2002. Job-shop scheduling with blocking and no-wait constraints. Eur. J. Oper. Res. 143 (3), 498–517.

Maurice Queyranne, Schulz, Andreas S., 1994. Polyhedral approaches to machine scheduling. Technical Report 408/1994, Technische Universitat Berlin..

Tolebi Sailauov, Zhao Wei Zhong, 2016. An optimization model for large scale airspace. Int. J. Model. Optim. 6 (2), 86..

Samà, Marcella, D'Ariano, Andrea, Corman, Francesco, Pacciarelli, Dario, 2017. Metaheuristics for efficient aircraft scheduling and re-routing at busy terminal control areas. Transp. Res. C Emerg. Technol. 80, 485–511.

Nina Schefers, Miquel Angel Piera, Juan José Ramos, Jenaro Nosedal, 2017. Causal analysis of airline trajectory preferences to improve airspace capacity. Procedia Comput. Sci. 104, 321–328..

SESAR Joint Undertaking, 2020. Single European Sky ATM Research (SESAR) Joint Undertaking. https://www.sesarju.eu, (visited April 2020)..

Vaaben, Bo, Larsen, Jesper, 2015. Mitigation of airspace congestion impact on airline networks. J. Air Transp. Manage. 47, 54–65.

Wolsey, Laurence A., Nemhauser, George L., 1999. Integer and Combinatorial Optimization, vol. 55. John Wiley & Sons.

Zhao Wei Zhong, 2018. Overview of recent developments in modelling and simulations for analyses of airspace structures and traffic flows. Adv. Mech. Eng. 10 (2)..