Aircraft landing planning under uncertain conditions

Vié, Marie-Sklaerder; Zufferey, Nicolas; Leus, Roel

Journal of Scheduling
https://doi.org/10.1007/s10951-022-00730-0

Aircraft landing planning under uncertain conditions

Marie-Sklaerder Vié[1] · Nicolas Zufferey[1] · Roel Leus[2]

Accepted: 25 February 2022

Abstract
Aircraft Landing Planning is challenging because the inherently limited capacity of airport runways causes bottlenecks. This type of planning involves different stakeholders (e.g., airlines, air traffic services providers, airport authorities, and passengers) and faces various uncertainties (e.g., take-off time variability, or wind speeds). This study, conducted in collaboration with the European Organization for the Safety of Air Navigation (EUROCONTROL), proposes a mathematical formulation of the problem and a simulation framework that accounts for uncertainties. We also propose different solution methods: a descent and a tabu search, as well as a mechanism for guiding restarts, to diversify the search process. These methods provide, in our simulated environment, more effective and stable solutions than the popular first-come-first-served practice regarding three objective functions (namely, delay, fuel, and landing sequence stability), which are considered lexicographically. Indeed, the average delays and fuel costs are reduced by 50% and 10%, respectively, at the cost of a small number of landing-sequence modifications, as each flight is repositioned an average of 0.5 times. Moreover, the computations can be performed quickly, which is crucial because re-optimization needs to be done online when flight information is updated.

Keywords Aircraft landing planning · Uncertainty · Scheduling · Online lexicographic optimization · Tabu search

1 Introduction

The inherently limited capacity of airport runways causes bottlenecks. Therefore, *Aircraft Landing Planning* (ALP), i.e., making a wise decision about the landing time of each plane, along with choosing how to make planes meet these arrival times, is crucial. One way to delay an airplane and ensure it meets the desired arrival time is instructing the pilot to perform a *holding-stack pattern* (i.e., waiting for the planned landing time by making loops, in a holding area, near the airport) before landing, but this technique increases fuel consumption since this is performed at lower altitudes. Another option is to adjust the plane's cruising speed (i.e., at higher altitudes, where fuel consumption is lower), or to stretch its path (i.e., deviate slightly from its planned route to lengthen it). However, planes are subject to many uncertainties while cruising (e.g., wind speeds and air traffic), which increase if the cruise duration is extended. Furthermore, as *air traffic controllers* (ATC) have multiple tasks to handle and prioritize to ensure the safe and efficient flow of traffic, we do not want the landing sequence to change too often because of uncertainties or delays.

This study was conducted in collaboration with the *European Organization for the Safety of Air Navigation* (EUROCONTROL), which is working to achieve safe and seamless air traffic management across Europe. This organization was founded in 1960, and its main activities consist of operations and service provision, research and concept development, Europe-wide project implementation, performance improvements, coordination with key aviation players, and providing support to the future evolution and strategic orientations of aviation. These are becoming more challenging for ALP because the demand for flights increases over time, but the number of landing runways remains rather stable.

The considered planning horizon covers all the flights of a typical peak duration in the core area of the European airspace, which lasts an average of three hours. A

✉ Nicolas Zufferey
n.zufferey@unige.ch

Marie-Sklaerder Vié
marie-sklaerder.vie@unige.ch

Roel Leus
roel.leus@kuleuven.be

[1] Geneva School of Economics and Management, GSEM - University of Geneva, UniMail, Bd du Pont-d'Arve 40, 1211 Geneva 4, Switzerland

[2] Faculty of Economics and Business, KU Leuven, Leuven, Belgium

Published online: 04 April 2022

rolling planning window of 45 min is used because, according to EUROCONTROL, a larger planning window would be associated with uncertainties that are difficult to forecast and manage efficiently, whereas a smaller planning window would neither allow enough delay absorption while planes are cruising nor allow enough possible changes in the landing sequence. In such a planning window, most flights are already cruising, except so-called *pop-up* flights (i.e., 'short' flights for which take-off and landing occur within the considered planning window), which are only taken into account after their departure. Pre-planning them only improves overall performance if departure time accuracy is less than five minutes (Vanwelsenaere et al. 2017), which is too small in practice.

There are different stages in ALP. First, an initial schedule is created when a flight enters the planning horizon (i.e., when it has taken off and its planned arrival time is close enough, according to the planning strategy). Second, the landing position and corresponding arrival time are modified during the cruise and the airport approach phases. Finally, each flight is frozen for the landing phase (i.e., the last 15 min), meaning that no modification can be performed on its landing position in the landing sequence of flights, and its landing time is definitely determined as there are only negligible variations at this stage. In this paper, we consider the second stage only, that is the cruise period and the airport approach, but not the take-off or frozen landing-time intervals. The corresponding planning window (of 45 min, or less if we consider a pop-up flight) is represented in Fig. 1, and it is separated in two parts. In the first part, just after its entry in the planning window, the flight is still in the cruise period, where it can be delayed by a path detour or speed adjustment. In the second part, the flight is within the airport approach, waiting for its landing time by performing holding-stack patterns; however, if it does not have to wait, the plane begins to land immediately, and this second part is reduced to zero. The initial schedule is provided by the popular *First-Come-First-Served* (FCFS) rule, which ranks the flights with respect to the entry time in the planning window, and therefore, with respect to increasing planned arrival times. Even though it used to be the most commonly employed approach (Erzberger 1995), Caprì and Ignaccolo (2004) show that FCFS does not usually lead to optimal sequences with respect to throughput or average delay. It is, however, a relevant method because, for a single-machine job-scheduling problem, it minimizes the maximum delay (Pinedo 2016).

The optimization problem faced here consists in deciding, first, the landing sequence of the planes, then their associated landing times, and finally, how to meet these landing times by making operational decisions, such as speed adjustments, path stretching, or the plane maintaining a holding pattern in the airport area. These decisions are associated with the distribution of delays to the flights, where a flight delay is

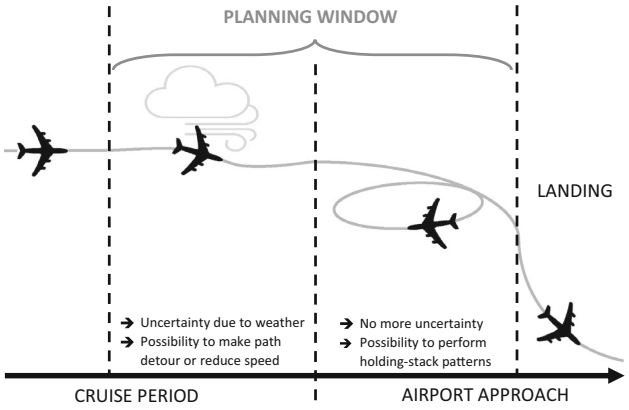


Fig. 1 Position of the planning window in the different stages of a flight

defined as the difference between (1) the currently expected landing time and (2) the initially scheduled one (i.e., by the airline company). Note that the expected landing time can be earlier than the initially scheduled one (e.g., if the flight has encountered less delay than expected, or if the uncertainties have had a positive impact during cruise), therefore leading to a negative delay. In such a case, the operational decisions are only useful for satisfying the *separation constraint*, which imposes minimum threshold times between planes, typically 90 or 120 s. They also help in reducing delays, which have a negative effect on passenger satisfaction if the arrival/departure times differ considerably from published schedules.

Three different objective functions (to be minimized) are considered and ranked lexicographically (i.e., no lower-level objective can be improved at the cost of deteriorating a higher-level objective) based on the EUROCONTROL priorities for this study: the summation of positive delays, the estimated total fuel cost, and the number of *rescheduling actions* (i.e., the number of times that a plane is moved to another position in the landing sequence). Separation and domain constraints (e.g., minimum speed and maximum path stretching) must also be satisfied.

This paper makes the following contributions. First, we propose a lexicographic ALP model that integrates the operational decisions for distributing delays while accounting for the concerns of different stakeholders. Second, we propose a simulation process that evaluates the performance of our solution methods empirically and chooses how to distribute delays and estimate actual arrival times to cope with uncertainty more effectively. Third, we propose quick solution methods that account for uncertainties encountered by flights, thanks to the simulation process. When compared to FCFS (which is a popular rule in practice), our best method leads to significant improvements in the first two objective functions (approximately 50% fewer delays and 10% lower fuel costs), while encountering a very reasonable degradation

of the third objective function (on average, each second aircraft changes its position once over the whole 3-h horizon). Finally, we present computational results for data generated by EUROCONTROL.

The paper is organized as follows. A literature review is conducted in Sect. 2, with a focus on more comprehensive existing reviews. Section 3 describes the considered problem formally. The simulation process is explained in Sect. 4, and the solution methods are proposed in Sect. 5. Finally, the results are presented in Sect. 6, followed by the conclusions in Sect. 7.

2 Literature review

A comprehensive review of ALP and ATP (*Aircraft Take-off Problem*)—considered together or separately—covering the period up to 2011 can be found in Bennell et al. (2011), and different aspects of the period up to 2018 are covered in Vié et al. (2018); Zulkifli et al. (2018). Since then, we have only found a model taking the arrival-time uncertainties into account (Liu et al. 2018), where the authors solve the problem with a sample average approximation algorithm. Within the scope of this study, the main points to highlight from these reviews are the following.

(1) Various objective functions have been considered for ALP, and they can be divided into four categories: airline objectives (e.g., maximize punctuality with respect to published landing times and minimize total passenger delays), safety and efficiency objectives (e.g., maximize the runway throughput and minimize the rescheduling workload), airport objectives (e.g., minimize gate changes due to delays), and government objectives (e.g., minimize noise and air pollution). The literature either considers a single objective or proposes a multi-objective approach. Surprisingly, we did not find any optimization approach that covers the various abovementioned stakeholders efficiently, at least as it relates to EUROCONTROL's priorities. We will show that the proposed lexicographic approach is straightforward from a practical standpoint, and the reader is referred to Gallay and Zufferey (2018) for various examples of successful lexicographic optimization models in industry, including one in transportation, and to Chern and Hsieh (2007) for a supply-chain situation where the lexicographic method provides better results than the aggregated multi-objective method (and where the first objective also consists of minimizing delays in order to satisfy the consumers at best, before minimizing the costs of the involved company).
(2) Different exact methods (e.g., dynamic programming (Balakrishnan and Chandran 2010), branch-and-bound (Artiouchine et al. 2008)) and several metaheuristics (e.g., some local-search procedures (Dear and Sherif 1991; Soomer and Franx 2008), and numerous genetic algorithms (Beasley et al. 2001; Caprì and Ignaccolo 2004; Hu and Chen 2005; Hu and Di Paolo 2009; Pinol and Beasley 2006; Yu et al. 2011), but surprisingly, no tabu search) have been developed for ALP. The exact methods are restricted to instances with up to 50 planes preparing to land, whereas some of the metaheuristics can tackle instances with up to 500 planes; however, such a large number is unnecessary in practice, as there are at least 90 s between two landings, and therefore, 500 flights can cover a period of at least 12 h.
(3) Many studies favor the consideration of arrival time slots (e.g., from 10:00 to 10:15) instead of accurate arrival times (e.g., at 10:05). For instance, Schummer and Vohra (2013) investigate how to re-assign such time slots (using ground delays or holding patterns only) when bad weather reduces visibility or limits the number of usable runways, which results in the reduction of the landing capacity. Interestingly, an extension of this work is proposed by Schummer and Abizada (2017), and shows how the model can be adapted to a set of flights with different priorities. Another example where the authors assign a landing-time window to each flight instead of an accurate point in time can be found in Heidt et al. (2016), with similar uncertainties as those considered in this paper. Note that they study the robustness of these assignments, and also tried an exact method, but it requires hours of computing time.
(4) Most of the papers consider the static case, where the arrival time in the airport area is known in advance for all flights, which is obviously not the case in practice, as aircraft encounter many uncertainties, mainly due to wind and departure-time changes. Furthermore, the robustness (i.e., sensitivity to variations in problem characteristics or data quality) of the provided solutions is never analyzed, which means that if uncertainty is considered, it could completely change the quality of these solutions. Recently, a two-stage stochastic programming approach exploiting the CPLEX solver was proposed by Khassiba et al. (2019, 2020). Taking into account multiple uncertainty scenarios, the authors first minimize the landing-sequence length (computed based on the minimum separation constraints only) from two to three hours before landing, under a pre-defined time-window constraint for each plane. Second, when the uncertainties are progressively revealed, the actual landing times are computed but without changing the landing sequence, while minimizing the total delay (positive or negative: an overly early flight is also penalized). However, the results are restricted to 10-plane instances, and the quality of the results decreases when uncertainty or the time win-

dows increase. Interestingly, Brentnall and Cheng (2009) show that deterministic algorithms are sub-optimal in a dynamic environment and that the FCFS rule is surprisingly robust and has several advantages over many existing algorithms. First, the generated flight sequence is stable and understandable by ATCs. Second, it leads to easy-to-estimate delays with respect to the airlines' published schedules. Finally, it is already in place in many airports. In our study, an accurate landing time will be determined with the use of simulation while accounting for various uncertainties.

(5) To our knowledge, recent works consider only two axes with respect to the occurrence of random events. One research stream quantifies the delays with a Monte-Carlo simulation using queuing theory (Nikoleris and Hansen 2012; Shone et al. 2018) and shows that queuing delays must be distributed between the cruise and approach phases to minimize fuel consumption (i.e., queuing delays on cruise only, or approach only, is not optimal). Another research avenue aims to add online optimization to a static method with the use of a rolling-window simulation (Bennell et al. 2017). Nevertheless, uncertainties are not accounted for, except the entry time of a flight in the planning window.

Representative recent works are in Faye (2015); Lieder et al. (2015); Sabar and Kendall (2015). However, in line with our above observations, we can highlight the following limitations of these references compared to our work. First, a deterministic problem is considered (i.e., no simulation tool is required for evaluating any solution). Second, different objective functions are not integrated in a same optimization problem (i.e., only one objective function is considered at a time). Third, no real data is considered, and the numerical experiences are based on a very small number of instances (i.e., below 15 instances for each paper). In contrast, the following features characterize our paper, which makes it unique and much closer to reality. (1) We consider uncertainties (e.g., wind effects) and we show (with numerical results) how CPLEX is not appropriate anymore when uncertainties are considered. (2) In addition to the popular penalties associated with the non-punctual flights, we consider two additional objective functions that are highly relevant for practice. (3) Our numerical experiments rely on 149 instances that are real and were performed under real computational conditions (where solutions must be provided within a few seconds).

In summary, a considerable amount of work has been done on ALP, but mainly on the static case (i.e., where release dates and flight durations are all known in advance) or dynamic case (e.g., where the take-off times are only known once the planes have taken off) but without considering uncertainties (i.e., unplanned changes impacting the arrival times, such as wind effects). Therefore, there is still a big gap to be bridged to include the uncertainties properly in a model. Furthermore, some quick and efficient metaheuristics, such as tabu search, and the use of lexicographic optimization are missing in the existing solution methods. This is somewhat surprising as tabu search is generally fast and has a great intensification ability, which is relevant when decisions have to be made quickly (Gendreau and Potvin 2019; Respen et al. 2017), and lexicographic optimization is usually in line with such decision-making environments for job scheduling (Respen et al. 2016; Solnon et al. 2008; Thevenin et al. 2018).

3 Mathematical model

Let t be the current time, updated every 30 s. Let H^t be the rolling planning window, and w its length (45 min in this study). It is discretized by time steps of $\Delta t = 30$ s. At each time unit t, among the flights that have already taken off, we only consider the flights with planned landing times up to time $t + w$ (i.e., $H^t = [t, t + w[$). We also denote as l the time period (fixed here to 15 min) that is considered as frozen for the landing phase (i.e., a flight cannot be modified within 15 min before it lands).

In this section, for any fixed time period t, a mathematical model is proposed for ALP. The resulting optimization problem is denoted as (P^t). The following elements are presented sequentially in the next subsections: the given data, the decision variables, the structural constraints, and the different objective functions. Finally, we show that the considered problem generalizes some well-known sequencing problems, and we discuss its complexity.

3.1 Given data

Let J^t be the set of flights considered in the rolling planning window $H^t = [t, t + w[$ and n be the size of J^t. The airport area corresponds to the geographic area where each flight can start its airport approach (defined in Sect. 1), as opposed to the cruise period where there is still some distance to travel. Typically, it consists of a 50-nautical-mile radius around the airport, but its shape and size can vary from one airport to another. For each flight $j \in J^t$, the following data are given:

- r_j: take-off time (or release date).
- d_j: planned (published) landing time by the involved airline company (or due date).
- v_j: cruising speed (in nm/s, i.e., nautical miles per second). This speed depends on the plane type and is the maximum allowed speed. The speed of j can be reduced down to $\alpha \cdot v_j$, with $\alpha = 92\%$.
- p_j^t: remaining distance (in nm) to be covered until the airport area, with respect to the flight coordinates at time t. It is reduced step-by-step when moving from one time

Table 1 Setup times $s_{j,j'}$ (in s) for each category of plane (SKYbrary 2017)

j'						
j	CAT-A	CAT-B	CAT-C	CAT-D	CAT-E	CAT-F
CAT-A	90	120	150	150	180	240
CAT-B	90	90	120	120	150	210
CAT-C	90	90	90	90	120	180
CAT-D	90	90	90	90	90	150
CAT-E	90	90	90	90	90	120
CAT-F	90	90	90	90	90	90

period to the next. Therefore, $t + p_j^t/v_j$ corresponds to the minimum time (in seconds) at which flight j can reach the airport area (i.e., j cannot land before this time).

- q_j^t: largest allowed path stretching (in nm) at time t. For each flight j, it is initially fixed to $300 \cdot v_j$ (i.e., the distance covered in 300 s at cruising speed), then progressively reduced when rolling the planning window and stretching the flying path. Such a small value of 300 s is used as EUROCONTROL does not want a flight to change its ATC zone because of path stretching only. Indeed, any time a flight j enters a zone of the airspace, an ATC is in charge of j.
- k_j^C, k_j^A: fuel cost while cruising and while maintaining a holding pattern in the airport area, respectively. As $k_j^C < k_j^A$, ATCs prefer to distribute the delays to the cruise phase as much as possible. These two weights have different values depending on the plane category. More precisely, $(k_j^C, k_j^A) = (2, 3)$ if j belongs to the categories CAT-D, CAT-E, or CAT-F, and $(k_j^C, k_j^A) = (6, 9)$ otherwise.
- π_j^t: planned landing position (with respect to the landing sequence) at the beginning of time period t, where new flights entering H^t are also considered and have been inserted according to the FCFS rule (the very few ties, where two flights enter the planning window at the same time, are broken at random). For instance, $\pi_2^t = 3$ means that flight 2 is planned to be the third flight to land if no other decision is made.

Finally, each pair (j, j') of flights such that j has to land before j' must have their landing times separated by $s_{j,j'}$ seconds. Such a setup time depends on the wake turbulence categories of the two involved planes. More precisely, planes are divided into six categories, labeled from CAT-A to CAT-F, and the setup times are given in Table 1 (in seconds). This table is based on the recent RECAT-EU categorization (SKYbrary 2017), which defines turbulence categories and their associated separation thresholds to increase airport capacities.

To illustrate how the computation works, consider a small theoretical example, with four flights numbered from 1 to 4, for which the detailed information is given in Table 2. As we start with time $t = 0$, only flight 1 is in the planning horizon (fixed here to 45 min, thus 2700 s). For this reason, only flight 1 has a planned landing position $\pi_1 = 1$. It is too early for flights 2 and 4 to take off (as their $r_j > 0$). Flight 3 has taken off, but its due date is currently out of the planning horizon (as $d_3 = 3600 > t + 2700$). Finally, we can see that flight 4 is a pop-up flight, as its expected flight duration is less than the planning horizon (it is $\frac{p_j^t}{v_j} = \frac{756000\text{ nm}}{420\text{ nm/s}} = 1800$ s = 30 min).

3.2 Decision variables

Two types of decision variables are defined. Vector Π gives the flight landing positions and is the main variable type on which the optimization will work. This vector gives the positions of the same flights as the ones involved in π, but after the employed optimization process, which will be presented later. The other variables allow computing feasible landing times (C_j^t) and determine how to reach them based on speed adjustments (V_j^t), path stretching (Q_j^t), and waiting times (i.e., maintaining a holding pattern above the airport area, where the duration W_j^t of the loop has to be decided). The full list of variables is as follows:

- Π_j^t: landing position of flight $j \in J^t$ at time t.
- C_j^t: expected arrival time (or completion time), considering the current situation of flight $j \in J^t$ at time t.
- V_j^t: speed (in nm/s) of flight $j \in J^t$ at time t.
- Q_j^t: path stretching (in nm) of flight $j \in J^t$ at time t. More precisely, this is a value we decide, at time t, to add to the remaining distance p_j^t of flight j, irreversibly (i.e., flight j engages on an alternative path that is longer by Q_j^t nm). Note that Q_j^t is not the portion of path stretching allocated for the next time step, but the path stretching decided for flight j at time t, which will be reached over the whole remaining cruise.
- W_j^t: waiting time (in s) assigned at time t to flight $j \in J^t$ for maintaining a holding pattern above the airport area.

Note that the variables W_j^t are continuous (i.e., not restricted to integer values). For a flight, maintaining a holding pattern can either be making loops of various sizes in the airport area or deviating from its path (including the loops) in an 'S-shaped' pattern. With these holding-pattern possibilities, all values for any waiting time can be met.

Table 2 Example of data for a four-flight situation

Flight j	1	2	3	4
Take-off time r_j (in s)	0	1200	0	600
Planned landing time d_j (in s)	2700	4200	3600	2400
Cruising speed v_j (in nm/s)	420	420	420	420
Remaining distance p_j^t (in 10^3nm)	1134	1260	1512	756
Largest path stretching q_j^t (in 10^3nm)	126	126	126	126
Planned landing position π_j	1	–	–	–

3.3 Structural constraints

The decision variables have to satisfy three types of constraints: separation, linking, and domain. Separation constraints ensure that two flights are not scheduled in the same position, which is enforced by Constraints (1), and the setup times are satisfied due to Constraints (2). The linking constraints (3) compute each expected landing time according to the different decision variables having an impact on it. It states that C_j^t must be equal to the current time t plus the planned remaining cruise time $(p_j^t + Q_j^t)/V_j^t$ (distance to travel divided by speed) and the approach time $W_j^t + l$ (waiting in the airport area plus the landing time). Finally, constraints (4–8) are the domain constraints.

$$\Pi_j^t \neq \Pi_{j'}^t \quad \forall j, j' \in J^t \tag{1}$$

$$C_{j'}^t \geq C_j^t + s_{j,j'} \quad \forall j, j' \in J^t \text{ such that } \Pi_j^t + 1 = \Pi_{j'}^t \tag{2}$$

$$C_j^t = t + \frac{p_j^t + Q_j^t}{V_j^t} + W_j^t + l \quad \forall j \in J^t \tag{3}$$

$$\Pi_j^t \in \{1, \ldots, n\} \quad \forall j \in J^t \tag{4}$$

$$C_j^t \geq 0 \quad \forall j \in J^t \tag{5}$$

$$V_j^t \in [\alpha \cdot v_j, v_j] \quad \forall j \in J^t \tag{6}$$

$$Q_j^t \in [0, q_j^t] \quad \forall j \in J^t \tag{7}$$

$$W_j^t \geq 0 \quad \forall j \in J^t \tag{8}$$

3.4 Objective functions

As imposed by EUROCONTROL, we consider three different objectives with lexicographic priorities $f_1 > f_2 > f_3$. First, f_1 (presented in Eq. (9)) is the sum of all positive delays (i.e., the total tardiness), and minimizing it aims at satisfying the passengers, which is the main priority of EUROCONTROL. Second, f_2 computes (see Eq. (10)) the total fuel cost, and reducing it satisfies different stakeholders: the airline companies (as it reduces their costs) and the governments (as it reduces the environmental impact). Note that the plane sizes are taken into account here: the larger a plane is, the more fuel it consumes, therefore minimizing f_2 after f_1 will favor delaying a small plane rather than a big one (the latter typically involving more passengers). Finally, f_3 (given in Eq. (11)) aims to maximize the stability of the flight landing positions (see Pfeiffer et al. (2007) for a good review of stability measures). Indeed, each time a flight landing position must be modified, ATCs must inform the involved pilots accordingly, and this increases their workload, whereas their priority is ensuring safety. Maximizing the stability of the flight landing positions corresponds to minimizing the number of perturbations that we operate on the landing sequence, and therefore it strongly contributes to the airport's objectives by reducing the ATCs workload regarding rescheduling. In this paper, and more generally in practice, the only rescheduling actions are *Reinsert* moves: moving one plane, forward or backward, in the landing sequence (and therefore all the flights between the initial position of the plane and its new position are shifted by one position). Therefore, we define f_3 as the number of rescheduling actions that must be performed to generate the current solution Π^t under evaluation from the initial solution π^t, which is based on $\Pi^{t-\Delta t}$, and where the new flights entering H^t at time t have been inserted. The accurate computation of f_3 is detailed in the next subsection.

$$f_1 = \sum_{1 \leq j \leq n} \max\{C_j^t - d_j, 0\} \tag{9}$$

$$f_2 = \sum_{1 \leq j \leq n} \left[k_j^C \cdot \frac{p_j^t + Q_j^t}{V_j^t} + k_j^A \cdot \left(W_j^t + l \right) \right] \tag{10}$$

$$f_3 = \text{ number of } \textit{Reinsert} \text{ moves to change } \pi^t \text{ into } \Pi^t \tag{11}$$

Ranking the objective functions lexicographically presents many advantages. First and most importantly, it perfectly reflects the priorities of EUROCONTROL for this problem, as EUROCONTROL wants to satisfy the passengers first (i.e., minimize delay via f_1), the airlines and government second (i.e., minimize fuel through f_2), and the ATCs third (i.e., reduce their workload via f_3). The employed lexicography is efficiently generalizable, as it allows to add easily additional features to the problem. For instance, one could assign a weight γ_i to each flight j in order to better reduce the delays of the more important flights (e.g., involving many passengers, typically the bigger planes) or the more

urgent flights (e.g., involving a passenger that needs a medical intervention), without impacting how the different objectives interact with each other. In such a case, f_1 would become $\sum_{1 \leq j \leq n} \gamma_j \cdot \max\{C_j^t - d_j, 0\}$. Moreover, one could add an additional objective function f_0, prior to f_1, that would penalize the delays that lead to passengers missing their connections. In such a context, we can define f_0 as $\sum_{1 \leq j \leq n} \max\{C_j^t - \delta_j, 0\}$, where δ_j is the first connection departure of the passengers from flight j. Finally, note that the fuel consumption (f_2) depends directly on the size of the planes. Therefore, if the optimization has to choose between several planes to assign a delay (f_1), it will naturally prefer to delay the smallest. Other features could be envisioned according to the needs of the ATCs.

3.5 Computation of f_3

We denote as $\hat{\pi}$ the vector such that each component $\hat{\pi}(i)$ is the index of the flight landing at the position i. In other words, $\hat{\pi}$ is the *landing-sequence vector* associated with the *position vector* π. Therefore, for any flight j, we have $\hat{\pi}(\pi(j)) = j$. For instance, if four flights numbered from 1 to 4 are landing in the positions 2, 4, 3, 1 (i.e., $\pi = (2, 4, 3, 1)$), the corresponding landing sequence is $\hat{\pi} = (4, 1, 3, 2)$, as flight 4 lands in the first position, flight 1 in the second, and so on. We define the function $g(\hat{\pi}, \hat{\Pi})$ as the size of the longest common sub-sequence between the two landing-sequence vectors $\hat{\pi}$ and $\hat{\Pi}$. A sub-sequence is defined here as a sequence obtained from the initial sequence by deleting some elements (anywhere in the initial sequence), and without changing the order of the remaining elements. For instance, if $\hat{\pi} = (2, 4, 3, 1)$ and $\hat{\Pi} = (4, 2, 3, 1)$, we have $g(\hat{\pi}, \hat{\Pi}) = 3$, as $(2, 3, 1)$ and $(4, 3, 1)$ are common sub-sequences of size 3, but there is no common sub-sequence of size 4. We can see that one *Reinsert* move is enough to change $\hat{\pi}$ into $\hat{\Pi}$, as we could just move flight 2 right after flight 4. We can now formulate f_3 as in Eq. (12), where n is the number of flights in the planning window.

$$f_3 = n - g(\hat{\pi}^t, \hat{\Pi}^t) \tag{12}$$

An efficient algorithm to compute g in quadratic time can be found in Hirschberg (1975). The function $(n - g)$ is inspired from the *edit* distance between two strings, which consists of counting the minimum number of 'simple' moves (e.g., a *Reinsert* move consisting of shifting one character within the string) needed to transform one string into another. Different definitions of this distance can be found, relying on different move types. With the moves *Delete* (i.e., remove a character from one of the two strings) and *Insert* (i.e., add a character to one of the two strings), we have $n - g \leq edit \leq 2 \cdot (n - g)$ (Navarro 2001). In (P^t), as the two vectors $\hat{\pi}^t$ and $\hat{\Pi}^t$ are made of exactly the same characters, we have $n - g(\hat{\pi}^t, \hat{\Pi}^t) = edit(\hat{\pi}^t, \hat{\Pi}^t)$ if the *edit* distance uses only the *Reinsert* move. One can easily see that there exist $n - g(\hat{\pi}^t, \hat{\Pi}^t)$ *Reinsert* moves that transform $\hat{\pi}^t$ into $\hat{\Pi}^t$, by reinserting sequentially all flights that are not in the longest common sub-sequence. Similarly, it is impossible to transform $\hat{\pi}^t$ into $\hat{\Pi}^t$ with only $n - g(\hat{\pi}^t, \hat{\Pi}^t) - 1$ *Reinsert* moves, or we would be able to find a common sub-sequence of size $g(\hat{\pi}^t, \hat{\Pi}^t) + 1$, composed of all flights that are not reinserted. Hence, $n - g(\hat{\pi}^t, \hat{\Pi}^t)$ is the *edit* distance between $\hat{\pi}^t$ and $\hat{\Pi}^t$ using only the *Reinsert* move.

3.6 Problem complexity

The way the delays are distributed to the flights before they land does not have any impact on f_1 (i.e., the total tardiness). Therefore, when considering f_1 only, instead of working with all the three types of decision variables W_j^t, V_j^t and Q_j^t, one could only consider the waiting times W_j^t to perform the optimization and set $V_j^t = v_j$ and $Q_j^t = 0$ (i.e., use the maximum speed and do not stretch any path). This way, each flight j has a remaining flight duration of p_j^t / v_j. As a result, we now have single-machine total-tardiness problem with setup times, which is NP-hard even without setup times (Du and Leung 1990). Additionally, single-machine scheduling with sequence-dependent setup times where the objective consists of minimizing the total setup time is also NP-hard, as it corresponds to the *Traveling Salesman Problem* (Laporte 2010). The problem faced here is, therefore, NP-hard.

The interest in using speed adjustments (V_j^t) or path stretching (Q_j^t) is mainly due to f_2 (as $k_j^C < k_j^A$). When considering the three objectives, our problem is nonlinear. Indeed, although f_1 is linearizable, as there are well-known techniques to linearize the 'max', f_2 is not linearizable, because of the ratio Q/V (as the speed V is also used in the ratio p/V, and as there are two different domain constraints on Q and V). We also did not see obvious ways to linearize f_3.

Overall, we can definitely conclude that (P^t) is a difficult problem and that (meta)heuristics are more appropriate than exact methods to find efficient solutions in a practical context where decisions have to be made every 30 s and for up to 30 flights in a planning window.

4 Simulation

As mentioned previously, (P^t) considers one planning window (of length $w = 45$ min) starting at time t. In this section, we present how this planning window can be rolled into the full planning horizon (of three hours), and we explain how to perform the simulation. We stop rolling the planning window

when all flights of the instance have landed. The uncertainties are described first. Next, the rolling-window algorithm is designed. Finally, the rules to determine the landing times are presented. Based on the flight positions, the latter computation relies on distributing delays through the use of speed adjustments, path stretching, or waiting times. This distribution is a very sensitive aspect, as it helps to cope with uncertainties. If it is performed efficiently, the uncertainties should not change the landing sequence drastically when moving from one time step to the next.

4.1 Uncertainties

The model proposed here for uncertainties has been validated by EUROCONTROL, both theoretically and experimentally. It captures the different uncertainty elements that have the most significant impact on solutions to the considered problem.

At time t, the uncertainties faced by an aircraft j impact its speed by a percentage u_j^t, which can be positive or negative. Therefore, instead of flying at the planned speed V_j^t, the aircraft will fly at speed $(1 + u^t) \cdot V_j^t$ during the time interval $[t, t + \Delta t[$. For two different flights, the uncertainties are entirely independent unless they come from the same sector, in which case they face the same uncertainties, typically winds. The sectors correspond to angle intervals centered toward the airport, as represented in Fig. 2. For instance, the two flights belonging to the sector $[30, 60[$ have the same uncertainties, but the two flights in sectors $[90, 120[$ and $[120, 150[$ face different and uncorrelated uncertainties. The consideration of sectors is a relevant simplification of the reality, where the flights directions are grouped into sectors when entering the planning window: the aircraft align themselves within just a few different possible directions. This can be observed in Fig. 3 (used with the permission of EUROCONTROL, the owner), which shows 1000 real flight trajectories approaching the *Paris-Charles-de-Gaulle* airport.

The relation $u_j^t = u_j^{t-\Delta t} + 0.1 \cdot N(0, 0.07^2)$ is used to compute the speed uncertainty at time t based on the speed uncertainty encountered at time $t - \Delta t$, where N is a random variable following a normal law of average 0 and standard deviation 0.07, which were chosen based on real-data observation, and $u_j^0 = N(0, 0.07^2)$. The speed uncertainties are then summed over the cruise time steps. For the practical data analyzed in this paper, this leads to an overall speed uncertainty percentage of a flight during its cruise (i.e., from entering the planning window to entering the airport area) that ranges from -16% to $+20\%$, with a mean of 2.5% and a standard deviation of 7%. These statistics are not centered on zero percent, as when encountering positive uncertainties, a flight cruise lasts longer, and hence has more chance

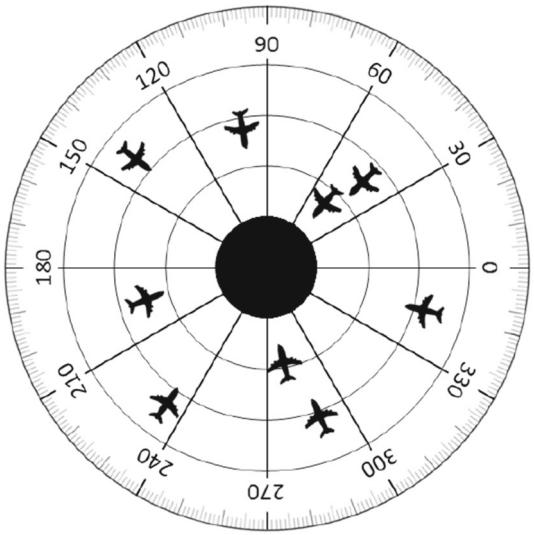


Fig. 2 Flight sectors (the central black circle is the airport area)

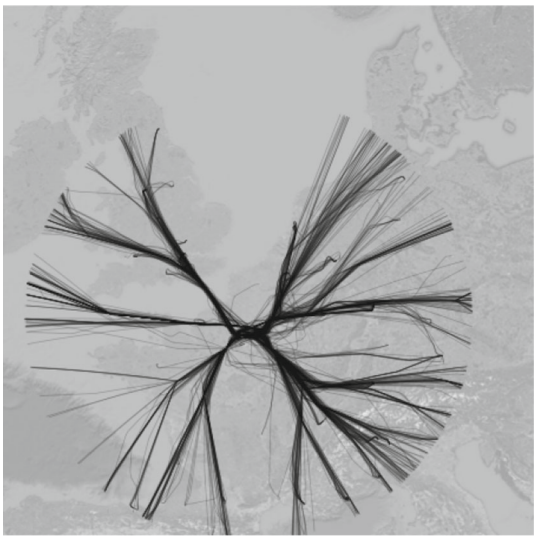


Fig. 3 Trajectories of 1000 flights landing in *Paris Charles de Gaulle* (within a 400-nautical-mile radius)

to encounter again positive uncertainties (especially as the uncertainties are linked from one time-step to another). One can note that at each time step, the actual traveled distance is $(1 + u_j^t) \cdot V_j^t \cdot \Delta t$, instead of the planned distance $V_j^t \cdot \Delta t$. Therefore, the traveled distance difference is $u_j^t \cdot V_j^t \cdot \Delta t$. Assuming the plane should have traveled at a speed of V_j^t, it results in a timing difference of $u_j^t \cdot \Delta t$. As the difference between $u_j^{t+\Delta t}$ and u_j^t follows a normal distribution with an average of zero, a reasonable forecast for the overall delay caused by uncertainty for a flight during its cruise is, therefore, the remaining cruise duration multiplied by the estimated timing difference for one time step: $\frac{p_j^t}{V_j^t} \cdot (u_j^t \cdot \Delta_t)$.

4.2 Simulation algorithm

The simulation procedure, based on a rolling planning window, is presented in Algorithm 1. In Step 2 of Algorithm 1, the landing positions π^t of the new flights are computed with the following rules, and ties are broken randomly.

Algorithm 1 Rolling-window simulation

Initialization

Set $t = 0$, $J^t = J^{t-\Delta t} = \emptyset$ and $\Pi^t = \Pi^{t-\Delta t} = ()$

While (all flights have not landed yet), **do**

1. Update J^t: remove the flights that have started landing (i.e., flights for which $t \geq C^t_j - l$) and add the flights that have just entered the updated planning window H^t (i.e., flights for which $t \geq r_j$ and $t \geq d_j - w$, as w is the length of the planning window).
2. Compute the positions of the new flights (i.e., the flights that are in J^t but not in $J^{t-\Delta t}$) to obtain the vector π^t, based on $\Pi^{t-\Delta t}$ and the insertion rules.
3. For each flight j, generate the uncertainty u^t_j.
4. Update the remaining distances: set $p^t_j = p^{t-\Delta t}_j + Q^{t-\Delta t}_j - \Delta t \cdot (1 + u^t_j) \cdot V^{t-\Delta t}_j$.
5. Update the maximum allowed path stretching: set $q^t_j = q^{t-\Delta t}_j - Q^{t-\Delta t}_j$.
6. Set the current landing positions $\Pi^t = \pi^t$, and compute variables (C^t, W^t, V^t, Q^t) accordingly.
7. Try to improve the solution $(\Pi^t, C^t, W^t, V^t, Q^t)$ based on (π^t, d, v, p^t, q^t), with any optimization method.
8. Move to the next time step: set $t = t + \Delta t$ and $H^t = [t, t + w[$.

- The just-added pop-up flights, namely the flights that just entered the planning window and just took off (i.e., the flights j that verify $t \geq r_j$ but $t - \Delta t < r_j$, and $t \geq d_j - w$) are added to the sequence at a position π^t such that their due dates are respected (i.e., a pop-up flight j is placed before all flights j' such that $C^t_{j'} \geq d_j$ but after all the other flights).
- The other just-added flights, namely the flights that took off a while ago and have entered the planning window (i.e., the flights j that verify $t \geq r_j$ and $t \geq d_j - w$, but $t - \Delta t < d_j - w$) are inserted at the end of the sequence (i.e., the earliest-due-date rule is applied). This corresponds to the FCFS rule, as flights have smaller positions if they entered earlier in the planning window.

Note that other insertion rules have been tested (e.g., insert the pop-up flights at the end of the sequence, just as for the other 'new' flights, or consider them before they take off) but lead to worse average delays. Moreover, as the above-presented rules are currently used in practice, no further investigations are proposed.

4.3 Variables computation

Let t be the current time. After each modification of vector Π^t (i.e., the landing positions), the variables (C^t, W^t, V^t, Q^t) are computed accordingly. The following current-practice rules are applied. First, for the sake of clarity, we re-number all flights of J^t as $j_1, j_2, \ldots, j_n$ such that $\Pi^t_{j_1} < \Pi^t_{j_2} < \ldots < \Pi^t_{j_n}$. Next, for $k = 1$ to n, we perform the following Steps (S1) to (S5).

(S1) $C^t_{j_k} = \max\{C^t_{j_{k-1}} + s_{j_{k-1}, j_k}, t + \frac{p^t_{j_k}}{v_j} + l\}$
The arrival time of flight j_k is chosen as close as possible to the arrival time of the previous flight j_{k-1}, while respecting the setup time s_{j_{k-1}, j_k} between the two flights, or as soon as j_k can land.

(S2) $W^t_{j_k} = \min\{C^t_{j_k} - (t + \frac{p^t_{j_k}}{v_j} + l), T\}$
If flight j_k is too early, we make it wait (by maintaining a holding pattern in the airport area) by an amount that cannot exceed a tolerance parameter T, which is typically fixed to two minutes in current practice. Note that this value cannot be negative, as the first term in the min function is always positive thanks to Step (S1), and as the second term is T which is also positive.

(S3) $V^t_{j_k} = \max\{\alpha \cdot v_j, \frac{p^t_{j_k}}{C^t_{j_k} - W^t_{j_k} - t - l}\}$
Flight j_k adapts its speed to meet $C^t_{j_k}$, but without being below the allowed minimum speed.

(S4) $Q^t_{j_k} = \min\{V^t_{j_k} \cdot (C^t_{j_k} - t - W^t_{j_k} - l) - p^t_{j_k}, q^t_j\}$
The possible additional path assigned to flight j_k (i.e., added to the remaining distance of the next time step $p^{t+\Delta t}_j$) cannot exceed the upper bound q^t_j.

(S5) $W^t_{j_k} = C^t_{j_k} - (t + \frac{p^t_{j_k} + Q^t_{j_k}}{V^t_{j_k}} + l)$
The flight maintains a holding pattern in the airport area if it is too early with respect to the planned landing time).

One can remark that waiting-time decisions appear in Steps (S2) and (S5). This is relevant from both practical and cost standpoints, as detailed below. Step (S2) is important to cope with uncertainties. Consider a flight j that is still in its cruise phase, and for which we decide at time t that it should spend W^t_j seconds in waiting when reaching the airport area (i.e., some time periods later). If in the next time step $t + \Delta t$, j faces uncertainties that delay it by 10 s, then we can decide that it should spend only $W^{t+\Delta t}_j = W^t_j - 10$ s in waiting, and therefore, the arrival time will not be impacted, assuming that W^t_j is larger than 10 s. Conversely, if at time t, we stretch the path or change the speed of j, these decisions cannot be

revoked later, as they will have been already implemented by the involved pilot. However, as waiting in the airport area consumes more fuel than changing speed or path stretching, we do not want all of the delay to be distributed to j using Step (S2). This is why parameter T is employed as a threshold in (S2). Finally, as the possibilities to distribute the delay using speed adjustment and path stretching are also upper bounded, Step (S5) re-uses the waiting-time opportunity if there is still a delay to be distributed after the previous steps.

We propose two improvements to the above current-practice rules. First, instead of staying with a static T, set to two minutes in practice, we propose to use a dynamic value T_j^t, depending on the time t, on each flight j and its remaining cruise time (p_j^t/v_j^t). More precisely, for each flight j, we set $T_j^t = \beta \cdot (p_j^t/v_j^t)$, where β is a given percentage. As the remaining cruise time decreases, the less uncertainty there should be, and therefore, the smaller T should be. If β is tuned efficiently, this is likely to reduce both the delay f_1 (as the Step (S2) will capture the uncertainties sufficiently) and fuel consumption f_2 (as T is progressively reduced, Steps (S3) and (S4) should be increasingly more involved). Second, instead of considering the planned speeds V_j^t in the simulation, we propose to use the estimated speed $(1 + u_j^t) \cdot V_j^t$. This is likely to lead to more accurate arrival times (i.e., it obviously provides different ones and, therefore, changes the evaluation of a solution) and, hence, to smaller values of T for both the static and the dynamic cases and, thus, allows reducing both f_1 and f_2.

Without optimization (in Step 7 of Algorithm 1) and using the planned speeds (i.e., without any uncertainty), the best value (regarding f_1) for a static parameter T is 120 s, which is currently used in practice. However, using estimated speeds (i.e., involving the uncertainty component) leads to having the same delay penalty, whichever value the parameter T takes in the range $[30, 600]$ s. The best value for T will, therefore, be 30 s because if the delay does not change, the smaller T is, the less fuel is consumed. This means that the resulting expected arrival times are more accurate (i.e., they capture actual arrive times more effectively), and therefore, the delay can be absorbed earlier when compared to using the planned speeds.

With optimization (in Step 7 of Algorithm 1), using the estimated speeds is also beneficial, but better results (with respect to the lexicography) are obtained by tuning T to a dynamic value (here, 25% of the remaining cruise time, by setting $\beta = 0.25$). In both static and dynamic cases, smaller T values increase the delay (f_1), whereas larger T values do not deteriorate f_1 but increases fuel consumption (f_2). This dynamic mechanism for updating T (i.e., a proportion of the remaining cruise time) leads to smaller average delays than using a single constant value (e.g., 120 s). Larger values of T are more appropriate when the flight will not be landing soon and, therefore, gives more possibilities to change its landing position at this point in time. Additionally, this way of gradually reducing T decreases f_2 drastically, as it leads to smaller values of T when the flights are close to the airport, and therefore, it uses other possible options more effectively to distribute the delay gradually as the flight approaches the landing phase (i.e., as the uncertainties are revealed).

Following these observations, in Sect. 6, we will compare the results obtained with the current practice (i.e., planned speeds with a static T tuned to 120 s) and with our proposed simulation (i.e., estimated speeds with a dynamic T tuned to 25% of the remaining cruise time).

5 Solution methods

An optimization method can be used in Step 7 of Algorithm 1. As (1) the considered problem is NP-hard, (2) the instances have dozens of flights, and (3) the allowed computing-time limit is very short (30 s), the possible choices for an appropriate solution method are drastically reduced. Exact methods, cumbersome population-based metaheuristics (e.g., genetic algorithms and ant algorithms) or metaheuristics using a somewhat long learning process (e.g., simulated annealing) are too slow. In contrast, fast and efficient local-search methods, such as descent search or tabu search, seem to be perfectly relevant, as they can provide good solutions quickly, even if they are stopped early in their search process. Tabu search has proven to be an excellent trade-off with respect to quality, robustness, speed, and adaptability (Gendreau and Potvin 2019; Zufferey 2012), whereas other metaheuristics tend to favor one of these criteria at the expense of the others.

Considering the computing-time limit of $\tau^{total} = \Delta t = 30$ s, as allowed in practice, we propose four different local-search methods for improving a solution to the considered problem: a *Descent Search with Restarts* (DSR), a *Tabu Search with Restarts* (TSR), a *Descent Search with Guided Restarts* (DSGR), and a *Tabu Search with Guided Restarts* (TSGR). To use the full computing-time budget τ^{total}, we perform restarts if a descent search is used, as the first local optimum is usually encountered within a second. Tabu search can be applied one time during τ^{total} seconds or restarted if specific conditions are met. Preliminary experiments have shown that the latter option for tabu search is significantly more effective than the former. Two neighborhood structures are investigated, namely *Reinsert* and *Rebuild* (we have also tried *Swap*, but this leads to poor performance). In the three following subsections, we first present the two neighborhood structures *Reinsert* and *Rebuild*, followed by the two local-search frameworks (descent and tabu search), and finally, the guided-restart mechanism.

A solution S at time t is defined by by the variable set $(\Pi^t, C^t, W^t, V^t, Q^t)$. For two solutions S and S', we say

that $F(S') < F(S)$ if S' is better than S with respect to the lexicographic ranking $f_1 > f_2 > f_3$, more precisely if either one of the following cases is true:

(1) $f_1(S') < f_1(S)$;
(2) $f_1(S') = f_1(S)$ and $f_2(S') < f_2(S)$;
(3) $f_1(S') = f_1(S)$ and $f_2(S') = f_2(S)$ and $f_3(S') < f_3(S)$.

5.1 Neighborhood structures *Reinsert* and *Rebuild*

The move *Reinsert* consists of changing the position Π^t_j of a flight $j \in J^t$ within the landing sequence. To reduce the neighborhood size efficiently, a pragmatic constraint is added: the new position must be in $[\Pi^t_j - 5; \Pi^t_j + 5]$. This kind of limitation is standard and is commonly referred to in the literature as *Constrained Position Shifting* (CPS) (Balakrishnan and Chandran 2010; Dear and Sherif 1989; Trivizas 1998). It is, however, not a significant restriction for the overall search process, where thousands of iterations are performed, as it is only applied when generating a neighbor solution from the current solution. Note that after each modification of the landing positions, the variables (C, W, V, Q) must be re-computed to evaluate the neighbor solution with respect to f_1, f_2 and f_3. Let $N(S)$ denote the set of neighbor solutions of the current solution $S = (\Pi, C, W, V, Q)$. To perform more moves per second and increase the exploration capability of the method, only a random proportion ρ of $N(S)$ is generated. This proportion has been tuned to $\rho = 50\%$ after preliminary experiments (testing $\rho \in \{25\%, 50\%, 75\%, 100\%\}$).

The neighborhood structure *Rebuild* is comprised of the two steps presented below: *Deconstruct* and *Reconstruct*. Let $N_i(S)$ (where i is a parameter) be the set of such neighbor solutions of $S = (\Pi, C, W, V, Q)$.

1. *Deconstruct*: remove i flights from the landing sequence and re-compute the variables (C, W, V, Q) accordingly.
2. *Reconstruct*: re-insert the i flights sequentially into the sequence in a greedy fashion and re-compute the variables (C, W, V, Q) accordingly.

Note that if $i = 1$, *Rebuild* is reduced to *Reinsert*. For this reason, the value $i = 1$ will not be considered for *Rebuild*. Let Π and Π' be the landing-position vectors of solutions S and S', respectively. As stated in Sect. 3, the distance measure $g(\Pi, \Pi')$ is exactly the minimum number of *Reinsert* moves that are required to transform S' into S. One can observe that the distance between S and any solution in $N_i(S)$ is upper bounded by i. *Rebuild* and *Reinsert* have very different properties, as any *Reinsert* move has a small (i.e., at a distance of 1) and well-tuned impact on the solution structure to improve it, whereas *Rebuild* relies on deconstructing (significantly, if i is large) to escape from the region of the solution space where the search process is currently trapped, as distances up to i are possible. The complementarity of *Reinsert* and *Rebuild* will be used in different phases of the local-search methods proposed below. *Reinsert* plays an intensification role (i.e., it has the ability to explore the zone of the solution space that contains the current solution efficiently), whereas *Rebuild* plays a diversification role (i.e., it has the ability to explore various and distant regions of the solution space).

5.2 Descent-search and Tabu-search frameworks

The frameworks of *Descent Search* (DS) and *Tabu Search* (TS) are presented in Algorithms 2 and 3, respectively. DS is a simple heuristic that moves from the current solution S to the best solution in $N(S)$ iteratively. In DSR, DS is restarted anytime S cannot be improved. When the total computation time limit τ^{total} is reached, the overall best-generated solution, denoted as $S^\star$ in this paper, is returned. The metaheuristic TS (Glover 1998) relies on the same principles, but with a different stopping criterion and a different rule for selecting the neighbor solution. More precisely, to escape from local optima, anytime a move is performed, the reverse move is forbidden for tab (parameter) iterations. At the end of each iteration, the parameter tab is generated randomly (with a uniform distribution) in the interval [2, 8]. Preliminary experiments have shown that using such a random value in a small interval leads to better results than employing a fixed duration. This idea was first introduced in Taillard (1991), and it is now commonly employed in tabu-search methods to prevent cycling in the search space (Gendreau and Potvin 2019). At each iteration, the best non-tabu neighbor solution is selected, even if it is worse than the current solution, and the process stops when a specific condition is met (i.e., it does not stop if the current solution is not improved). The best-encountered solution $S^\star$ is returned at the end. In our implementation of TS, we stop generating neighbor solutions when the computation time has reached a given time limit τ^{time} (parameter tuned to 2 s), or when the number of iterations without improvement of $f(S^\star)$ has reached a given limit τ^{iter} (parameter tuned to 100 iterations).

Algorithm 2 Descent Search (DS)

Initialization

Set S to the solution resulting from Step 6 of Algorithm 1.

While (S is improved), **do**

1. Generate the best neighbor solution $S' \in N(S)$.
2. If $F(S') < F(S)$, set $S = S'$.

Algorithm 3 Tabu Search (TS)

Initialization

- Set S to the solution resulting from Step 6 of Algorithm 1.
- Initialize the best-encountered solution: set $S^\star = S$.

While (neither τ^{time} nor τ^{iter} is met), **do**

1. Generate the best non-tabu neighbor solution $S' \in N(S)$ (let j denote the flight reinserted).
2. Update the current solution: set $S = S'$.
3. Update the best-encountered solution: if $F(S) < F(S^\star)$, set $S^\star = S$.
4. Update the tabu status: flight j cannot be moved again during tab (parameter) iterations.

Algorithm 4 Guided Restarts (GR)

Initialization

- Set S to the solution resulting from Step 6 of Algorithm 1.
- Initialize the best-encountered solution: set $S^\star = S$.
- Set $i = i^{\min}$.

While (the computing time has not reached τ^{total}), **do**

1. Generate a neighbor solution $S' \in N_i(S)$ at random (as presented in Section 5.1).
2. Perform a local search on S' (i.e., either DS or TS).
3. Update the current solution: set $S = S'$.
4. Update i: if $F(S) < F(S^\star)$, set $i = i^{\min}$; otherwise if $i < i^{\max}$, set $i = i + 1$.
5. Update the best-encountered solution: if $F(S) < F(S^\star)$, set $S^\star = S$.

Note that the lexicographic approach employed retains flexibility to optimize f_2 and f_3. In preliminary experiments, averaged over all optimization methods and all instances, we had equivalent solutions for f_1 (resp. for both f_1 and f_2) for 22% (resp. 20%) of the generated neighbor solutions. Having enough room to optimize f_3 is somewhat expected. The delay penalty f_1 is often unchanged when moving a flight in the landing sequence, as some delays are just switched from one flight to others. The same observation holds for the fuel penalty f_2, especially if the concerned flights involve the same category of planes. Furthermore, when a new best solution is found, with respect to the lexicographic priorities, it improves f_1 in 91% of the cases, reduces f_2 without increasing f_1 in 8% of the cases, and decreases f_3 in 1% of the cases without increasing f_1 and f_2. Such percentages make sense as a new best solution is generally more distant, in terms of f_3, from the initial solution than from the current or the previous best solution.

5.3 Guided restart mechanism

The straightforward restart mechanisms used in DSR and TSR have been explained previously. When a stopping condition is met, the involved method is restarted from the same initial solution $S^{(init)}$, and the best-visited solution $S^\star$ is returned at the end of the search process. Preliminary experiments showed that DSR and TSR are less efficient if all the neighbor solutions are investigated at each iteration (i.e., if $\rho = 100\%$). This likely means that a strong diversification technique is needed to escape from the solution-space region containing $S^{(init)}$. This conjecture motivates us to implement more refined restarts, leading to the *Guided Restart* mechanism described in Algorithm 4.

The parameter $i^{\min}$ is set to 2, as *Reinsert* moves, used in the local search involved in Step 2, correspond to a distance of 1. Moreover, after preliminary experiments, $i^{\max}$ has been set to a third of the size of J^t. With larger values, we are likely to lose control of the search process, and the guided-restart mechanism would, instead, be a random restart mechanism.

Note that Algorithm 4 differs from *Variable Neighborhood Search* (Hansen and Mladenović 2001) because we always update the current solution. In contrast, VNS relocates the search by updating the current solution S only if the solution returned by the intensification mechanism (i.e., the employed local search applied to S', which is the output of Step 2 here) is better than S. We have tested VNS for our problem, but it does not lead to better solutions than Algorithm 4. Our conjecture is that the search must be frequently diversified due to the very short time limit, and the exploration capability (Step 1) is higher when the current solution is updated at each iteration (Step 3).

6 Results

All the algorithms were coded in C++ under Linux and run on a 3.4 GHz Intel Quad-core i7 processor with 8 GB of DDR3 RAM. In this section, we compare the following optimization methods presented in Sect. 5.

- FCFS: First-Come-First-Served heuristic, a common practice rule (see Algorithm 1 without Step 7).
- DSR: descent search (DS) with restarts.
- DSGR: descent search with guided restarts.
- TSGR: tabu search (TS) with guided restarts.

We also compare the two above-described simulation approaches (see Sect. 4.3).

– *STATIC*: based on planned speeds and a single value of T. This procedure corresponds to the current practice.
– *DYNAMIC*: based on estimated speeds and a variable T. This mechanism integrates the proposed improvements in the current-practice simulation.

The instances are presented in the following subsection. Next, we benchmark our best method (TSGR) with the CPLEX solver and provide initial insights. Then, we compare the above-identified approaches for f_1, f_2, and f_3. Tables 5, 6, 7 present the results per cluster, as well as over all instances. Finally, we conduct a sensitivity analysis to assess the effect of different magnitudes of uncertainty, we analyze the impact of the lexicographic optimization approach on the lower-level objectives (i.e., how much it is detrimental for f_2 and f_3), and we propose insights on how to integrate take-off and gate assignment in our approach.

6.1 Presentation of the instances

EUROCONTROL provided 149 instances (which have been made accessible in Vié and Zufferey (2021) in an anonymized format that satisfies our non-disclosure agreement with EUROCONTROL), taken from three different large European airports: Frankfurt-am-Main (FRA), Amsterdam-Schiphol (AMS), and Paris-Charles-de-Gaulle (CDG). As proposed by EUROCONTROL, we separated them into three clusters: 48 instances for Cluster 1, 68 for Cluster 2, and 33 for Cluster 3. The instance separation was made with respect to three different indicators: (1) the total number of flights; (2) the average delay (positive or negative) obtained without uncertainties and without optimization; (3) the scaled density, which is the maximum number of flights belonging to a 15-minute planning window within the full 3-h horizon. Statistics of these different indicators (i.e., the minimum, maximum, and average values) are given in Table 3 for the three clusters.

As one can see in Table 3, the three indicators tend to increase from Cluster 1 to Cluster 3. Therefore, an instance from Cluster i is likely to be more difficult than one from Cluster $i-1$ (for $i \in \{2, 3\}$). This instance clustering will, therefore, allow us to see how the different algorithms react to the increasing difficulty of the instances. For a better view of the distribution of the three indicators, see Figs. 4, 5, 6.

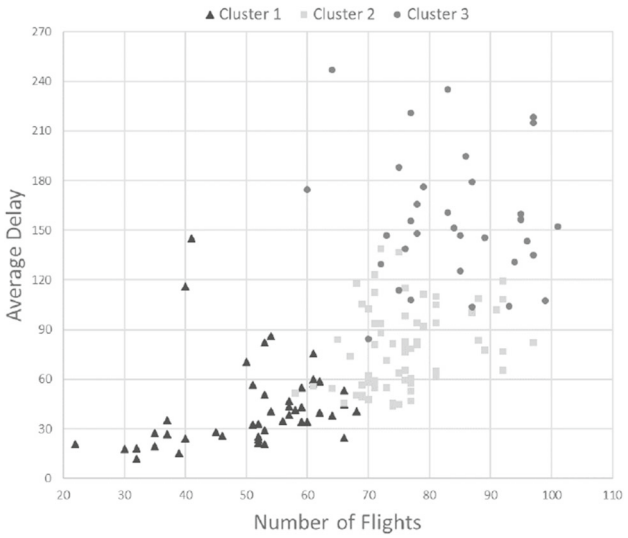


Fig. 4 Instance distribution by number of flights and average delay

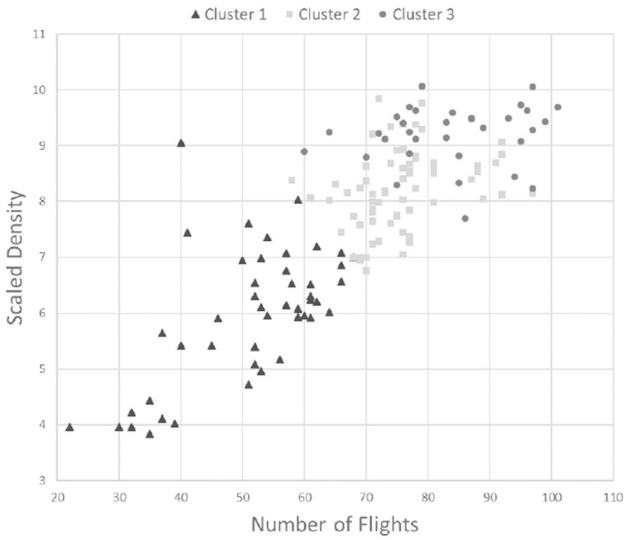


Fig. 5 Instance distribution by number of flights and scaled density

6.2 Performance of CPLEX and initial insights

To provide the first benchmark, we propose to compare FCFS, CPLEX, and TSGR. Note that FCFS was previously the most employed current-practice approach (Erzberger 1995), and it is still a very popular rule in practice. Although more sophisticated heuristics than FCFS are used today, they mainly concern the short planning horizon before landing (100NM), without uncertainty, and are intended to fine tune the landing sequence.

In order to obtain a linear model, the following restrictions are considered here (only in Sect. 6.2 and not for the following subsections): the comparison is performed for the first objective f_1; the delay is distributed on the waiting phase only (see Sect. 3.6); the planned speeds are always used to compute the arrival times. This results in the linear optimization problem defined by Eqs. (13)–(20). In order to linearize f_1, we introduce the additional variables $M_j^t = \max\{C_j^t - d_j, 0\}$. Also, as the position variables Π_j^t only appear in f_1, to linearize Constraints (2), we replace them by precedence variables $A_{j,j'}^t$, which equal 1 if j' lands after j (i.e., $\Pi_{j'}^t > \Pi_j^t$), and a very big integer I. Constraints (14) and (15) ensure anti-symmetry and transitivity, respectively. Finally constraints

Table 3 Presentation of the instance clusters

Cluster	Number of flights			Average delay			Scaled density		
	Minimum	Maximum	Average	Minimum	Maximum	Average	Minimum	Maximum	Average
1	22	68	51.31	11.88	144.95	42.69	3.83	9.05	5.94
2	58	97	75.82	43.32	138.92	78.61	6.76	9.84	8.21
3	60	101	83.76	84.19	246.88	156.40	7.70	10.07	9.19

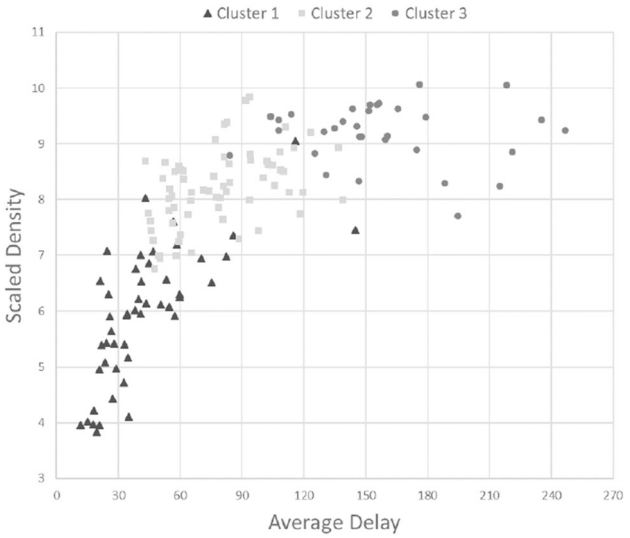


Fig. 6 Instance distribution by average delay and scaled density

(16) enforce the minimum safety separation with these new variables. At each time step, FCFS provides CPLEX and TSGR an initial solution. This ensures that a feasible solution can always be reached. The imposed time limit of 30 s is employed for TSGR, whereas CPLEX has the advantage of using five minutes, because smaller time limits do not allow CPLEX to propose interesting results.

$$\min f_1 = \sum_{1 \le j \le n} M_j^t \tag{13}$$

$$\text{s.t. } A_{j,j'}^t + A_{j',j}^t = 1 \quad \forall j, j' \in J^t, \text{ such that } j \neq j' \tag{14}$$

$$A_{j,j'}^t + A_{j',j''}^t - A_{j,j''}^t \le 1 \quad \forall j, j', j'' \in J^t, \text{ such that } j \neq j', j \neq j'', j' \neq j'' \tag{15}$$

$$C_j^t + s_{j,j'} - C_{j'}^t \le I \cdot A_{j',j}^t \quad \forall j, j' \in J^t, \text{ such that } j \neq j' \tag{16}$$

$$C_j^t = t + \frac{p_j^t}{v_j} + W_j^t + l \quad \forall j \in J^t \tag{17}$$

$$M_j^t \ge C_j^t - d_j \quad \forall j \in J^t \tag{18}$$

$$A_{j,j'}^t \in \{0,1\} \quad \forall j, j' \in J^t, \text{ such that } j \neq j' \tag{19}$$

$$C_j^t, W_j^t, M_j^t \ge 0 \quad \forall j \in J^t \tag{20}$$

FCFS, CPLEX, and TSGR are compared with and without uncertainties. Table 4 presents the average delay obtained for each method and each cluster. In the situations with uncertainty, the presented values are also averaged over five runs. The following observations can be made. First, the FCFS results can be improved by both CPLEX and TSGR. Second, without uncertainty, TSGR and CPLEX have comparable results, despite the much larger time limit assigned to CPLEX. Third, with uncertainty (i.e., when the problem becomes more complex and more realistic), TSGR outperforms CPLEX significantly, and interestingly, the gap between TSGR and CPLEX increases with the instance difficulty. Indeed, 10 s are saved for Cluster 1, 45 s for Cluster 2, and 100 s for Cluster 3.

These initial results provide interesting insights that can be used in practice. They confirm that the use of an exact method is inappropriate, as a non-realistic time limit is needed to propose efficient results for a simplified problem. More sophisticated heuristics (but ignoring uncertainty) are sometimes used when planes are close to the airport to fine tune the landing sequence provided by FCFS. However, we show here that the consideration of uncertainty allows for significant improvements.

Additional results that are not reported here show that CPLEX finds optimal solutions efficiently with up to six flights per planning window, and only a few seconds are required. CPLEX encounters difficulty if more flights are involved, and it never provides optimality within the allowed time limit of five minutes when there are more than 11 flights in the planning window. As expected, there are, on average, more flights per planning window in the case of uncertainty, as some flights are delayed, and thus, their flight duration is longer. This can also explain why CPLEX performs poorly with uncertainty.

6.3 Results for f_1 (delays)

Table 5 shows averaged (over the considered instances) delay values for the following KPIs, in seconds: the average, max-

Table 4 Average delays (in seconds)

	Without uncertainty			With uncertainty		
Method	FCFS	CPLEX	TSGR	FCFS	CPLEX	TSGR
Cluster 1	41.89	38.29	38.89	158.00	94.54	83.77
Cluster 2	77.59	70.12	70.87	207.75	150.75	104.08
Cluster 3	154.52	131.50	130.71	291.91	349.69	147.79
Average	83.13	73.46	73.82	210.36	176.70	107.22

imum, and median delay. First, we can see that *DYNAMIC* significantly outperforms *STATIC*. The benefit of *DYNAMIC* is greater for the larger instances (see Cluster 3) and the more refined optimization methods (see TSGR): an average of 0.5 s is saved for FCFS on Cluster 1, whereas 34 s are saved for TSGR on Cluster 3, when using *DYNAMIC* instead of *STATIC*. Such observations highlight the accuracy and flexibility of *DYNAMIC* with respect to *STATIC*. Delays are evaluated more accurately, and the dynamic value of T allocates enough room for optimizing the solution at appropriate times, providing flexibility.

Second, we can see that optimization (in Step 7 of Algorithm 1) can lead to a substantial improvement in the average and median delays. For instance, with *DYNAMIC*, DSR reduces the average delay by almost half when compared to FCFS. More precisely, the degree of improvement increases along with the difficulty of the instance: 42% for Cluster 1, 47% for Cluster 2, and 49% for Cluster 3. We can also measure the benefit of the guided-restart mechanism. This likely means that the local optima are difficult to escape and that a strong diversification procedure is needed for the considered problem. Finally, as TSGR outperforms DSGR, we can conclude that the tabu escaping mechanism drives the search process efficiently toward promising solutions.

Globally, when comparing our best method (i.e., TSGR-*DYNAMIC*) with respect to a common practice rule (i.e., FCFS-*STATIC*), the average (over all instances) percentage improvement of the average delay is 49.42%; our average delay is 132.41 s, compared to 261.79 s for the common practice rule. Moreover, the improvement increases with cluster difficulty, as there is a gain of 43.80% for Cluster 1, 50.11% for Cluster 2, and 52.70% for Cluster 3.

With the median results, with *DYNAMIC*, we can observe that half the planes land with less than 73 s of delay with TSGR, whereas half had more than 250 s of delay with FCFS. The median value is reduced more significantly by optimization than the average value. This likely means that to reduce the average delay, TSGR penalizes just a few planes and allows most of the other ones land on time, whereas with FCFS, the delay is distributed amongst all the planes in a more balanced fashion. For each method and each simulation approach, the median values double from Cluster 1 to Cluster 3. For instance, with *DYNAMIC*, it moves from 175 s with FCFS (resp. 53 s with TSGR) for Cluster 1 to 365 s (resp. 109 s) for Cluster 3. This shows how the difficulty of the instances increases with the cluster type.

FCFS is the best method regarding the maximum-delay KPI. This makes sense, as FCFS guarantees optimality for minimizing the maximum delay for single-machine job-scheduling contexts. Comparing FCFS and TSGR, the maximum delay is less impacted when moving from Cluster 1 to Cluster 3. Indeed, for both *STATIC* and *DYNAMIC*, it moves from 9 (Cluster 1) to 14 min (Cluster 3) for FCFS, whereas it moves from 9 to 14 min for TSGR.

The first managerial insight involves understanding how our best method (i.e., TSGR with the *DYNAMIC* simulation) reduces the delays (f_1) when compared to the common practice rule (i.e., FCFS with the *STATIC* simulation). For the largest instance, which is likely to be the most difficult one, and the whole three-hour horizon, Fig. 7 shows the due dates and landing times of the common practice rule and our best method. We can see that for the common practice rule, almost all the planes have a small delay, and that the landing sequence follows the order given by the due dates, which is the goal of this heuristic. However, using our best method, some planes are rescheduled (i.e., moved in the sequence) to make many of the planes land on time or close to it, as shown in Table 5.

6.4 Results for f_2 (fuel costs)

Table 6 shows results, averaged over the considered instances, for (1) the average fuel cost per flight and (2) the maximum fuel cost for a flight. For confidentiality purposes, instead of the real cost values, we report the fuel-cost increase (as a percentage) with respect to the ideal fuel cost. The latter corresponds to the case where no flight has any delays to absorb (i.e., each plane flies at its standard cruising speed, makes no path detours, and does not maintain a holding pattern before landing). Therefore, the presented values indicate the percentage increase of f_2 due to delays during the cruise phase and while maintaining a holding pattern, as well as encountering uncertainties.

Similar observations (as regarding delays) can be made regarding fuel costs. First, *DYNAMIC* outperforms *STATIC*. More precisely, there is a constant improvement of 1.5% with

Table 5 Delay results (in seconds)

KPI	Simulation	Optimization	Cluster 1	Cluster 2	Cluster 3	ALL
Average	*STATIC*	FCFS	194.46	255.31	373.08	261.79
		DSR	120.60	150.92	225.76	157.73
		DSGR	115.65	143.80	213.13	150.08
		TSGR	114.85	143.33	210.70	149.08
	DYNAMIC	FCFS	193.97	253.94	367.09	259.68
		DSR	112.09	134.14	188.07	138.98
		DSGR	111.98	130.67	180.19	135.61
		TSGR	109.28	127.37	176.46	132.41
Median	*STATIC*	FCFS	175.92	252.13	369.62	253.60
		DSR	67.71	99.18	164.55	103.52
		DSGR	62.44	92.31	152.93	96.11
		TSGR	61.78	91.98	154.19	96.03
	DYNAMIC	FCFS	175.20	250.93	364.82	251.76
		DSR	55.50	78.54	120.30	80.37
		DSGR	56.86	72.97	112.76	76.59
		TSGR	52.54	70.30	108.98	73.15
Maximum	*STATIC*	FCFS	528.81	618.84	818.33	634.02
		DSR	578.12	794.34	1265.66	829.07
		DSGR	549.84	739.29	1131.86	765.21
		TSGR	545.76	733.88	1096.13	753.51
	DYNAMIC	FCFS	527.57	616.32	807.33	630.03
		DSR	569.27	731.82	1160.95	774.50
		DSGR	553.96	734.60	1127.15	763.35
		TSGR	558.06	712.23	1110.22	750.71

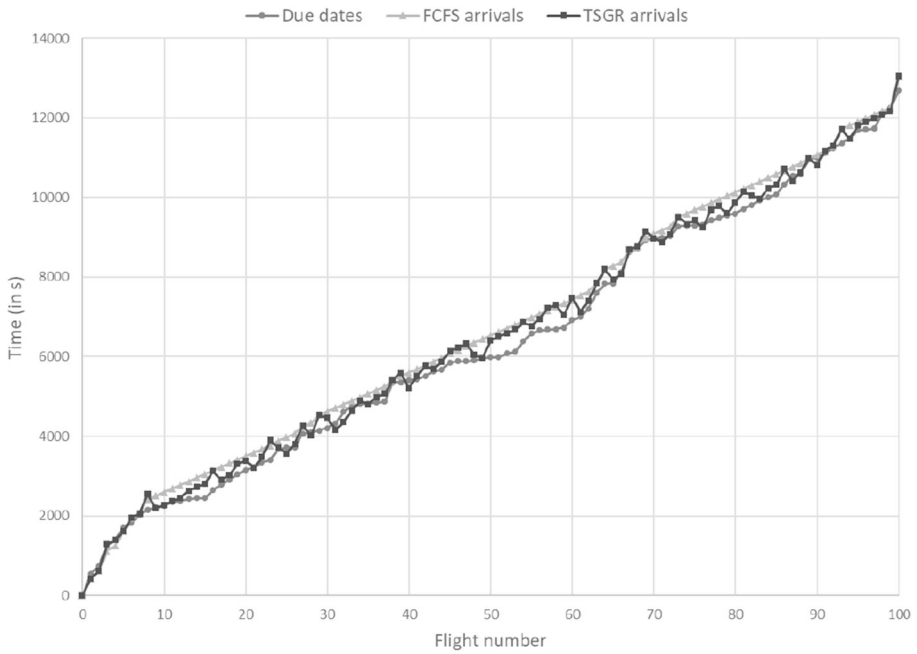


Fig. 7 Comparison of FCFS-*STATIC* (the common practice rule) and TSGR-*DYNAMIC* (our best method) regarding arrival times

FCFS for each cluster, which is probably because the cluster separation was done following delay indicators, but not fuel indicators. In contrast, with TSGR, the improvement increases from 1.54% (Cluster 1, 5.57%-4.03%) to 3.5% (Cluster 3, 13.64%-10.14%). Indeed, the larger the instance, the more flights are in the rolling planning window, and therefore, there are more possibilities to move flights from different categories to absorb the most expensive delays (i.e., the delays of the heaviest planes) during the cruise phase (i.e., by speed adjustments or path stretching instead of maintaining a holding pattern in the airport area). Each optimization method is able to reduce the average fuel cost when compared to FCFS, and the more tools the optimization method include, the better the results are on average. More precisely, the benefit of the following features can be measured: (1) a descent search, when compared to the greedy construction rule that characterizes FCFS; (2) guided restarts, when compared to non-guided ones; (3) escaping some local optima by the use of tabu tenures, when compared to always restarting the search from a different initial solution.

On average (over all the instances), when comparing TSGR-*DYNAMIC* (our best method) to FCFS-*STATIC* (a common practice rule), the fuel-cost reduction amounts to 10.51% of the ideal fuel cost (i.e., 16.83%–6.32%). Interestingly, the improvement increases with the cluster difficulty: we have an additional gain of 7.83% for Cluster 1, 10.60% for Cluster 2, and 14.24% for Cluster 3. This could be only due to f_1 and f_2 being somewhat connected. Indeed, the larger the delay is (f_1), the more fuel is consumed (f_2).

Furthermore, with respect to FCFS, despite TSGR increasing the maximum delays by 20% (see Table 5), TSGR increases only the maximum fuel cost by 10%. This shows that TSGR decreases fuel consumption not only by reducing the average delay values (f_1) but also by sequencing the flights more efficiently according to the different plane categories (i.e., planes that have the largest delay to absorb belong to smaller categories) and distributing delays more effectively (i.e., planes spend less time in holding patterns prior to landing).

An additional managerial insight involves understanding how our best method (TSGR-*DYNAMIC*) distributes delays when compared to the common practice rule (FCFS-*STATIC*). Figure 8 shows how the delays are distributed among the three different options: path detour, holding pattern, and speed adjustment (on average, over all the instances). As there is less delay to absorb when our best method is employed, we can see that each of the three possibilities is used less frequently than with the common practice rule. However, the amount of time spent in holding patterns is reduced the most, by approximately 90%, which is, as explained previously, a direct consequence of using a dynamic T. Furthermore, the speed adjustment is significantly reduced, by approximately 65%, more than the path

Fig. 8 Average delay distribution over all instances (comparing the common practice rule and our best method)

detour, which is reduced by approximately 20%, even though it is used earlier in the variables computation. Again, this is likely to be a direct consequence of using a dynamic T. When the flight is far away from the airport, as the dynamic T is proportional to the remaining cruise time, T is very large at this point, and therefore, Step (S2) of the variable computation absorbs the entire delay. The speed adjustment is only used later, when the flight is closer to the airport and T is reduced. This can be seen in Fig. 9, which shows the evolution of the decisions over time for one representative flight. We can see that for the common practice rule, the speed adjustment is used all along the cruise time, the path detour is used at its maximum, and the waiting decision increases when the flight gets closer to the airport. However, with our best method, the path detour and speed adjustment are initially not used and the waiting is high, but when the flight gets closer to the airport, the delay is switched the waiting to the two other possibilities, which is good because it reduces the fuel cost. This leads to less delay at the end, as using the waiting option first allows for more flexibility in the optimization process and probably copes better with the uncertainties. Indeed, the waiting option is a reversible decision because it is only implemented when the flight is in the airport area, whereas the other two options are implemented during the cruise phase.

6.5 Results for f_3 (landing sequence stability)

Table 7 provides the average number of *Reinsert* moves per plane performed during the whole time horizon. More precisely, at each time step, we (1) compute the minimum number of *Reinsert* moves between the solution at the beginning of the time step and the one after the optimization process (i.e., we compute f_3, as explained in Sect. 3), (2) sum

Table 6 Fuel-cost reduction results (as a percentage of the expected cruise fuel)

KPI	Simulation	Optimization	Cluster 1	Cluster 2	Cluster 3	ALL
Average	*STATIC*	FCFS	11.86	16.68	24.38	16.83
		DSR	6.01	9.19	14.73	9.39
		DSGR	5.63	8.75	13.86	8.88
		TSGR	5.57	8.71	13.64	8.79
	DYNAMIC	FCFS	10.52	15.01	22.53	15.23
		DSR	4.28	6.48	10.81	6.73
		DSGR	4.24	6.30	10.39	6.54
		TSGR	4.03	6.08	10.14	6.32
Maximum	*STATIC*	FCFS	191.80	195.18	220.27	199.65
		DSR	187.28	212.73	288.57	221.33
		DSGR	183.96	209.03	267.65	213.94
		TSGR	184.29	206.72	265.92	212.60
	DYNAMIC	FCFS	188.52	191.98	217.57	196.53
		DSR	181.54	196.18	263.18	206.30
		DSGR	180.82	198.89	265.87	207.90
		TSGR	181.16	198.61	264.09	207.49

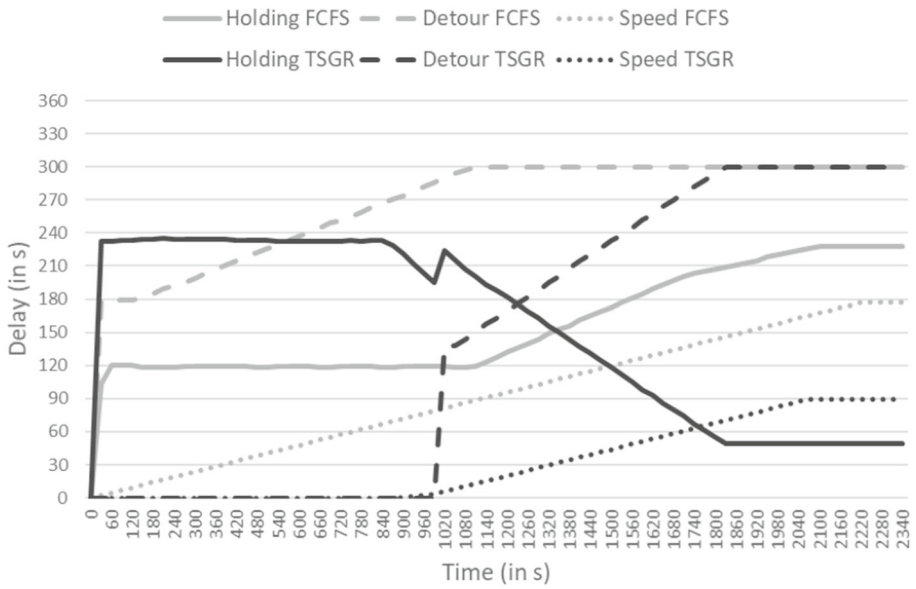


Fig. 9 Example of delay evolution for one representative flight (comparing the common practice rule and our best method)

these numbers over the whole time horizon (if we reinsert a plane three positions backward in a time step, and then reinsert it three positions forward in another time step, it counts as two *Reinsert* moves), and finally (3) divide it by the total number of flights in the instance. For example, an average of 0.5 means that if we have 60 flights, we have performed a total of 30 *Reinsert* moves. We perform this computation over the whole time horizon, instead of only between the initial solution and the final one, for two reasons. First, any action increases the ATCs' workload even if one action can compensate for or cancel another action. Second, there are no flights in the initial solution (i.e., at time 0, no plane has taken off), and thus, we cannot compute the number of *Reinsert* moves between the initial and final solution. Note that it is impossible to give the number of *Reinsert* moves operated on one plane, as sometimes, different *Reinsert* moves lead to the same solution. For instance, if the landing sequence 1–2–3–4 is changed to 1–3–2–4, it could be the consequence of a *Reinsert* of flight 2 after flight 3 or of flight 3 before flight 2. Therefore, the average number of *Reinsert* moves is the only relevant KPI for f_3.

Table 7 shows that *DYNAMIC* leads to fewer *Reinsert* moves than *STATIC*. It better evaluates the arrival times, and therefore, the decisions made at a time step are more robust. As expected, regardless of the simulation used, there are no *Reinsert* moves with FCFS. In contrast, TSGR-*DYNAMIC*

performs an average of 0.49 *Reinsert* moves per flight over the whole planning horizon of 3 h and over all the instances. This means that at most half of the planes are reinserted, as each plane can be reinserted multiple times. There is minimal variability in the results for DSR, DSGR and TSGR. However, we can see that TSGR uses fewer *Reinsert* moves than the descent-based methods. This occurs because TS does not have to stop when a local minimum is encountered. Consequently, fewer restarts are performed when compared to DS, and obviously, performing restarts from different solutions, as is the case when employing guided restarts, increases the likelihood of visiting distant solutions.

When comparing our best method (i.e., TSGR-*DYNAMIC*) with the common practice rule (i.e., FCFS-*STATIC*), in contrast with the highlighted significant improvements obtained with respect to f_1 and f_2, one can observe a very reasonable degradation of f_3. Here, the degradation increases along with the instance difficulty. Over the whole 3-h planning horizon and the considered instance set, on average, 0.33 *Reinsert* moves per flight are performed for Cluster 1, 0.50 moves per flight for Cluster 2, and 0.72 moves per flight for Cluster 3. This progressive degradation of f_3 with the instance difficulty can be explained by the lexicographic hierarchy of the objectives, as a counterpart of the gradual improvements obtained for the higher-level objectives f_1 and f_2 when the instance difficulty is increased.

Another managerial insight can be found in the statistics about the position difference per flight for our best method, which are shown in Fig. 10. On the left side of the figure, the average, maximum, and median of the absolute difference between the first planned position and the actual landing position are represented. For instance, if a plane was first planned at position 2 but landed at position 5, it counts as a difference of 3 and vice versa, as it is an absolute difference. For each cluster, we can see that the average is around 1, and the median is less than 1. However, there is a significant difference between the clusters for the maximum values: it goes from around 4.5 for Cluster 1 to almost 10 for Cluster 3. The instances of Cluster 3 have more flights and a larger scaled density and, therefore, more flights in the planning window on average. Hence, there are more possibilities to reinsert a plane forward or backward. Although it does not appear in the figure, note that with the common practice rule, the median is at zero, but the average is around 0.25, and the maximum is around 1.8. The planes are never reinserted, but a pop-up flight can be inserted earlier and, therefore, change the position of the planes in front of which it is inserted. When examining the relative position difference instead of the absolute one (right side of Fig. 10), we can see that the absolute maximum is the same as the relative maximum (i.e., between 4 and 10 positions), whereas the relative minimum is smaller (i.e., between 2 and 4 positions), which means the maximum position difference always concerns a plane moved backward. We can also observe that the average is around 0.5. It would have been zero if only *Reinsert* moves could change the position of a plane, but there are also pop-up-flight insertions, which move planes backward. Finally, we did not add the medians on the right side of the figure, but they are equal to zero for each cluster, which means there are, on average, as many flight positions moved forward as backward.

The final managerial insight involves understanding how often our best method performs *Reinsert* moves. Figure 11 shows the number of *Reinsert* moves made over time for one instance of each cluster. To capture the evolution of the number of *Reinsert* moves performed with respect to the number of flights in an instance, we took the smallest instance for Cluster 1, a medium-sized instance for Cluster 2, and the largest instance for Cluster 3. These three instances contain 22, 72, and 101 planes, respectively. Figure 12 shows the same information as Fig. 11, but it is cumulated over time. It shows that the *Reinsert* moves occur far from each other (i.e., 'the peaks', corresponding to the *Reinsert* moves, are distant from each other), which is appropriate in practice as it corresponds to a well-balanced workload for ATCs. We can also observe that the maximum number of *Reinsert* moves that can happen in one iteration increases from Cluster 1 to Cluster 3 (i.e., there is a maximum of 1 *Reinsert* move for the instance of Cluster 1, 3 *Reinsert* moves for the instance of Cluster 2, and 6 *Reinsert* moves for the instance of Cluster 3), and so does the total number of *Reinsert* moves, which is 6 for the instance of Cluster 1, 24 for the instance of Cluster 2, and 43 for the instance of Cluster 3. As for the increase in the maximum position differences discussed above, it is probably due to the average increase of the number of flights in the rolling window from Cluster 1 to Cluster 3.

6.6 Sensitivity analysis on the magnitude of the uncertainties

In order to measure the impact of the uncertainties on the results, we perform a sensitivity analysis by varying the uncertainty standard deviation from 0.03 to 0.25 (the value proposed by EUROCONTROL being 0.07). Two methods are compared here, namely the common practice FCFS-*STATIC*, and our best method TSGR-*DYNAMIC*. The results obtained on the average delays (resp. on the fuel costs) are presented in Fig. 13 (resp. Fig. 14). Unsurprisingly, we can see that, for each method and each cluster, the bigger the standard deviation of the uncertainties is, the larger are the delays and the fuel costs. However, the increase is significantly more important for the common practice rule than for our best method. Indeed, with a standard deviation of 0.03, both methods have average delays between 0 and 200 s (resp. average fuel costs between 0% and 10% of the expected fuel consumption during cruise), whereas with a

Table 7 Distance results (number of *Reinsert* moves per plane)

KPI	Simulation	Optimization	Cluster 1	Cluster 2	Cluster 3	ALL
Average	*STATIC*	FCFS	0.00	0.00	0.00	0.00
		DSR	0.35	0.55	0.84	0.55
		DSGR	0.35	0.54	0.89	0.56
		TSGR	0.35	0.54	0.88	0.55
	DYNAMIC	FCFS	0.00	0.00	0.00	0.00
		DSR	0.34	0.51	0.72	0.51
		DSGR	0.33	0.51	0.75	0.51
		TSGR	0.33	0.50	0.72	0.49

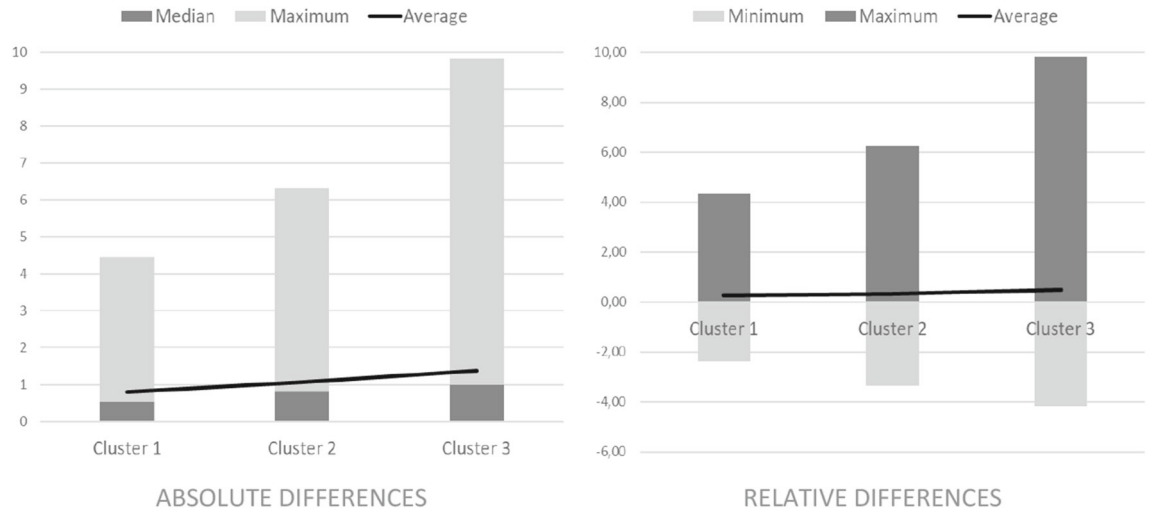


Fig. 10 Statistics of the position differences per cluster for our best method

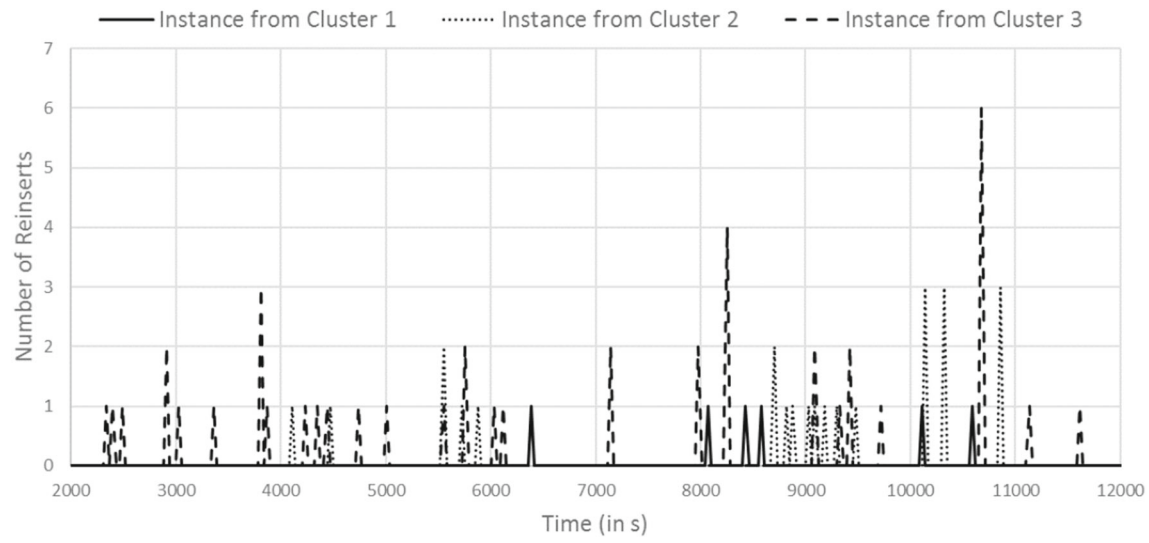


Fig. 11 Number of *Reinsert* moves over time with our best method, for one representative instance in each cluster

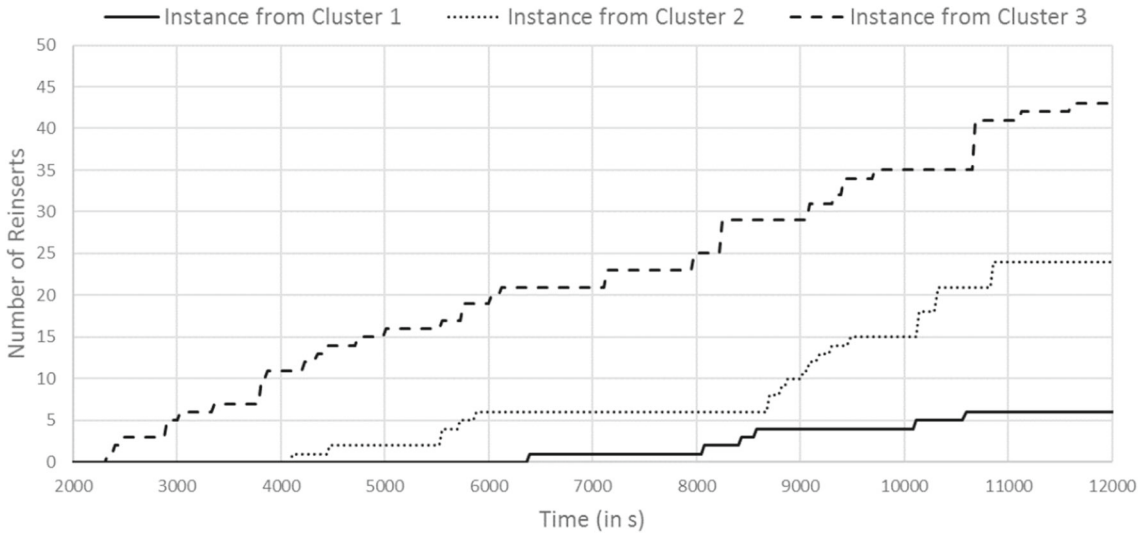
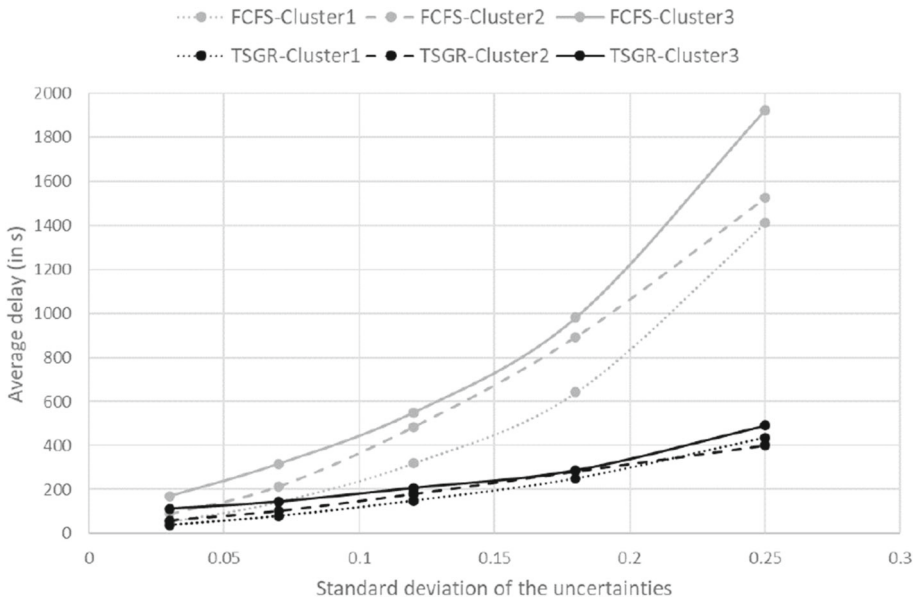


Fig. 12 Cumulated number of *Reinsert* moves over time with our best method, for one representative instance in each cluster

Fig. 13 Average flight delay per cluster with respect to different magnitudes of uncertainty (comparing the common practice rule and our best method)



standard deviation of 0.25, the common practice rule has a delay performance between 1400 and 2000 s (resp. a fuel-cost performance between 100% and 160% of the fuel consumption during cruise), while our best method has an average delay performance between 300 and 500 s (resp. a fuel performance between 20% and 30% of the fuel consumption during cruise).

Moreover, considering all the instances for each cluster, we can observe in Fig. 15 that the number of rescheduling actions performed by TSGR-*DYNAMIC* increases proportionally with the uncertainty standard deviation. For instance, in Cluster 1, with a standard deviation of 0.25, we have on average 1.5 *Reinsert* move per flight, instead of 0.4 with the standard deviation of 0.07 used by EUROCONTROL. Note that the number of *Reinsert* moves performed by FCFS-*STATIC* stays zero, as the method does not employ any rescheduling action. For this reason, no curve is showed in Fig. 15 for FCFS-*STATIC*. One can observe here that even with large uncertainty standard deviations (e.g., 0.25), the average number of rescheduling actions remains low (i.e., it mostly ranges between 1 and 2). This also indicates that the increase of f_1 and f_2 with respect to the increase of the uncertainty standard deviation remains reasonable also because a moderate deterioration of f_3 is encountered.

6.7 Evaluation of the lexicographic optimization approach

The three considered objectives (f_1, f_2, f_3) could be integrated in a single, weighted objective function by setting

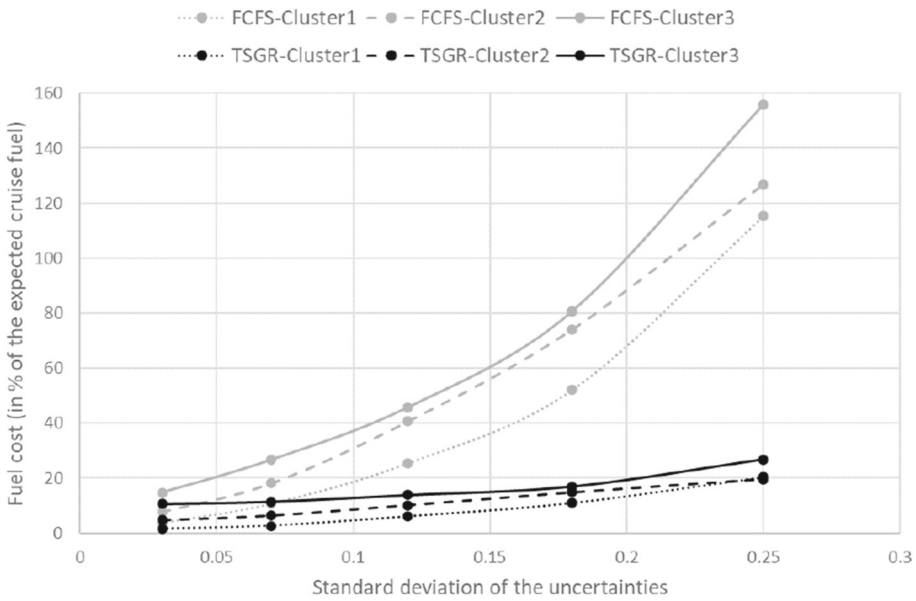


Fig. 14 Average fuel cost per cluster with respect to different magnitudes of uncertainty (comparing the common practice rule and our best method)

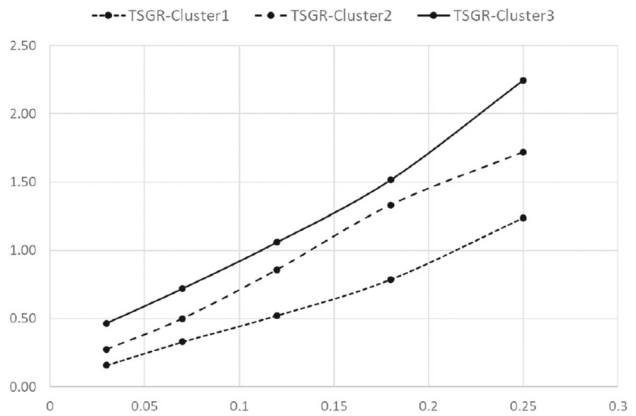


Fig. 15 Average number of *Reinsert* moves per cluster with respect to different magnitudes of uncertainty (comparing the common practice rule and our best method)

$f = w_1 f_1 + w_2 f_2 + w_3 f_3$, where the weights (w_1, w_2, w_3) have to be decided by the involved decision makers. In order to preserve the lexicography (i.e., a lower objective function can only be improved without deteriorating a higher objective), one can simply choose very discriminating weights, such as $(w_1, w_2, w_3) = (1000000, 1000, 1)$. To make trade-offs between the objectives possible, by contrast, the weights (w_1, w_2, w_3) should be closer to each other. On the one hand, it is important to be aware that each f_i has its own range of possible values. In other words, to keep control on the optimization and on the search process, it is recommended to always normalize all the f_i values in the same interval, like [0, 1] (this would however require additional computational steps and thus slow down the solution method, which has to provide a solution within seconds in practice). On the other hand, as various stakeholders are involved in f, tuning (w_1, w_2, w_3) once and for all would lead to endless discussions and is thus an impossible task in practice. This is in line with other studies, in which no trade-off among objectives is possible. Among them, one can cite applications in the automotive industry (Solnon et al. 2008), in the luxury industry (Respen et al. 2017), in telecommunication (Dupont et al. 2004), in electricity (Aghaei et al. 2011), in vehicle routing (Lehuédé et al. 2020), and in aviation (Prats et al. 2010).

As a consequence, instead of presenting various artificial, speculative results with different combinations of (w_1, w_2, w_3), we have compared the employed lexicographic approach ($f_1 > f_2 > f_3$) with ($f_2 > f_1 > f_3$). This will allow to measure the potential possible improvements on f_2, and the degradation of f_1 if it is constrained by f_2. This is indeed relevant from a managerial perspective. Note that considering f_3 as a priority simply means that it is forbidden to reschedule a flight in a different position in the landing sequence (this is thus not investigated further).

Considering TSGR-*DYNAMIC* (i.e., our best method), the experiments are presented in Table 8 for each cluster, and for all the clusters together. We show first the objective-function values for both approaches, then the variation and percentage deviation between those values. We can observe that prioritizing f_2 over f_1 deteriorates all the objectives on all the clusters (only a slight improvement of 4.72% is encountered on f_3 for Cluster 1). The degradation of f_1 is significant (on average, 32.3%) and it increases with the instance difficulty (i.e., when moving from Cluster 1 to Cluster 3, for which the delays augment by 51.6% on average). After all, such results are not surprising as less delay leads to less fuel consumption. In other words, a reduction on f_1 has a positive impact on the reduction of f_2, which highlights the relevance of prioritizing f_1 over f_2. Also, from a practical standpoint, it is likely to mean that giving the priority to large aircraft over small

Table 8 Comparison of $(f_1 > f_2 > f_3)$ with $(f_2 > f_1 > f_3)$ (i.e., prioritizing f_2 instead of f_1)

	Cluster 1	Cluster 2	Cluster 3	ALL
Objective	Approach $(f_1 > f_2 > f_3)$: objective-function values			
f_1	109.28	127.37	176.46	132.41
f_2	4.03	6.08	10.14	6.32
f_3	0.33	0.5	0.72	0.49
Objective	Approach $(f_2 > f_1 > f_3)$: objective function values			
f_1	128.04	163.58	267.52	175.15
f_2	4.33	8.11	16.99	8.86
f_3	0.31	0.55	0.73	0.51
Objective	Variation of the objective-function values from $(f_1 > f_2 > f_3)$ to $(f_2 > f_1 > f_3)$			
f_1	18.76	36.21	91.06	42.74
f_2	0.30	2.03	6.85	2.54
f_3	−0.02	0.05	0.01	0.02
Objective	Percentage deviation of the values from $(f_1 > f_2 > f_3)$ to $(f_2 > f_1 > f_3)$			
f_1	17.2%	28.4%	51.6%	32.3%
f_2	0.30%	2.03%	6.85%	2.54%
f_3	−4.72%	9.08%	1.61%	4.53%

ones in order to reduce fuel is mainly interesting if it does not increase the overall delay. Finally, given the computing-time restrictions imposed in practice (i.e., 30 s), optimizing f_1 first is computationally less cumbersome than optimizing f_2 first (see the discussion in Sect. 3.6). This is another motivation for the employed approach $(f_1 > f_2 > f_3)$.

6.8 Insights on the take-off integration

Many airports, in particular larger ones, use dedicated and independent runways for landing and take-off, with enough space between the runways to avoid any additional complexity due to take-off/landing interactions. This operating mode is called 'segregated parallel' and is used by the three major airports from which we obtained data, which allows us not to consider the take-offs of the corresponding airports; for additional details and a list of all airports using it, see SKYbrary (2019). However, the take-off uncertainty is indirectly covered by our approach. Indeed, if a flight take-off is delayed prior to the considered planning window of 45 min, it will appear later than expected in our planning window, and the proposed optimization will then try to reduce the final delay, computed as the gap between the actual arrival time and the planned landing time published by the airline company. Further investigations on the take-off integration are not performed because of the following reasons.

- The problem stated by EUROCONTROL mainly aims at helping the decisions of the ATCs that plan aircraft landing in a big airport. The ATC activity concerns the flights sufficiently close to their landing airport, which is exactly what we model here with the use of a 45-minute planning window. The duration of such a planning window is in line with the real needs of the involved ATCs, which is to schedule or reschedule efficiently the landing of flights within a couple of seconds. This corresponds to the computation time required by the real conditions.
- Allowing decisions on departure times would imply the consideration of all departures from all the involved airports (i.e., dozens of airports), and the simulation of many flight sectors associated with the wind effects. The resulting planning window would be much larger than 45 min and, as highlighted by EUROCONTROL, it would employ unreliable forecasts with respect to wind effects and flight interactions. Therefore, the problem complexity would increase unnecessarily, the simulation quality would decrease significantly, and the resulting decisions would not be reliable for the ATCs.
- The observations made by Vanwelsenaere et al. (2017) lead to the conclusion that taking into account flights before their departure (without being able to decide on their departure times) only achieves improvement in the landing planning if the actual departure time has a gap smaller than five minutes with respect to the planned departure time. This is obviously not the case in practice.

6.9 Insights on the gate-assignment integration

Despite the fact that the gate assignment problem is very different from our problem, we provide some comments on how this feature could be integrated to our solution methods.

Even if gate assignment is usually determined well in advance of our 45-minute planning window, in reality, last-minute changes can occur. In this case, as mentioned in Dorndorf et al. (2016), decision-makers can manually postpone the flight arrival times, which means that the aircraft has to wait until a gate is available. In other words, from a practical standpoint, the gate assignment will be done manually within our planning window. If a gate is not available (which can be considered like an uncertainty), we can simply apply holding stack patterns (i.e., make circles above the airport). Note however that the unavailability of a gate is rare in practice (e.g., companies have often dedicated gates in the big airports), and that landing as expected and then simply waiting on the ground (until the planned gate is available) is a commonly employed option.

7 Conclusion

Aircraft Landing Planning is a complex optimization problem. It consists of determining the flight landing sequence, including the arrival times and how to meet these times, while satisfying safety (i.e., minimum threshold distance between two landing planes) and feasibility (e.g., admissible speeds) constraints. A substantial amount of work has been done on ALP, considering various objective functions with a single or a multi-objective function. However, most of the literature either does not consider the various unavoidable uncertainties or uses too much computing time to be operational in practice.

In this work, we consider three different objective functions to optimize, in a lexicographic fashion, the total delay, fuel cost, and sequence stability, as well as design a new way to model the latter. Next, we propose efficient, flexible, and quick solution methods, employing a simulation procedure that integrates the most impactful uncertainties. TSGR, the best proposed simulation-optimization method, relies on a tabu search with a generalizable guided-restart mechanism. On average, within the considered simulated environment and when compared to a common practice rule, TSGR reduces the delay by around 50% and the average fuel cost by around 10%. Reasonable modifications of the initially planned landing sequences are performed to achieve such significant improvements. Indeed, a flight is repositioned an average of only 0.5 times. This means more than half of the flights land as expected with respect to the initial landing sequence, which is based on the landing times published by the airline companies. Finally, these landing-sequence modifications are evenly distributed over the whole considered horizon of 3 h, which means the additional workload is acceptable from the air traffic controllers' perspective (i.e., no peaks are created in their workload).

A possible extension of this study is to investigate the optimal size of the rolling planning window (i.e., not necessarily fixed to 45 min) and ways to integrate the uncertainties in a larger planning window. Next, it could be interesting to include the take-offs and their corresponding uncertainties in the optimization problem. Furthermore, as the proposed approach is flexible enough, we could consider other objective functions, such as minimizing the number of flights that land too early, as it impacts the required resource of the airport (e.g., the parking duration/space needed for the planes or staff availability for the arrivals). Finally, we could consider flights that have higher priorities to evaluate how our solution methods react to them.

Acknowledgements This study was funded in part by EUROCONTROL (the *European Organization for the Safety of Air Navigation*) through the SESAR2020 programme. The authors would like to thank Mr. Raphaël Christien from EUROCONTROL for his availability and advice. Thanks are also due to the reviewers for their valuable comments.

Funding Open access funding provided by University of Geneva

Declaration

Conflicts of interest The authors declare that they have no conflicts of interest.

References

Aghaei, J., Amjady, N., & Shayanfar, H. (2011). Multi-objective electricity market clearing considering dynamic security by lexicographic optimization and augmented epsilon constraint method. *Applied Soft Computing, 11*(4), 3846–3858.

Artiouchine, K., Baptiste, P., & Dürr, C. (2008). Runway sequencing with holding patterns. *European Journal of Operational Research, 189*(3), 1254–1266.

Balakrishnan, H., & Chandran, B. G. (2010). Algorithms for scheduling runway operations under constrained position shifting. *Operations Research, 58*(6), 1650–1665.

Beasley, J. E., Sonander, J., & Havelock, P. (2001). Scheduling aircraft landings at London Heathrow using a population heuristic. *Journal of the Operational Research Society, 52*(5), 483–493.

Bennell, J. A., Mesgarpour, M., & Potts, C. N. (2011). Airport runway scheduling. *4OR: A Quarterly Journal of Operations Research, 9*(2), 115–138.

Bennell, J. A., Mesgarpour, M., & Potts, C. N. (2017). Dynamic scheduling of aircraft landings. *European Journal of Operational Research, 258*(1), 315–327.

Brentnall, A. R., & Cheng, R. C. H. (2009). Some effects of aircraft arrival sequence algorithms. *Journal of the Operational Research Society, 60*(7), 962–972.

Caprì, S., & Ignaccolo, M. (2004). Genetic algorithms for solving the aircraft-sequencing problem: the introduction of departures into the dynamic model. *Journal of Air Transport Management, 10*(5), 345–351.

Chern, C. C., & Hsieh, J. S. (2007). A heuristic algorithm for master planning that satisfies multiple objectives. *Computers and Operations Research, 34*(11), 3491–3513.

Dear, R. G., & Sherif, Y. S. (1989). The dynamic scheduling of aircraft in high density terminal areas. *Microelectronics Reliability, 29*(5), 743–749.

Dear, R. G., & Sherif, Y. S. (1991). An algorithm for computer assisted sequencing and scheduling of terminal area operations. *Transportation Research Part A, 25*(2/3), 129–139.

Dorndorf, U., Jaehn, F., & Pesch, E. (2016). Flight gate assignment and recovery strategies with stochastic arrival and departure times. *OR Spectrum, 39*(1), 65–93.

Du, J., & Leung, J. Y. T. (1990). Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research, 15*(3), 483–495.

Dupont, A., Alvernhe, E., & Vasquez, M. (2004). Efficient filtering and Tabu search on a consistent neighbourhoodfor the frequency assignment problem with polarisation. *Annals of Operations Research, 130,* 179–198.

Erzberger, H. (1995). Design principles and algorithms for automated air traffic management. AGARD Lecture Series No 200: Knowledge-Based Functions in Aerospace Systems, Madrid, Paris, and San Francisco 7(2)

Faye, A. (2015). Solving the aircraft landing problem with time discretization approach. *European Journal of Operational Research, 242*(3), 1028–1038.

Gallay, O., & Zufferey, N. (2018). Metaheuristics for lexicographic optimization in industry. In: Proceedings of the 19th EU/ME Workshop on Metaheuristics for Industry, EU/ME 2018, Geneva, Switzerland, March 22–23

Gendreau, M., & Potvin, J. Y. (2019). *Handbook of Metaheuristics, International Series in Operations Research and Management Science* (Vol. 146). New York: Springer.

Glover, F. (1998). Tabu search: wellsprings and challenges. *European Journal of Operational Research, 106*(2–3), 221–225.

Hansen, P., & Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research, 130*(3), 449–467.

Heidt, A., Helmke, H., Kapolke, M., Liers, F., & Martin, A. (2016). Robust runway scheduling under uncertain conditions. *Journal of Air Transport Management, 56,* 28–37.

Hirschberg, D. S. (1975). A linear space algorithm for computing maximal common subsequences. *Communications of the ACM, 18*(6), 341–343.

Hu, X. B., & Chen, W. H. (2005). Genetic algorithm based on receding horizon control for arrival sequencing and scheduling. *Engineering Applications of Artificial Intelligence, 18*(5), 633–642.

Hu, X. B., & Di Paolo, E. (2009). An efficient genetic algorithm with uniform crossover for air traffic control. *Computers and Operations Research, 36*(1), 245–259.

Khassiba, A., Bastin, F., Gendron, B., Cafieri, S., & Mongeau, M. (2019). Extended aircraft arrival management under uncertainty: A computational study. Journal of Air Transportation pp 1–13

Khassiba, A., Bastin, F., Cafieri, S., Gendron, B., & Mongeau, M. (2020). Two-stage stochastic mixed-integer programming with chance constraints for extended aircraft arrival management. *Transportation Science, 54*(4), 897–919.

Laporte, G. (2010). A concise guide to the traveling salesman problem. *Journal of the Operational Research Society, 61*(1), 35–40.

Lehuédé, F., Péton, O., & Tricoire, F. (2020). A lexicographic minimax approach to the vehicle routing problem with route balancing. *European Journal of Operational Research, 282*(1), 129–147.

Lieder, A., Briskorn, D., & Stolletz, R. (2015). A dynamic programming approach for the aircraft landing problem with aircraft classes. *European Journal of Operational Research, 243*(1), 61–69.

Liu, M., Liang, B., Zheng, F., Chu, C., & Chu, F. (2018). A two-stage stochastic programming approach for aircraft landing problem. In: Proceedings of the 15th International Conference on Service Systems and Service Management, Hang Zhou, China

Navarro, G. (2001). A guided tour to approximate string matching. *ACM Computing Surveys, 33*(1), 31–88.

Nikoleris, T., & Hansen, M. (2012). Queueing models for trajectory-based aircraft operations. *Transportation Science, 46*(4), 501–511.

Pfeiffer, A., Kádár, B., & Monostori, L. (2007). Stability-oriented evaluation of rescheduling strategies, by using simulation. *Computers in Industry, 58*(7), 630–643.

Pinedo, M. L. (2016). *Scheduling: Theory, algorithms, and systems*. New York: Springer.

Pinol, H., & Beasley, J. E. (2006). Scatter search and bionomic algorithms for the aircraft landing problem. *European Journal of Operational Research, 171*(2), 439–462.

Prats, X., Puig, V., Quevedo, J., & Nejjari, F. (2010). Lexicographic optimisation for optimal departure aircraft trajectories. *Aerospace Science and Technology, 14*(1), 26–37.

Respen, J., Zufferey, N., & Amaldi, E. (2016). Metaheuristics for a job scheduling problem with smoothing costs relevant for the car industry. *Networks, 67*(3), 246–261.

Respen, J., Zufferey, N., & Wieser, P. (2017). Three-level inventory deployment for a luxury watch company facing various perturbations. *Journal of the Operational Research Society, 68*(10), 1195–1210.

Sabar, N. R., & Kendall, G. (2015). An iterated local search with multiple perturbation operators and time varying perturbation strength for the aircraft landing problem. *Omega, 56,* 88–98.

Schummer, J., & Abizada, A. (2017). Incentives in landing slot problems. *Journal of Economic Theory, 170,* 29–55.

Schummer, J., & Vohra, R. V. (2013). Assignment of arrival slots. *American Economic Journal: Microeconomics, 5*(2), 164–85.

Shone, R., Glazebrook, K.D., & Zografos, K. (2018). Stochastic modelling of aircraft queues: A review. In: Proceedings of the OR60 Annual Conference, Lancaster

SKYbrary (2017) RECAT: Wake Turbulence Re-categorisation. https://www.skybrary.aero/index.php/RECAT_-_Wake_Turbulence_Re-categorisation

SKYbrary. (2019). Parallel Runway Operation. https://www.skybrary.aero/index.php/Parallel_Runway_Operation

Solnon, C., Cung, V. D., Nguyen, A., & Artigues, C. (2008). The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF 2005 challenge problem. *Europeran Journal of Operational Research, 191*(3), 912–927.

Soomer, M. J., & Franx, G. J. (2008). Scheduling aircraft landings using airlines preferences. *European Journal of Operational Research, 190*(1), 277–291.

Taillard, E. (1991). Robust taboo search for the quadratic assignment problem. *Parallel Computing, 17*(4–5), 443–455.

Thevenin, S., Zufferey, N., & Potvin, J. Y. (2018). Graph multi-coloring for a job scheduling application. *Discrete Applied Mathematics, 234,* 218–235.

Trivizas, D. A. (1998). Optimal scheduling with maximum position shift (MPS) constraints: A runway scheduling application. *Journal of Navigation, 51*(2), 250–266.

Vanwelsenaere, A., Ellerbroek, J., & Hoekstra, J.M. (2017). Analysis on the impact of pop-up flight occurrence when extending the arrival management horizon. In: Proceedings of the 12th USA/Europe Air Traffic Management Rechearch and Development Seminar, Seattle

Vié, M.S., & Zufferey, N. (2021). Real instances for aircraft landing planning. https://data.mendeley.com/datasets/rcdvm6y7sm/2

Vié, M.S., Zufferey, N., & Leus, R. (2018). Aircraft landing planning: Past, present and future. In: Proceedings of the 19th ROADEF Conference, ROADEF 2018, Lorient, France, February 21–23

Yu, S. P., Cao, X. B., & Zhang, J. (2011). A real-time schedule method for aircraft landing scheduling problem based on cellular automation. *Applied Soft Computing, 11*(4), 3485–3493.

Zufferey, N. (2012). Metaheuristics: some principles for an efficient design. *Computer Technology and Application, 3*(6), 446–462.

Zulkifli, A., Aziz, N.A.A., Aziz, N.H.A., Ibrahim, Z., & Mokhtar, N. (2018). Review on computational techniques in solving aircraft landing problem. In: Proceedings of the 2018 International Conference on Artificial Life and Robotics, Oita, Japan