



## INFORMS Journal on Applied Analytics

Publication details, including instructions for authors and subscription information:  
<http://pubsonline.informs.org>

### Decision Support for Schedule Recovery at Lufthansa Group Airlines

Toby O. Davies, Daniel Duque, Emily Masten, Tom Tangl, Eric Bruneton, Alejandra Estanislao, Jon Orwant, Daniel Bogado Duffner, Michael Frey, Christian Most

To cite this article:

Toby O. Davies, Daniel Duque, Emily Masten, Tom Tangl, Eric Bruneton, Alejandra Estanislao, Jon Orwant, Daniel Bogado Duffner, Michael Frey, Christian Most (2026) Decision Support for Schedule Recovery at Lufthansa Group Airlines. INFORMS Journal on Applied Analytics 56(1):58-75. <https://doi.org/10.1287/inte.2025.0296>

Full terms and conditions of use: <https://pubsonline.informs.org/Publications/Librarians-Portal/PubsOnLine-Terms-and-Conditions>

This article may be used only for the purposes of research, teaching, and/or private study. Commercial use or systematic downloading (by robots or other automatic processes) is prohibited without explicit Publisher approval, unless otherwise noted. For more information, contact [permissions@informs.org](mailto:permissions@informs.org).

The Publisher does not warrant or guarantee the article's accuracy, completeness, merchantability, fitness for a particular purpose, or non-infringement. Descriptions of, or references to, products or publications, or inclusion of an advertisement in this article, neither constitutes nor implies a guarantee, endorsement, or support of claims made of that product, publication, or service.

Copyright © 2026, INFORMS

Please scroll down for article—it is on subsequent pages



With 12,500 members from nearly 90 countries, INFORMS is the largest international association of operations research (O.R.) and analytics professionals and students. INFORMS provides unique networking and learning opportunities for individual professionals, and organizations of all types and sizes, to better understand and use O.R. and analytics tools and methods to transform strategic visions and achieve better outcomes.

For more information on INFORMS, its publications, membership, or meetings visit <http://www.informs.org>



THE FRANZ EDELMAN AWARD  
*Achievement in Operations Research*

## Decision Support for Schedule Recovery at Lufthansa Group Airlines

Toby O. Davies,<sup>a,\*</sup> Daniel Duque,<sup>b</sup> Emily Masten,<sup>b</sup> Tom Tangl,<sup>c</sup> Eric Bruneton,<sup>c</sup> Alejandra Estanislao,<sup>d</sup> Jon Orwant,<sup>b</sup> Daniel Bogado Duffner,<sup>e</sup> Michael Frey,<sup>e</sup> Christian Most<sup>f</sup>

<sup>a</sup>Google Research, Pyrmont, New South Wales 2009, Australia; <sup>b</sup>Google Research, Cambridge, Massachusetts 02142; <sup>c</sup>Google Research, 75009 Paris, France; <sup>d</sup>Google Research, 8004 Zürich, Switzerland; <sup>e</sup>Swiss International Airlines, 8302 Zürich, Switzerland; <sup>f</sup>Lufthansa Airlines, 60546 Frankfurt, Germany

\*Corresponding author

Contact: tobyodavies@google.com,  <https://orcid.org/0000-0001-5328-4316> (TOD); danielduque@google.com,  <https://orcid.org/0000-0003-3364-9597> (DD); masten@google.com (EM); tangltom@google.com (TT); ebruneton@google.com (EB); alestanis@google.com (AE); orwant@google.com (JO); daniel.bogadoduffner@swiss.com (DBD); michael.frey@swiss.com (MF); christian.most@lufthansa.com (CM)

Accepted: November 7, 2025

<https://doi.org/10.1287/inte.2025.0296>

Copyright: © 2026 INFORMS

**Abstract.** In 2019, Lufthansa and Google began a five-year collaboration to apply operations research techniques to operational airline schedule recovery, jointly optimizing over aircraft, passengers, and crew to ensure that disruptions such as flight delays are mitigated smoothly. The resulting Operations Decision Support Suite (OPSD) implements several novel decomposition techniques and heuristics. The service is in daily use by Lufthansa subsidiary Swiss International Airlines, with EUR 12 million and 14 kilotons of CO<sub>2</sub> already saved. It is currently being deployed to all airlines in the Lufthansa Group and is on track to save EUR 30 million and 50 kilotons of CO<sub>2</sub> annually.

**Keywords:** decision support • optimization • airlines • disruptions • Edelman Award

### Lufthansa Group and Swiss International Airlines

The Lufthansa Group (LHG) is a global aviation company. In its home market of Europe, it takes a leading role operating with over 700 aircraft. With about 100,000 employees, LHG generated revenue of more than EUR 35.4 billion in the 2023 financial year. LHG is organized into business segments, which include network airlines and point-to-point airlines, and the aviation services sector, which includes the logistics and technik business units.

Swiss International Air Lines (SWISS), a subsidiary of LHG, is Switzerland's largest airline and a premium carrier. Operating a global flight network from Zurich and Geneva, SWISS connects Switzerland with Europe and the rest of the world. With one of Europe's most advanced aircraft fleets, SWISS is dedicated to delivering the highest product and service quality.

### Irregular Operations

As frequent fliers know well, flight disruptions are common: strong headwinds delay landings, deicing delays takeoffs, crew fall ill, passengers miss their connections, and issues are identified during preflight checks. Disruptions caused by events like these are referred to as "irregular operations."

Airlines know to expect irregular operations and have well-established processes for recovering from them. At SWISS, like most airlines, these processes were largely manual, requiring teams of operations controllers to develop a plan to repair the schedule, communicate the plan to ground and air crew to execute the new schedule, and accommodate passengers whose itineraries have been disrupted.

The specific problem addressed in the Google–Lufthansa collaboration is how best to repair these last-minute "day-of-operations" disruptions, or disruptions that happen during the course of daily operations and must be repaired quickly to mitigate impact. Sometimes, mitigating daily disruptions is simple: a passenger who misses a connection can be rebooked on a later flight. Other times, it is more difficult. If a large flight to an airline's hub is delayed, the airline might choose to delay some subsequent flights, creating a potential succession of delayed flights. If a large number of passengers are rebooked onto a flight, a larger aircraft may be required, necessitating different crew with the proper qualifications to operate the larger aircraft. Airline crew have extremely complicated flight duty restrictions that vary by role, airline, and country, imposing additional constraints.

To make these decisions, airlines juggle multiple objectives: minimizing operating cost while providing exemplary customer service, adhering to safety guidelines, and maximizing crew satisfaction. Multiple discrete optimization problems are involved, and historically the industry has solved them in isolation. For instance, one software system might assign aircraft to flights, whereas another assigns crew based on this assignment. When disruptions in the schedule occur, this decoupling requires manual work to check and repair, and therefore significant staff capacity to handle these situations, leading to suboptimal solutions due to time pressure and the limited abilities of staff to evaluate scenarios. For example, a longer-than-expected flight might trigger a compulsory break for the crew, rendering that crew unavailable for a planned subsequent flight.

There are various ways to resolve these problems, but many of the possible decisions must be made within minutes of the disruption occurring. For example, we cannot rebook passengers on a flight departing in 30 minutes if making that decision requires 45 minutes. This means the first problem is to quickly find an acceptable decision, which is preferable to an optimal one that requires hours to compute. These fast but suboptimal decisions inevitably lead to opportunities for reoptimization after recovery operations put the schedule back on track.

Fixing tomorrow's suboptimality introduced by quickly repairing today's disruptions is the second, but vitally important, part of the recovery problem. It is simpler and also has a less strict deadline because the solver can run overnight. During the night, the majority of aircraft are either on the ground at an airport or operating relatively predictable long-haul overnight flights. Therefore, new disruptions are far less likely to occur than during the day.

Because of these similarities and differences, we implemented multiple solvers as part of the integrated Operations Decision Support Suite (OPSD). OPSD consists of several user interfaces that interact with two distinct solvers: the IrrOptimizer, which solves the disruption recovery problem, and the Schedule Optimizer, which solves the reoptimization problem. We define these problems in the next sections.

## The Disruption Recovery Problem

Our implementation of the disruption recovery problem has three component subproblems: aircraft, passengers, and crew, reflecting the three operations departments in the SWISS operations control center. The problem is sufficiently general that subproblems can be added (or omitted) without significant changes.

The problem is logically defined by the following:

- a set of candidates for each flight,
- a set of subproblems,
- a set of subsolutions to each subproblem,
- a set of candidates that must be chosen in order to choose each subsolution, and
- a cost for each subsolution.

A solution chooses a subset of candidates (at most one per leg) and a subset of subsolutions (exactly one per subproblem). Appendix B, IrrOptimizer Mathematical Formulation, includes a mathematical model.

A candidate for a given flight represents a valid pair of departure and arrival times, an aircraft type (e.g., A320) employed to operate the flight, and the prior flight that must use the same aircraft, if applicable. All aircraft belong to one type (or class), chosen such that these aircraft are interchangeable within a given class for all subproblems except the aircraft subproblem (described below), as no aircraft are ever equivalent in this subproblem. The set of candidates are derived from various regulations and operational constraints, and for the purposes of defining a mathematical model, we treat them as an input. We use the term “leg” interchangeably with “flight” throughout this paper.

A solution must pick a candidate for each flight or cancel the flight. The set of candidates selected must satisfy the constraints of all subproblems. Optimal solutions minimize the sum of subproblem costs, although we do not expect to solve this problem to optimality in practice.

The integrated disruption recovery problem has been discussed in the literature, although many approaches consider only a subset of these considerations: several problems in the literature consider aircraft and passenger recovery (Bisaillon et al. 2011, Artigues et al. 2012, Jozefowicz et al. 2013), others consider only aircraft and crew (Desaulniers et al. 1997, Kohl and Karisch 2004, Maher 2015b, Parmentier and Meunier 2020), and several consider all three subproblems (Peterson et al. 2012, Maher 2015a, Arikan et al. 2017). Many constraints and costs considered by our subproblems are significantly more nuanced than in these prior problems. Some examples of these differences are highlighted in the relevant subsections below.

A key property of this definition is that if the selected set of candidates is fixed, each subproblem can be optimized independently. Conceptualizing the problem in this way also allowed the project to proceed in parallel, with a single Google engineer learning about a given department in collaboration with domain experts from that department at SWISS and Lufthansa. No one team member had to be an expert on everything in order to understand the overall problem.

In the next subsections, we will describe at a high level the decisions and costs considered in each subproblem.

### Aircraft

The aircraft (or rotation) subproblem assigns a specific aircraft of the correct type to each chosen candidate, minimizing aircraft operating costs (e.g., fuel use) and penalties for missing scheduled maintenance events. The aircraft operating costs and missed maintenance event penalties are given as input to the solver. The solver outputs a sequence of legs assigned to each aircraft. The

sequence of legs assigned to an aircraft is called a rotation, because they typically start and end at the same hub airport.

The aircraft solver must assign aircraft to candidates subject to minimum ground time and continuity constraints. For a crew to operate a flight, the assigned aircraft must be at the correct departure airport, not be undergoing maintenance overlapping that flight, and have sufficient turnaround time after its prior flight arrives.

Turnaround times guarantee that the aircraft has sufficient time between consecutive flights for passengers to disembark, the aircraft to be cleaned and refueled, and the next flight's passengers to board. The minimum times are a function of the inbound flight, outbound flight, and the aircraft type. For example, an aircraft departing from a different terminal than its last arrival will need significantly longer time before its next flight departs than if it can stay at the same gate; some flights have more passengers with large pieces of baggage brought onboard and stored within the cabin, slowing expected boarding and disembarkation times; and larger aircraft take longer to clean and refuel than smaller ones. In contrast, prior descriptions consider turnaround time to depend only on aircraft type (Artigues et al. 2012).

The aircraft subproblem plays a critical role in ensuring the feasibility of the rotations selected for the recovery problem. We use the term "rotation solver" interchangeably with "aircraft solver" throughout this paper.

### Passengers

The passenger (or pax) subproblem assigns each "reservation" (i.e., one or more passengers who booked together, e.g., a family or tour group) to a sequence of flights to route these passengers to their destination or cancel their itinerary. A solution must assign passengers to a sequence of flights that starts and ends at the correct airport, and starts no earlier than their original itinerary (to ensure they are at the airport on time).

Similar to aircraft, passengers also need to be at the correct airport with sufficient time to make their connections in order to travel on their next flights. The minimum connection time a passenger needs between two flights depends on both of those flights and the passenger. For example, connecting between flights that require the passenger to go through immigration will take longer than if both flights leave from gates that are on the same side of immigration. The passport a passenger holds will significantly impact how much time that passenger needs to clear immigration. In contrast, prior problems in the literature use a constant connection time (Artigues et al. 2012).

This subproblem minimizes the combined cost of all passenger disruptions, including delays, cabin reassignments, and cancellations. These costs differ for each reservation and are partially derived from penalties prescribed by regulations, augmented with an estimate of the long-term cost of customer satisfaction, which

captures the future revenue impact of customers being less likely to book reservations with an airline that has previously inconvenienced them. These costs are manually calibrated in consultation with operations controllers, and for the purposes of an optimization model, we consider them as input.

The passenger subproblem is the most significant component of the overall solution cost. We note that a feasible passenger solution is always available for any selection of candidates. This is guaranteed by the existence of a trivial, albeit prohibitively expensive, fallback: the cancellation of all passenger reservations.

### Crew

The crew subproblem assigns crew members to specific roles on each leg candidate such that all required roles are filled. The number of crew members needed in each role depends both on the flight and on the type of aircraft assigned. For example, larger aircraft usually need more cabin crew, and long flights often need additional crew to ensure they can take sufficient breaks. Qualifications depend on the role and type of aircraft: flight crew are typically only qualified to operate one specific aircraft type (or a small number of sufficiently similar types), whereas cabin crew have more flexibility. A distinction between our approach and that of Peterson et al. (2012) is that their approach considers only cockpit crew.

As with the aircraft and passenger subproblems, crew members also need to be at the right airport with sufficient time between flights to make their connection. These times depend on the inbound and outbound flights, the specific crew member, and if the two flights are flown using the same aircraft.

The crew assignment must also take into account constraints on the maximum time crew can work per day, minimum break durations between work days, and extremely complex contractual obligations that vary between airlines (even within LHG), and even between different subsets of crew within the same airline. Because the specifics of these rules fill multiple books, we do not describe them here. Instead, we will describe the very general class of rules our approach can support in the subsection Crew Subsolver.

The crew subproblem is extremely important to consider in checking the feasibility of a recovery plan. However, it typically has a negligible direct contribution to the overall objective.

### The Reoptimization Problem

Mitigating a disruption, either manually or by solving the disruption recovery problem heuristically, may lead to suboptimal assignments of aircraft to flights. The reoptimization problem aims to mitigate the cost of these suboptimal assignments.

Note that there are still opportunities for reoptimization even in the absence of any disruption due to more

up-to-date fuel efficiency data for each aircraft. For example, a new aircraft that has undergone recent maintenance may be as much as 10% more fuel efficient than older aircraft of the same type. This is a major contributor to measurable impact described in Impact and Adoption.

The reoptimization problem is characterized by a set of legs, a set of aircraft, and a set of crews. A solution assigns an aircraft to each leg, and a set of crew to each leg, subject to the same types of constraints described in the aircraft and crew subproblems above. Unlike the disruption recovery problem, however, the reoptimization problem may not cancel or reschedule flights, nor can it cancel aircraft maintenance events. In addition to hard turnaround time constraints, the reoptimization problem also has soft constraints on turnaround times, which may be violated for a penalty. The assignment should first minimize the aircraft assignment cost plus the penalty of violated soft turnaround time constraints, and then minimize the crew assignment cost.

Assigning aircraft to flights is known in the literature as the tail assignment problem (Khaled et al. 2018, Fuentes et al. 2021). The term “tail” refers to the identifying “tail numbers” painted on the tails of aircraft. The term “tail” is interchangeable with the word “aircraft.”

There are a few key differences between the reoptimization problem and the simpler tail assignment problem. First, tail assignment typically reassigns whole rotations, whereas the reoptimization problem may change rotations. Second, the reoptimization problem is allowed to change the type of aircraft assigned to flights, whereas the tail assignment problem is typically restricted to assigning aircraft of a predetermined aircraft type to any given flight. Third, additional soft constraints are added for the turnaround times of an aircraft to introduce robustness into the schedule. These additional types of changes can potentially impact the feasibility of the crew; thus, the reoptimization problem also considers (and may change) the assignment of crew to flights.

In the first case, if two consecutive legs are assigned to two different aircraft, an existing assignment of crew to legs can become infeasible because the crew may not have enough time to travel to the next aircraft and prepare it for an on-time departure. Some descriptions of the tail assignment problem assign legs to tails rather than rotations, but do not consider this impact on crew assignment. As a result, these approaches can only be used in practice if the assignment is generated sufficiently far in advance to provide enough time for the crew to be assigned later.

In the second case, crew members may be impacted because many crew qualifications are specific to one type of aircraft, and the number of crew members required differs between aircraft types. Thus, changing the aircraft type nearly always requires some crew changes.

The final difference compared with the tail assignment problem is the additional soft constraints on turnaround

times. Without such a penalty, the optimal solution will inevitably pack the most efficient aircraft’s schedule as full as possible, leaving little slack to absorb minor delays encountered in the future. These soft constraints are also used to incentivize the solver to leave large gaps in some aircraft schedules. Aircraft with such gaps are useful to improve schedule robustness: if an aircraft requires unscheduled maintenance, another aircraft with a large gap in its schedule is likely to be available to cover the next few flights assigned to the unavailable aircraft, simplifying the disruption recovery problem.

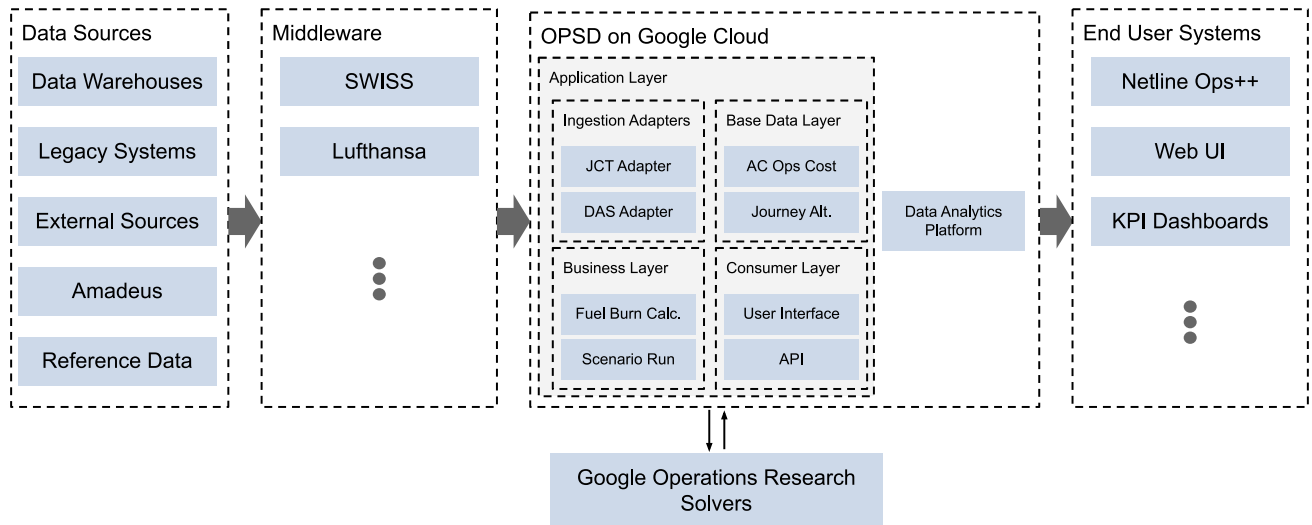
## The Operations Decision Support Suite

As we can see from these problem definitions, solving the disruption recovery and reoptimization problems requires input data describing many different aspects of an airline’s operations. Operational data for large businesses are often spread across multiple operational databases, and the airlines of LHG are no exception: crew data are stored in one database, which is separate from that of the plan of record for aircraft maintenance, which is separate from that used for passenger bookings. Complicating our processes further, some of these operational database systems differ across airlines within LHG.

To address this, we implemented a standardized base data layer in OPSD to which these disparate systems publish updates, minimizing the airline-specific code required. This airline-specific middleware typically chooses which system-specific adapters should be used to input data for that airline, and to which it should export data, based on the systems that airline’s operations controllers use. A high-level overview of this process is shown in Figure 1.

For tactical or strategic problems, solved at most once every few weeks, any minor incompatibilities due to operational changes can be either ignored or manually fixed with little effort because the solves are infrequent. However, for operational problems like the ones addressed by OPSD, which are solved multiple times per day under time pressure, these inconsistencies need to be automatically resolved, or at least have efficient specialized user interfaces. For example, if one database shows that a flight departed part way through its scheduled maintenance, the details of which are stored in another database, knowing if that maintenance was completed early or was canceled and must be rescheduled at the aircraft’s next stop at a maintenance hub is important. Similarly, if a flight landed a few minutes late and does not have adequate turnaround time before its next scheduled departure, the data specifying the amount of delay time required for subsequent flights must be kept up to date.

Consequently, the most significant effort in building OPSD was in preparing the input data for the solvers. Approximately 20 software developers over four years were dedicated to preparing the data and

**Figure 1.** (Color online) An Overview of OPSD's Data Preparation Architecture

*Notes.* Data flow from existing data sources (which vary by airline) into a middleware layer that communicates with a subset of the technical ingestion adapters. These translate data from specific systems used by that airline into a common format shared by all airlines, which is then stored in the base data layer. These data are combined with input from users via the consumer layer via logic in the business layer to construct self-contained requests sent to one of the two operations research solvers developed by Google, described in the rest of this paper. Responses to these requests are stored, transformed, and displayed to users via various components in the application layer. All of these processes record important metadata such as how frequently they are called and the run times of requests via the data analytics platform. The specific example systems and acronyms in this figure are not important to know, but are left for those familiar with the airline industry. API, application programming interface.

the user interfaces necessary to reconcile these data sources; in contrast, only five were involved in building the solvers.

In the next sections, we describe the OPSD solvers. We first present the Schedule Optimizer, which addresses the reoptimization problem described earlier. It was also implemented first, is substantially simpler than the IrrOptimizer, is valuable even if disruption recovery is performed manually, and is responsible for the vast majority of monetary and CO<sub>2</sub> savings achieved from implementing OPSD.

## The Schedule Optimizer

The core of the Schedule Optimizer is a network flow with side constraints solved as a mixed-integer programming (MIP) problem, described in more detail in Appendix A, Schedule Optimizer Mathematical Formulation. Our approach is broadly similar to the compact model described in Khaled et al. (2018).

### Tail Assignment

The tail assignment problem assigns aircraft to legs, minimizing assignment costs, subject to the following constraints:

- every leg must be assigned a single aircraft,
- aircraft must be assigned to valid connections between two consecutive legs,
- aircraft must fulfill their required maintenance events,

- minimum turnaround times must not be violated, and

- crew-approximation constraints, which we describe in the section Approximating Crew Constraints, must not be violated.

One of the more interesting features of the model is a penalty function, which is applied to the turnaround time to improve the robustness of solutions returned by the solver. A turnaround time that is too short does not provide enough time between legs for an aircraft to swap rotations and increases the risk of delay propagation if a small delay were to occur the next day, whereas a moderate turnaround time of one or two hours is more than needed to mitigate the propagation of small delays, but not long enough to use the aircraft as a replacement for another that requires unscheduled maintenance. A piecewise linear penalty function is applied to the amount of time between two consecutive legs assigned to an aircraft to incentivize the solver to include a few minutes of buffer time between most flights, or to include sufficiently long buffers to allow the aircraft to fit additional flights into its schedule. With this feature, the solutions produced by the solver both mitigate the need for schedule changes due to minor disruptions and allow for easier handling of minor disruptions by operations controllers during the day of operations.

### Approximating Crew Constraints

Assigning tails to legs requires appropriately qualified crew to be available to operate those legs. A simple

model works well for the reassignment problem. If the current solution has a valid crew assignment (i.e., a feasible assignment of crew to legs given the original tail assignment), we can restrict the allowed changes relative to the original plan to ensure the same crew assignment remains valid given the new tail assignment. To achieve this, we need to add two restrictions: first, the type of aircraft used for all scheduled flights should be fixed, and second, some flights should be forced to occur sequentially on the same aircraft.

These two restrictions guarantee that the solutions returned by the tail solver are compatible with the current crew assignment, and thus are guaranteed to be usable by operations controllers. Fixing aircraft types to the type used in the original schedule ensures the currently assigned crew members have the required certifications to operate the flights to which they are assigned. Forcing flights with tight crew connections to be flown using the same aircraft guarantees that crew members do not have to swap aircraft if they do not have time to do so.

The addition of these crew approximations in the tail assignment solver allows operations controllers to use the solutions provided. However, this restricts the solution space and reduces the impact the solver can have compared with a more detailed crew model.

### Exploiting a Black-Box Crew Model

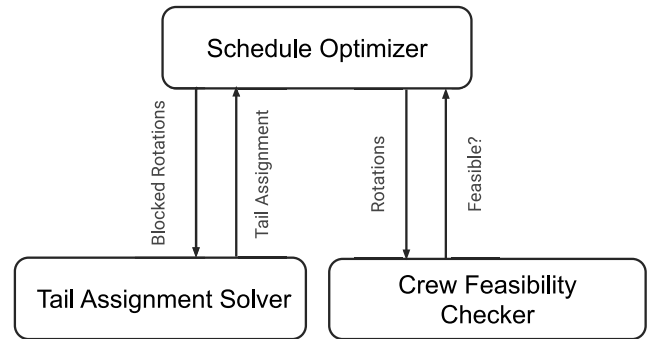
Crew scheduling is a notoriously complicated problem. It is governed by a complex set of laws, and each airline has its own set of additional rules. Instead of hard coding a fixed set of constraints, we use a black-box crew feasibility checker provided by the user to verify that a feasible crew solution exists. Users can write their crew rules in a programming language of their choice, compile this to a portable format, and provide this to the solver in order to allow for different rules per airline, or even per request. This is combined with logic-based Benders decomposition (Hooker and Ottosson 2003) to construct the crew-aware Schedule Optimizer solver (Figure 2).

### Schedule Optimizer Architecture

The Schedule Optimizer solver is decomposed into two main components: the tail assignment solver and crew feasibility checker. The tail assignment solver is the core decision maker of the solver. It solves the tail assignment problem and finds a valid tail-to-leg assignment (i.e., set of rotations) for a given schedule. The black-box crew feasibility checker is used to verify if the solution produced by the tail assignment solver has a feasible crew assignment. If a feasible crew assignment exists (i.e., the feasibility checker returns true), the solution is returned to the user.

If the crew feasibility checker returns false, a feasible crew assignment for the solution produced by the tail assignment solver does not exist. These crew-infeasible

**Figure 2.** The Schedule Optimizer Solves the Tail Assignment Subject to Crew Feasibility Constraints Using Logic-Based Benders Decomposition



solutions are added to a set of restricted rotations, which are used to generate cuts (i.e., additional constraints) for the tail assignment solver to block the solver from generating these solutions again.

The restricted rotation cuts are weak, because each cut removes only a small fraction of the potential tail assignments found by the solver. Additionally, because we only add feasibility cuts, the only feasible solution this approach finds will be provably optimal. This leads to impractically slow convergence for the Schedule Optimizer solver. To help improve the solver's convergence speed, we add discrepancy constraints, inspired by limited discrepancy search (Harvey and Ginsberg 1995), as a kind of primal heuristic. Each request from the user includes a crew-feasible initial assignment. Discrepancy constraints added to the tail assignment solver limit the number of tail class changes the solution can have relative to the initial assignment. Each tail class requires different crew numbers or qualifications; therefore, limiting the number of tail class swaps in a solution helps improve the solver's chances of finding a crew-feasible solution. The allowed discrepancy from the initial assignment is slowly increased over the course of the solving process, as new solutions are found, or if the problem is proven infeasible with the current discrepancy limit. This focuses the search on solutions that are similar to one compatible with the initial crew solution, making it more likely to find crew-feasible solutions. This also has the additional benefit of finding solutions with a smaller number of changes earlier, which are easier for operations controllers to implement.

The crew feasibility checker is a simplified version of the crew solver, which we describe in greater detail in the subsection Crew Subsolver.

### The IrrOptimizer

The IrrOptimizer solves the disruption recovery problem by coordinating the aircraft, pax, and crew subsolvers using a common interface.

The core concept for the subproblem interface is that of candidates for each leg, described in the section [Disruption Recovery Problem](#). Subsolvers receive a “candidate cost” for each candidate as an input (not necessarily the same costs for each subproblem) and return a subset of candidates plus the “subproblem cost” for that subproblem (e.g., the cost of calling reserve crew for the crew subproblem, or the cost of passenger compensation for the pax subproblem). Subproblems also return lower bounds on their objectives (minimizing the sum of chosen candidate costs and the subproblem cost).

To ensure we always have a feasible solution, we require that all flights departing in the future must be cancelable. This ensures there is always a trivial solution: cancel all remaining flights. Ensuring that a trivial feasible solution always exists guarantees that the subsolvers will still propose a solution for the remaining flights instead of simply returning infeasible, because it might if some of those problematic flights were not cancelable. This helps users identify which flights may have inconsistent input data or require manual resolution.

This simple interface allows the IrrOptimizer to implement multiple decompositions and heuristics without any changes to the subsolvers. Each subsolver can be implemented independently using portfolios that combine fast heuristics and slower but (near-)optimal MIP-based models. This allows the solver to both find an initial solution quickly and to refine the solution over time.

The overall flow of the IrrOptimizer is shown in [Figure 3](#).

The core of the IrrOptimizer is a column generation procedure; all subsolutions found by any subsolver are added to a packing integer program, and the duals of the constraints in the linear relaxation of the packing problem are used to set candidate costs. See [Appendix B](#) for details.

However, only relying on this column generation does not reliably find reasonable solutions quickly. To combat this, we first employ an initialization phase to generate some initial columns using heuristically computed costs.

The initialization phase starts using heuristic candidate costs, computed assuming that passengers will not be rerouted and no other flights are delayed. Thus, the candidate cost provided to subsolvers is reduced by the cancellation cost of all passengers booked on the flight. Additionally, if a candidate is delayed, its cost is increased by the delay cost of all passengers booked on the flight, plus the cancellation cost of any passengers who would miss their next connection (assuming that flight departs on time).

The sequential heuristic (shown in [Figure 4](#)) is a key component of both the initialization and improvement phases. A key property is that, because all flights must be cancelable, it will always converge to a feasible solution.

All intermediate solutions generated by the heuristic are added as columns to the packing integer program.

Another key technique is large neighborhood search (LNS), where most candidates are omitted except for a small number close to the initial schedule or incumbent, as we describe in more detail in the next subsection.

### Large Neighborhood Search

The performance of a column generation procedure is usually bounded by the speed at which new solutions can be generated by the subsolvers. Typically, subsolvers take the majority of the computational time budget. The time it takes to solve any subproblem depends primarily on the number of candidates. To speed up the generation of subsolutions, we define a neighborhood of candidates that determines whether to consider a candidate in the subproblems. Such a neighborhood reduces the solution space of the recovery problem in favor of speeding up the computation of a subsolution. Reducing the solution space usually diminishes the quality of subsolutions. To overcome such a behavior, we embed LNS within the column generation procedure. The LNS updates the neighborhood at each iteration of the column generation with the intention of exploring a different region of the solution space. Broadly, the principles that guide the generation of a new neighborhood by the LNS controller are as follows:

- If a leg is disrupted in the current incumbent solution (e.g., delayed), more candidates of such a leg will be considered in the next neighborhood.
- If the incumbent solution of the recovery problem has not improved in recent iterations, more candidates (for all legs) will be considered in the next neighborhood.
- All candidates in the current incumbent solution are considered in the next neighborhood.

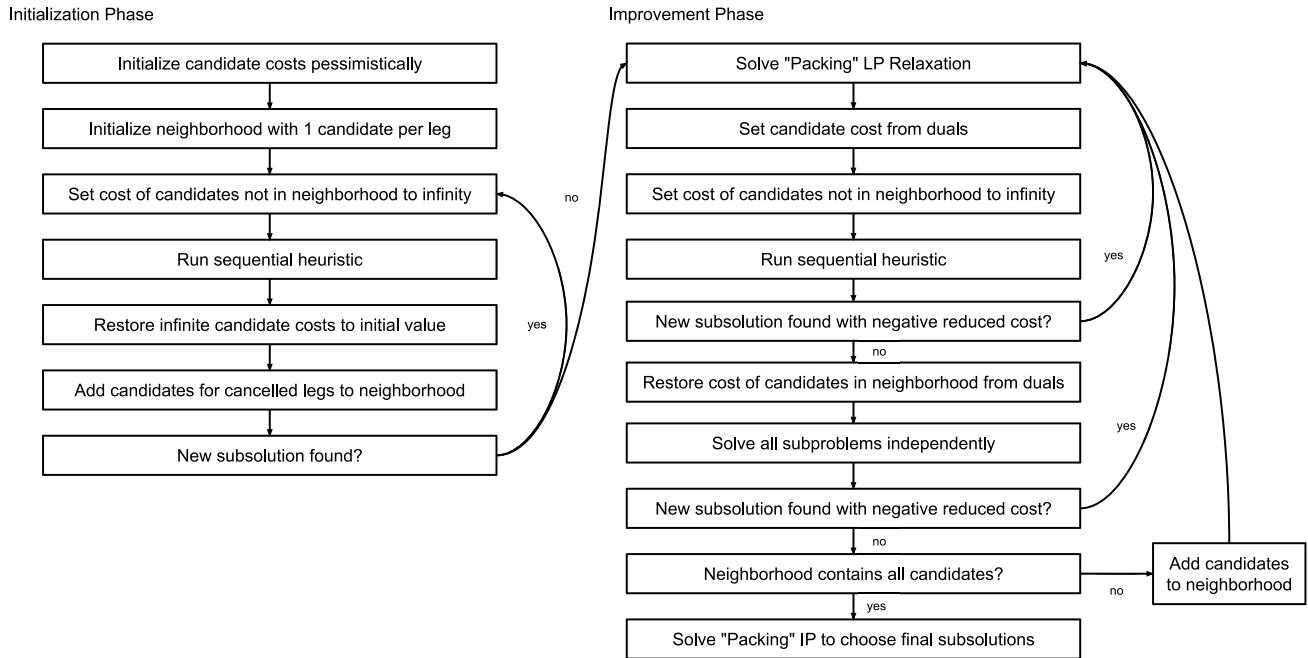
Embedding the LNS procedure in the column generation procedure modifies the termination criteria. We terminate before reaching the time limit if no subproblem can find any new column with negative reduced cost *and* the neighborhood includes all candidates. In practice, given the scale and complexity of the problem, the procedure will almost always terminate due to reaching the time limit.

### Aircraft (Rotation) Subsolver

The rotation and tail assignment subproblem is essentially the same as the tail assignment described above. Aircraft must be chosen to operate flights with sufficient turnaround time between consecutive flights assigned to the same aircraft, and aircraft must fulfill their required scheduled maintenance.

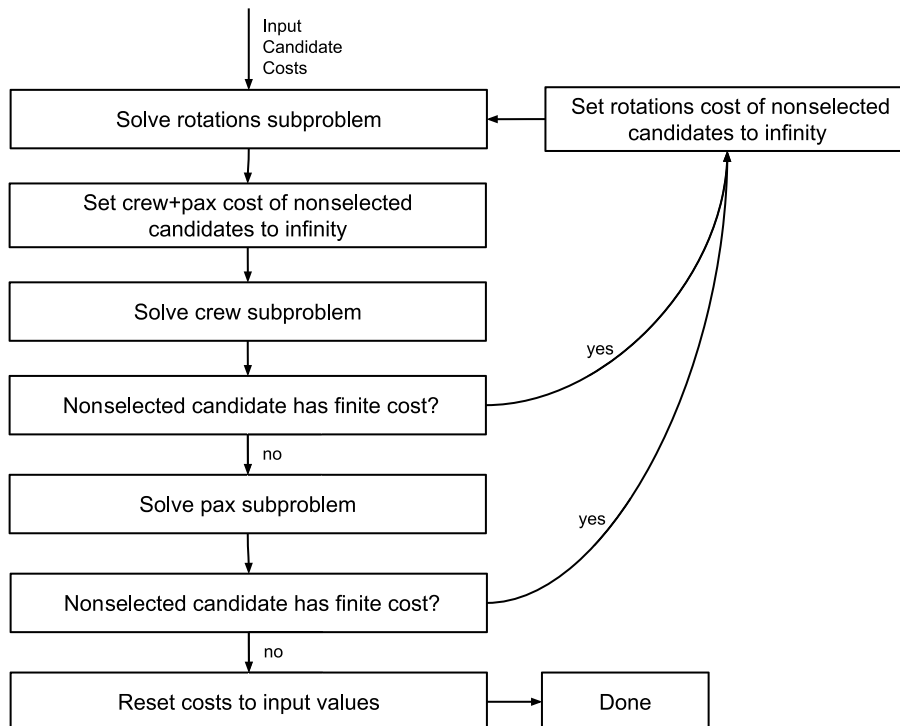
The major difference in this subproblem is that irregular operations need to be modeled: flights can be canceled or delayed, aircraft can be swapped, and

**Figure 3.** The IrrOptimizer Coordinates the Aircraft, Pax, and Crew Subsolvers to Solve the Disruption Recovery Problem Using a Common Subsolver Interface



*Notes.* The IrrOptimizer works in two phases: initialization, which uses heuristic costs chosen to pessimistically overestimate disruption cost, and improvement, which uses costs derived from a linear programming relaxation described in Appendix B. Both phases share two key techniques: The first is a neighborhood of candidates that restrict the solution space, so the subsolver calls perform large neighborhood search until the neighborhood eventually contains all candidates. The second shared technique is a sequential heuristic (see Figure 4) to find compatible subproblem solutions.

**Figure 4.** The Sequential Heuristic Converges to a Feasible Solution to the Disruption Recovery Problem



*Note.* "Nonselected candidates" refers to candidates that are not included in the solution of the most recent subproblem solved.

scheduled maintenance can be canceled. This significantly changes the structure of the problem; therefore, in spite of their apparent similarity, the tail assignment problem in the IrrOptimizer shares relatively little code with this subsolver.

**Solution Approach.** The rotation subsolver is a portfolio of the following:

1. *Delay propagation*: A heuristic that assigns the same sequence of legs to each aircraft, and assigns the earliest possible departure time given minimum turn-around times.

2. *Multicommodity flow*: A complete MIP model that combines a multicommodity flow (where each tail number is a separate commodity) and a set packing problem, where each leg is covered by at most one tail number. This is similar to the tail assignment model previously described.

3. *Column generation*: A column generation approach, with a resource-constrained shortest-path subproblem per aircraft. Once the optimal solution to the linear relaxation is found, integrality constraints are introduced for the variables that have been added so far, and the problem is solved as an MIP to choose a rotation for each aircraft.

When the IrrOptimizer runs the rotation subsolver, both the multicommodity flow solver and the column generation solver are run, and the best solution from the two solvers is returned. The delay propagation heuristic is used by both solvers to find a quick, but usually sub-optimal, solution that is used as a warm start.

### Passenger Subsolver

The objective of any airline is to transport passengers to their destinations. The benefits that accrue from a recovery plan should be determined primarily by its impact on passengers. Passenger recovery does not have an impact on the feasibility of the final solution because canceling all passenger itineraries is a feasible recovery solution. Instead, passenger recovery is key to evaluating quality.

**Model Description.** The passenger recovery problem aims to minimize disruptions to passengers while simultaneously minimizing costs to the airline. Passenger disruptions include flight misconnections, departure/arrival delays, downgrades, and cancellations. Airline costs arise from late arrivals, itinerary changes, class upgrades, and other recovery efforts. Effectively balancing these competing objectives is crucial.

The passenger recovery problem involves several key decisions. These include flight rescheduling to facilitate connections, aircraft swaps or configuration adjustments to increase capacity, and, most importantly, passenger rerouting to alternative flights. Hubs, central airports of an airline's network where the majority of passenger

transfers occur, can also be strategically used to find recovery solutions. For example, at hubs, LHG can expedite important passenger connections using a limited fleet of ramp direct service (RDS) vehicles, which are vehicles that can quickly transport passengers directly to their next gates.

The passenger recovery problem for LHG is further complicated by the heterogeneous nature of its passengers. Passengers often travel in groups (e.g., families), have varying connection speeds within airports, and/or may be subject to specific constraints and cost implications based on factors like disability, nationality, or frequent flyer status. These individual factors must be considered during the recovery process. For example, connection times can vary depending on both the specific reservation and the connection itself (i.e., the connecting flight legs). Non-EU passport holders will usually require more time to connect between legs if they enter or leave the Schengen zone (requiring passengers to queue for immigration). Frequent fliers can usually use priority security lanes, giving them more time to make their connections in case of delays.

These individual factors, coupled with the large scale of the problem—often involving hundreds of thousands of passengers across the network—create an immense optimization challenge. Effective recovery requires not only considering individual passenger needs but also efficiently exploring the vast solution space to identify globally near-optimal rerouting strategies in minutes.

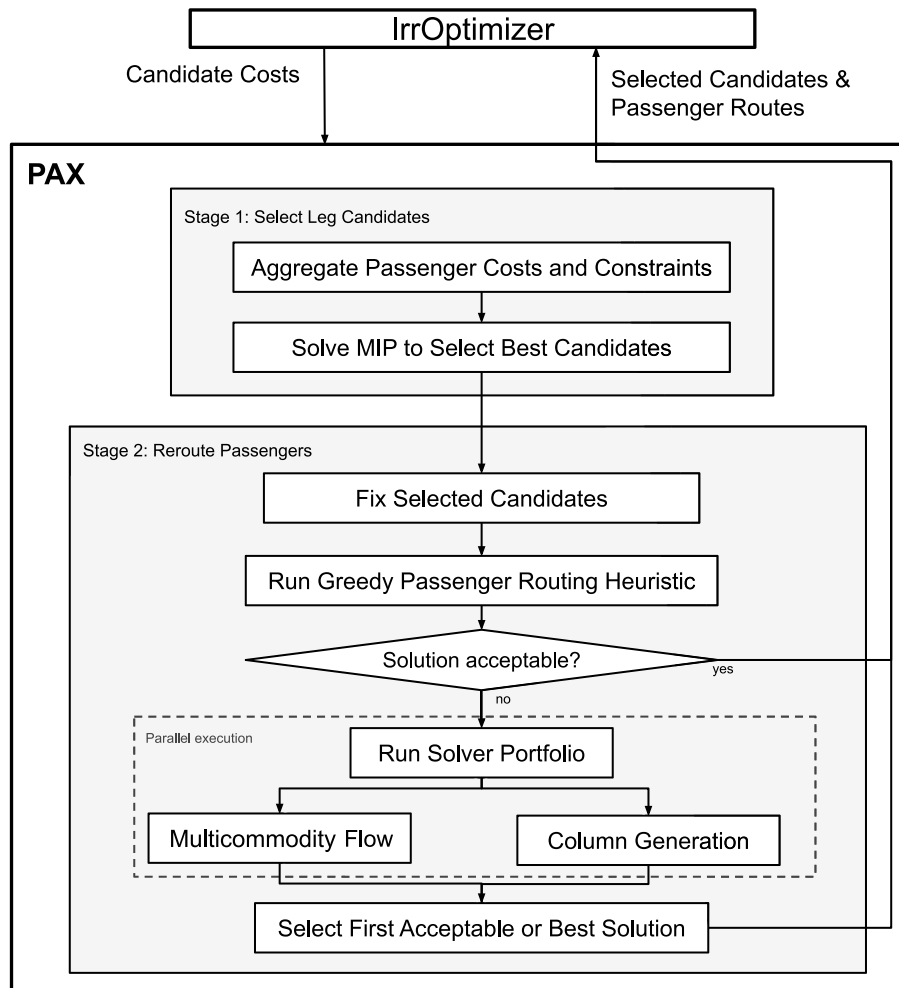
**Solution Approach.** As shown in Figure 5, our approach to the passenger recovery problem is a two-stage process, inspired by the work of Zhang et al. (2016): a candidate selection (flight rescheduling) stage, followed by a passenger rerouting stage. In the first stage, we use an aggregated MIP model to choose a subset of the candidates and assign aircraft classes. The second stage focuses on rerouting reservations, swapping aircraft classes for larger or smaller aircraft depending on the need, and strategically utilizing the RDS fleet to expedite passenger connections.

For the passenger rerouting stage, we implemented a portfolio of distinct approaches to balance solution quality and computational efficiency. This portfolio-based strategy ensures that the system can dynamically select the best tool for a given disruption scenario, leveraging the strengths of different optimization techniques to find high-quality solutions quickly. The mathematical formulation for this stage is given in Appendix B, Section B.4 (Passenger Subsolver: Reroute Passengers Formulation).

The rerouting process is as follows:

1. *Greedy routing heuristic*: A fast, greedy algorithm is always run first to generate an initial feasible solution. This heuristic quickly finds a baseline solution by rerouting reservations in order of importance, initially favoring the original itinerary.

Figure 5. The Pax Subsolver Selects Candidates and Then Reroutes Passengers



2. *Parallel portfolio execution*: If the initial solution from the greedy routing is not sufficient, a portfolio of more advanced solvers is executed in parallel to find a better solution. This portfolio includes the following:

- *Multicommodity flow*: This model is inspired by the multicommodity flow model of Zhang et al. (2016). Our enhancements to this model include strategic search space pruning, incorporating aircraft capacity adjustments, and the use of RDS vehicles to reduce connection times. This approach strives for holistic optimization.

- *Column generation*: A column generation approach provides a middle ground. We iteratively add feasible itineraries as columns to a linear program until convergence, generate a 0-1 integer program with the same set of columns constrained to be binary, and then solve this model. This method balances solution quality and computational effort.

3. *Final solution selection*: The first solver in the portfolio to return a solution that meets a predetermined acceptance threshold from the IrrOptimizer is selected.

If neither provides a satisfactory solution within the time limit, the best solution found by either solver is chosen. The initial solution from the greedy algorithm also serves as an initial solution for both of the advanced solvers, further improving performance.

This parallel, portfolio-based approach is a key feature of the passenger solver, because it allows the system to leverage the strengths of different optimization techniques to find high-quality solutions quickly and efficiently.

### Crew Subsolver

**Model Description.** The crew recovery problem aims to find feasible crew schedules to operate a given set of flights, while minimizing the number of changes from the original schedules (or some other cost function). The original crew schedules can be disrupted because of certain events such as departure/arrival delays, cancellations, change of aircraft class, or sick crew members. The goal of the crew subsolver is to *repair* these

disruptions, that is, to find new schedules satisfying the following feasibility constraints.

**Trip Constraints.** A *trip* is composed of one or more crew members, and of a sequence of flights assigned to them. A trip can comprise several flight duty periods (FDPs), which roughly correspond to a day of work. A trip must satisfy many feasibility constraints, including the following:

1. A trip must start and end at the home base of its crew members.
2. There must be enough time between consecutive flights. This minimum connection time can depend on many factors, such as the flight IDs or whether the crew stays on the same aircraft.
3. The FDPs must not be too long, and there must be enough rest time between consecutive FDPs. These time limits also depend on many factors, such as the start of the FDP and its number of flights.
4. The trip crew members must be qualified to operate each aircraft class used in this trip.

**Global Crew Constraints.** A set of flights can be operated by the crew if there is a set of trips that satisfies all the global crew constraints, including the following:

1. A sufficient number of crew members must be available in each role to operate each flight (e.g., at least one captain, one first officer, one cabin manager, and three flight attendants). Additional crew members, if any, travel as passengers (this is called *deadheading*), which reduces the number of seats that can be used by the passenger solver.
2. Crew members should stay together during each FDP to minimize the impact of disruptions: if the crew members on one delayed inbound flight are all assigned to the same next flight, then the inbound delay has a smaller effect on subsequent flights than if each crew member were scheduled to connect to a different flight, all of which might have to be delayed or find replacement crew.

The crew subsolver aims to minimize the cost of the crew recovery solution. This cost can be computed in various ways, depending on the desired goal. For example, we typically use the number of changes compared with the initial schedules to minimize the total number of changes that must be communicated to crew members. This is a reasonable metric because crew members generally prefer to avoid changes, and each change requires additional work for an operations controller to review and implement. These costs are very small compared with those in other solvers because we do not typically want to incur significant additional operating costs or degrade passenger experiences to reduce crew changes. However, if we can achieve the same aircraft

and passenger costs with fewer crew changes, we would prefer to do so.

**Solution Approach.** The crew subsolver takes as input a set of flight candidates (generally one per flight ID) and a list of initial trips. It returns a subset of flight candidates (at most one per flight ID because some flights can be canceled), and a list of modified trips satisfying all the above constraints, with a total cost as small as possible. It does this in two stages, illustrated in Figure 6 and presented below.

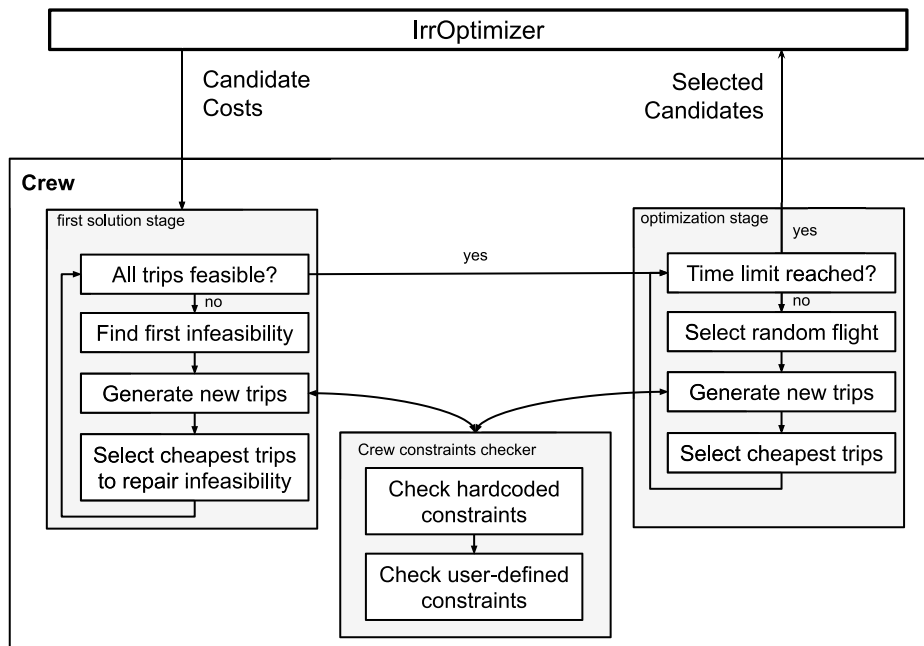
**Stage 1.** The first stage of the crew subsolver is to find a feasible solution. For this, the crew subsolver tries to repair the input trips by looking at all the infeasibilities in chronological order, as follows (see Figure 6):

1. Identify all the infeasible flights, that is, those breaking one of the above crew constraints. If there are no infeasibilities, the problem is solved; proceed to Stage 2.
2. Find the earliest departing infeasible flight, and select a subset of trips  $\mathcal{T}$  and a subset of legs  $\mathcal{L}$  that could be changed to repair it (with some heuristics, mostly based on feasible paths in the graph of legs, using a relaxed version of the above constraints).
3. For each trip  $T$  in  $\mathcal{T}$ , generate a list of alternative, feasible leg sequences for its crew members, as follows: start with a fixed prefix of  $T$ , append some legs in  $\mathcal{L}$ , and finish with a sequence as close as possible to the end of  $T$ . This step is done with a depth-first search based on a crew constraint checker component (see Figure 6 and the subsection Crew Constraint Checker below). It also computes the cost of the generated alternative trips.
4. For each trip  $T$ , select at most one of its alternative leg sequences generated above, so that these alternative trips satisfy the global crew constraints as defined above, while minimizing their total cost. We model these constraints as an MIP and solve them with an MIP solver. Note that the resulting model is a set covering problem (each flight must be covered by enough trips in each role) with side constraints (because of the second global constraint stating that crew members should stay together).
5. Go back to the first step.

**Stage 2.** Once a feasible solution has been found, the second stage tries to reduce its cost as much as possible. For this, it uses a specific LNS, implemented with the following algorithm (see Figure 6):

1. Select a flight at random.
2. Assume that it is infeasible and repair it using the same method as in Steps 3 and 4 of Stage 1 above.
3. If the repaired trips have a smaller cost than the current ones, make them the new current trips.

Figure 6. The Crew Subsolver Selects and Assigns Crew to Candidates



4. Repeat the above steps until a time limit is reached.

**Crew Constraint Checker.** The goal of the crew constraint checker is to check whether a trip is feasible and to compute its cost if feasible. Many trip feasibility constraints apply to only one specific airline. Some of these constraints are updated somewhat regularly (e.g., once every year or two, because of new regulations or collective labor agreements). Similarly, the cost of a trip can be defined in many ways. To avoid implementing all the constraints and cost models from all LHG airlines in the solver, and to avoid recompiling and redeploying the solver when a constraint changes, the crew constraint checker is divided in two parts (see Figure 6). The most fundamental constraints and some basic costs are implemented in the solver (such as the constraint that each flight must have a sufficient number of crew members in each role). The others must be implemented by the user (via a well-defined application programming interface), compiled to the portable binary WebAssembly format (Rossberg 2019), and sent to the solver in the optimization request. The solver then runs this WebAssembly code to check these custom constraints and to compute custom costs. The solver uses several techniques to improve performance, such as a cache of previously checked leg sequences.

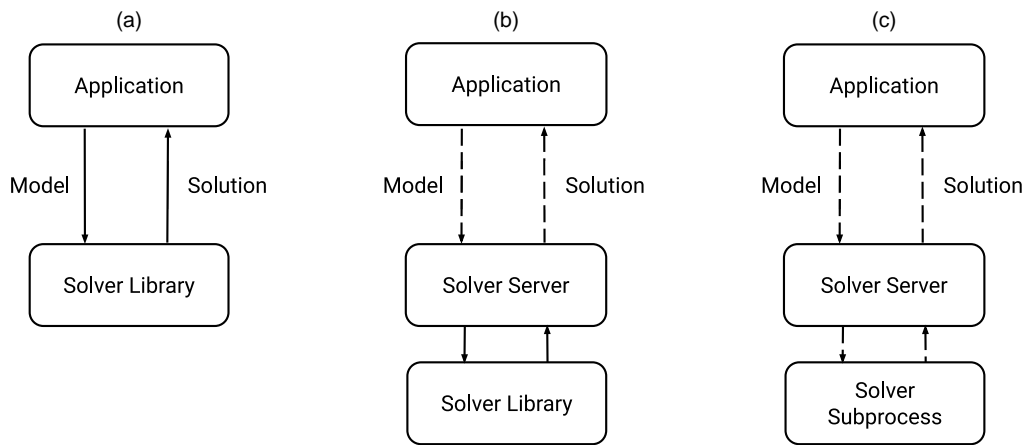
### Reliable Distributed Model Solving

Many components of the OPSD solvers we have described above solve one or more MIP models, either for a whole (sub)problem or as part of a heuristic. In a

traditional application designed to use an MIP solver, the solver is linked into the same binary (as in Architecture A in Figure 7). This has a significant disadvantage when using any parallel portfolio of solvers or MIP-based heuristics that require multiple models to be solved concurrently: these MIP models would compete for computational resources on the same machine at the same time, which can have serious consequences for the reliability of the application.

Exceeding the random access memory (RAM) assigned to an application, for example, by solving too many large MIPs simultaneously, can cause the application to stop. If no per application memory limits are configured, the consequences can be even worse: approaching the physical memory limits of the machine can catastrophically degrade the performance of all applications running on the same machine. Exceeding the available processing capacity can cause all computations on the machine to slow down because they must share the available processors, and incur additional overhead as a result of this sharing.

Architecture B in Figure 7 largely mitigates these problems: an application can safely solve many MIPs concurrently because they are being solved on a larger number of machines. However, if many users share the same set of solver servers, this architecture has similar problems to Architecture A, but on the solver server instead of the application: if one server receives many MIPs to solve simultaneously, these MIPs will still share resources. If one hard MIP being solved on a server uses all of the available RAM, the operating system will stop the entire solver server. When this happens, none of the

**Figure 7.** Comparison of Different Application Architectures Using a Solver Library

*Notes.* In this figure, solid lines represent in-process communication via traditional function calls, and dashed lines represent interprocess communication using, for example, a remote procedure call library such as gRPC (van Winkel 2016). Architecture A shows the simplest and most common architecture, but has limits on the number of MIPs that may be solved concurrently by an application because they all must be solved on the same machine. Architecture B allows for distributed solving, so one application can solve more MIPs on more machines. Architecture C enables significant reliability improvements compared with Architecture B where many applications share the same set of solver servers.

users whose MIPs were being solved on the stopped server at the time will receive a response, even those whose MIPs did not require significant RAM.

Because the server process itself is stopped, there is no way to return an error; therefore, applications must detect that the server has crashed and retry their request. However, naively retrying all failed requests is dangerous—the hard MIP that used all the available RAM is among those that will be retried, and will most likely cause a different server to stop when retried. If retried indefinitely, hard MIPs received by solve servers will build up, even if they are only a small percentage of new requests, they can become the overwhelming majority of all requests being processed by the servers. Potentially, this can reach the point where *all* servers are solving multiple hard MIPs, quickly exhaust their memory limit, and are stopped, then restarted, and immediately receive multiple hard MIP retries again.

To combat these issues, most Operations Research projects at Google solve MIPs using a shared internal service called the Universal Optimization Solver Service (UOSS). Although UOSS is not part of OPSD, it is a critical service on which OPSD depends. UOSS implements Architecture C in Figure 7, running each solve in a separate subprocess. This has two main advantages over the simpler Architecture B:

- When a solver uses too much memory on a request, the operating system handling that request can stop only the subprocess handling that request rather than stopping the entire server.
- When a solve crashes or is stopped, the server can detect this and notify the client application that an error occurred without crashing the client's application.

Each solver subprocess started by UOSS runs inside its own container, which enables the operating system to

enforce memory and processor limits per subprocess. This allows a single server to safely run multiple solves concurrently without the risk of exceeding the server's available resources.

The ability to end individual solve subprocesses quickly and reliably also simplifies the implementation of load shedding. Load shedding allows the service to continue functioning well for most users when there is more demand for solves than can be served concurrently. When overloaded by a small number of users, UOSS continues to serve the majority of other users by choosing a running solver process to end and thus free up the processors and RAM required for the new request. When choosing which solve to interrupt, UOSS considers how many other solves from the same user are in progress, whether a human is waiting for an answer or whether this is an offline batch computation that can be delayed, and the amount of time the solve has been running thus far.

This fair load shedding allows batch pipelines and large-scale experiments to share capacity with critical operational requests while most users do not experience any of the symptoms of overload described above. This ensures that Google's large internal MIP users cannot consume all the available resources, and thus cannot prevent OPSD from starting MIP solves on UOSS or vice versa.

## Impact and Adoption

As of January 2025, the total monetary savings directly attributed to the Schedule Optimizer are CHF 8.5 million in reduced operating costs (fuel cost and landing, parking, and handling fees), and 14.7 kilotons of CO<sub>2</sub> due to reduced fuel consumption. The solver was used on 1,038

days between January 2022 and January 2025, and its results were published as the plan of record six or seven times a day (a total of 6,797 times). Operations controllers accepted 222,714 changes suggested by OPSD solvers overall, representing 26% of all changes made by operations controllers since January 2022. Those solver-initiated changes affected 61,192 aircraft or equipment, representing 24.7% of all flights operated by SWISS. The other nonsolver changes were considered, decided upon, and executed by operators manually.

The aforementioned metrics are calculated daily. We can distinguish manual changes by operations controllers from the implemented scenario runs proposed by our solution. Savings are considered as the decrease of operating costs after the implementation of a proposed scenario by a user. A user might also make changes to a proposed scenario because of factors that are as yet unknown to the system.

Before the Schedule Optimizer was available, one full operations controller shift per day was dedicated to preparing the next day's tail assignment, and the operations controller was mainly preoccupied with fixing problems, such as ensuring aircraft were available for their scheduled maintenance that they would otherwise miss if the aircraft were reassigned earlier in the day. The time and effort required to fix these problems usually meant there was little opportunity for the operations controllers to make any improvements to the assignment, and they simply accepted the first reasonable solution that satisfied the constraints. In contrast, OPSD allows operations controllers to simultaneously fix these issues and optimize the schedule, even in the busiest situations. The ability of the Schedule Optimizer to optimize globally allows for more significant changes to the schedule than were ever achieved manually: manual optimization would change at most 20 rotations for the next day, whereas OPSD regularly changes over 100 rotations at a time over a three-day period. This improves both operational robustness and saves CO<sub>2</sub> and operating costs beyond what could ever be achieved manually because of the additional flexibility available over the longer time horizon.

The introduction of OPSD also gave full transparency of the operating costs of a flight to operations controllers for the first time. In addition to operational cost considerations, the solver assists the operations controllers in adding robustness to the schedule. By using various incentives and penalties, it achieves improved outcomes, such as

- reducing the number of crews changing aircraft during the day,
- enhancing aircraft utilization before maintenance events,
- selecting optimal aircraft size based on passenger bookings, and
- favoring desired flight cycles or flight time limits.

Unfortunately, these effects are not quantitatively measured: doing so is quite challenging because, compared

with the cost savings, these dimensions are influenced by many factors other than our tool. We primarily rely on the user feedback on the optimized and proposed scenarios as an indicator of the robustness of our solutions.

A human could not simultaneously consider all these factors and consistently weigh their importance relative to each other across dozens of aircraft and multiple days, thus highlighting the OPSD's significant advantage in enhancing operational efficiency and decision making.

Another impact has been a change in how passengers are rerouted when flights are delayed. Historically, passengers would be rerouted after they reached a hub for the airline (e.g., Zurich for SWISS), because there are typically many options for connecting flights on which to rebook the passengers. Now, as a result of OPSD, passengers are instead rerouted as soon as it becomes apparent that their itinerary will be disrupted. This allows greater flexibility for rerouting passengers with connections when flights to a hub are canceled or delayed, so passengers can often reach their destination sooner by routing via a different hub either within the airline, within LHG, or via a partner airline. More flexibility can provide financial benefits, because passengers are less likely to require an overnight stay in a hotel at the airline's expense and are less likely to be sufficiently delayed to be entitled to compensation, in addition to the obvious customer satisfaction benefits of fewer delayed passengers.

An overall financial impact of this policy change is difficult to estimate because the passenger subsolver has only been in operation since February 2025, and passengers have a multiyear grace period in which to claim compensation under EU rules. However, we can share an anecdote: shortly before the system was to be officially rolled out, a single long-haul aircraft needed unexpected maintenance. A member of the OPSD project happened to be in the operations center and suggested the solver's output be compared with the intuition of the operations controller. The estimated cost (assuming all passengers claim their full compensation, plus hotel costs, passenger rebooking, and the operating cost of canceled flights) for the solution generated by the solver was over USD 200,000 less expensive than the solution proposed by the operations controller with over a decade's experience. This estimation came directly from the tool OPSD uses to evaluate the costs of various scenarios and is well calibrated.

## Conclusions

OPSD solves two important interacting optimization problems that must be solved every day by commercial airlines: the disruption recovery problem and the reoptimization problem. Our definitions of these problems better reflect the way airline disruptions must be handled in practice, taking into account the need to quickly mitigate

a disruption and the opportunity to later perform nondisruptive reoptimization once operations have stabilized.

OPSD has saved SWISS significant operating expenses, with reduced fuel usage, which also reduced CO<sub>2</sub> emissions, and passengers are rerouted much sooner after disruptions, leading to reduced delays for passengers.

OPSD is in daily use at several airlines within LHG. Prior to OPSD, these airlines did not have shared data or process standards for handling irregular operations. The successful deployment of OPSD at these airlines is strong evidence that OPSD could be deployed at nearly any commercial airline with only changes to the small middleware layer that imports and exports data to OPSD's base data layer.

OPSD's mitigate-then-reoptimize approach is even more general: organizations in many industries face daily disruptions that must be mitigated faster than executing an optimal repair plan. Implementing solvers and processes for regular reoptimization has the potential for significant impact in many such organizations.

## Acknowledgments

The authors wish to thank the Edelman Committee and the reviewers for their helpful suggestions. This manuscript is much clearer for their efforts.

## Appendix A. Schedule Optimizer Mathematical Formulation

This section presents a simplified tail assignment model used by the Schedule Optimizer.

### A.1. Tail Assignment Model

Below is a simplified tail assignment model. Some of the more complicated side constraints that are not discussed in the text have been omitted.

#### Sets and Constants

- $l \in L$ : Set of legs
- $s \in S$ : Set of indivisible leg sets  $s \subset L$ ; an indivisible leg set  $s$  is a set of legs that must be flown by the same aircraft
- $t \in T$ : Set of tails
- $t \in T_s \subseteq T$ : Set of tails that can fly leg set  $s$
- $a \in A$ : Set of airports
- $s' \in TA(t, s)$ : Set of leg sets  $s'$  that depart after leg set  $s$ , which cannot be assigned tail  $t$  because of turnaround times if tail  $t$  is assigned to leg set  $s$ 
  - $j \in DA_{ta}$ : Departure and arrival times of the leg sets that depart or arrive at airport  $a$  and which tail  $t$  can fly
    - $s \in S_{aj} \subset S$ : Leg sets departing or arriving at or before timestamp  $j$  at airport  $a$
    - $s \in S_{aj} \subset S_{aj}$ : Leg sets departing at or before timestamp  $j$  from airport  $a$
    - $s \in S_{aj} \subset S_{aj}$ : Leg sets arriving at or before timestamp  $j$  to airport  $a$
  - $j \in M_{ta}$ : Set of maintenance event start times at airport  $a$  for tail  $t$ 
    - $S_a$ : Set of all leg sets (arriving or departing) at airport  $a$
    - $(p, q) \in (S_a, S_a)$ : Pair of valid arrival and departure leg sets; the last leg of leg set  $p$  arrives before the first leg of leg set  $q$  departs and they connect at the same airport

- $Dep_{pq} \subset S_a$ : Set of leg sets that depart from airport  $a$  between leg sets  $p \in S_a$  and  $q \in S_a$
- $c_{st} \geq 0$ : Cost of assigning tail  $t$  to leg set  $s$
- $c_{pq} \geq 0$ : Cost associated with the amount of ground time between leg sets  $p$  and  $q$
- $i_{ta} \in \{0, 1\}$ : Initial location of tail  $t$ , equal to one if tail  $t$  is at airport  $a$  at the start of the solving period

#### Decision Variables

- $x_{st} \in \{0, 1\}$ : Binary variable that takes a value of one if tail  $t$  is assigned to leg set  $s$ ; this variable is zero if  $t \notin T_s$
- $y_{pqt} \in \{0, 1\}$ : Binary variable that takes a value of one if tail  $t$  is assigned to consecutive leg sets  $(s_p, s_q)$

#### Formulation

$$\min \sum_{s \in S} \sum_{t \in T_s} c_{st} x_{st} + \sum_t \sum_{pq} c_{pq} y_{pqt} \quad (\text{A.1})$$

$$\text{s.t.} \quad \sum_{t \in T_s} x_{st} = 1 \quad \forall s \in S, \quad (\text{A.2})$$

$$x_{st} + \sum_{s' \in TA(t, s)} x_{s't} \leq 1 \quad \forall s \in S, t \in T, \quad (\text{A.3})$$

$$i_{ta} - \sum_{s \in S_{aj}} x_{st} + \sum_{s \in S_{aj}} x_{st} = 1 \quad \forall t \in T, a \in A, j \in M_{ta}, \quad (\text{A.4})$$

$$i_{ta} - \sum_{s \in S_{aj}} x_{st} + \sum_{s \in S_{aj}} x_{st} \leq 1 \quad \forall t \in T, a \in A, j \in DA_{ta}, \quad (\text{A.5})$$

$$x_{pt} + x_{qt} - \sum_{r \in Dep_{pq}} x_{rt} \leq 1 + y_{pqt} \quad (\text{A.6})$$

$$\forall t \in T, (p, q) \in (S_a, S_a).$$

Equation (A.1) minimizes the sum of the cost of assigning tails to legs and the cost of the ground time between two flights. Equation (A.2) forces every leg to be assigned exactly one tail. Equation (A.3) ensures that assigned tails do not violate the minimum turnaround time between two legs. Equation (A.4) ensures that all aircraft fulfill required maintenance checks. Equation (A.5) models the network commodity flow constraints as inventory constraints. Equation (A.6) models the ground time robustness constraint—it ensures that variable  $y_{pqt}$  takes a value of one when tail  $t$  is assigned to consecutive leg sets  $p$  and  $q$ .

The crew-approximation constraints are encoded in the sets  $T_s$  and  $S$ . First, the assigned aircraft type of a flight cannot change, and this restricts the set of tails that can be assigned to legs (i.e., reduces the size of  $T_s$ ). Second, because of tight crew connections, some flights are forced to be flown by the same aircraft. These sets of legs are the “leg sets” defining  $S$ .

## Appendix B. IrrOptimizer Mathematical Formulation

We refer to the core of the disruption recovery problem, which chooses from the set of available subproblem solutions, as the packing problem because of its similarity to a set-packing problem. For the purpose of introducing the packing problem, assume that we can enumerate all possible subsolutions from each subproblem. Let  $P$  be the set of subproblems (each solved with a specific subsolver), and let  $S_p$  be the set of all subsolutions of subproblem  $p$ . Furthermore, let  $C$  be the set of all leg candidates in

the recovery plan, and let  $K$  be the set of all cabin classes. A particular subsolution  $j \in S_p$  is characterized by the following:

- $c_{pj}$ : The cost of implementing subsolution  $j \in S_p$  of subproblem  $p \in P$
- $u_{pjc}$ : A binary parameter that takes the value one if candidate  $c \in C$  is selected in subsolution  $j \in S_p$  of subproblem  $p \in P$ , and takes the value zero otherwise
- $s_{pjck}$ : The number of seats in cabin class  $k \in K$  used by candidate  $c \in C$  in subsolution  $j \in S_p$  of subproblem  $p \in P$

**Decision Variables**

- $y_{pj}$ : Binary variable that takes the value one if subsolution  $j \in S_p$  of subproblem  $p \in P$  is selected for the recovery plan, and takes the value zero otherwise
- $z_c$ : Binary variable that takes the value one if candidate  $c \in C$  is selected in the recovery plan, and takes the value zero otherwise

**Formulation**

$$\min \sum_{p \in P} \sum_{j \in S_p} c_{pj} y_{pj} \tag{B.1}$$

$$\text{s.t. } \sum_{j \in S_p} y_{pj} = 1 \quad \forall p \in P, \tag{B.2}$$

$$\sum_{j \in S_p} u_{pjc} y_{pj} - z_c = 0 \quad \forall p \in P, c \in C, \tag{B.3}$$

$$\sum_{p \in P} \sum_{j \in S_p} s_{pjck} y_{pj} \leq a_{ck} \quad \forall c \in C, k \in K, \tag{B.4}$$

$$\begin{aligned} y_{pj} &\in \{0, 1\} & \forall p \in P, j \in S_p, \\ z_c &\in \{0, 1\} & \forall c \in C. \end{aligned} \tag{B.5}$$

Equation (B.1) minimizes the cost of the selected subsolutions. Equation (B.2) ensures that exactly one subsolution is selected for each subproblem. Equation (B.3) ensures that a candidate  $c$  is selected if and only if all of the selected subsolutions include such a candidate. Equation (B.4) ensures that the seat capacity of each candidate for each cabin class is satisfied. Finally, (B.5) models the numerical domain of the decision variables.

**B.1. Restricted Model**

Model (B.1)–(B.5) is not practical given that the sets of subsolutions,  $S_p$  for all  $p \in P$ , are exponentially large. Instead, we will consider a restricted version of the model in which we will dynamically enlarge  $S_p$ , generating new subsolutions on demand for each of the subproblems. Generating a subsolution to include in  $S_p$  creates a new variable,  $y_{p,j}$ , in the restricted model (i.e., a column when the packing problem is viewed in its matrix form). Consider a specific subproblem,  $p$ , and let  $j' \notin S_p$  be the index of a new subsolution to be generated. Subsolutions  $j'$  is characterized by parameters,  $c_{pj'}$ ,  $u_{pj'c}$ , and  $s_{pj'ck}$ , which are produced by the corresponding subsolver.

Naturally, not all subsolutions  $j'$  are worth including in  $S_p$ . To determine whether a new subsolution should be added to  $S_p$ , consider the linear relaxation of the restricted model, and let  $\sigma_p$ ,  $\pi_{pc}$ , and  $\gamma_{ck}$  be the dual variables associated with constraints (B.2), (B.3), and (B.4), respectively. With these definitions, we can compute the reduced cost of a new variable,  $y_{pj'}$ , as

$$r_{pj'} = c_{pj'} - \sigma_p - \sum_{c \in C} u_{pj'c} \pi_{pc} - \sum_{c \in C} s_{pj'ck} \gamma_{ck}. \tag{B.6}$$

If  $r_{pj'} < 0$ , we add subsolution  $j'$  to  $S_p$ . On the other hand, if we can prove that there is no subsolution  $j'$  such that  $r_{pj'} < 0$ , then we terminate the column generation procedure. To obtain an integer solution to the packing problem, we solve the restricted packing problem with the generated columns, but enforce integrality. Optimality of the integer packing problem is more challenging and requires using a branch and price approach, which we have not implemented.

**B.2. Subproblems**

Each subproblem in the IrrOptimizer generates the bundle of parameters  $c_{pj'}$ ,  $u_{pj'c}$ , and  $s_{pj'ck}$  for a potentially new subsolution  $j'$ . In the context of column generation, subproblems are modeled as optimization problems where the objective function is to minimize  $r_{pj'}$ . To formalize this idea for a specific subsolver  $p$ , let  $x$  be a vector of all its decision variables and  $\mathcal{X}$  be the set of feasible solutions. With a slight abuse of notation, let  $c_p(x)$ ,  $u_{pc}(x)$ , and  $s_{pck}(x)$  be the mappings from the decision variables of the subproblem to the bundle of parameters that constitutes a new subsolution (note the absence of  $j$  in the parameters, because we are not referring to a specific solution  $j$ , but rather, we are generating it by optimizing over  $x \in \mathcal{X}$ ). Last, let  $r_p(x)$  be the reduced cost of a subsolution as a function of the decision vector  $x$ . With the above definitions, we can define a general subproblem as

$$\begin{aligned} \min \quad & r_p(x) = c_p(x) - \sigma_p - \sum_{c \in C} u_{pc}(x) \pi_{pc} - \sum_{c \in C} s_{pck}(x) \gamma_{ck}. \\ \text{s.t.} \quad & x \in \mathcal{X} \end{aligned} \tag{B.7}$$

For the rotation subsolver,  $x$  is a vector that includes the decision variables associated with the selection of leg candidates and aircraft on-ground events, whereas  $\mathcal{X}$  considers, among others, constraints to ensure turnaround times between consecutive legs using the same aircraft, aircraft flow balance across all airports, and that connected legs are flown by the same aircraft. The cost of a solution to the rotation subsolver,  $c_p$ , includes on-ground event cancellation costs, leg cancellation costs, and departure arrival option costs. Notice that there are no decision variables related to seats, and therefore  $s_{pck}(x) = 0$  for any  $x$  in this subproblem. Note that  $u_{pc}(x)$  is trivially mapped from the decision variables associated with leg candidates.

For the pax subsolver,  $x$  is a vector that includes the decision variables associated with the selection of leg candidates, how the reservations are routed over the selected candidates, and whether a reservation has been canceled. The set  $\mathcal{X}$  considers, among others, constraints for minimum connection times for itineraries with multiple legs and pax flow balance (i.e., they should depart from and arrive at the airports specified in the reservation). The mapping  $s_{pck}(x)$  can be computed from the decision variables associated with routing pax. As above,  $u_{pc}(x)$  is trivially mapped from the decision variables associated with the leg candidate selection. The cost of a solution to the pax subsolver,  $c_p(x)$ , includes reservation cancellation costs, reservation delay costs, and seat (re)assignment costs.

For the crew subsolver,  $x$  is a vector that includes the decision variables associated with the selection of leg

candidates, how the crews are routed over the selected candidates and in which roles (e.g., as flight attendant or cabin manager, for work or deadhead), and whether a trip has been canceled. The set  $\mathcal{X}$  considers, among others, constraints ensuring that enough crew members in each role are assigned to operate the aircraft on each candidate, that they have the qualifications to operate it, and that crews depart from and arrive at their home base. It also considers several legal and business rules about maximum work periods, minimum rest time, maximum delays, and training requirements. The mapping  $s_{pck}(x)$  can be computed from the decision variables associated with routing crews. As above,  $u_{pc}(x)$ , is also trivially mapped from the decision variables associated with leg candidates. The cost of a solution to the crew subsolver,  $c_p(x)$ , includes trip change costs, trip cancellation costs, aircraft change costs along a trip, and duty period usage costs.

Upon solving Equation (B.7) and obtaining a solution,  $\bar{x}$ , the new subsolution,  $j'$ , is constructed as  $c_{pj'} = c_p(\bar{x})$ ,  $u_{pj'c} = u_{pc}(\bar{x})$ , and  $s_{pj'ck} = s_{pck}(\bar{x})$ , and appended to the packing problem if  $r_{pj'} = r_p(\bar{x}) < 0$ .

### B.3. Dual Stabilization

The sequence of dual variable values can have large fluctuations, especially on early iteration of the column generation procedure. This is a well known behavior, and different techniques are available to circumvent this issue. In particular, we focus on the stabilization of dual variable values associated with Constraints (B.3). Let  $\pi_{pc}$  be the dual variables associated with this constraint. We use a proximal term in the objective function of the dual problem penalized by a scalar,  $\rho$ , and centered on  $\bar{\pi}_{pc}$  that is chosen from previous iterations. This center is updated whenever there is sufficient improvement in an iteration of the column generation procedure. In practice, we map back the proximal term in the dual problem back to the primal problem and solve the following mathematical program instead:

$$\begin{aligned} \min \quad & \sum_{p \in P} \sum_{j \in S_p} c_{pj} y_{pj} - \sum_{p \in P} \sum_{c \in C} \left( \bar{\pi}_{pc} \phi_{pc} + \frac{1}{\rho} |\phi_{pc}|_q \right) \\ \text{s.t.} \quad & \sum_{j \in S_p} y_{pj} - x_p = 0 \quad \forall p \in P, \\ & \sum_{j \in S_p} u_{pj} y_{pj} - z_c = \phi_{pc} \quad \forall p \in P, c \in C, \\ & \sum_{p \in P} \sum_{j \in S_p} s_{pjck} y_{pj} - d_{c,k} = 0 \quad \forall c \in C, k \in K, \\ & x_p \in \{0, 1\} \quad \forall p \in P, \\ & y_{pj} \in \{0, 1\} \quad \forall p \in P, j \in S_p, \\ & z_c \in \{0, 1\} \quad \forall c \in C, \\ & d_{ck} \in \{0, \dots, a_{ck}\} \quad \forall c \in C, k \in K. \end{aligned} \quad (\text{B.8})$$

Equation (B.8) extends the formulation presented in (B.1) with variables  $\phi_{pc}$ , which represent the dual proximal term in the primal problem.

### B.4. Passenger Subsolver: Reroute Passengers Formulation

This section presents a formulation for rerouting passengers. The set of leg candidates over which to reroute passengers is fixed and used as input to this model. Candidate selection

for the pax subsolver is a network flow where the leg candidates are nodes and arcs exist for leg connections that are valid. Exactly one candidate is selected for each leg, and thus, the following model indexes on legs instead of candidates. Formally, the formulation is defined by:

#### Sets and Constants

- $r \in R$ : Set of passenger reservations
- $l \in L$ : Set of flight legs (candidates)
- $L_r \subseteq L$ : Set of legs that can be used by reservation  $r$
- $L_r^o \subseteq L_r$ : Set of legs that can be chosen to be the first leg in an itinerary for reservation  $r$
- $L_r^d \subseteq L_r$ : Set of legs that can be chosen to be the last leg in an itinerary for reservation  $r$
- $L_r^l \subseteq L_r$ : Set of legs that can be immediately after leg  $l$  in an itinerary for reservation  $r$
- $k \in K$ : Set of cabin classes (e.g., business or economy)
- $P^r \geq 1$ : Number of passengers in reservation  $r$
- $M_r \geq 1$ : Maximum number of legs on an itinerary for a reservation  $r$
- $S_k^l \geq 0$ : Seating capacity of class  $k$  on leg  $l$
- $C^r \geq 0$ : Cancellation cost of reservation  $r$
- $C^{r,l,k} \geq 0$ : Seat assignment cost; cost of assigning reservation  $r$  to cabin class  $k$  on leg  $l \in L_r$

#### Variables

- $z_r \in \{0, 1\}$ : Binary variable that takes a value of one if reservation  $r$  is canceled
- $y_{r,l,k} \in \{0, 1\}$ : Binary variable that takes a value of one if cabin class  $k$  is used for reservation  $r$  on leg  $l \in L_r$

#### Objective

$$\min \sum_{r \in R} C^r z_r + \sum_{r \in R} \sum_{l \in L_r} \sum_{k \in K} C^{r,l,k} y_{r,l,k} \quad (\text{B.9})$$

$$\text{s.t.} \sum_{r \in R} P^r y_{r,l,k} \leq S_k^l, \quad (\text{B.10})$$

$$z_r + \sum_{l \in L_r^o} \sum_{k \in K} y_{r,l,k} = 1, \quad (\text{B.11})$$

$$z_r + \sum_{l \in L_r^d} \sum_{k \in K} y_{r,l,k} = 1, \quad (\text{B.12})$$

$$y_{r,l,k} \leq \sum_{l' \in L_r^l} \sum_{k' \in K} y_{r,l',k'} \quad \forall l \in L, k \in K, \quad (\text{B.13})$$

$$\sum_{l \in L_r} \sum_{k \in K} y_{r,l,k} \leq M_r \quad \forall l \in L, k \in K. \quad (\text{B.14})$$

Equation (B.9) minimizes the sum of the cost of canceling reservations and the seat assignment cost for a reservation. Moving some passengers to new legs may require upgrades or downgrades to their cabin class. These changes incur a cost. Equation (B.10) ensures that the passengers assigned to a leg do not exceed the leg's seating capacity for a given cabin class. Equations (B.11), (B.12), and (B.13) model network flow. They ensure that a leg is assigned to the start, end, and all intermediary stops of a reservation, if it has not been canceled. Equation (B.14) ensures that the maximum number of allowed legs for a reservation is not exceeded.

### References

- Arikan U, Gürel S, Akturk MS (2017) Flight network-based approach for integrated airline recovery with cruise speed control. *Transportation Sci.* 51(4):1259–1287.

- Artigues C, Bourreau E, Afsar HM, Briant O, Boudia M (2012) Disruption management for commercial airlines: Methods and results for the ROADEF 2009 challenge. *Eur. J. Indust. Engrg.* 6(6):669–689.
- Bisaillon S, Cordeau JF, Laporte G, Pasin F (2011) A large neighbourhood search heuristic for the aircraft and passenger recovery problem. *4OR* 9:139–157.
- Desaulniers G, Desrosiers J, Dumas Y, Marc S, Rioux B, Solomon M, Soumis F (1997) Crew pairing at Air France. *Eur. J. Oper. Res.* 97(2):245–259.
- Fuentes M, Cadarso L, Vaze V, Barnhart C (2021) The tail assignment problem: A case study at Vueling Airlines. *Transportation Res. Procedia* 52:445–452.
- Harvey WD, Ginsberg ML (1995) Limited discrepancy search. Mellish C, ed. *Proc. 14th Internat. Joint Conf. Artificial Intelligence* (Morgan Kaufmann Publishers, San Francisco), 607–613.
- Hooker JN, Ottosson G (2003) Logic-based Benders decomposition. *Math. Programming* 96(1):33–60.
- Jozefowicz N, Mancel C, Mora-Camino F (2013) A heuristic approach based on shortest path problems for integrated flight, aircraft, and passenger rescheduling under disruptions. *J. Oper. Res. Soc.* 64(3):384–395.
- Khaled O, Minoux M, Mousseau V, Michel S, Ceugniet X (2018) A compact optimization model for the tail assignment problem. *Eur. J. Oper. Res.* 264(2):548–557.
- Kohl N, Karisch SE (2004) Airline crew rostering: Problem types, modeling, and optimization. *Ann. Oper. Res.* 127(March):223–257.
- Maher SJ (2015a) A novel passenger recovery approach for the integrated airline recovery problem. *Comput. Oper. Res.* 57(May):123–137.
- Maher SJ (2015b) Solving the integrated airline recovery problem using column-and-row generation. *Transportation Sci.* 50(March): 216–239.
- Parmentier A, Meunier F (2020) Aircraft routing and crew pairing: Updated algorithms at Air France. *Omega* 93(June):102073.
- Peterson JD, Solveling G, Clarke JP, Johnson EL, Shebalov S (2012) An optimization approach to airline integrated recovery. *Transportation Sci.* 46(4):482–500.
- Rossberg A (2019) WebAssembly core specification. Accessed October 15, 2025, <https://www.w3.org/TR/wasm-core-1>.
- Van Winkel JC (2016) The production environment at Google, from the viewpoint of an SRE. *Site Reliability Engineering: How Google Runs Production Systems*, 1st ed. (O’Reilly Media, Inc., Sebastopol, CA), 13–22.
- Zhang D, Yu C, Desai J, Lau H (2016) A math-heuristic algorithm for the integrated air service recovery. *Transportation Res. Part B: Methodological* 84(February):211–236.

---

**Toby O. Davies** is a staff software engineer on Google’s Operations Research team, where his research mainly focuses on improving the CP-SAT lazy clause generation constraint programming

solver, and applying approaches from operations research, constraint programming, and Boolean satisfiability to solve problems inside and outside of Google. He holds a PhD and BSc from the University of Melbourne, Australia.

**Daniel Duque** is a senior software engineer at Google. His work focuses on decomposition algorithms and stochastic optimization. He holds a PhD in operations research from Northwestern University.

**Emily Masten** is a senior software engineer on Google’s Operations Research team and is based in Cambridge, Massachusetts. She holds a BA from Amherst College.

**Tom Tangl** is a senior software engineer on Google’s Operations Research team, where he combines mathematical optimization and metaheuristics with Google’s computing infrastructure to tackle large-scale industrial challenges. His work spans Lufthansa’s schedule recovery, shipping network design, and internal data center optimization. He holds a BSc from the Technical University of Munich.

**Eric Bruneton** is a senior software engineer on Google’s Operations Research team and is based in Paris. He holds a PhD from Institut National Polytechnique de Grenoble, and worked for INRIA (Institut National de Recherche en Sciences et Technologies du Numérique) before joining Google.

**Alejandra Estanislao** worked on the Operations Research team at Google for over 10 years. From optimizing internal operations to leading a team to solve large-scale enterprise problems, she’s worked across many fields including human resources, infrastructure, logistics, and aviation. She holds an MSc from École des Ponts ParisTech and Université Pierre et Marie Curie.

**Jon Orwant** was the director of Google’s Operations Research team. Prior to Google, he was the director of research at France Telecom and chief technology officer of O’Reilly Media. He received his PhD from the Massachusetts Institute of Technology.

**Daniel Bogado Duffner** has worked as a data scientist, software engineer, and product manager on the Operations Decision Support Suite. In his software engineering role, he ensured the proper preparation of the data and rule compliance of the operations processes for the optimizers and as product manager the successful adoption by end users. He holds a MSc from the Federal Institute of Technology Zurich (ETH Zurich).

**Michael Frey** is the product manager for the OPSD tail optimizer at Lufthansa Group. His background includes roles as a flight dispatcher, and duty manager of network operations control, giving him unique insight into user needs and operational complexities. He holds a BSc in business engineering with a specialization in computer science from the Lucerne University of Applied Sciences and Arts.

**Christian Most** is senior director of digital operations optimization at Lufthansa Group, where he leads the development of decision support tools for the group’s operations control centers. He holds an MSc in management and technology from the Technical University of Munich.