ORIGINAL - OR MODELING/CASE STUDY

Robust flight schedules with stochastic programming

Sujeevraja Sanjeevi[1] · Saravanan Venkatachalam[2]

Abstract
Limiting flight delays during operations has become a critical research topic in recent years
due to their prohibitive impact on airlines, airports, and passengers. A popular strategy for
addressing this problem considers the uncertainty of day-of-operations delays and adjusts
flight schedules to accommodate them in the planning stage. In this work, we present a
stochastic programming model to account for uncertain future delays by adding buffers to
flight turnaround times in a controlled manner. Specifically, our model adds slack to flight
connection times with the objective of minimizing the expected value of the total propagated
flight delay in a schedule. We also present a parallel solution framework that integrates an
outer approximation decomposition method and column generation. Further, we demonstrate
the scalability of our approach and its effectiveness in reducing delays with an extensive
simulation study of five different flight networks using real-world data.

Keywords Airline planning · Robust scheduling · Stochastic programming · Column
generation · L-shaped method

1 Introduction

The Bureau of Transportation Statistics reports that between October 2018 and October 2019,
delays caused by late aircraft arrivals amounted to 40+ million minutes, which is 39.47%
of the total delays experienced by the flights of reporting carriers (https://www.transtats.bts.
gov/OTDelay/OTDelayCause1.asp). This highlights that operational delays are a significant
problem on both an absolute and a relative basis even today, with propagated delays being
the biggest offender. Propagated delays occur when the arriving flight for a connection is
delayed and causes a departure delay for the onward flight, kicking off a chain reaction of
delays on the aircraft's route. Such propagation is primarily due to the creation of "tight"
schedules with very limited buffers for connection times. Such schedules are created to
maximize utilization of assets such as equipment and crew (Klabjan et al. 2001). This leaves
no room for the schedule to absorb fluctuations in flight arrivals and departures, resulting in
significant delays and costs.

✉ Saravanan Venkatachalam
saravanan.v@wayne.edu

[1] Sabre, 3150 Sabre Drive, Southlake, TX 76092, USA

[2] Wayne State University, Detroit, MI, USA

The idea of making an airline schedule robust seeks to counteract this problem by adjusting the schedule to better absorb time fluctuations in aircraft arrivals and departures during operations. As robustness-based decisions need to be made much earlier than actual operational delays are known, it is necessary to consider the stochasticity of such delays. The downside of this approach is a reduction in resource utilization and an increase in planned operational costs. This creates the need for solution strategies that can balance planning and operational costs. Optimization-based approaches, which are inherently equipped with mechanisms for such balancing acts, are therefore a great fit for this problem.

Schedule robustness has been tackled in the literature from several perspectives. A two-stage stochastic programming model is proposed in Yen and Birge (2006), where crew assignments are made in the first-stage and swap opportunities are anticipated in the second-stage. Another two-stage stochastic programming model is presented in Froyland et al. (2013), where the first-stage is a tail assignment problem and the second-stage is a schedule recovery problem. This model uses penalties to minimize changes between the planning and recovery solutions. A mixed integer program (MIP) with stochastic data to minimize expected propagated delays is presented in Lan et al. (2006). The study in Marla et al. (2018) compares the performance of chance-constrained programming, robust optimization, and stochastic optimization approaches using a solution space similar to the one in the model presented in Lan et al. (2006). Methodologies to solve integrated aircraft routing and crew pairing problems to reduce uncertain propagated delays are considered in Yen and Birge (2006), Dunbar et al. (2012), Dunbar et al. (2014). More recently, the robust optimization approach presented in Yan and Kung (2016) uses column and row generation to solve a routing problem with delays coming from a bounded uncertainty set by minimizing worst-case propagated delay costs. An alternate perspective in Ahmadbeygi et al. (2010), Chiraphadhanakul and Barnhart (2013) retains a given planned routing but re-times flights in order to add time buffers or "connection slacks" to flight connections that are likely to be missed. Other related work can also be found in Arıkan et al. (2013), Kang (2004), Rosenberger et al. (2004), Shebalov and Klabjan (2006), Talluri (1996), Weide et al. (2010).

To motivate our research, we present some concerns we observed with scheduled robustness models proposed so far. First, there is no clear differentiation between the cost of rescheduling flights a few weeks before the day-of-operations versus delaying them a few hours before departure. This difference can be significant in practice. Second, the stochastic programming approaches proposed in literature use very complex first-stage models with a wide variety of first-stage decisions. This may be undesirable, as each adjustment of a schedule can affect other operational considerations such as staff scheduling, maintenance scheduling, crew and passenger connectivity, among others. Also, there is no clarity on how to reduce the scope of such models while still generating useful results for scheduling practitioners. Computationally, the size and complexity of first-stage models proposed in literature makes it difficult to scale them and use them for real-world airline schedules.

In this research, we seek to fill the aforementioned gaps in literature. Our main contributions are (i) a two-stage stochastic programming model that re-times flights of a given schedule in a controlled manner while minimizing the sum of first-stage rescheduling costs and expected cost of propagated delays on the day of operations; (ii) a parallel decomposition framework based on the L-shaped method (Van Slyke and Wets 1969) that uses column generation to solve recourse second-stage models; (iii) extensive computational study using disruptions caused by randomly generated flight delays that show a significant reduction in propagated delays in schedules adjusted by the proposed model; and (iv) recommendations and insights that can boost the performance of decomposition techniques like Benders decomposition and column generation for flight schedule models. The proposed model and

solution framework allow to solve much larger instances than those solved so far in literature. For example, one of the networks we consider has 324 flights and 71 aircraft, much larger in size than networks used in recent works like (Froyland et al. 2013; Yan and Kung 2016). Furthermore, we use a dynamic delay approach similar to Yan and Kung (2016) to solve our recourse problems. This approach uses the least required delay on each flight while building paths. This eliminates the need for discrete delay copies which can generate unnecessary flight delays due to discretization and cause significant run time increases (see Figure 7 in Froyland et al. (2013)). The path-based recourse formulation in our model can be easily extended to incorporate requirements from other operational domains of airlines. This includes hard constraints like minimum crew/passenger connection times and soft requirements like the loss of passenger goodwill that can be incorporated into path costs.

The remainder of this paper is organized as follows. In Sect. 2, we present a two-stage stochastic programming formulation to minimize the expected value of propagated delays, along with a simpler mixed integer programming formulation based on sample mean values of primary delays. In Sect. 3, we describe a column-generation procedure for recourse problems and the L-shaped algorithm for the complete two-stage problem. In Sect. 4, we report the results of extensive computational studies that highlight the qualitative and quantitative benefits of our approach. In Sect. 5, we conclude the article with a summary and discussion of future research directions.

2 Stochastic delay models

In this section, we present our two-stage stochastic programming formulation of the delay mitigation problem. We also present an alternate approach that we use to benchmark our computational results. The latter approach is based on an MIP model that uses the mean values of individual flight delays. We begin by introducing the required notation.

Given a valid flight schedule, we model it as a connection network on a directed acyclic graph $G = (F, A)$ in which the set of nodes F represent flights and the arcs A represent flight connections. A connection (i, j) is valid if and only if (i) the incoming arrival and outgoing departure airports match, and (ii) the connection slack s_{ij}, defined as the difference between the departure time of the outgoing flight j and the arrival time plus the turnaround time of the incoming flight i, is non-negative. The set A contains only valid connections.

Our modeling of uncertain flight delays is similar to that in Lan et al. (2006), Dunbar et al. (2012), Yan and Kung (2016). A flight can experience *primary delays* that are independent of routing and rescheduling, and *propagated delays* that are caused by upstream flights on that flight's route. Let ω be a random variable representing a delay scenario, and let Ω be a finite set of delay scenarios. Let pd_f^ω be the realized non-negative integer-valued primary delay in minutes experienced by flight $f \in F$ in scenario $\omega \in \Omega$. Let R^ω be the set of possible routes in scenario ω. For any route $r \in R^\omega$ and connection (i, j) in r, the parameter d_{rj} representing the delay propagated to the outgoing flight j by the connection is defined as:

$$d_{rj} = max(0, d_{ri} + pd_i^\omega - s_{ij}). \tag{1}$$

2.1 Two-stage model

Let $x_f \geq 0$ be an integer decision variable representing the number of minutes by which flight $f \in F$ needs to be rescheduled, and let c_f, $f \in F$, be the per-minute reschedule cost. The formulation of the two-stage model (TSM) can then be stated as:

$$(TSM) \quad \text{Minimize} \quad \sum_{f \in F} c_f x_f + \mathbb{E}_\Omega[\phi(x, \tilde{\omega})]$$

$$\text{s.t.} \qquad x_i \leq s_{ij} + x_j, \qquad\qquad (i, j) \in A^{orig}, \qquad (2)$$

$$\sum_{f \in F} x_f \leq B, \qquad\qquad\qquad (3)$$

$$x_f \in \mathbb{Z} \cap [0, l], \qquad\qquad f \in F. \qquad (4)$$

The objective of this model is to minimize the sum of the total reschedule cost and the expected flight delay costs. Constraints (2) protect the time connectivity for all connections in the original routing $A^{orig} \subseteq A$. Constraints (3) provide a control factor in the form of a time budget B that limits the total reschedule time. We also limit the x_f values with a fixed bound l to prevent exorbitant reschedules of individual flights. Given a reschedule x and the scenario probabilities $p_\omega, \omega \in \Omega$, the expected value $\mathbb{E}_\Omega[\phi(x, \omega)] = \sum_{\omega \in \Omega} p_\omega \phi(x, \omega)$ can be computed by solving the following set partitioning model for each scenario $\omega \in \Omega$, which is the second-stage formulation for a given x and scenario ω:

$$\phi(x, \omega) = \text{Min} \sum_{f \in F} e_f z_f^\omega$$

$$\text{s.t.} \quad \sum_{r \in R^\omega} a_{rt} y_r = 1, \qquad\qquad t \in T, \qquad (5)$$

$$\sum_{r \in R^\omega} b_{rf} y_r = 1, \qquad\qquad f \in F, \qquad (6)$$

$$\sum_{r \in R^\omega} b_{rf} d_{rf} y_r - x_f \leq z_f^\omega, \qquad\qquad f \in F, \qquad (7)$$

$$z_f^\omega \geq 0, \ f \in F, \ y_r \in \{0, 1\}, \ r \in R^\omega. \qquad (8)$$

The second-stage model minimizes the propagated delay costs incurred in scenario $\omega \in \Omega$ computed as per-minute costs e_f for each flight f. It uses two sets of decision variables: continuous variables z_f^ω that represent the excess delay propagated to each flight $f \in F$ and binary variables y_r that take the value 1 to indicate the selection of the route $r \in R^\omega$. The parameters a_{rt} and b_{rf} are binary and respectively indicate whether route r is for the tail t and whether it contains flight f. Constraints (5) and (6) enforce the assignment of one route per aircraft and one route per flight. Constraints (7) are linking constraints that capture the excess propagated delay that has not been accounted for by the first-stage rescheduling.

Next, we present an MIP formulation that reschedules flights based on the average values of the primary delays. This model is used in the comparative studies presented in the computational results section.

2.2 Mean delay model

Let $\bar{\omega}$ be the scenario in which each flight experiences the mean primary delay across all scenarios in Ω, i.e., $d_f^{\bar{\omega}} = \sum_{\omega \in \Omega} p_\omega d_f^\omega$ for $f \in F$. The mean delay model aims to reschedule flights to accommodate the average delay scenario $\bar{\omega}$ without changing the original routing. To simplify the notation, we set $d_f^{\bar{\omega}}$ to be the delay propagated to flight f in scenario $\bar{\omega}$ in the original schedule. The mean delay model can be stated as follows:

$$\text{Minimize} \sum_{f \in F} \left(c_f x_f + e_f z_f^{\tilde{\omega}} \right)$$

$$
\begin{aligned}
\text{s.t.} \quad & x_i \leq s_{ij} + x_j, & (i, j) \in A^{orig}, \\
& \sum_{f \in F} x_f \leq B, & \\
& d_f^{\tilde{\omega}} - x_f \leq z_f^{\tilde{\omega}}, & f \in F, \\
& z_f^{\tilde{\omega}} \geq 0, \, x_f \in \mathbb{Z} \cap [0, l], & f \in F.
\end{aligned}
$$

The objective function minimizes the total reschedule and delay costs, with the latter carrying a higher penalty. The first two sets of constraints are the first-stage constraints (2) and (3). The third set of constraints is obtained from (7) by selecting only the original route for each aircraft.

3 Solution approach

In this section, we present our solution framework that uses the L-shaped method in Van Slyke and Wets (1969) to solve the TSM. We first present details about how we solve the recourse problems of the TSM.

3.1 Column-generation framework

Solving the TSM using the L-shaped method requires computing $\phi_{LP}(\bar{x}, \omega)$, the solutions to linear programming (LP) relaxations of the recourse models for any fixed first-stage solution $\bar{x}$. For a given scenario ω, we use a column-generation approach to generate the required routes. We iterate between solving a version of the recourse problem restricted to a subset of routes $\tilde{R} \subseteq R^\omega$ and solving a pricing problem to find new routes that can improve the solution. Optimality of the linear program can be declared when no such route can be found. For ease of exposition, we state here the dual formulation of the recourse problem in full. Let μ_t and ν_f be unbounded dual variables for the coverage constraints (5) and (6) for a scenario ω. Given a first-stage solution $\bar{x}$, we write the constraints (7) as

$$z_f^\omega - \sum_{r \in R^\omega} b_{rf} d_{rf} y_r \geq -\bar{x}_f, \; f \in F,$$

and we let π_f be the non-negative dual variables for these constraints. Let $a(r) \in T$ be the aircraft for which the route $r \in R^\omega$ was generated. Using this notation, the dual formulation can be written as:

$$\text{Maximize} \sum_{t \in T} \mu_t + \sum_{f \in F} \left(\nu_f - \bar{x}_f \pi_f \right)$$

$$
\begin{aligned}
\text{s.t.} \quad & \mu_{a(r)} + \sum_{f \in F} b_{rf} \left(\nu_f - d_{rf} \pi_f \right) \leq 0, & r \in R^\omega, \\
& \mu_t \text{ free}, & t \in T, \\
& \nu_f \text{ free}, \, 0 \leq \pi_f \leq e_f, & f \in F. \quad (9)
\end{aligned}
$$

Our column-generation procedure begins by solving the LP relaxation of the recourse problem with a subset $\tilde{R}$ of routes. One way to initialize $\tilde{R}$ is with routes of the original

schedule that have delays propagated sufficiently enough to protect minimum turnaround times. With the dual solution of this restricted problem, a pricing problem is solved to find columns with the least reduced cost rc_r, where

$$rc_r = \sum_{f \in F} b_{rf} \left(d_{rf} \pi_f - \nu_f \right) - \mu_{a(r)}. \tag{10}$$

The dual formulation provides some intuition for rc_r; we want routes that violate the constraints (9). Once such a route is found, it is added to $\tilde{R}$ and we repeat the above steps. If no such route can be found, optimality can be declared. As there are potentially a large number of pricing problems to be solved, it is critical to determine the useful routes quickly. Next, we present our version of the labeling algorithm, an extension of the algorithm presented in Dunbar et al. (2012), Yan and Kung (2016), which we use to solve this problem.

3.2 Pricing problem

We solve the pricing problem by searching for routes in the graph G with negative values for the reduced cost as defined in (10). As we assume that the original schedule is already available, the airports from which each aircraft should depart at the beginning of the schedule and at which it should arrive at the end of the schedule are fixed. To reflect this, we introduce separate source and sink nodes for each aircraft and separately search for candidate routes for each aircraft. This approach is quite practical, as it can easily be extended to consider aircraft-specific business constraints during route generation. Each aircraft's source node connects only to flights departing from the aircraft's initial departure airport. Similar restrictions apply to sink nodes based on final arrival airports.

To search for candidate routes, we use a label-setting algorithm similar to the one proposed in Dunbar et al. (2012), Yan and Kung (2016). This algorithm relies on building *labels* that represent partial routes and extending them along valid flight connections given by A to generate full routes from the source to the sink. The combinatorial explosion in the number of routes is controlled using the notion of *dominance* between labels. More formally, each label l denotes a partial path stored in a tuple $(f_l, pred_l, red_l, prop_l)$, where $f_l \in F$ is the last flight on the path, $pred_l$ is the label from which l was extended, red_l is the reduced cost accumulated so far, and $prop_l$ is the delay propagated to f_l on the partial route corresponding to l. Note that $pred_l$ is empty for labels at source nodes. When a label u is extended with a connection $(f_u, f') \in A$, the algorithm generates a new label $v = (f', u, red_v, prop_v)$ in which red_v and $prop_v$ are updated using (1) and (10), respectively. Once a label is extended to the sink node, the route that it corresponds to becomes a full route and can be obtained by traversing backward along the chain of predecessors.

Definition 1 (*Label dominance condition*) Let u and v be two labels with $f_u = f_v$. The label u dominates v if (i) $red_u \leq red_v$, (ii) $prop_u \leq prop_v$, and at least one of the inequalities is strict.

Given two labels u and v, if we know that any feasible extension of v is also feasible for u, any route that can be generated by successively extending v to the sink can also be generated by u, meaning that we can safely ignore v. This was proved in Lemma 1 in Yan and Kung (2016). For clarity, we restate the lemma here using the notation of the present article:

Lemma 1 *Let u and v be labels such that u dominates v. If u' and v' are labels obtained by extending u and v with a connection $(f_u, f') \in A$, then u' dominates v'.*

Lemma 1 allows us to store and extend only non-dominated labels at each node and thus implicitly remove large numbers of candidate paths from consideration. We have observed that the label-setting algorithm in Yan and Kung (2016) provides at most one negative reduced-cost route in each iteration. As any route with a negative reduced cost is likely to improve the recourse solution, we enhance the algorithm by considering three possible alternatives for generating multiple negative reduced-cost columns:

(i) *All paths* Store and return all negative reduced-cost paths.
(ii) *Best paths* Store all negative reduced-cost paths, but return only the N most negative reduced-cost paths.
(iii) *First paths* Stop the search as soon as N negative reduced-cost paths are found, and return them.

We found that all three strategies produce a significant speedup over generating a single path per pricing problem. Among the three, the "first paths" strategy gave us the best runtime with $N=10$. We present a more detailed comparative study of these strategies in the computational results section. We present the label-setting algorithm of Dunbar et al. (2012), Yan and Kung (2016) with our enhancements below, in Algorithm 1. As the original initial-departure and final-arrival airports can be different for each aircraft, the algorithm is used to separately generate routes for each aircraft. The input includes augmented sets of nodes F' and arcs A'; $F' = F \cup \{so, si\}$, where so and si are dummy source and sink nodes, respectively, and A' contains all eligible connections in A, connections from so to every valid first flight in F, and connections from every valid last flight to si for the selected aircraft. The output of the algorithm is a set of negative reduced-cost columns for the selected aircraft.

Algorithm 1 Label-setting algorithm

---

function GENERATECOLUMNS(F', A', so, si)
    $M_f \leftarrow \emptyset$, $f \in F'$.   ▷ Processed labels container
    $I_f \leftarrow \emptyset$, $f \in F'$.   ▷ All labels container
    $I_{so} \leftarrow \{(so, \emptyset, -\mu^\omega_{a(r)}, 0)\}$.   ▷ Source label creation
    while $\bigcup_{i \in F'}(I_i \setminus M_i) \neq \emptyset$ and SHOULDSTOP(I_{so}) $\neq true$ **do**
        Choose $i \in F'$ and a label $l \in I_i \setminus M_i$ with a minimal reduced cost.
        for $(i, j) \in A'$ **do**
            $l' \leftarrow$ EXTEND(l, j).
            if l' is not dominated by any label in I_j **then**
                $I_j \leftarrow I_j \cup \{r(i), j\}$.
            end if
            if $j = si$ and SHOULDSTOP(I_{so}) = $true$ **then**
                break.   ▷ Stop processing labels
            end if
        end for
        $M_i \leftarrow M_i \cup \{l\}$.
    end while
    return BUILDCOLUMNSFROMLABELS(I_{so})
end function

Algorithm 1 initializes a single label at the source node as $(so, \emptyset, -\mu^\omega_{a(r)}, 0)$, without a predecessor. Given a label $l = (i, pred_l, red_l, prop_l)$ and a connection (i, j), the EXTEND procedure creates a new label l' at node j by updating $prop_{l'}$ using (1) and the reduced cost $red_{l'} = red_l + d_j\pi_j - \nu_j$, as obtained from (10). Labels become complete when they are extended to si. The implementation of SHOULDSTOP depends on the column-generation

strategy that is used. It always returns *false* for the all-paths and best-paths strategies. For the first-paths strategy, it returns *true* if the number of negative reduced-cost labels at *si* have exceeded N, and *false* otherwise. When the while loop ends, the BUILDCOLUMNS-FROMLABELS procedure builds columns using negative reduced-cost labels at *si*. It returns all columns for the all-paths strategy, and the N most negative reduced-cost columns for the other two strategies. The LP solution to the recourse problem is optimal if Algorithm 1 returns an empty set.

3.3 Solution framework for the TSM

Now that we have established the machinery to solve recourse models, we are ready to present the L-shaped method to solve the TSM. The method has two variants: a single-cut and a multi-cut version. We present the multi-cut method here and show later in this section how it can be modified to obtain the single-cut method. The multi-cut L-shaped method works with the following approximation of the TSM:

$$(MP) \quad \text{Minimize} \sum_{f \in F} c_f x_f + \sum_{\omega \in \Omega} \eta^\omega$$
$$\text{s.t.} \quad (2)-(4),$$
$$\eta^\omega \text{ free}, \omega \in \Omega.$$

We refer to this version of the formulation as the "master problem" (MP). Our solution procedure iterates between solving the MP and the recourse LP problems. Solutions to the latter can provide *optimality cuts* that bound η from below or *feasibility cuts* generated from infeasible recourse problems. As we can always get a feasible solution for any delay scenario by propagating delays along the original routing, our recourse problems are always feasible. So we only need to consider optimality cuts. To describe these cuts, we introduce the following additional notation for each scenario $\omega \in \Omega$:

$$\alpha^\omega = p_\omega \left(\sum_{t \in T} \mu_t + \sum_{f \in F} \nu_f \right), \text{ and } \beta_f^\omega = p_\omega \pi_f, \ f \in F.$$

Using this notation, the multi-cut procedure is presented in Algorithm 2. We found that $x_f^0 = 0, f \in F$ is a reasonable starting solution. The parameter `MaxNumIterations` provides a practical way to limit the algorithm's runtime. To convert the algorithm into the single-cut L-shaped method, we use a single variable η in the MP and add only the single cut (11) that is computed using the optimal dual values of all recourse problems in each iteration:

$$\eta \geq \sum_{\omega \in \Omega} \alpha^\omega - \sum_{f \in F} \left(\sum_{\omega \in \Omega} \beta_f^\omega \right) x_f. \tag{11}$$

We note here that the Benders cuts are valid only when the binary restrictions of the second-stage problems are relaxed. Making our approach exact requires embedding Algorithm 2 in a branch-and-bound scheme that finds integer solutions to all second-stage y_r variables. However, as we found that most of the optimality gap was closed in the root node, we did not explore branching. As we shall see in Section 4, even these solutions can provide rescheduling values that significantly improve the preparedness of a schedule for uncertain delays.

Algorithm 2 Multi-cut L-shaped method for the SM

Solve the MP without η^ω variables to get an initial solution x^0.
Add η^ω variables to the MP.
Set $UB \leftarrow \infty$, $LB \leftarrow -\infty$, $k \leftarrow 0$, $x^* \leftarrow x^0$.
while $UB - LB > \epsilon$ and $k \leq$ `MaxNumIterations` **do**
    for each scenario $\omega \in \Omega$ **do**
        Find $\phi_{LP}(x^k, \omega)$ using column generation.
        Compute β^ω, α^ω using optimal dual values.
        Add cut $\eta^\omega \geq \alpha^\omega - \sum_{f \in F} \beta_f^\omega x_f$ to the MP.
    end for
    Set $UB \leftarrow min\left(UB, \sum_{f \in F} c_f x_f^k + \sum_{\omega \in \Omega} \phi_{LP}(x^k, \omega)\right)$.
    if UB changed **then**
        Update incumbent solution $x^* \leftarrow x^k$.
    end if
    Solve the updated MP to get the objective value obj_k.
    Set $LB \leftarrow max(LB, obj_k)$, $k \leftarrow k + 1$.
end whilereturn x^*.

Table 1 Instance details

Instance	Number of flights	Number of aircraft	Number of paths
s1	210	41	48,674
s2	248	67	20,908
s3	112	17	39,242
s4	110	17	56,175
s5	80	13	190,540
s6	324	71	113,892

4 Computational experiments

In this section, we demonstrate the efficacy of our proposed formulation and solution approach using real-world data for five flight networks. We used Java for the implementation, with CPLEX 12.9 as the solver. The experiments were conducted on an Intel(R) Xeon(R) CPU E5-2640 computer with 16 logical cores and 80 GB RAM. We implemented parallel processing using the thread-safe Java "actors" provided by the Akka actor library (available at https://akka.io). All code and data used for our experiments is publicly available at https://github.com/sujeevraja/stochastic-flight-scheduler.

4.1 Network data and experiment setup

Table 1 presents details about the flight networks we used. Each network is based on daily schedules of two different airlines on different days in early 2017, and is the planned schedule for a single equipment type. We avoid solving multiple equipment types together as such swaps can cause operational issues like unfilled seats or passenger spillage. Each flight in our data has a minimum turnaround time that applies to connecting flights departing after the arrival of the flight. As the costing data for our networks is quite complex, we simplify the calculations with a first-stage reschedule cost of one per minute and a recourse delay cost of 10 per minute for each flight. This costing serves to encode the significant increase of costs incurred by operational delays as opposed to planned reschedules. The "Number of paths" values are the maximum number of paths that can be built during column generation.

To calculate them, we build a flight network and add a dummy source and dummy sink node for each aircraft based on its original first-departure and last-arrival stations. We then add dummy source arcs to flights departing from the source node station and dummy sink arcs from flights arriving at the sink node station. The number of paths for each aircraft is recursively computed as the number of paths from the aircraft's dummy source to the aircraft's dummy sink. The total number of paths is the sum of paths of all aircraft.

We simulate primary delays by constructing 30 randomly generated delay scenarios for each run. The scenarios are generated by varying two parameters: the distribution used for delay generation and the flights that experience primary delays. We follow the recommendation of Yan and Kung (2016) in using truncated normal, gamma, and log normal distributions for primary delays, with log normal being the default. We select flights that experience primary delays using two strategies, which we call "hub" and "rush". The hub strategy selects flights from a hub, which we define as the airport with the most departures in a given schedule. The rush strategy calculates the duration between the earliest departure and the latest arrival for a schedule and selects flights departing during the first quarter of the window. This idea stems from the morning runway congestion that frequently occurs in most airports. Our model limits first-stage rescheduling with two control factors, an individual limit of l for each flight and a limit of B minutes on the total delay. We fix l to 30 minutes in all of our runs. We make B adaptive to the problem data by computing the total primary flight delay for each recourse scenario, taking the average of these values, and allowing B to be a fraction of the average total primary delay. Unless specified otherwise, we default to 0.5 for B, $LogNormal(15, 15)$ as the delay distribution, "hub" as the flight selection strategy, the multi-cut L-shaped method, the first-paths column generation strategy outlined in Sect. 3.2, and use 30 threads to solve 30 second-stage problems in parallel. Solution times in all tables are reported in seconds.

4.2 Results and insights

Our computational study contains three sets of results. The first set presents the performance metrics of our algorithm, as shown in Table 2. The *Strategy* column shows the strategy we use to select flights, as explained above. We report two gaps: the percentage gap computed as $100 \times (UB - LB)/UB$ from Algorithm 2 in the Gap column, and the optimality gap of the solution in the *Opt Gap* column. To compute the latter, we first find an upper bound **ub** by fixing the first-stage reschedule values to the solution found by Algorithm 2, solving all second-stage problems without relaxing the binary restrictions, and computing the objective value as the sum of the fixed reschedule cost and the mean value of the second-stage delay costs. As the objective value of the solution found by Algorithm 2 is a lower bound (denoted by **lb**) for the optimal solution, we report the optimality gap as $100 \times$ **(ub-lb)/ub**. The columns *Cuts* and *Iter* report the number of Benders cuts added and the number of iterations, respectively. The main takeaways from Table 1 are that the Benders gap is almost completely closed for most instances and that the root node closes more than 90% of the optimality gap. We believe that the low optimality gap occurs because of the set partitioning structure in the second-stage model in TSM. As set partitioning models are known to have a property called *quasi-integrality* (Balas and Padberg 1975, 1972; Tahir et al. 2019), their linear relaxations typically yield integer solutions in most cases.

For the second set of experiments, we report solution quality results in Figs. 1, 2, 3, and 4. Numbers used for these figures are available in the Appendix in Tables 7, 8, 9 and 10 respectively. In these experiments, we first randomly generate 30 delay scenarios and use

Table 2 Solution quality and performance

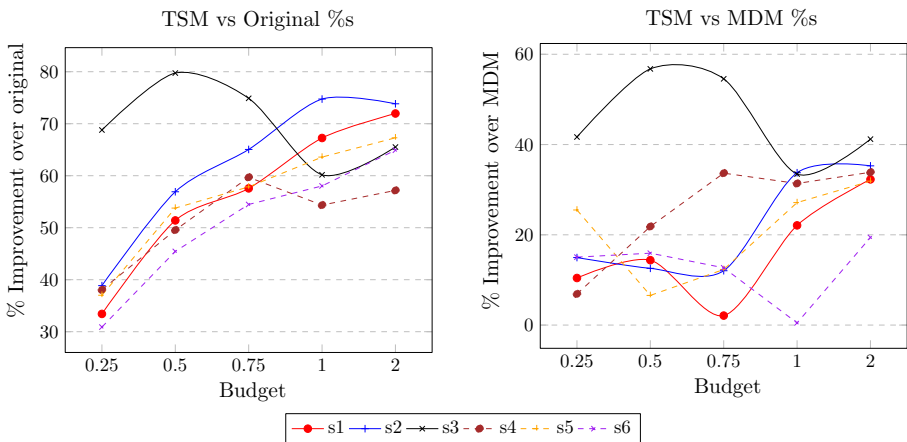| Strategy	Instance	Time	Gap (%)	Opt gap (%)	Cuts	Iter
Hub	s1	78.42	0.35	3.42	886	30
	s2	53.94	2	3.87	900	30
	s3	15.94	0	0	93	6
	s4	14.04	0.05	7.61	304	15
	s5	73.16	0	6.18	352	16
	s6	377.3	3.54	11.85	900	30
Rush	s1	90.64	0.09	7.52	861	30
	s2	71.07	0.5	7.94	888	30
	s3	11.73	0.03	8.75	79	4
	s4	6.37	0	0.41	115	6
	s5	47.92	0	0.09	188	8
	s6	144.34	0.04	1.82	302	13

Fig. 1 Illustration of performance of TSM by budget

this data to solve the two-stage and mean delay models. The same scenarios are used for both models for a fair comparison. Next, we generate a new set of 100 random delay scenarios different from those used for solving. For each new scenario, we compute the total propagated delay incurred by three variants of the original schedule: (i) no adjustments, (ii) adjustments based on the reschedule solution of the mean delay model, and (iii) adjustments based on the reschedule solution of the TSM. By "adjustment", we mean that the departure time of a flight is changed based on its corresponding reschedule value. The propagated delay for any scenario, measured in minutes, is found by solving the integer-valued recourse model to optimality. We then take the average value of the total propagated delay of the 100 scenarios as a comparison metric for the three approaches. Each set of Figs. 1, 2, 3, and 4 has two charts that measure the relative reduction in the average value of total propagated delay when using the TSM compared with that of the original schedule and MDM solution.

To study the quality of the solution over the entire parameter space, we vary one parameter in each figure that reports propagated delay comparisons. Figure 1 reports
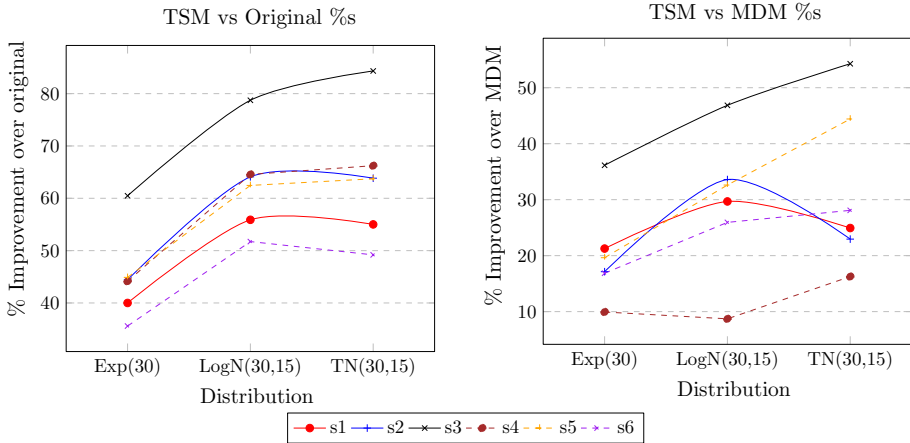
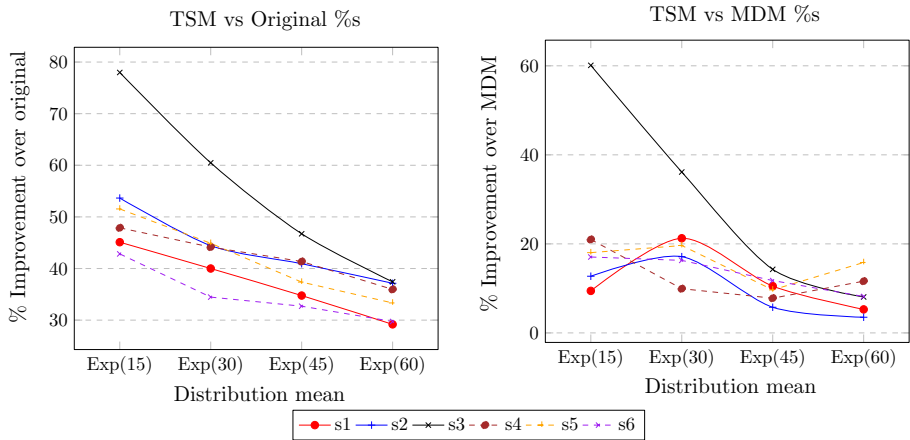**Fig. 2** Illustration of performance of TSM by distribution

Fig. 3 Illustration of performance of TSM by distribution mean

a comparison for the reschedule budget fractions in {0.25, 0.5, 0.75, 1, 2}. Given a budget fraction, the corresponding reschedule budget is computed by multiplying the average value of the total primary flight delay of each of the 30 recourse scenarios with the budget fraction value. Figure 3 reports comparisons for distributions in $\{Exponential(30), LogNormal(30, 15), TruncatedNormal(30, 15)\}$. Figure 3 fixes the distribution as exponential and reports comparisons for mean values of {15, 30, 45, 60} minutes. Figure 4 reports comparisons for the number of training scenarios in {10,20,30,40,50}. These figures show that the reduction in propagated delay achieved with TSM is significantly better than the original schedule and mean delay model, and that this reduction is agnostic of the underlying data.

In addition to the data-related parameters discussed so far, our approach also has several technical parameters, such as the type of column-generation strategy and the use of single versus multiple cuts for the L-shaped method. We use our final set of experiments to empir-
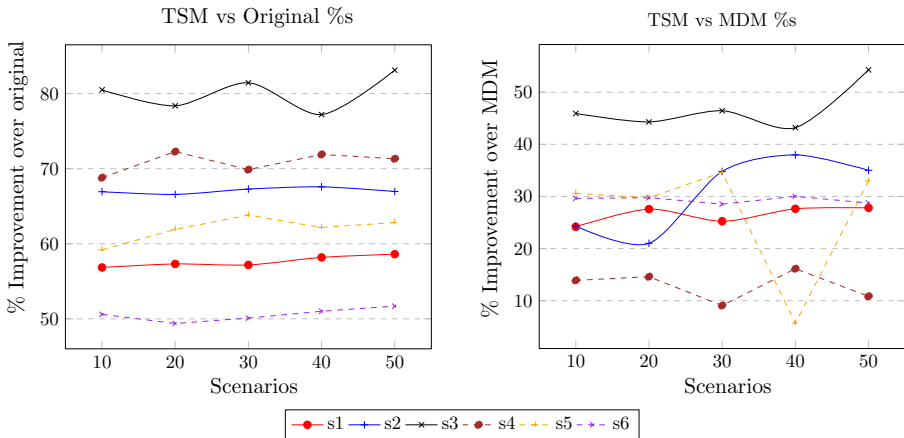
Fig. 4 Illustration of performance of TSM by scenarios

Table 3 Runtime comparison for column-generation strategies

Instance	Enumeration	All paths	Best paths	First paths
s1	958.58	112.33	75.61	77.45
s2	161.19	63.45	47.46	49.87
s3	170.61	19.64	9.87	9.49
s4	417.46	28.32	15.2	14.28
s5	3086.92	121.61	65.81	69.34
s6	1860.32	1854.57	461.2	524.48

ically select a set of these parameters that give the best runtime performance. The results are reported in Tables 3, 4, 5, and 6.

We obtain the values for each row in these tables as follows. First, we generate 30 random delay scenarios using the default parameters specified in Sect. 4.1. Then we run Algorithm 2 for each value of the tested parameter and collect the solution time. We smooth out aberrations by repeating this 5 times and reporting the average of these values as the time. The same procedure applies for values other than the solution time reported in Table 5.

Table 3 reports a comparison between the different column-generation strategies presented in Sect. 3.2. In this test, the first-paths and best-paths strategies are run with $N=10$, i.e., by selecting the first 10 and the 10 most negative reduced-cost columns, respectively. The results reported in this table are in line with the intuition that enumerating all columns should take much longer than using a delayed column-generation procedure with pricing. Among the pricing strategies, the best-paths and first-paths strategies are both clearly better than the all-paths strategy, which adds all negative reduced-cost columns to the restricted recourse problems.

Table 4 reports a run-time comparison with an increase in the number of threads. While it is indeed true that parallel solving should be faster, it is not practically obvious that this should be true. Specifically, we expected that the performance should stagnate or worsen when the number of threads exceeds the number of logical cores, but Table 4 shows that this is not the case. Though the gain in performance declines with increasing threads, on an absolute basis, increasing the number of threads up to 30 seems to improve the overall

Table 4 Runtime comparison for multiple threads

Instance	Number of parallel solvers			
	1	10	20	30
s1	692.71	123.49	98.63	77.88
s2	402.31	74.53	60.54	48.52
s3	64.64	12.58	10.16	8
s4	117.12	22.5	18.53	14.4
s5	607.55	104.76	88.55	74.04
s6	6490.08	986.12	736.96	519.8

Table 5 Comparison of single- vs multi-cut L-shaped method

Instance	Multi-cut				Single-cut			
	Time	Gap	Cuts	Iter	Time	Gap	Cuts	Iter
s1	686.81	0.4	883.4	30	708.91	24.28	30	30
s2	406.39	2.51	899.8	30	455.63	33.08	30	30
s3	58.16	0	85	4.8	214.2	0	19.4	19.6
s4	105.99	0.01	304.6	14	223.77	11.88	30	30
s5	579.02	0.03	340	14.4	1172.27	12.36	30	30
s6	6488.77	3.87	900	30	6724.13	19.02	30	30

Table 6 Runtime comparison for caching columns between iterations

Instance	Caching	No caching
s1	686.78	715.77
s2	399.28	422.96
s3	62.31	61.52
s4	112.8	105.6
s5	615.87	585.76
s6	6372.9	6198.92

runtime. Increases beyond this are not helpful, as the maximum number of problems that can be solved in parallel is the number of recourse problems, which is 30. Table 5 reports a runtime comparison between the single- and multi-cut versions of Algorithm 2. Clearly, the multi-cut version is better than the single-cut version in terms of the solution time, the Benders percentage gap (reported in the *Gap* column), and the number of iterations. As the memory used to store and add cuts is minuscule in comparison to the rest of the data, the greater number of cuts in the multi-cut version does not affect performance at all. In Table 6, we present the results of caching the columns between the iterations for Algorithm 2. We noticed that the columns generated in an iteration of the L-shaped method require only flight data and propagated delay data, and are unaffected by changes in the first-stage reschedule solution. This allows them to be cached and reused in future iterations, which in turn allows pricing problems to be warm-started with promising columns. As Table 6 indicates, we were not able to find a clear advantage of this approach. While we certainly do not discard this idea, we recommend against using it, based purely on an ease-of-implementation perspective.

Table 7 Total propagated delay improvements for different budgets

Budget fraction	Average total propagated delay					
	Instance	Original	MDM	TSM	RR over original (%)	RR over MDM (%)
0.25	s1	845	628.06	562.57	33.42	10.43
	s2	850.82	611.65	520.17	38.86	14.96
	s3	50.24	26.88	15.68	68.79	41.67
	s4	219.37	145.93	135.86	38.07	6.9
	s5	254.29	215.02	160.18	37.01	25.5
	s6	1221.54	993.74	844.16	30.89	15.05
0.5	s1	836.37	474.79	406.51	51.4	14.38
	s2	844.62	416.29	363.95	56.91	12.57
	s3	42.45	19.89	8.6	79.74	56.76
	s4	232.55	150.1	117.32	49.55	21.84
	s5	250.1	123.74	115.61	53.77	6.57
	s6	1231.86	799.37	672.06	45.44	15.93
0.75	s1	861.65	373.57	365.71	57.56	2.1
	s2	868.94	345.26	303.68	65.05	12.04
	s3	46.81	25.88	11.76	74.88	54.56
	s4	218.15	132.55	87.93	59.69	33.66
	s5	242.06	116.37	102.03	57.85	12.32
	s6	1244.04	648.78	566.47	54.47	12.69
1	s1	832.36	349.93	272.63	67.25	22.09
	s2	829.33	316.21	209.45	74.74	33.76
	s3	49.48	29.62	19.71	60.17	33.46
	s4	233.37	155.23	106.54	54.35	31.37
	s5	246.86	123.38	89.9	63.58	27.14
	s6	1197.72	505.05	502.65	58.03	0.48
2	s1	849.18	351.68	238.15	71.96	32.28
	s2	851.63	344.38	222.88	73.83	35.28
	s3	49.12	28.81	16.94	65.51	41.2
	s4	222.53	144.08	95.3	57.17	33.86
	s5	243.47	116.92	79.63	67.29	31.89
	s6	1237.37	538.89	434.22	64.91	19.42

5 Conclusions and future research

In this research, we present a two-stage stochastic programming model that adds time buffers to flight connections in order to make a schedule more robust to uncertain delays. By "robust", we mean that the schedule is more accommodating to changes in scheduled times and has fewer delays propagated to downstream flights. To solve the two-stage model, we present a solution framework that combines an outer approximation method with a delayed column-generation routine. We conduct a thorough qualitative and quantitative analysis of the proposed framework and report extensive computational results. To efficiently solve

Table 8 Total propagated delay improvements for different distributions

Distribution	Average total propagated delay					
	Instance	Original	MDM	TSM	RR over original (%)	RR over MDM (%)
Exp(30)	s1	2050.08	1562.11	1230.08	40	21.26
	s2	1993.59	1336.85	1107.84	44.43	17.13
	s3	141.43	87.52	55.89	60.48	36.14
	s4	701.25	434.87	391.68	44.15	9.93
	s5	599.99	411.45	330.68	44.89	19.63
	s6	3281.99	2542.58	2113.81	35.59	16.86
LogNormal(30,15)	s1	1966.24	1233.31	867.31	55.89	29.68
	s2	1849.07	999.45	663.77	64.1	33.59
	s3	116.12	46.47	24.7	78.73	46.85
	s4	575.49	223.43	203.98	64.56	8.71
	s5	580.18	378.54	210.44	63.73	44.41
	s6	3187.22	2251.7	1619.3	49.19	28.09
TruncNormal(30,15)	s1	2008.96	1204.15	903.91	55.01	24.93
	s2	1919.41	900.75	693.00	63.84	22.95
	s3	115.87	39.72	18.16	84.33	54.28
	s4	615.21	238.11	207.77	66.23	16.26
	s5	580.18	378.54	210.44	63.73	44.41
	s6	3187.22	2251.7	1619.3	49.19	28.09

large-scale instances of the model, we adopt various software engineering techniques such as caching and parallelism. Our results highlight that the operational delay reduction can be significant using our proposed methodology compared to a deterministic approach.

There are several interesting directions for extending this work, and we highlight a few here. First, the model can be made into a closer approximation of reality by considering more business constraints such as maintenance events and crew-friendliness. Another direction would be to study the scalability of our approach when more complex modifications such as cancellations, diversions, and overbooking are allowed in the first-stage. We have observed that, in practice, strategies to minimize delays can be quite diverse. While some airlines want to spread out delays among several flights to prohibit exorbitant delays for a single flight, other airlines want almost the exact opposite with the idea of minimizing the number of flights with delays. Making our model flexible enough to allow such variety in rescheduling and delay strategies is a worthwhile idea to pursue in the future. Also, from a modelling perspective, appropriate risk-averse objectives other than the risk-neutral expectation function can be evaluated in the second-stage.

Appendix

In this section, we report numbers used for the charts in Figs. 1, 2, 3 and 4.

Nomenclature common to all the following tables is listed below. All total propagated values are reported in minutes.

- *Instance* name of instance.
- *Original* average total propagated delay for the original schedules.
- *MDM* average total propagated delay with the schedule adjusted by the mean delay model solution.
- *TSM* average total propagated delay with the schedule adjusted by the TSM.
- *RR over original (%)* relative improvement of the MDM solution over the original ($100 \times (Original - TSM)/Original$).
- *RR over MDM (%)* relative improvement of the TSM solution over the original ($100 \times (MDM - TSM)/MDM$).

Table 9 Total propagated delay improvements for different distribution means

Distribution	Average total propagated delay					
	Instance	Original	MDM	TSM	RR over Original (%)	RR over MDM (%)
Exp(15)	s1	860.09	521.5	472.28	45.1	9.44
	s2	853.49	453.26	395.58	53.65	12.73
	s3	42.41	23.41	9.34	77.98	60.10
	s4	235.08	155.06	122.52	47.88	20.99
	s5	252.87	149.5	122.54	51.54	18.03
	s6	1233.16	849.91	705.01	42.83	17.05
Exp(30)	s1	2050.08	1562.11	1230.08	40	21.26
	s2	1993.59	1336.85	1107.84	44.43	17.13
	s3	141.43	87.52	55.89	60.48	36.14
	s4	701.25	434.87	391.68	44.15	9.93
	s5	599.99	411.45	330.68	44.89	19.63
	s6	3242.44	2538.83	2125.24	34.46	16.29
Exp(45)	s1	3504.48	2554.76	2286.38	34.76	10.51
	s2	3079.29	1930.02	1818.79	40.93	5.76
	s3	267.3	166.12	142.39	46.73	14.28
	s4	1199.09	762.59	703.14	41.36	7.80
	s5	1042.92	723.06	653.13	37.37	9.67
	s6	5715.73	4359.76	3846.91	32.7	11.76
Exp(60)	s1	5247.03	3922.66	3715.71	29.18	5.28
	s2	4674.16	3045.3	2938.05	37.14	3.52
	s3	412.07	280.51	257.87	37.42	8.07
	s4	1825.04	1322.64	1168.95	35.95	11.62
	s5	1437.66	1138.58	958.5	33.33	15.82
	s6	8822.83	6750.83	6198.65	29.74	8.18

Table 10 Total propagated delay improvements for different numbers of training scenarios

Scenarios	Average total propagated delay					
	Instance	Original	MDM	TSM	RR over original (%)	RR over MDM (%)
10	s1	1960.17	1115.96	846.1	56.84	24.18
20		1945.01	1146.31	830.28	57.31	27.57
30		1941.91	1112.12	831.43	57.18	25.24
40		1943.8	1123.35	812.97	58.18	27.63
50		1949.21	1117.64	806.86	58.61	27.81
10	s2	1857.88	811.84	614.34	66.93	24.33
20		1867.09	789.97	624.08	66.57	21
30		1851.5	929.35	605.88	67.28	34.81
40		1856.5	970.19	601.81	67.58	37.97
50		1845.01	938.06	609.64	66.96	35.01
10	s3	108.81	39.26	21.23	80.49	45.92
20		110.76	43.02	23.96	78.37	44.3
30		112.25	38.89	20.83	81.44	46.44
40		118.27	47.47	26.97	77.2	43.19
50		114.05	42.11	19.25	83.12	54.29
10	s4	587.27	212.83	183.25	68.8	13.9
20		583.26	189.41	161.75	72.27	14.6
30		610.97	202.41	184.06	69.87	9.07
40		580.37	194.53	163.15	71.89	16.13
50		586.65	188.69	168.23	71.32	10.84
10	s5	562.62	330.84	229.6	59.19	30.6
20		560.79	303.95	213.6	61.91	29.73
30		554.21	306.81	200.55	63.81	34.63
40		544.98	218.56	206.14	62.17	5.68
50		548.48	304.2	203.92	62.82	32.97
10	s6	3053.12	2143.16	1508.29	50.6	29.62
20		3082.73	2221.58	1560.49	49.38	29.76
30		3062.17	2137.81	1527.92	50.1	28.53
40		3099.17	2169.34	1518.15	51.01	30.02
50		3078.29	2086.7	1487.11	51.69	28.73

References

Ahmadbeygi, S., Cohn, A., & Lapp, M. (2010). Decreasing airline delay propagation by re-allocating scheduled slack. *IIE Transactions, 42*(7), 478–489.

Arıkan, M., Deshpande, V., & Sohoni, M. (2013). Building reliable air-travel infrastructure using empirical data and stochastic models of airline networks. *Operations Research, 61*(1), 45–64.

Balas, E., & Padberg, M. (1975). On the set-covering problem: Ii. An algorithm for set partitioning. *Operations Research, 23*(1), 74–90.

Balas, E., & Padberg, M. W. (1972). On the set-covering problem. *Operations Research, 20*(6), 1152–1161.

BTS: Bureau of Transportation Statistics. https://www.transtats.bts.gov/OTDelay/OTDelayCause1.asp

Chiraphadhanakul, V., & Barnhart, C. (2013). Robust flight schedules through slack re-allocation. *EURO Journal on Transportation and Logistics, 2*(4), 277–306.

Dunbar, M., Froyland, G., & Wu, C. L. (2012). Robust airline schedule planning: Minimizing propagated delay in an integrated routing and crewing framework. *Transportation Science, 46*(2), 204–216.

Dunbar, M., Froyland, G., & Wu, C. L. (2014). An integrated scenario-based approach for robust aircraft routing, crew pairing and re-timing. *Computers & Operations Research, 45,* 68–86.

Froyland, G., Maher, S. J., & Wu, C. L. (2013). The recoverable robust tail assignment problem. *Transportation Science, 48*(3), 351–372.

Kang, L.S.: Degradable airline scheduling: an approach to improve operational robustness and differentiate service quality. Ph.D. thesis, Massachusetts Institute of Technology (2004)

Klabjan, D., Schaefer, A.J., Johnson, E.L., Kleywegt, A.J., & Nemhauser, G.L.: Robust airline crew scheduling. In *Proceedings of TRISTAN IV* (pp. 275–280) (2001)

Lan, S., Clarke, J. P., & Barnhart, C. (2006). Planning for robust airline operations: Optimizing aircraft routings and flight departure times to minimize passenger disruptions. *Transportation science, 40*(1), 15–28.

Marla, L., Vaze, V., & Barnhart, C. (2018). Robust optimization: Lessons learned from aircraft routing. *Computers & Operations Research, 98,* 165–184.

Rosenberger, J. M., Johnson, E. L., & Nemhauser, G. L. (2004). A robust fleet-assignment model with hub isolation and short cycles. *Transportation Science, 38*(3), 357–368.

Shebalov, S., & Klabjan, D. (2006). Robust airline crew pairing: Move-up crews. *Transportation Science, 40*(3), 300–312.

Tahir, A., Desaulniers, G., & El Hallaoui, I. (2019). Integral column generation for the set partitioning problem. *EURO Journal on Transportation and Logistics, 8*(5), 713–744.

Talluri, K. T. (1996). Swapping applications in a daily airline fleet assignment. *Transportation Science, 30*(3), 237–248.

Van Slyke, R. M., & Wets, R. (1969). L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics, 17*(4), 638–663.

Weide, O., Ryan, D., & Ehrgott, M. (2010). An iterative approach to robust and integrated aircraft routing and crew scheduling. *Computers & Operations Research, 37*(5), 833–844.

Yan, C., & Kung, J. (2016). Robust aircraft routing. *Transportation Science, 52*(1), 118–133.

Yen, J. W., & Birge, J. R. (2006). A stochastic programming approach to the airline crew scheduling problem. *Transportation Science, 40*(1), 3–14.