Constraint-Specific Recovery Network for Solving Airline Recovery Problems

Niklaus Eggenberg* Matteo Salani Michel Bierlaire

TRANSP-OR – ENAC
EPFL – École Polytechnique Fédérale de Lausanne
CH-1015 Lausanne, Switzerland

Abstract

In this paper, we consider the recovery of an airline schedule after an unforeseen event called *disruption*, making the planned schedule infeasible. We present a modeling framework that allows the consideration of operational constraints within a Column Generation (CG) scheme. We introduce the general concept of *recovery network*, generated for each individual *unit* of the problem, and show how unit-specific constraints are modeled using *resources*. We fully illustrate the concept by solving the Aircraft Recovery Problem (ARP) with maintenance planning, we give some insights into applying the model to the Passenger Recovery Problem (PRP) and we present computational results on real data.

keywords: Airline scheduling, Recovery algorithms, Column generation

1 Introduction

Air travel is one of the most frequently used modes of transportation for both business and leisure. As the airline market is no longer protected both in Europe and in the US, airlines can decide their own routes and fares. This makes it crucial for airlines to manage efficiently their operations to provide a high-quality and reliable service while optimizing their profits.

The strategic and operational decisions of an airline are to choose the destinations to serve, plan routes between them and determine routes for aircraft and crew members, in order to meet technical constraints for aircraft routes and to comply with union of workers constraints for crew pairings. The whole decisional process starts approximately a year before the day of operations.

Unfortunately, the original schedule is rarely executed as planned: in reality, the schedule often becomes infeasible because of unexpected events called *disruptions*, such as bad meteorological conditions, unpredicted maintenance requirements or propagated delays. When a disruption occurs, aircraft, crew and passengers have to be re-accommodated in order to retrieve the initial schedule, which incurs huge costs. In the European airline punctuality report 2007[1], the Association of European Airlines (AEA) reports that for European flights 21.1% of departures

*corresponding author niklaus.eggenberg@epfl.ch

[1] www.aea.be

and 22.3% of arrivals of the European flights are delayed by more than 15 minutes and that the yearly average is increasing. A report released by EuroControl (EUROCONTROL, 2004) studies four scenario-based forecasts of air traffic demand for the next 20 years. In the highest growth scenario, the annual demand rises up to 21 million flights a year with more than 60 airports congested, the top 20 airports being saturated at least $8-10$ hours a day.

Given this forecast it is obvious that a disruption would have deeper operational and economic consequences. This is a clear signal for airlines to develop fast and reliable recovery methods.

Recovery decisions are taken at the Operations Control Center (OCC). The challenge is to determine a new route, called a *recovery scheme*, for each plane, crew member and passenger within a certain makespan $[0, T]$ called the *recovery period*; the disruption is assumed to occur at time 0 and the original schedule must be recovered within time T.

Each *unit* (that is, a plane, a crew or a passenger) is associated with an *initial state*, which is the first ground location of the unit after the disruption, time t_0 being the earliest time at which the unit is ready for action at this location. A *final state* is the required location of a unit at time T for the original schedule to be recovered.

The recovery scheme is then defined as a succession of flights connecting the initial to the final state for each aircraft, crew and passenger.

The options for OCC operators are to delay or cancel flights, swap planes, reassign crew and reroute or cancel passenger itineraries. For canceled itineraries, the airline must possibly book flight tickets with other companies. The recovery costs are determined by the airline, which estimates the cancellation cost for a flight, the delay cost (typically measured in dollars per minute) and other operational costs such as plane and crew swapping costs.

The underlying difficulty of recovery is to provide recovery schemes that satisfy all operational constraints: plane recovery schemes must satisfy maintenance constraints, crew recovery schemes must comply with all union constraints, and the individual passenger recovery schemes must ensure arrival at the final destination within some delay limits, which depends on the length of the original itinerary. We refer to these constraints as *unit-specific constraints*. In addition to the unit-specific constraints, operators must ensure that the aggregated solution of the chosen recovery schemes is feasible, i.e. that neither crew nor passengers are assigned to a canceled flight, connection time for crew and passengers between two consecutive flights are sufficient, etc. The constraints ensuring that the combination of the recovery schemes is feasible are called *structural constraints*.

This paper introduces an original and flexible model to solve airline recovery problems: the *constraint-specific recovery network*. The corresponding model is then integrated within a Column Generation (CG) algorithm that combines the individual recovery schemes in an optimal way satisfying all structural constraints. We then illustrate in detail its application to a less studied recovery problem, the Aircraft Recovery Problem (ARP) with maintenance planning, in which maintenance constraints are explicitly taken into account in the optimization process. Moreover, we briefly illustrate its application to the Passenger Recovery Problem (PRP). After a discussion on instance complexity, we provide computational experiments on real-world data.

The structure of the paper is as follows: section 2 summarizes the state of the art in disrupted airline schedule recovery; section 3 presents a framework based on the constraint-specific recovery network with definitions, algorithms and an appli-

cation to the ARP; section 4 provides a computational study for the ARP with maintenance planning on real data and section 5 reports on preliminary results for the PRP; finally, in section 6, we present some concluding remarks and future research directions.

2 Literature survey

The field of recovery algorithms was developed mainly in the past 10 years following the growth of airline networks and consequently the increase of irregularities: for each 1% increase in airport traffic it is estimated that there will be a corresponding 5% increase in delays (Schaefer et al., 2005) and additional costs (Shavell, 2000).

For general surveys on airline scheduling in the recovery perspective, we refer to Kohl et al. (2007) and Clausen et al. (2009).

Teodorvić and Gubernić (1984) are the pioneers of the ARP. Given that one or more aircraft are unavailable, the objective is to minimize the total passenger delay by flight re-timing and aircraft swaps. The algorithm is based on a branch-and-bound framework where the relaxation is a network flow with side constraints.

There are few contributions in which maintenance operations planning is considered in combination with the ARP. In Stojković et al. (2002) the authors consider maintenance constraints and provide a real-time algorithm that does not affect routing decisions. Only Sriram and Hagani (2003) consider maintenance and routing decisions together, but aircraft maintenance checks can be performed only during the night. In an unpublished report, Clarke (1997) introduces a model for the ARP enforcing maintenance operations within a given time slot but, in the reported computational tests, all the flights are constrained to be operated either on time, with 30 minutes delay or are canceled, restricting drastically the degrees of freedom of the algorithm and thus the overall complexity of the problem.

However, the literature contains several works on different aspects of the recovery problem. Many of them are based on a multi-commodity flow problem solved on a time-band network, see for example Jarrah et al. (1993), Yan and Yang (1996), Yan and Young (1996), Argüello et al. (1997), Wei et al. (1997), Yan and Lin (1997), Yan and Tu (1997), Thengvall et al. (2000), Bard et al. (2001) or Thengvall et al. (2001).

Jarrah et al. (1993) use two separate approaches to the ARP: cancellation and re-timing. The problem is modeled with a time-line network and three methods are reported: the successive shortest path method for cancellations, and two network flow models for cancellations and re-timings.

Yan and Lin (1997) and Yan and Tu (1997) are related to the same underlying model, which is a time-line network in which flights are represented by edges. The network has position arcs corresponding to the potential shortage of an aircraft. The possibility of flight re-timing is modeled by several arc copies. In Yan and Lin (1997) an instance of 39 flights is solved. In Yan and Tu (1997) the authors solve larger instances, up to 273 flights, within a small optimality gap and below 30 minutes of computation.

Argüello et al. (1997) and Bard et al. (2001) use a time-band model to solve the ARP. In the former article the authors propose a fast heuristic based on a randomized neighborhood search. The latter article presents a heuristic based on an integral minimum cost flow in the time-band network. Furthermore, the method proves to be effective for some medium-sized instances up to 162 flights serviced by 27 aircraft.

An extension to the network model of Argüello et al. (1997) is presented by Thengvall et al. (2000). The authors present a model in which deviations from the original schedule are penalized in the objective function and they allow human planners to specify preferences related to recovery operations. Computational results are presented for a daily schedule recovery of two homogeneous fleets of 16 and 27 aircraft. Disruption scenarios are simulated grounding one, two or three planes.

Thengvall et al. (2001) introduce a multi-commodity flow model based on a time-band network to solve the ARP after a hub closure. Computational results for a fleet with 332 aircraft divided into 12 fleets and 2921 flights are provided. Computational times remain below 450 seconds for instances with up to 1434 flights.

Yu et al. (2003) introduce a decision aid algorithm (CALEB) tested on Continental Airlines data. The authors test their algorithm on probably the worst day ever for aviation, namely September 11th 2001. They show impressive results on how fast the return to normal schedule is achieved when such a severe disruption happens. The estimated savings for 9/11 are up to $\$29,289,000$, almost half coming from avoided flight cancellations.

Kohl et al. (2007) give a survey of the previous works on airline scheduling and schedule-recovery. They also develop a *crew solver* and describe a prototype of a multiple-resource decision-support system (the Descartes project), which includes independent algorithms to solve the ARP, CRP and PRPs.

The literature lacks papers that deal with integrated models combining aircraft, crew or passenger recovery problems. The only relevant article we could find on integrated aircraft and passenger recovery is Bratu and Barnhart (2006). The authors present an explicit approach, leading to mixed integer models with an exponential number of constraints, as all possible passenger itineraries are explicitly modeled. To control the exponential size of the model, only itineraries with up to two legs are considered and flight re-timing is restricted to a time window around the original departure time. The main drawback of this approach is that routing decisions are taken after the passenger routing, which implies that flight capacities are not correctly determined at the passenger recovery stage.

Interestingly, even for the airline scheduling problem, where no final location is imposed in order to recover a baseline schedule, only a few works on integrated models exist. We refer to Clausen et al. (2009) for a survey on integrated approaches for both scheduling and recovery.

Mercier and Soumis (2007) present an algorithm based on the Benders' decomposition for solving simultaneously the crew pairing and the aircraft routing problems. The combination of the two problems is done using some linking constraints, imposing minimum connection times for crew depending on aircraft departure times.

Weide et al. (2008) extend the model of Mercier and Soumis (2007) and develop a CG framework similar to that of Cordeau et al. (2001). Indeed, the original combined formulation is divided into two subproblems corresponding respectively to the original crew pairing and aircraft routing problems. Each of these subproblems is then solved at the pricing phase using dual information of the master problem combining the two formulations. The difference is that all possible connections for crew are explicitly enumerated in the master formulation. This methodology cannot be applied to recovery problems, since the connections cannot be explicitly enumerated because of flight re-timing.

The common weakness of multi-commodity flow models for disruption recovery presented in the literature is that they are not appropriate to take into account exact

delays and unit-specific constraints (see Bard et al., 2001): on the one hand, if the model aims at describing exactly the recovery problem at each point in the network, then exponentially many constraints (or variables) are required. On the other hand, approximating delays and constraints satisfaction leads to sub-optimal methods. Using lower bounds may lead to unfeasible solutions or to cost underestimations; using upper bounds may lead to discard feasible solutions.

The main advantage of the constraint-specific recovery network model we present in this paper is that structural and unit-specific constraints are separated and checked independently. Indeed, all unit-specific constraints, modeled as resources that are consumed, are handled in each unit's recovery network, whereas the structural constraints are considered when combining the different recovery schemes. When embedded in a CG scheme, the recovery networks are used to solve the pricing problem for generating new promising recovery schemes, involving the computation of exact resource consumptions.

As a final comment to our literature overview, we note that the largest solved instance reported (Thengvall et al., 2001) involves up to 1434 flights and 332 aircraft. The tests we perform on the ARP in section 4 show that the difficulty of an instance does not directly depend on the number of flights and/or aircraft. Instead, a possible way to characterize the complexity of a problem is the ratio flights/planes. This ratio represents the average length of a recovery scheme, which is directly associated with the combinatorial nature of the recovery problem. In our study we solve instances with a flights/planes ratio up to 18.4, while the largest instances of Thengvall et al. (2001) have a ratio of 4.3, and the biggest ratio reported in the survey of Clausen et al. (2009) equals 7.2.

3 The constraint-specific recovery network model

In this section we present the constraint-specific recovery network, or recovery network model, which can be seen as an extension of the time-band model by Bard et al. (2001).

Each unit (that is, a plane, a crew member or a passenger) is associated with a specific *recovery network*, which is a set of nodes and arcs, such that each possible recovery scheme of the unit corresponds to a path. For each unit p, let F_p be the subset of flights and S_p be the subset of final states that the unit is allowed to cover within the entire sets of flights and final states F and S.

In addition, each unit-specific constraint is modeled as a *resource* and an associated resource limit. For example, aircraft have limits on the consecutive flown hours, crew members have limits on the duration of a duty and passengers have limits on the delay of their itinerary. A resource is either consumed (e.g. by flights) or renewed (e.g. by maintenances). Let r identify a resource and u^r be the associated limit. The vectors of all resources and limits are denoted by R, and U, respectively.

Special resource limits are associated with each final state to ensure the feasibility of the original schedule after time T.

A unit's recovery network is based on a two-dimensional coordinate system (A, t). The discrete horizontal A-axis represents the location a of each airport. The vertical t-axis represents time and is discretized with time intervals of equal size. A node is characterized by a pair $\{a, t\}$ in the coordinate system. We distinguish three types of nodes. The unique *source node* $\{a, t_0\}$ corresponds to the unit's initial state. *Sink nodes* $\{a, T\}$ correspond to the possible final states that the unit is allowed to cover. Finally, all the other nodes are *transition nodes* and are denoted $\{a, t\}$. An arc $(\{a, t\},\{a', t'\})$ connects two nodes and represents an *action* of the unit.

We distinguish four types of arcs $(\{a,t\},\{a',t'\})$ in the networks, corresponding to four different actions: a *flight arc* represents a flight performed by the unit between airports a and a'; a *renewing arc* in which the unit performs a renewing operation *before* the flight; a *termination arc* is a vertical arc connecting the source node or a transition node $\{a,t\}$ to a sink node $\{a,T\}$. A *renewing termination arc* is a termination arc for which a renewing operation is performed before the sink is reached. Flight and renewing arcs are created only if activity and renewing slots are available at the corresponding airport. Note that several arcs associated with the same flight may exist, each defining a different departure time or action. Each arc incurs a cost that is determined by the corresponding action and potential additional costs such as delay costs or unit swapping costs.

This modeling scheme is sufficiently general and flexible to represent a great variety of situations. We discuss in the rest of the section how to make this modeling framework operational.

3.1 Recovery network generation and preprocessing algorithms

The recovery network generation algorithm for each unit is a dynamic programming algorithm where the nodes and arcs are created iteratively. For a formal definition of the notation see Appendix A. See Appendix B for a detailed description of the algorithm dedicated to the ARP.

Each node is associated with a label h^r, for each resource r, modeling one unit-specific constraint and corresponding to the consumed resource when the node is reached. As several different paths may reach the same node, we compute for each node the upper and lower bounds $\overline{h}^r$ and $\underline{h}^r$ on the resource consumption. A node j is said to be feasible if and only if $\underline{H}_j \leq U_j$. Accounting for bounds instead of real resource consumptions allows encoding at construction time of all feasible recovery schemes plus, possibly, some infeasible ones. We identify a node j with the associated labels by $[\{a,t\},\{\underline{H}_j,\overline{H}_j,U_j\}]$.

The source node is described by $[\{a,t_0\},\{H_0,H_0,U_0\}]$, where H_0 is the deterministically known vector of initial resource consumptions. The sinks are described by $[\{a,T\},\{\underline{H},\overline{H},U_T\}]$, where U_T is the vector of maximal allowed resource consumptions at the sink.

A feasible recovery scheme is thus a path from the source node to a sink node such that the maximal allowed resource consumption is never exceeded.

The generation algorithm can be described as follows:

1. initialize the node set $S = [\{a,t_0\},\{H_0,H_0,U_0\}]$;
2. for each non-explored node $[\{a,t_j\},\{\underline{H}_j,\overline{H}_j,U_j\}]$, consider the four arc types for flights in F_p departing from a and final states in S_p associated with a;
3. for all arcs check that the resource consumption at destination is respected, create the destination node and add it to S if it does not exist, or update the resource consumptions;
4. go to step 2.

The discretization of time guarantees a pseudo-polynomial number of nodes (Bard et al., 2001). Additionally, we limit the feasible extensions of step 2: a flight has a maximal allowed delay τ and a maximal waiting time before take-off ψ. The parameters are used both for flight and renewing arcs. In addition, we consider the resource consumption upper bound $\overline{H}$ and we introduce a parameter ρ which

sets a minimal percentage of resource consumption to permit a renewing operation to take place. We add a parameter Γ, which sets the earliest time for termination arcs: at node $\{a, t\}$, we must have $t \geq \Gamma$ for the arc to be created.

The preprocessing phase follows the generation of the networks in order to remove proven sub-optimal or infeasible arcs and nodes. We compute, for each node, the value $\underline{h}^r_{sink}$ of a shortest path with respect to resource r from node $[\{a, t\}, \{\underline{H}_j, \overline{H}_j, U_j\}]$ to a sink. $\underline{h}^r_{sink}$ corresponds to the minimal necessary resource potential to reach the sink from node $\{a, t\}$. If for some resource r, $\underline{h}^r_{sink} + \underline{h}^r_j > u^r_T$, the maximal allowed consumption at the sink is exceeded: there is no feasible path from the source to the sink traversing the node. If none of the sinks is reachable, no feasible path traverses the node, which can be removed.

Finally, all nodes (except the source) that have no predecessor and all nodes (except the sinks) that have no successor are removed from the network. Each time a node is removed, all ingoing and outgoing arcs are removed as well.

At the end of the preprocessing phase, each unit's recovery network contains only nodes belonging to at least one feasible path, i.e. at least one feasible recovery scheme for the unit. Notice that some unfeasible paths may remain in the recovery networks, as we used upper and lower bounds on resource consumption.

3.2 Illustration of ARP with maintenance planning

The ARP with explicit maintenance planning is becoming increasingly relevant for applications: although an extension of an aircraft maintenance limit can be obtained through written permission from the authorities, doing so is not a regular occurrence. Indeed, any airline that continually asks for extensions could be subject to an audit from the authorities. Airlines usually prefer to operate under the Joint Aviation Requirement for Operations (JAR-OPS) rules to avoid additional audits at all costs.

In the ARP a unit is an aircraft and the unit-specific constraints are maintenance constraints. The resources enforcing maintenance are the maximal allowed flight hours, modeled with label h^{FlH}, the maximal number of take-offs and landings, modeled as h^{ToL}, and the maximal absolute time elapsed between two maintenances, denoted as h^{Hrs}. The renewing operations are clearly maintenances. For clarity, we rename the renewing and renewing termination arcs as *maintenance* and *maintenance termination* arcs.

The ARP aims at assigning a recovery scheme to each aircraft such that the original schedule is recovered at T, i.e. that each final state is reached by an aircraft able to carry out the initial schedule after T; the global solution of the ARP is called a *recovery plan*. The allowed decisions are delaying or canceling flights, swapping aircraft and re-assign maintenances; early departures are not allowed.

The following example illustrates the ARP. In Table 1 we have a schedule for planes p_1 and p_2. We consider the number of take-offs and landings as the single resource h^{ToL}, with an upper bound of $u^{ToL} = 20$ and the initial consumption $h^{ToL}_0 = 6$ for p_1 and $h^{ToL}_0 = 10$ for p_2, corresponding to the state of each plane when the disruption occurs: plane 1 has already performed 6 take-offs and landings, and p2 has performed 10. At time $t_0 = 0905$, when p_1 lands in AMS, an unplanned maintenance of two hours is enforced on p_1 because of problems during the landing phase. This leads to a disruption as the schedule cannot be implemented as planned: p_1 cannot take off to MIL at 1000, as it is not ready for take off before 1105.

The initial state for p_1 is represented by source node $[\{\text{AMS},0905\}, \{20, 20, 20\}]$ ($h^{ToL}_0 = 20$ because maintenance is enforced). The source node of p_2 is $[\{\text{AMS},1000\}, \{6, 6, 20\}]$,

as p_2 is active until 0930 and requires, say, 30 minutes to prepare for the next flight. We assume that we want to recover the disrupted situation by $T = 1800$. As both planes will undergo maintenance during the night, resource consumption capacity limits at T are $h_T^{ToL} = U^{ToL} = 20$. We thus define two sink nodes $[\{\text{BCN},1800\}_T, \{0, 20, 20\}]$ and $[\{\text{GVA},1800\}_T, \{0, 20, 20\}]$. Moreover, we assume a homogeneous fleet, i.e. p_2 can cover the flights initially scheduled for p_1 and cover both final states, and vice versa.

A possible recovery plan is given is Table 2, consisting in swapping the two planes: flights F2, F3 and F4 are assigned to plane p_2 and flight F6 to plane p_1. The resource consumption constraints at the final states are clearly satisfied, as p_1 reaches the final state $[\{\text{BCN},1800\}_T, \{0, 20, 20\}]$ with $h^{ToL} = 2 < 20$ and plane p_2 reaches final state $[\{\text{GVA},1800\}_T, \{0, 20, 20\}]$ with $h^{ToL} = 18 < 20$.

To create the recovery network of p_2, we create the source node $[\{\text{AMS},1000\}_0, \{12, 12, 20\}]$ and apply Algorithm 1, with the set of flights $F_{p_2} = \{F2, F3, F4, F6\}$ (F_1 and F_5 are already covered), and the set of sinks $S_{p_2} = \{[\{\text{GVA}, 1800\}, \{0, 20, 20\}], [\{\text{BCN}, 1800\}, \{0, 20, 20\}]\}$. Activity slots and minimum connection time are $O_a = [0700, 1800]$ and $mct_a = 30$ for all airports a; the only maintenance slot is $M_a = [0900, 1800]$, with maintenance duration $d_{m_a} = 60$ minutes for $a = \text{AMS}$. The generated recovery network is shown in Figure 1 (we remove flights delayed by more than $\tau = 240$ minutes).

The recovery network enables us to identify the possible recovery schemes. Each of them corresponds to a path from the source to a sink. In Figure 1 we identify eight feasible paths from the source to a sink, i.e. eight different recovery schemes; the path corresponding to the recovery scheme of plane p_2 in Table 2 is the succession of the nodes $\{\text{AMS},1000\}_0$, $\{\text{MIL},1200\}$, $\{\text{BCN},1410\}$, $\{\text{GVA},1620\}$ and $\{\text{GVA},1830\}_T$. The succession of flight arcs corresponds to flights F2, F3 and F4, respectively.

In this example, there is no maintenance termination arc, as there is no maintenance slot at GVA nor at BCN, the locations of the final states. Notice that there are several arcs, both flight or maintenance, associated with the same flight at different times.

3.3 Column Generation algorithm

In this section, we briefly describe a CG algorithm designed to exploit the constraint-specific recovery networks. For a detailed description of CG algorithms see Desaulniers et al. (2005), Lübbecke and Desrosiers (2005) or Vanderbeck (2005a). Let Ω be the set of all possible single-unit recovery schemes. The structural constraints are modeled in the following Master Problem (MP):

$$\min z_{MP} = \sum_{r\in\Omega} c_r x_r + \sum_{f\in F} c_f y_f \tag{1}$$

$$\sum_{r\in\Omega} b_r^f x_r + y_f = 1 \qquad \forall f \in F \tag{2}$$

$$\sum_{r\in\Omega} b_r^s x_r = 1 \qquad \forall s \in S \tag{3}$$

$$\sum_{r\in\Omega} b_r^p x_r \le 1 \qquad \forall p \in P \tag{4}$$

$$x_r \in \{0, 1\} \qquad \forall r \in \Omega \tag{5}$$

$$y_f \in \{0, 1\} \qquad \forall f \in F \tag{6}$$

Each recovery scheme r has a cost c_r, which can be uniquely determined by operational and delay costs, and is associated with a binary variable x_r that is equal to one if it is considered in the solution, 0 otherwise. A recovery scheme is described by the binary coefficients b_r^f, b_r^s and b_r^p, which form a column of the constraint matrix. Those constants take value one if scheme r covers flight f, ends with final state s and is associated with unit p, respectively.

Constraints (2) ensure that each flight is either covered or canceled; constraints (3) ensure that each final state is covered, i.e. the schedule is recovered at time T. Finally, constraints (4) ensure that each unit is associated with at most one recovery scheme. Constraints (5) and (6) ensure the integrality of the variables.

The pricing problem of finding negative reduced cost columns corresponds to finding the (feasible) recovery scheme with minimal reduced cost. We thus have to solve a pricing problem for each unit and determine which recovery scheme is optimal. Recall that a unit's recovery scheme corresponds to a path in its associated constraint-specific recovery network. To solve the pricing, we solve a Resource Constrained Elementary Shortest Path Problem (RCESPP) on each unit's recovery network to determine the minimal reduced cost column.

To solve the RCESPP, we use the algorithm proposed by Righini and Salani (2006). The algorithm creates labels associated with nodes; a label encodes a feasible path to reach the node it is associated with. If several labels are active at a same node, it is possible to eliminate some of them that are dominated, i.e. that cannot lead to an optimal path. If a label is not eliminated by dominance, it is extended through all feasible arcs (j, k) to a new label at node k. The optimal solution is encoded by the label with lowest cost at the sinks.

The same flight can be associated with different arcs in the network to model delay decisions. Although the network is acyclic, the elementarity of the used flights must be enforced, as it cannot be ensured by the only cost minimization objective of a resource constrained shortest path. To enforce elementarity, we use the method of Beasley and Christofides (1989), adding a dummy vector of binary resources, one for each flight. To tackle the computational effort issued by the additional elementarity constraint we exploited the Decremental State Space Relaxation (DSSR) technique introduced by Righini and Salani (2008).

As most of the unfeasible paths in the recovery networks are removed in the preprocessing phase, the number of labels generated by the pricing algorithm is reduced. Furthermore, the available bounds on the minimal required resource consumption to reach a sink from each node allow to detect infeasible paths earlier.

3.4 Implementation issues

The algorithm is implemented in C++ exploiting BCP, an open source framework implementing a Branch&Cut&Price algorithm, provided by the Computational Infrastructure for Operations Research (*COIN-OR*, 2008) project. Tests are run on a computer with a 2GHz processor and 2GB memory.

To control the number of non-dominated labels in the RCESPP algorithm, resource consumption is discretized as we do for time: resource consumptions falling into the same interval are considered as equivalent.

We introduce a logarithmic resource discretization controlled by a parameter θ. It corresponds to the number of logarithmic intervals resources are divided into, with the length of the intervals being proportional to $\log(\theta)$. The intervals for resource h are denoted by I_j^h, $j = 1 \cdots \theta$. The idea of the logarithmic discretization is that resource-renewing operations are unlikely to occur for low resource consump-

tions. As a consequence of resource discretization dominance criteria are applied by comparing the interval indices j instead of the exact resource consumption; all pairs of labels falling into the same interval are compared based on the remaining resources only, so that more labels are dominated with respect to the exact dynamic programming algorithm.

Notice that in a linear discretization, two labels might belong to the same or to different intervals for increasing values of θ. In the logarithmic case however, this does not occur: the example in Figure 2 illustrates this phenomenon for increasing θ with linear interval lengths on the left and with logarithmic lengths on the right for a resource h and resource limit $U_h = 100$. We see that the two labels corresponding to resource consumption 49 and 51, represented by $\updownarrow$, always fall in different intervals in the logarithmic case for $\theta \geq 2$. In the linear case however, they are in the same interval for odd θ and they are not for even θ, as long as the gap between the two values is larger than $\frac{1}{\theta}$. Figure 2 also shows the saturation effect in logarithmic discretization when θ grows too large, i.e. intervals for big θ become infinitely small.

In the implemented algorithm, CG is done only at the root node of the search tree: we solve the linear relaxation of the root node to optimality and obtain a valid lower bound. An integral solution is then obtained by branching without generating new columns. The obtained algorithm is therefore an optimization-based heuristic with an a posteriori guarantee on the optimality gap.

Moreover, CG is known as a *primal method*: primal feasibility of the linear relaxation of the Restricted Master Problem (RLMP) is guaranteed, whereas the feasible dual vector is searched by adding valid dual cuts in the dual space. Indeed each cut corresponds to a feasible column in the primal space. It is known that for an efficient implementation of CG methods (see for example Vanderbeck, 2005b), one needs to provide a relevant set of columns to obtain a good estimation of the dual vector and to prove its optimality at the end of the generation.

As the dual vector estimation during the early iterations of the method is poor, it is common practice to solve the pricing problem heuristically to produce quickly negative reduced cost columns. We derive three pricing heuristics from the exact dynamic programming method using two relaxations. In the first relaxation we keep the elementarity constraint during the construction of partial paths but we relax it in dominance tests. This increases the possible number of dominated labels, and thus accelerates the algorithm.

The second method consists in ordering the labels by increasing reduced costs and bounding the number of active labels for each node by a constant. Thus, thanks to time discretization, the resulting heuristic is polynomial in time and space since both the number of nodes and the number of labels are bounded by a polynomial function.

We combine the two relaxations to obtain a third and fast heuristic applied first. If this heuristic fails in finding new columns, we apply the heuristic with a fixed number of labels for each node. The heuristic in which we relax the elementarity dominance criteria is applied next. If none of the heuristic methods returns a column with negative reduced cost, we resort to exact pricing.

Finally, we add to the master problem all the new columns with negative reduced cost found in the heuristic phase to accelerate the convergence of CG; useless columns are then removed from the LP by variable fixing based on reduced costs.

3.5 Illustration of PRP

In the PRP, all passengers must be brought to their final destination within a maximum delay limit that depends on the original itinerary length. Passengers can either be rerouted on the available flight network or canceled, which implies the booking of a ticket with another airline. In the proposed framework, each unit corresponds to a single passenger or a group of passengers *with the same itinerary*, and the unit-specific constraints are the maximal allowed delay for a passenger and the maximal number of flights to which a passenger can be allocated. In our case, recovery schemes with more than η flights are discarded, where η is the parameter limiting the maximal number of flights. The input of the PRP is the output of the ARP, i.e. a feasible recovery plan; the objective is to operate the maximal number of passengers to their final destination while minimizing the total delay and canceling costs. In the recovery network, we create an additional *cancellation* termination arc from the source to the sink to model the itinerary cancellation.

The structural constraints of the PRP are:

1. the number of assigned passengers cannot exceed the aircraft capacity;
2. each passenger must be assigned to exactly one recovery scheme.

Finally, since the network is acyclic, the pricing problem is a Resource Constrained Shortest Path Problem (RCSPP), i.e. elementarity constraints do not need to be enforced as with the ARP. The DSSR technique we have adopted for pricing takes advantage of this. In section 5 we report on computational results for the PRP.

The general concept of units allows the application of the modeling framework also to crew recovery: a recovery scheme for a crew is also a succession of flights and we can define a unique initial state and a final state for each crew corresponding to its base. Swaps within crew teams might be allowed, and then the subsets S_p are used to model possible crew swaps at the end of the recovery period. Moreover, crew-specific constraints can be modeled as resources: contract constraints are measured in terms of maximal duty time without breaks, maximum time spent away from a base, etc. Renewing operations are thus rest periods. Thus, also for crew recovery, the same method and algorithms hold.

4 Computational results for the ARP

The data used in the computational tests have been obtained from Thomas Cook Airlines (TC). TC is a medium-size airline relying on a heterogeneous fleet of 16 aircraft and operating around 250 flights a week. In our instances, the number of flights varies from 40 to 760 flights; instances with more than 250 flights are artificially built from the real ones by duplicating schedules, assigning each copy to a different fleet.

The largest real instance in our tests has 242 flights and 16 aircraft, while the largest instance has 760 flights and 100 aircraft. These instances are smaller than the largest reported instance in literature which has 1434 flights and 332 aircraft (Thengvall et al., 2001). However, our largest instance has a flights/planes ratio of 18.4 compared to a ratio of 4.3 for Thengvall et al. (2001) and 7.2. for the biggest ratio of an instance reported in the survey of Clausen et al. (2009).

As no disruption is provided in the data, we use the TC schedule of May 2006 and simulate some disruption scenarios using the following experimental setup:

- size of the fleet concerned by the disruptions: 5, 10 and 16 aircraft;

- recovery period $\mathtt{T}$: from 1 to 7 days;
- $0-6$ delayed planes;
- $0-6$ grounded planes;
- airport closures: 0, 300, 500 minutes or 3×100 minutes;

The instances are divided into two classes: the first is composed of more than 20 instances using groundings, delays and combining both. The name of the instance is related to its size: `xD_yAC`, where `x` is the number of days considered in the recovery period, `y` is the number of aircraft. The second class is derived from a hub-and-spoke situation (where the hub airport is Denver) with 10 aircraft and 36 flights for which we generate 12 different disruption scenarios: scenarios consider either delayed planes only, grounded planes only, a mixture of delayed and grounded planes or airport closure(s). The name of each instance describes the simulated disruption: we denote the number `n` of grounded planes by `ngrd` and the number `m` of delayed planes by `mdel`. When the name of the instance is followed by `_R`, the initial resource consumption is randomly generated.

Generation and preprocessing algorithms. The statistics of the recovery network generation and preprocessing algorithms on 49 different instances with various sets of parameters are as follows: the average computation time for both generation and preprocessing of the networks is lower than 0.3 seconds. In the biggest network we get 14,467 arcs and 3,634 nodes. The generation time is 1.344 seconds and the preprocessing phase is done in 0.891 seconds.

In average over the 49 instances, the preprocessing reduced by 20.53% the total number of arcs (20.22% for flight arcs and 21.96% for maintenance arcs) and by 23.64% the number of nodes. In the best case, it removes up to 63.64% of the arcs and up to 66.40% of the nodes. The number of nodes and arcs is directly linked with the computational effort of the pricing problem, since it is a dynamic programming algorithm that explores each node and extends them through all outgoing arcs. These results thus show that the use of recovery networks is useful in order to save computational time, since around 20% of the computation time is saved at each call of the pricing algorithm.

Interestingly, the more restrictive the time discretization parameters, the fewer nodes (and arcs) are removed at the preprocessing phase. Indeed, when time discretization intervals increase, a larger number of paths traverse a same node with respect to smaller intervals; it is therefore unlikely that none of these paths corresponds to a feasible recovery scheme, explaining why less nodes are removed.

Solvable instances. Table 3 shows the size and the required computation time for some representative instances. When a schedule can be carried out almost as initially planned, the algorithm solves the problem to optimality at the root node within 1.0 second. Only bigger instances require branching, which drastically increases computation time. We see that the number of flights is not crucial: it is more difficult for the algorithm to solve instance `7D_16AC` with only 242 flights than `2D_100AC` with 760 flights. Comparing the flights/planes ratio, we see however that instance `7D_16AC` has a ratio of 15.1, whereas instance `2D_100AC` has a ratio of only 7.6.

We report that with the standard settings of parameters, instance `7D_16AC` fails because of memory excess. The results presented for this instance are obtained with more restrictive values for the delay and inactivity time parameters (τ and ψ).

Impact of disruptions. To measure the impact of a disruption on a solution, we use the Denver data (10 aircraft, 36 flights) with 12 different scenarios. The instances `3x100` and `1x300` simulate a closure of the hub airport, i.e. Denver. In the first instance, Denver airport is closed during 3 periods of 100 minutes, with a gap of 100 minutes between each closure. The second instance simulates a longer closure of 300 minutes in a row. We also simulate a storm affecting several local airports. In instance `Storm1` four airports are closed for 300 minutes and in instance `Storm2` the same airports are closed for 500 minutes.

Table 4 shows the results for the different instances. The second line reports the number of flights directly involved by the disruption without any forecast on disruption propagation; it thus represents the minimum number of flights on which the planner must take a recovery decision. It is evident (see for example instance `6grd`) that the final number of affected flights is more important because of delay propagation: 18 flights are canceled or delayed while the minimum number is estimated to be 16.

We see from Table 4 that a grounded plane more often incurs flight cancellation than a delayed plane. This follows intuition, as when a plane is grounded, the original schedule must be recovered with one plane less than when a plane is simply delayed and can still operate. In the instances combining grounded and delayed planes, the effects of cancellations due to grounded planes and delays incurred by delayed planes are combined. This is a direct consequence of the network density, meaning that if there are not enough available planes, the other planes' schedules do not permit the insertion of supplementary flights.

In general, we see that the bigger the number of directly affected flights, the higher the delay or cancellation rates, except for the two Denver closure scenarios. Even though instance `3x100` has more affected flights, the solution is better than for `1x300`. The explanation is that the closure is split and covers more take-offs and landings at Denver, but the slots between closures allow for planes to leave and start rotations from Denver. Landing and take-off at other airports are then possible even during the hub closure. In the `1x300` instance, planes located at Denver when the closure occurs must wait the whole 300 minutes before taking off. We see from Table 4 that the closure of the hub airport has, as expected, dramatic impact due to delay propagation. Surprisingly, for the storm instances, all the flights are covered, inducing however huge delays.

Sensitivity to parameters. The sensitivity tests with respect to the different parameters show that the approach is stable. Increasing time discretization significantly decreases computational time both at generation and pricing phase. The resulting loss of optimality is, however, rather small.

One sensitive parameter is the estimated delay cost per minute $\mathtt{cd}$, which determines the trade-off between delays and flight cancellations. The lower the delay cost, the more the algorithm tends to cover all the flights regardless of the produced delay. Reversely, if the delay cost is high, the recovery plan avoids delays, canceling more flights if necessary. Since our approach does not consider repositioning flights, a single cancellation rarely occurs alone.

Finally, we test the logarithmic resource discretization against the linear one. Solutions computed with the logarithmic resource discretization are better than those obtained by the linear one for a number of intervals up to $\theta = 10$. For $\theta > 10$, solutions obtained from the linear discretization are globally better, but the improvement is not necessarily homogeneous as discussed in section 3.4.

A low number of discretization intervals is favorable to control the memory

usage, making the logarithmic resource discretization more attractive.

Maintenance scheduling. We want to test the added value of maintenance planning. To this extent we compare different recovery approaches that can be implemented at Operations Control Centers (OCCs). The first approach is to neglect maintenance planning, focusing only on resource consumption limits. We consider three possible resource extensions: 5%, 10% and 20% extension on U_h. We refer to these approaches by NM+y% (for No Maintenance), where y is the allowed percentage of consumption excess.

The second approach, which is probably closer to human planner behavior, is to schedule a maintenance as soon as resource consumption gets critical. We refer to it as the Greedy Maintenance (GM) algorithm. This approach is achieved by setting parameter ρ to a high value (ρ is set to 0.9 in our tests, meaning maintenance can be performed when at least 90% of the resource is consumed). Finally, the third approach is Maintenance Optimization (MO), where the algorithm plans the maintenances in an optimal way ($\rho = 0$).

We compare the approaches on a set of 10 instances derived from instance 4D_10AC with 147 flights, allowing maintenances at any time at half of the airports. The initial resource consumption has been randomly generated. We show the results in Table 5.

Even when allowing up to 20% more resource consumption, we get a massive cancellation rate and huge delays. However we mention that NM+20% finds a better solution than MO for 1 of the 10 instances, where actually this 20% increase is sufficient to perform the whole schedule without any maintenance. In this instance, the only additional costs in the solution of MO are the maintenance costs; neither delay nor flight cancellation is required. Remarkably, even with the 20% increase in resource capacity, only 7 solutions out of the 10 instances are feasible, i.e. cover all the final states.

The GM algorithm clearly outperforms the NM+5%, NM+10% and NM+20% algorithms, reducing the average cost by one order of magnitude. However, GM computes solutions whose cost is 7.5% higher than those given by MO. The main savings are made thanks to delay reductions: GM finds the same solutions as MO for 3 out of the 10 instances, but never finds a better one.

We see from these results that considering maintenances is not only necessary to ensure feasibility of the recovery scheme. Indeed, the solution may be significantly improved (mainly by reducing delays) when maintenance operations are rescheduled. The results show that NM+y% approaches, i.e. an extension of y% of the resource consumption, is not efficient to solve the recovery problem: the computed solutions are not necessarily feasible and, when they are, the costs are one order of magnitude higher. Furthermore, airlines must obtain the permission from the authorities to extend resource consumption limits, which involves further negociations and an exposure to a possible expensive audit.

Sensitivity analysis for T. We show in Table 6 the different solutions when increasing the recovery period T. We solve instance 5D_10AC with one plane grounded up to time 2160, and compute a solution for increasing recovery periods, going from 720 minutes up to 6480 minutes. Table 6 shows the details of the solutions, where *additional costs* are the aggregated delay and cancellation costs over the whole period, assuming that the schedule is recovered at T. Figure 3 shows the Pareto frontier, i.e. the additional costs against the length of the recovery period. Note that the solution for $T = 5760$ and $T = 6480$ are the same as for $T = 5040$, ex-

cept computation time of 76.6 and 183.6 seconds, respectively, and are thus not presented in Table 6.

For $T \leq 2160$, the solutions are infeasible, which is trivial: as one plane is grounded until time 2160, at least one final state cannot be covered before time 2160.

If a feasible solution exists for a given T (here $T > 2160$), increasing T leads to a *stable* solution, i.e. we generate the same recovery plan even when a longer recovery period is considered. No additional recovery costs are incurred for a stable solution, even when T is increased further. Notice that because of computational complexity, the delay bound is set to 800 minutes, ignoring therefore potential recovery schemes with longer delays.

Finally, Table 6 and Figure 3 show the conflict between the two objectives of minimizing T and the recovery costs simultaneously: minimizing T incurs higher recovery costs.

Summary. The computational results show that the algorithm is efficient, solving instances of reasonable complexity comparable to the state of the art in low computation time. We see that the introduced parameters are useful to accelerate computation without dramatically decreasing the quality of the solution. The resulting recovery plans follow intuition, deleting as few flights as possible by swapping or delaying planes. Furthermore, we show that scheduling maintenances during the recovery period significantly improves the solution. Finally, we see that the solution of the recovery algorithm depends on the initial schedule as much as on the actual disruption.

5 Preliminary results for the PRP

In section 3.5, we illustrate the application of the framework to the PRP. In this section, we provide some insight into the benefits of applying the proposed method without discussing performances nor speed-up strategies.

As remarked by Kohl et al. (2007), conventional airline wisdom is to get back to the original plan with little modifications, while minimizing additional costs and passenger inconvenience. Thus, it is common practice that all passenger itineraries that are still feasible in the new schedule are assigned to their original scheduled flights. Then, each disrupted passenger is reaccommodated, minimizing cancellation and delay costs mainly. Kohl et al. (2007) also show that a thorough reoptimization of all itineraries could improve the quality of the solution.

To validate the framework for the PRP we use eight out of ten `A` instances of the ROADEF Challenge 2009[2], namely instances A01−A04 and A06−A09.

Instances A01−A10 are based on the same daily schedule with 35 airports and 85 planes; A01−A04 and A06−A09 are instances of same size, i.e. 608 flights, with different number of passengers: A01−A04 have 1,943 OD pairs for 36,010 passengers in total, whereas instances A06−A09 have 1,872 OD pairs and 46,619 passengers. Each instance differs in the level of disruption and in the allowed recovery period. Table 7 reports on the disruptions for each instance.

A cost checker evaluating the quality of the recovery plan is available according to a specified cost structure. The cost structure is a combination of several objectives, namely operating costs for the fleet and passenger inconvenience costs for trip cancellation, delay or downgrading, i.e. change to a lower cabin class on all

[2]http://challenge.roadef.org/2009/index.en.htm

or part of the trip. In our implementation we neglect the passenger downgrading costs.

We use our implementation of the ARP limited to 10 minutes of computation to obtain a feasible initial solution for the PRP. Table 8 reports on the number of canceled flights and the total delay of the new schedule with respect to the original one.

Although a rigorous comparison goes beyond the scope of this illustration, we implemented a flow-based algorithm called `FlowPRP` to estimate the potential benefits. It is a two-stage process: first, all passengers whose itinerary is still feasible in the new schedule are confirmed; then, for each disrupted passenger, alternative itineraries are computed solving a minimum cost network flow problem on a flight connection network where residual seat capacity is considered. Passengers are accommodated sequentially, ordered by decreasing cancellation cost, i.e. starting from business class.

The same recovery scheme limitations are used for both `PRP` and `FlowPRP`, namely $\eta = 5$ and the maximal allowed delay is 800 minutes in both cases.

Table 9 summarizes the results obtained for the 8 instances. Computations are performed on a 2.53GHz processor with 3GB of memory.

The results show that, on average, using `PRP` leads to better solutions in terms of both costs and number of canceled passengers; to achieve this, the number of rerouted and delayed passengers is increased. We also note that the more canceled flights in the ARP solution, the more `PRP` outperforms `FlowPRP`. This is because the heuristic algorithm is close to optimal when operations are close to normal; this happens for instance A01 mainly, where `FlowPRP` actually gives a solution with lower cost than `PRP`. The difference in recovery costs for this instance is only due to passenger downgrading penalties. The actual implementation of the `PRP` algorithm focuses only on cancellation and delay reduction.

On average, `PRP` reduces the number of canceled passengers by 54.9%, the recovery costs by 51.7%, and the total and average delays by 3.0% and 0.1%, respectively. We see that the overall solutions are better. From the passengers' point of view, although total and average delays are slightly reduced, the number of rerouted passengers is multiplied by a factor of 4.8. The average computation time for `PRP` is of 3,102 seconds, against 1.0 second for `FlowPRP`.

The results are consistent with our expectations: a significant improvement in the quality of the solutions, at the price of more computational time.

These preliminary results for the PRP show that the constraint-specific network model is flexible enough to solve the recovery problem for different types of units. Given that the actual implementation is a prototype, computational times are still acceptable even with the very large number of units in the PRP case, i.e. more than 30,000 passengers. Indeed, for `PRP`, we need to solve multiple pricing problems. The pricing problem of finding a negative reduced cost route for each passenger is computationally harder than finding one feasible route in a connection network. In the actual implementation the pricing problem requires more than 90% of the total computation time. Finally, thanks to the framework, we can identify non-trivial solutions.

6 Conclusions and future work

In this paper, we present a general modeling approach to solving airline recovery problems. The general model considers unit-specific constraints using resource consumption, whereas the use of a CG algorithm ensures feasibility according to

the structural constraints of the problem, i.e. that the combination of each unit's recovery scheme is globally feasible; this approach overcomes the main drawbacks of usual multi-commodity approaches, that struggle to consider exact unit-specific constraints. We show the efficiency of the approach by solving successfully with real data the ARP with maintenance planning and illustrate briefly its application to the PRP. Furthermore, as units can be either aircraft, crew members or passengers, the formulation is appropriate for the different aspects of the airline recovery problem.

The constraint-specific recovery network model is also applicable to connection networks, for which nodes correspond to flights and the time dimension is represented implicitly in the arcs: the departure of a flight is only determined by the path reaching its corresponding node. It is thus more difficult to determine departure and arrival times using activity periods as we do in the recovery network model. Moreover, the concept of renewing arcs and the computation of resource consumptions for the arcs seems less intuitive in a connection network.

The flexibility of the constraint-specific recovery network allows to easily impose user-specific constraints: activity and renewing operation slots at the airports allow to model airport disruptions such as airport closure; subsets F_p and S_p allow unit type differentiation such as heterogeneous fleet or different crew types, a plane of one fleet may not be allowed to cover a flight or a final state of another fleet and a particular crew may not have the required training to operate a specific flight. It also allows the enforcement of a unit to perform its initial schedule, when F_p is the set of initially scheduled flights for unit p and S_p contains only the final state initially allocated to unit p. It is also possible to consider a given set of repositioning flights: a repositioning flight is similar to any other flight, except it has zero cancellation and delay costs.

Reserve units (e.g. reserve aircraft) are included in the model as additional units p. Whether a reserve unit has a final state or not depends on the policy of the airline; if none is required, we define a dummy final state that is reachable from any airport and coverable by each unit that is allowed to become reserve.

A disruption making a unit unavailable for some time or even the whole recovery period is modeled using the initial state. Unpredicted resource renewing requirements are modeled by setting the concerned resources to their upper limits.

In terms of instance complexity analysis, we see that considering the number of flights and aircraft only is not necessarily the best approach. We consider in this paper the ratio flights/planes, i.e. the average number of flights per plane. A deeper theoretical analysis to compare efficiently instance complexity would allow a better understanding of the problem and also lead to a more efficient performance measure for the existing approaches.

Remaining work on the ARP with maintenance is to extend tests on data from a bigger airline with a heterogeneous fleet and to implement the full Branch&Price algorithm. Although repositioning flights are easy to integrate into the model when they are provided, the problem of generating repositioning flights remains: generating many repositioning flights increases the instance complexity; considering a limited number of them decreases the chances of a good repositioning flight to be generated. Deeper work on repositioning flights is thus certainly a research direction.

Finally, being an active field for researchers, deriving integrated models dealing with different types of units simultaneously is definitely a research direction to be explored and tested. Thanks to the uniform model for different unit types, we derived an integrated model with a pseudo-polynomial number of constraints. The

next step is to study this model more closely and eventually implement it to test its computational efficiency, which requires the model to be applied to crew and/or passengers as well.

7 Acknowledgments

The current work results from a collaboration between EPFL and APM Technologies, sponsored by the Swiss government within the fund for technology transfer (CTI - Project 8007.2 ESPP-ES). We especially thank CTI for its funding and APM Technologies, the industrial partner of this project, for its support and vital information on the problem. We also thank Thomas Cook Airlines for providing their data for our computational tests.

A Notation

We list here the notation we use through the paper:

- 0, T, the begin and end time of the recovery period;
- t_0, time for an initial state (first time on ground after time 0);
- t_T, time for a final state (latest time on ground before T);
- H_j, the resource consumption vector at node j (a single resource consumption is denoted h_j^r);
- U, the vector of maximal allowed resource consumptions (upper bound for a single resource r is u^r);
- upper and lower bounds on resource labels:
  - $\overline{h}_j^r$, the maximal consumed resource r when reach node j,
  - $\underline{h}_j^r$, the minimal consumed resource r when reach node j,
  - $\underline{h}_{sink,j}^r$, the minimal required amount of resource r to reach the sink from node j;
- A, the set of airports, and for each airport $a \in A$:
  - mct_a, the minimal connection time for a unit between two flights,
  - O_a, the set of activity slots, which are time intervals when take-off and landing operations can take place,
  - M_a, the set of resource-renewing operation slots, which are time intervals when resource-renewing operations can take place,
  - dm_a, the duration of resource renewing action;
- P, the set of units, and for each unit $p \in P$:
  - $[\{a, t_0\}, \{H_0, H_0, U_0\}]$, the initial state specified by initial time-location and initial resource consumption;
- S, the set of final states, where $S_p \subseteq S$ is the set of final states coverable by unit $p \in P$ and for each $[\{a, T\}, \{H_T, H_T, U_T\}] \in S_p$:
  - U_T, the maximal allowed resource consumption at the sink node;
- F, the set of flights, where $F_p \subseteq F$ is the set of flights that can be assigned to unit $p \in P$ and for each $f \in F$:
  - sdt_f, the scheduled departure time,

- d_f, the duration,
- c_f, the cancellation cost of the flight,
- edt_f, the earliest departure time for flight $\mathsf{f} \in \mathsf{F}$.

In addition, we define:

- cd: the delay cost per time unit ($/minute);
- Δ: the length of a time discretization interval;
- τ: the maximal allowed delay for a flight;
- ψ: the maximal allowed waiting time before a take-off;
- Γ: the maximal length of a termination or a renewing termination arc;
- ρ: the minimal percentage of a resource to be consumed to perform renewing operation;
- θ: the number of resource intervals for logarithmic resource discretization;
- η: the maximal number of flights of a disrupted passenger's recovery scheme.

B Recovery network generation

Algorithm 1 shows the dynamical structure of the recovery network generation algorithm for the ARP: here, we replace the concepts of units and renewal with plane and maintenance.

In the algorithm, N represents the set of time-location nodes ordered by increasing time. The way that nodes and arcs are created is described in Table 10. The remaining notational detail are given in Appendix A and the parameters are introduced in Section 3.4. For notational simplicity, we denote a final state $[\{\mathsf{a}, \mathsf{T}\}, \{0, \mathsf{H}_\mathsf{T}, \mathsf{U}_\mathsf{T}\}]$ by s.

Tables 11 and 12 give an overview of the different constraints that must be satisfied in each function (parametrized constraints are labeled by (P)).

References

Argüello, M., Bard, J. and Yu, G. (1997). A grasp for aircraft routing in response to groundings and delays, *Journal of Combinatorial Optimization* **5**: 211–228.

Bard, J., , Yu, G. and Argüello, M. (2001). Optimizing aircraft routings in response to groundings and delays, *IIE Transactions* **33**: 931–947.

Beasley, J. and Christofides, N. (1989). An algorithm for the resource constrained shortest path problem, *Networks* **19**: 379–394.

Bratu, S. and Barnhart, C. (2006). Flight operations recovery: New approaches considering passenger recovery, *Journal of Scheduling* **9**: 279–298.

Clarke, G. (1997). The airline schedule recovery problem, *working paper* (1997).

Clausen, J., Larsen, A., Larsen, J. and Rezanova, N.J. (2009). Disruption management in the airline industry - concepts, models and methods, *Computers & Operations Research*, doi:10.1016/j.cor.2009.03.027.

COIN-OR (2008).
URL: *www.coin-or.org*

Cordeau, J.-F., Stojkovic, G., Soumis, F. and Desrosiers, J. (2001). Benders decomposition for simultaneous aircraft routing and crew scheduling, *Transportation Science* **35**(4): 375–388.

Desaulniers, G., Desrosiers, J. and Solomon, M. (eds) (2005). *Column Generation*, GERAD 25th Anniversary Series, Springer.

EUROCONTROL (2004). Challenges to growth report.
URL: *www.eurocontrol.int*

Jarrah, A., Krishnamurthy, N. and Rakshit, A. (1993). A decision support framework for airline flight cancellations and delays, *Transportation Science* **27**(3): 266–280.

Kohl, N., Larsen, A., Larsen, J., Ross, A. and Tiourine, S. (2007). Airline disruption management - perspectives, experiences and outlook, *Journal of Air Transport Management* **13**(3): 149–162.

Lübbecke, M. and Desrosiers, J. (2005). Selected topics in column generation, *Operations Research* **53**: 1007–1023.

Mercier, A. and Soumis, F. (2007). An integrated aircraft routing, crew scheduling and flight retiming model, *Computer & Operations Research* **34**: 2251–2265.

Righini, G. and Salani, M. (2006). Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints, *Discrete Optimization* **3**(3): 255–273.

Righini, G. and Salani, M. (2008). New dynamic programming algorithms for the resource constrained shortest path problem, *Networks* **51**(3): 155–170.

Rosenberger, J., Johnson, E. and Nemhauser, G. (2003b). Rerouting aircraft for airline recovery, *Transportation Science* **37**(4): 408–421.

Rosenberger, J., Schaefer, A., Golldsman, D., Johnson, E., Kleywegt, A. and Nemhauser, G. (2003a). A stochastic model of airline operations, *Transportation Science* **36**(4).

Schaefer, A., Johnson, E., Kleywegt, A. and Nemhauser, G. (2005). Airline crew scheduling under uncertainty, *Transportation Science* **39**(3): 340–348.

Shavell, Z. A. (2000). The effects of schedule disruptions on the economics of airline operations, *Technical report*, The MITRE Corporation.

Sriram, C. and Hagani, A. (2003). An optimization model for aircraft maintenance scheduling and re-assignment, *Transportation Research Part A* **37**: 29–48.

Stojković, G., Soumis, F., Desrosiers, J. and Solomon, M. (2002). An optimization model for a real-time flight scheduling problem, *Transportation Research Part A* **36**: 779–788.

Teodorvić, D. and Gubernić, S. (1984). Optimal dispatching strategy on and airline network after a schedule perturbation, *European Journal of Operations Research* **15**: 178–182.

Teodorvić, D. and Stojković, G. (1990). Model for operational airline daily scheduling, *Transportation Planning and Technology* **14**(4): 273–285.

Thengvall, B. G., Bard, J. F. and Yu, G. (2000). Balancing user preferences for aircraft schedule recovery during irregular operations, *IIE Transactions* **V32**(3): 181–193.

Thengvall, B., Yu, G. and Bard, J. (2001). Multiple fleet aircraft schedule recovery following hub closures, *Transportation Research Part A* **35**: 289–308.

Vanderbeck, F. (2005a). Branching in branch-and-price: a generic scheme, *Technical report*, CNRS, Université de Bordeaux, http://hal.inria.fr/inria-00311274/fr/.

Vanderbeck, F. (2005b). Implementing mixed integer column generation, *in* G. Desaulniers, J. Desrosiers and M. Solomon (eds), *Column Generation*, Springer-Verlag, pp. 331–358.

Wei, G., Yu, G. and Song, M. (1997). Optimization model and algorithm for crew management during airline irregular operations, *Journal of Combinatorial Optimization* **1**: 305–321.

Weide, O., Ryan, D. and Ehrgott, M. (2008). Solving the robust and integrated aircraft routing and crew pairing problem in practice - a discussion of heuristic and optimization methods, *Technical report*, Department of Engineering Science, University of Auckland.

Yan, S. and Lin, C. (1997). Airline scheduling for the temporary closure of airports, *Transportation Science* **31**: 72–78.

Yan, S. and Tu, Y. (1997). Multifleet routing and multistop flight scheduling for schedule perturbation, *European Journal of Operations Research* **103**: 155–169.

Yan, S. and Yang, D. (1996). A decision support framework for handling schedule perturbations, *Transportation Research* **30**: 405–419.

Yan, S. and Young, H. (1996). A decision support framework for multi-fleet routing and multi-stop flight scheduling, *Transportation Research Part A* **30**(5): 379–398.

Yu, G., M.Argüello, G.Song, McCowan, S. and A.White (2003). A new era for crew scheduling recovery at continental airlines, *Interfaces* **33**(1): 5–22.

Algorithm 1 Recovery Network Generation for the ARP

Require: Set P of planes, sets F_p of coverable flights, initial states and set S_p of final states

for $p \in P$ **do**
  INITIALIZATION: Create source node $s = [\{a, t_0\}, \{H_0, H_0, U_0\}]$, set $N = \{s\}$
  while $N \neq \emptyset$ **do**
    Select the first node $j = [\{a, t_j\}, \{\underline{H}_j, \overline{H}_j, U_j\}] \in N$
    for $f \in F_p$ where a is the departure of f **do**
      if `FeasibleForFlightArc`(j, f) **then**
        $[\{a', t'\}, \{\underline{H}_k, \overline{H}_k, U_k\}] =$ `CreateFlight`(j, f)
        set $N \leftarrow N \cup [\{a', t'\}, \{\underline{H}_k, \overline{H}_k, U_k\}]$
      end if
      if `FeasibleForMaintArc`(j, f) **then**
        $[\{a', t'\}, \{\underline{H}_k, \overline{H}_k, U_k\}] =$ `CreateMaintenance`(j, f)
        set $N \leftarrow N \cup [\{a', t'\}, \{\underline{H}_k, \overline{H}_k, U_k\}]$
      end if
    end for
    for $s \in S_p$ where a_j is the airport of s **do**
      if `FeasibleForTermArc`(j, s) **then**
        `CreateTermination`(j, s)
      end if
      if `FeasibleForMaintTermArc`(j, s) **then**
        `CreateMaintTermination`(j, s)
      end if
    end for
    Set $N \leftarrow N \setminus \{j\}$
    Sort N by increasing time
  end while
end for

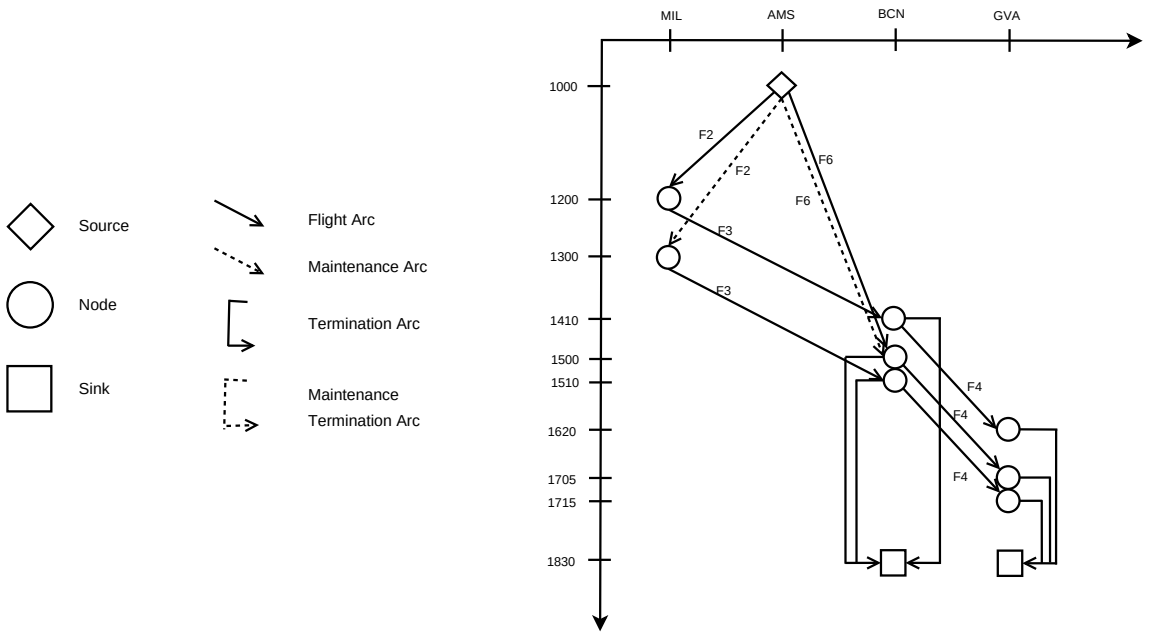

Figure 1: Recovery network of plane $\mathfrak{p}_2$ with initial schedule of Table 1 and initial state [BCN,0740,10].

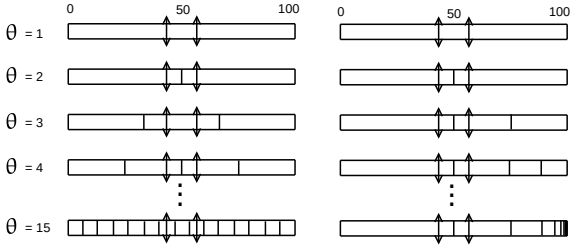

Figure 2: Linear (on the left) and logarithmic (on the right) discretization for increasing number of intervals ($\theta = 1, 2, 3, 4, 15$).

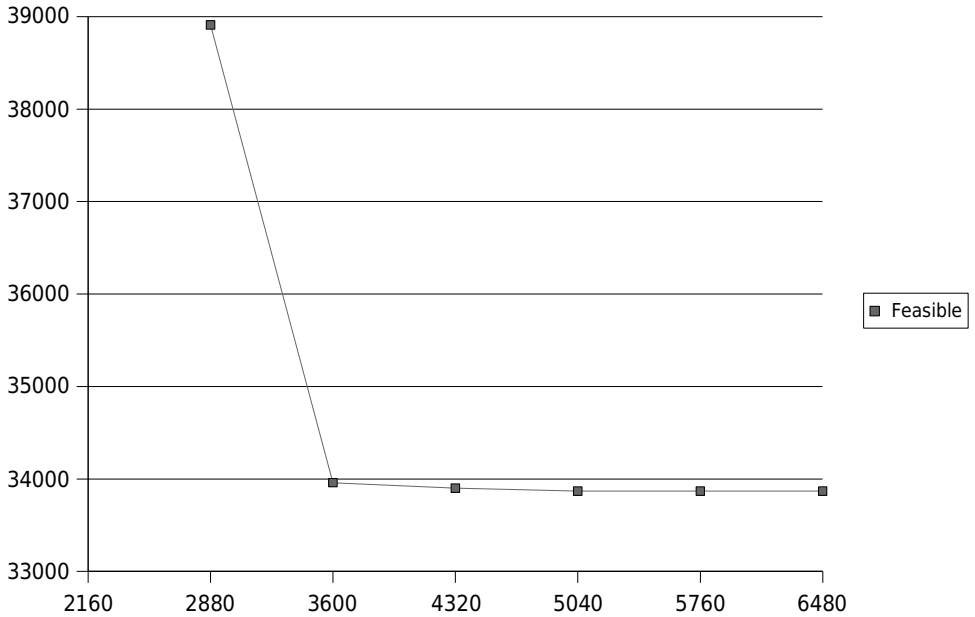

Figure 3: Pareto frontier: additional solution costs against recovery period length T

Plane 1	Flight ID	Origin	Destination	Departure time	Landing time
	F1	GVA	AMS	0830	0905
	F2	AMS	MIL	1000	1130
	F3	MIL	BCN	1200	1340
	F4	BCN	GVA	1415	1550
Plane 2	Flight ID	Origin	Destination	Departure time	Landing time
	F5	MIL	AMS	0740	0930
	F6	AMS	BCN	1120	1430

Table 1: The original schedule for two planes

Plane 1	Flight ID	Origin	Destination	Departure time	Landing time
	F1	GVA	AMS	0830	0905 (1105)
	F6	AMS	BCN	1120	1430
Plane 2	Flight ID	Origin	Destination	Departure time	Landing time
	F5	MIL	AMS	0740	0930
	F2	AMS	MIL	1000	1130
	F3	MIL	BCN	1200	1340
	F4	BCN	GVA	1415	1550

Table 2: A recovered schedule for two planes

Instance	2D_10AC	3D_10AC	4D_10AC	5D_10AC	7D_16AC	2D_100AC
# planes	10	10	10	10	16	100
# flights	75	113	147	184	242	760
# delayed planes	2	2	2	1	0	10
# canceled flts	2	2	4	2	0	20
# delayed flts	5	7	1	6	11	40
total delay [min]	989	1146	20	1126	310	9690
max delay [min]	370	370	20	370	45	370
cost	21745(*)	23695(*)	25930(*)	2425(*)	5600	557550(*)
tree size	1	1	1	1	2033	1
run time [s]	1.0	2.9	16.2	24.7	3603	62.9

Table 3: Results for some instances. Costs followed by (*) are proved to be optimal

Instance	2del	2grd	4del	4grd	2del2grd	6del
# affected flights	1	4	3	8	5	5
# canceled flts	0	2	0	8	4	0
# delayed flts	1	4	7	2	7	13
total delay	10	920	230	380	490	640
max delayed flight	10	275	85	200	200	100
cost	36100(*)	83200(*)	38300(*)	163800(*)	84900(*)	42400(*)
tree size	1	1	1	1	1	41
run time	0.7	0.5	0.6	0.3	0.5	1.6

Instance	6grd	3del3grd	3x100	1x300	St1	St2
# affected flights	16	9	11	7	3	6
# canceled flts	16	6	0	4	0	0
# delayed flts	2	12	11	11	6	6
total delay	380	950	675	2560	350	1550
max delayed flight	200	200	90	385	140	340
cost	251800(*)	127500(*)	42750(*)	125600(*)	39500(*)	51500(*)
tree size	1	1	1	35	1	3
run time	0.2	0.4	0.3	0.8	0.5	0.5

Table 4: Results for different disruption scenarios. Affected flights is the number of flights affected directly by the disruption without any propagation.

Algorithm	NM+5%	NM+10%	NM+20%	GM	MO
# canceled flts	52.7	46.7	33.2	2.2	2
# delayed flts	5	4.7	5.5	2.7	1.5
# uncovered final states	1.2	0.7	0.3	0.1	0.1
total delay [min]	851.3	635.7	712.5	89.6	52.3
max delay [min]	271.3	251.5	218.2	37.7	37.1
cost	289462	272067	144388	15881	14683
optimality gap [%]	0.61	0.54	1.27	0.73	0

Table 5: Average results for recovery algorithms with different maintenance operation handling on 10 randomly generated initial resource consumptions.

Recovery Period T	720	1440	2160	2880	3600	4320	5040
# canceled flts	1	3	5	6	5	5	5
# delayed flts	0	1	3	5	9	9	8
# uncovered sinks	1	1	1	0	0	0	0
total delay [min]	0	3	14	461	636	630	627
max delay [min]	0	3	8	153	153	153	153
additional costs	7000(#)	19555(#)	25415(#)	38910	33960	33900	33870
optimality gap	0%	0%	0%	0%	0.25%	0%	0%
tree size	1	1	1	1	7	9	3
run time [s]	< 0.1	< 0.1	0.4	1.7	8.3	25.0	81.8

Table 6: Results for the same instance with different recovery periods T. Cost followed by (#) correspond to infeasible solutions.

Instance	A01	A02	A03	A04	A06	A07	A08	A09
# flights	608	608	608	608	608	608	608	608
total duration	1680	1680	1680	1680	1680	1680	1680	1680
recovery period	960	720	840	1080	960	720	840	1080
# delayed flts.	63	106	79	41	63	106	79	41
# canceled flts.	0	1	4	0	0	1	4	0
# resting planes	0	0	1	0	0	0	1	0
# mod. capacity slots	0	0	0	4	0	0	0	4

Table 7: Description of the A instance set of the ROADEF Challenge 2009.

Instances	# canceled flights	total flight delay [min]
A01	0	1390
A02	2	900
A03	8	939
A04	14	7602
A06	0	1390
A07	2	900
A08	8	909
A09	14	7807

Table 8: Number of canceled flights and cumulated flight delays - ROADEF dataset.

Instance	A01		A02		A03		A04	
Algorithm	FlowPRP	PRP	FlowPRP	PRP	FlowPRP	PRP	FlowPRP	PRP
# canceled passengers	41	33	196	79	499	293	196	116
# rerouted passengers	235	2848	587	2468	900	3092	1875	6431
# delayed passengers	8664	7852	10430	9969	8798	8569	15612	14365
total delay [min]	280312	259133	581312	557593	511026	523042	1004023	841422
average delay [min]	32.3	33.0	55.7	55.9	58.1	61.0	64.3	58.6
recovery costs	89477	111351	342267	219789	703928	451378	289384	185004
run time [s]	0.66	3155	0.95	1806	1.17	2425	1.84	3755

Instance	A06		A07		A08		A09	
Algorithm	FlowPRP	PRP	FlowPRP	PRP	FlowPRP	PRP	FlowPRP	PRP
# canceled passengers	44	10	441	148	954	579	1161	334
# rerouted passengers	243	2779	445	2462	843	3167	1206	7427
# delayed passengers	11293	10469	13007	12997	10898	11323	18892	19367
total delay [min]	350257	332786	700504	715910	653078	678746	1154229	1171478
average delay [min]	31.0	31.8	53.9	55.1	59.9	59.9	61.1	60.5
recovery costs	99893	64859	632736	268534	1363047	775285	1459473	330232
run time [s]	0.47	3781	0.72	2562	0.97	3363	1.20	3973

Table 9: Results for the PRP - ROADEF dataset.

`CreateFlight`$([\{a_j, t_j\}, \{\underline{H}_j, \overline{H}_j, U_j\}], f)$ Given depart node $\{a_j, t_j\}$, computes the destination node $\{a_k, t_k\}$, and the flight arc $(\{a_j, t_j\};\{a_k, t_k\})_f$, where a_k is destination airport, and t_k is the earliest departure time at airport a_k. To compute this, first compute edt_f, the earliest departure time for the flight f according to the activity slots and the scheduled departure time sdt_f, then $t_k = edt_f + d_f + mct_{a_k}$. Labels $\underline{H}_k$ and $\overline{H}_k$ of node $\{a_k, t_k\}$ are updated according to $\underline{H}_j$, $\overline{H}_j$ and the consumed resources during flight f.

`CreateMaintenance`$([\{\{a_j, t_j\}, \underline{H}_j, \overline{H}_j, U_j\}], f)$ Similar to `CreateFlight`(j, f), it computes the maintenance time and the cost of the maintenance arc.

`CreateTermination`$([\{a_j, t_j\}, \{\underline{H}_j, \overline{H}_j, U_j\}], s)$ Given depart node $\{a_j, t_j\}$ and a sink node s, it creates the termination arc $(\{a_j, t_j\};\{a_j, T\})$.

`CreateMaintTermination`$([\{a_j, t_j\}, \{\underline{H}_j, \overline{H}_j, U_j\}], s)$ Given depart node $\{a_j, t_j\}$ and a sink node s, it creates the maintenance termination arc $(\{a_j, t_j\}; s)$. By convention, the first available maintenance slot is used.

Table 10: Functions used in Algorithm 1

`FeasibleForFlightArc`$([\{a_j, t_j\}, \{\underline{H}_j, \overline{H}_j, U_j\}], f)$ The flight arc can only be created if flight is actually departing from airport a_j and if feasible departure and landing times are available at airports a_j and a_k. The following constraints are checked:

- $\exists\ etd_f \geq \max\{sdt_f, t_j\}$ such that
  1. $\exists\ o_{a_j} \in O_{a_j}$ such that $etd_f \in o_{a_j}$
  2. $\exists\ o_{a_k} \in O_{a_k}$, such that $etd_f + d_f \in o_{a_k}$
- $delay \leq \tau$ (P)
- $edt_f - t_j \leq \psi$ (P)

where τ and ψ are representing the maximal delay bound and the maximal waiting bound, respectively.

`FeasibleForMaintArc`$([\{a_j, t_j\}, \{\underline{H}_j, \overline{H}_j, U_j\}], f)$ The maintenance arc can only be created if there is a maintenance slot available at airport a_j. t^M is the starting time of the maintenance if feasible, i.e. if we find a feasible departure time for take-off in a_j and landing in a_k. The following constraints are checked:

- $\exists\ t^M \geq t_j, m_{a_j} \in M_{a_j}$ such that $t^M \in m_{a_j}$
- $\exists\ etd_f \geq \max\{sdt_f, t^M + dm_{a_j}\}$ such that
  1. $\exists\ o_{a_j} \in O_{a_j}$ such that $etd_f \in o_{a_j}$
  2. $\exists\ o_{a_k} \in O_{a_k}$, such that $etd_f + d_f \in o_{a_k}$
- $delay \leq \tau$ (P)
- $edt_f - t_j - dm_{a_j} \leq \psi$ (P)
- $H_j \geq \rho U_j$

where τ and ψ are the same as in `FeasibleForFlightArc`(j, f) and ρ is the parameter of minimal resource consumption ratio before considering maintenance.

Table 11: Feasibility functions for flight and maintenance arcs used in Algorithm 1

`FeasibleForTermArc`$([\{a_j, t_j\}, \{\underline{H}_j, \overline{H}_j, U_j\}], s)$ A termination arc can be created between $\{a_j, t_j\}$ and the sink node s if the airports are matching, if there is at least a feasible path reaching the sink with respect to resource consumption ($\underline{H}_j \leq U_T$ for all resources) and if the required time t_T is not yet reached. A parameter Γ is used to bound the grounding time needed to reach the sink from $\{a_j, t_j\}$.

- $t_T - t_j \leq \Gamma$ (P)

where Γ is the grounding time bound.

`FeasibleForMaintTermArc`$([\{a_j, t_j\}, \{\underline{H}_j, \overline{H}_j, U_j\}], s)$ Similarly a maintenance termination arc can be created between $\{a_j, t_j\}$ and the sink node s if there is a maintenance slot available.

- $\exists\ t^M \geq t_j,\ m_{a_j} \in M_{a_j}$ s.t. $t \in m_{a_j}$
- $t^M + dm_{a_j} \leq t_T$
- $t_j \leq t_T$
- $t_T - t_j \leq \Gamma$ (P)
- $H_j \geq \rho U_j$

where Γ is the grounding time bound and ρ the resource consumption proportion.

Table 12: Feasibility functions for termination and maintenance termination arcs used in Algorithm 1