A column generation algorithm for disrupted airline schedules

N. Eggenebrg [*] M. Salani [*] M. Bierlaire [*]

December 3, 2007

Report TRANSP-OR 071203
Transport and Mobility Laboratory
Ecole Polytechnique Fédérale de Lausanne
transp-or.epfl.ch

[*]Transp-OR, Ecole Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland, niklaus.eggenberg@epfl.ch, matteo.salani@epfl.ch, michel.bierlaire@epfl.ch

Abstract

We consider the recovery of an airline schedule after an unforeseen event, called *disruption*, that makes the planned schedule unfeasible. In particular we consider the aircraft recovery problem for a heterogeneous fleet of aircrafts, made of regular and reserve planes, where the aircrafts' maintenances are planned in an optimal way in order to satisfy the operational regulations.

We propose a column generation scheme, where the pricing problem is modeled as a commodity flow problem on a dedicated network, one for each plane of the fleet. We present a dynamic programming algorithm to build the underlying networks and a dynamic programming algorithm for resource constrained elementary shortest paths to solve the pricing problem. We provide some computational results on real world instances.

1 Introduction

Air travel is nowadays one of the most frequent modes of transportation for business, leisure, and tourism. The market of airlines is no longer protected both in Europe as in the US and airlines have the possibility to decide their routes as well as their fares. It is crucial for them to manage their operations in an efficient way in order to lead the market and to optimize their profits and services.

Airlines need to coordinate a relevant number of resources to provide their service to the customers. Strategical and operational decisions are taken to provide a reasonable expected revenue for the company: routes must be planned in terms of location and arrival/departure time, aircrafts have to be affected to routes complying with all the safety regulations and technical constraints, crews must operate the aircrafts within the contractual specifications traded with the unions of workers.

From a computational point of view airline scheduling is one of the most challenging decisional problems. It has been addressed in the last decades by algorithms based on operations research techniques. The problem is usually decomposed into stages. The reasons are that airlines are organized in departments in which decisions on flights, planes and crews are taken separately, with different publication deadlines and different required knowledge on the problem. Moreover it is commonly believed that the entire scheduling problem is computationally intractable.

The first objective is the route choice, where the airline decides on the legs to be flown, which is typically done 6 to 12 months in advance. Next step is the fleet assignment, where fleets of planes are assigned to legs. The tail assignment then builds routes satisfying all technical constraints, such as maintenances for individual planes, which is done from 1 to 6 months in advance. Next step is to compute crew pairings that satisfy the corresponding union of workers' requirements. This is done between 1 and 2 months before the day of operations, as is the crew roistering, where individual crews are assigned to a pairing with respect to their working history and union constraints for individuals. Finally, usually up to the day before the day of operations, the passenger routing is done in order to determine the passenger's connections.

Unfortunately, on the day of operations, it is unlikely that the optimized schedule obtained by the airline scheduling is actually carried out as planned: most of the time, *disruptions* such as bad weather, unpredicted technical maintenances or propagated delays affect the planned schedule. Thus, when disruptions make the schedule unfeasible, aircrafts, crews and passengers have to be reaccommodated.

In the *European airline punctuality report* (2006), the Association of European Airlines reports that 20.6% of departures and 22.1% of arrivals of the European flights are delayed by more than 15 minutes and that the yearly average is increasing. Interestingly, weather conditions delay only 3.6% of the flights, while the propagation of the delays of late planes affects the whole schedule up to 15.1%.

The delay costs for airlines is estimated between 840 and 1200 million Euros in 2002 by the EuroControl Association (Cook et al., 2004) and the delay cost per minute is estimated to be around 72 Euros.

In the *Challenges to growth report* (2004) by EuroControl we find four scenario based forecasts on the air traffic demand for the next years. An estimation about the increase of the demand for flights lies between 2.5% and 4.3%, yearly based, for the next 20 years. With the highest growth scenario, annual demand increases to 21 million flights, a growth by a factor 2.5 compared to 2003. However, despite 60% potential capacity increase of the airport network, only twice the volume of 2003 traffic can be accommodated, and 17.6% of demand (i.e. 3.7 million flights per year) cannot be served. This is expected to have a significant impact on airport operations: in this scenario, more than 60 airports are congested, and the top-20 airports are saturated at least 8-10 hours per day. Given this forecast on the increase of air traffic and airport congestion it is obvious that a disruption in the airline schedule would have a deeper operational and economic consequence because of the cascade effect on other scheduled flights. Thus it is crucial for airlines to adopt more and more effective recovery strategies.

Schedule recovery decisions are taken at the Operations Control Center (OCC) with the aim of finding a set of feasible operations that rebuild the planned schedule as soon as possible. Moreover OCC operators are required to provide quickly reliable decisions without ensuring the total recover of the planned schedule in case of emergency situations. Because of the real-time nature of the problem, recovery decisions are taken thanks to their knowledge and experience. OCC operators will certainly benefit from the use of a decision support system based on a recovery algorithm able to provide several recovery alternatives.

The original contributions of this paper are related to the integrated re-scheduling of flights and maintenances for an heterogeneous fleet of aircrafts in case of unforeseen disruptions. In this work we adapt, implement and validate some state-of-the-art algorithms based on column generation and dynamic programming and we report a computational study based on real world instances to validate the importance of considering the optimization of maintenance operations.

The motivation for simultaneously optimizing flight and maintenances comes from actual rules in civil aviation. The Joint Aviation Authorities (JAA) is an associated body of the European Civil Aviation Conference (ECAC) representing the civil aviation regulatory authorities of a number of European States. It provides a set of rules, called JAR-OPS, that cover the operational requirements of any civil aircraft owned by an airline belonging to a JAA Member State. Under civil aviation rules, maintenance operations are enforced when aircrafts reach some resource limits (typically fly hours, number of take-off and landing operations, and total time between two maintenances). An extension of an aircraft maintenance limit can be obtained through written permission from the authority. As such, it is not a regular occurrence. Any airline that continually asks for exemptions could be subject to an audit from the authority. Usually airlines prefer to operate under JAR-OPS rules to avoid additional audits at all costs.

We first give in Section 2 an overview on the state of the art. We provide the problem description in Section 3. Section 4 describes the recovery algorithm based on column generation. Finally, Section 5 reports on some computational results.

2 Literature Survey

Schedule recovery plans, in opposition to deterministic or robust scheduling, usually use a deterministic schedule and an irregular event as an input an try to recover the now unfeasible schedule at lowest cost. This is an a posteriori approach to cope with irregularities, in the sense that decisions are made when the actual schedule is already unfeasible. Some also refer to this problem as the *day of operations* problem. This category has been developing in the last 10 years mainly, as the more the airline network develops, the more (proportionally) irregularities occur: for each 1% increase in airport traffic it is estimated that there will be a corresponding 5% increase in delays (Schaefer et al., 2005).

As a motivation for our work we refer to Shavell (2001), who studies the economical impact of schedule disruptions on airline companies.

For a general survey on airline scheduling in the recovery perspective, we refer to Kohl et al. (2007), who give an overview of the literature on airline scheduling and discuss different aspects of the problem when irregular events occur.

Wei et al. (1997) introduce a recovery method for crew management based on a multi-commodity integer network flow and also develop a heuristic branch-and-bound search algorithm. The originality of this work lies in the business-like criteria the built solution has to meet: the recovered solution has to be as close to the actual schedule as possible, i.e. there is an upper bound on the number of modified pairings, the number of impacted flights etc.

In his thesis, Sojkovic (1998) introduces three approaches to solve the *Day of Op-*

eration Scheduling problem (*DAYOPS*). The first method consists of regenerating a new flight schedule without changing the rest of the schedule. The Second approach allows for modifications of aircraft itineraries, crew rotations and the planned schedule. Optimization is done separately for aircrafts, pilots and flight attendants. The last approach is based on the Benders decomposition to separate the initial integral multi-commodity flow formulation and solves the resulting problems using the Dantzig-Wolfe formulation by branch-and-bound.

Yu et al. (2003) introduce a decision aid algorithm (*CALEB*) tested on data of Continental Airlines. They 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 the $9/11$ is up to $\$29,289,000$, almost half of it coming from the avoided flight cancellations.

Rosenberger et al. (2003a) present a stochastic approach to model the uncertainty that occurs in the schedule. The stochastic model is a discrete event semi-Markov process. The authors consider independent random events, not taking into account that a severe climatic perturbation could extend on several airports, for example.

Kohl et al. (2007) give a survey of the previous work on airline scheduling and schedule recovery approaches. They also develop a *crew solver* and describe a prototype of a multiple resource decision support system (*Descartes* project), which includes independent algorithms to solve the plane recovery, the crew recovery and the passenger recovery problems. The tests are run on data where small irregularities in a database of 4000 events are generated randomly, at most 10% of the flights being delayed from 15 to 120 minutes.

Rosenberger et al. (2003b) work on different aspects of the airline scheduling problem, mainly in automated recovery policies. One of these projects is based on the aircraft rerouting problem when a schedule has to be recovered. They develop a model that reschedules legs and reroutes aircrafts in order to minimize the rerouting and cancellation costs. They also develop a heuristic to choose which aircraft to reroute, and discuss a model that minimizes the crew and passenger disruption.

When only planes are involved in the recovery problem, we refer to the problem as the *Aircraft Recovery Problem* (ARP).

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

Teodorvić and Stojković (1990) is a direct extension. The authors consider both aircraft shortage and airport curfews and they try to minimize the number of canceled flights, with a secondary objective of minimizing the total passenger delay if the number of cancellations is equal. A heuristic based on dynamic programming is proposed to solve the problem. No experiments are reported.

Several articles published by S. Yan 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 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. Yan and Yang (1996) and Yan and Young (1996) are related to the previous ones.

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. The possibility of swapping aircrafts is taken into account. Instances with three airports with considerable air traffic are presented with several disruption scenarios.

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

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 they penalize in the objective function the deviation from the original schedule and they allow human planners to specify preferences related to the recovery operations. Computational results are presented for a daily schedule recovery of two homogeneous fleets of 16 and 27 aircrafts. Disruption scenarios are simulated grounding one, two or three planes.

There are few contributions in which the maintenance operations are considered as variable. In Stojković et al. (2002) the authors consider the maintenance constraints and provide a real time algorithm that does not affect the routing decision. 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) enforces the satisfaction of maintenance requirements within a given time slot but, in the computational experience, all the flights are constrained to be operated either on time or with 30 minutes delay or canceled, restricting drastically the degrees of freedom of the algorithm and thus the overall complexity of the problem.

In this paper, we consider the ARP problem where all the aircraft related operational decisions can be optimized simultaneously, namely plane re-routing, continuous flight delays, and maintenance operations. Such an integrated approach allows airlines to avoid the iterative procedure between OCC and technical department for validating the produced recovery plan. The results we show come from instances of a medium sized airline with similar sizes than the one we could find in the literature.

3 The Aircraft Recovery Problem

We focus on the Aircraft Recovery Problem (ARP), where decisions are taken on the plane's schedule. When irregularities occur, OCC operators need to find a way to get back to the initial schedule by delaying or canceling flights or reassigning them to other planes (plane swappings). They are given a planned (or initial) schedule and its disrupted state, i.e. the location and time of the planes at the moment the disruption occurs.

The objectives are to both minimize costs produced by delays, cancellations and plane swappings, and makespan. The makespan is the time needed to recover the initial schedule. We refer to it as the *recovery period* denoted by T. The two objectives are contrasting: minimizing T leads to more severe and costly decisions, typically canceling flights, while to minimize recovery costs more time is needed.

It is a common approach, for multi-objective optimization, to fix a threshold on an objective and to optimize the other. We solve the ARP by optimizing the recovery costs given a fixed recovery period and solve the problem iteratively. This decision is motivated by practitioners: it is usually appreciated to have several recovery scenarios based on different recovery periods.

The solution of the ARP is a *recovery plan*, i.e. a new global schedule up to the end of the recovery period T such that the initial schedule can be carried out as planned after T. The recovery plan is composed of one *recovery scheme* for every plane.

Notice that in our formulation, repositioning flights are prohibited due to their high cost even if they might be the only possibility to recover the schedule without canceling flights. Thus, only the initial set of flights can be used in a recovery scheme.

As we discuss in Section 1, the technical constraints involved in the ARP are the maintenance constraints. The maintenances are ruled by civil aviation regulatory authorities: they are enforced by a *resource capacity*, e.g. the number of flown hours, the number of take-offs and landings or the total time between two maintenances. Once the resource capacity limit is reached, the plane is forced to undergo maintenance before being able to perform any flight. The resources are consumed along the time between two maintenances, and are renewed when maintenance is performed. In a general formulation, without explicitly specifying their nature, the resources are given as a vector $\mathbf{H}$ and the capacity limits as a vector $\mathbf{U}$. Thus, u^i is the capacity limit of resource h^i, the i-th component of vector $\mathbf{H}$ and $\mathbf{U}$, respectively.

We define the *state* of a plane p by the vector $[a, t, \mathbf{H}]_p$, where a is an airport, t is a point in time and $\mathbf{H}$ is the vector of consumed resources since last maintenance.

Given a schedule, its disrupted situation and the length of the recovery period, the *initial state* of a plane p is defined as the vector $[a_0, t_0, \mathbf{H}_0]_p$, which is known deterministically as the disruption is known. Notice that for planes performing a flight, the initial state is at the destination airport of this flight, the time being the earliest possible take off time, that we refer to as the *earliest availability time*. A *final state* is a state $[a_T, t_T, \mathbf{H}_T]$ at the end of the recovery period a plane is required to be at in order to perform the schedule as planned after T. As the initial schedule is known, we know where and when the planes are expected and how much of each resource is needed (at least) to reach the next planned maintenance. Thus, $\mathbf{H}_T$ is the maximal resource consumption allowed for a plane to reach the final state. If in the initial schedule a flight with scheduled departure before and landing after T exists, then the final state is set at the take-off time at the origin airport. As we allow plane swappings, a final state is not necessarily associated with a unique plane, and therefore, final states are not indexed by planes.

We illustrate the ARP with a small example. In Table 1 we have a schedule for planes p_1 and p_2. We consider a single resource h^{ToL} being the number of take-offs and landings, where $u^{\text{ToL}} = 20$ and the initial consumption h_0^{ToL} is known for both planes. At time 0905, when p_1 lands in AMS, it comes up to knowledge that an unplanned maintenance has to be performed on p_1 because of problems incurred during the landing phase. It is known that this maintenance will take 2 hours. We are now in a situation of disruption because the schedule cannot be implemented as planned (p_1 cannot take off to MIL at 1000, it will be ready for take off at 1105). Thus we have an ARP where the initial state for p_1 is represented by the state $[\text{AMS}, 0905, 20]_{p_1}$. The initial state for p_2 is $[\text{AMS}, 0930, 10]_{p_2}$ and, assuming that we want to recover the disrupted situation by the evening ($T = 1800$) where both planes will undergo maintenance during the night, meaning that resource consumption can be at capacity limit at T, i.e. $h_T^{\text{ToL}} = u^{\text{ToL}} = 20$, we have two final states [BCN,1800,20] and [GVA,1800,20].

In terms of states, the initial schedule of plane p_2 is given by the following succession of states: $[\text{MIL}, 0740, 10]_{p_2}$, $[\text{AMS}, 1120, 12]_{p_2}$ and $[\text{BCN}, 1430, 14]_{p_2}$.

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

In this small example a possible recovery plan is to swap the assignment of flights F2, F3 and F4 to plane p_2 and of flight F6 to plane p_1 with the recovery schedule of Table 2. The resource consumption constraints at the final states are clearly satisfied, as p_1 reaches the final state [BCN,1800,20] with $h^{ToL} = 4 < 20$ and plane p_2 reaches final state [GVA,1800, 20] with $h^{ToL} = 18 < 20$.

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

The complexity of considering simultaneously the Fleet Assignment Problem and the Plane Routing Problem with explicit consideration of technical constraints (maintenances) makes the problem hard to solve, given that those problems are already NP-hard when considered separately. Nonetheless their combination is taking more and more interest in applications as pointed out in some recent AGIFORS conferences (see *Challenges to growth report*, 2004 and Scheidereit, 2006).

The solution of the ARP requires to reassign aircrafts to flights in order to minimize the recovery costs obeying the operational constraints on the maintenances. Each scheduled flight has either to be served by an aircraft or canceled. The cost of a recovery plan is determined considering the costs related to cancellations, delays, aircraft swappings and additional maintenances. The original schedule must be recovered within a given horizon such that the maintenance requirements and aircraft type at the end of the recovery period are compatible with the originally planned schedule.

3.1 Definitions

In this section we provide some definitions which we use through the paper.

We assume as given:

- the recovery period T,
- the number $|\mathbf{H}|$ of resources that enforce maintenance,
- the maximal resource consumption vector $\mathbf{U}$ enforcing maintenance,
- the delay cost per time unit c^D,
- the set of airports A and for each airport $a \in A$:
  - the minimal turn around time mtt_a, which is the time needed to prepare an aircraft for the next take-off,
  - the set of activity slots O_a, which are time intervals when take-off and landing operations can take place,
  - the set of maintenance slots M_a, which are time intervals when maintenance operations can take place,
  - the maintenance duration d_a^M,
- the set of planes P and for each plane $p \in P$:
  - the initial state $[a_0, t_0, \mathbf{H}_0]_p$, specified by initial time-location and initial resource consumption,
- the set of required final states S, where $S_p \subseteq S$ is the set of final states coverable by plane $p \in P$. For each $[a_T, t_T, \mathbf{H}_T] \in S_p$:
  - the required location a_T and point in time t_T,
  - the maximal allowed resource consumption given by $\mathbf{H}_T$,
- the set of flights F, where $F_p \subseteq F$ is the set of flights that can be flown by plane $p \in P$. For each $f \in F$:

– the scheduled departure time sdt_f,

– the duration d_f,

– the flight cost g_f

– the cancellation cost c_f,

The set of planes P and the initial state $[a_0, t_0, \mathbf{H}_0]_p$ for each plane $p \in P$ are used to model plane disruptions such as unavailability or plane delay by increasing the earliest availability time t_0. Moreover unpredicted aircraft maintenance can be modeled by setting the resource consumption in the initial state at the capacity $\mathbf{U}$, as shown in the example, where we create the initial state $[\text{AMS}, 0905, 20]_{p_1}$ to enforce maintenance. The activity slots and maintenance slots are used to model airport closure or maintenance disruptions, for example due to strikes.

The time t_T of a final state $[a_T, t_T, \mathbf{H}_T]$ might be smaller than T when a scheduled flight f leaps over T. In this case $t_T = sdt_f - mtt_{a_T}$.

Finally, remark that the final state is a crucial point in recovery problems. Final states are what differentiates recovery to usual scheduling problems. In a normal plane routing problem, we want to cover as many flights as possible but without having restrictions on the final plane's location, as for recovery, reaching the final states is crucial in order to carry out the initial schedule after the recovery period. Not covering a final state means recovery is not realized.

In the next section we introduce the recovery network model. Each plane has its own network given its initial state and the set of final states that are compatible with it. This is the *recovery network* of the plane.

4 The Column Generaion Algorithm for the ARP

The objective of ARP is twofold: minimize both the recovery period T and the recovery costs. Our approach is to optimize the costs given a fixed recovery period. We then generate several recovery plans for different values of T that will help the decision taker to identify the best trade-off between cost and time.

For each fixed value of T, we model the ARP as a set partitioning problem with additional constraints. This model is based on an integral combination of feasible recovery schemes, one for each plane of the fleet. The set of feasible schemes is exponential in size. Thus, we resort to column generation to solve the linear relaxation of this model, commonly referred as Restricted Linear Master Problem (RLMP). We refer the reader to Desaulniers et al. (2005) for the theoretical details of the method.

The key point of a column generation algorithms is the way we model and solve the pricing problem. In the case of the ARP as formulated in section 4.1, the pricing problem is to find a recovery scheme, i.e. a schedule for a plane within the recovery period, which is promising for improving the solution of the RLMP. The model we use for the computation of these recovery schemes is the recovery network described in section 4.2. Section 4.3 explains the main properties and parameters of the recovery network generation algorithm that is reported in Appendix A.

Section 4.4 describes the dynamic programming algorithm used to solve the pricing problem on the recovery networks, which is based on modern algorithmic techniques, namely decremental state space relaxation, presented by Righini and Salani (to appear). Finally section 4.5 reports some implementation details of the whole algorithm.

4.1 Applying Column Generation to the ARP

Let Ω be the set of all possible single-plane recovery schemes. They must be combined together to obtain a minimum cost recovery plan for the set of scheduled flights such that each flight is either serviced by exactly one plane or canceled. In addition, to carry out the original schedule after T, all the final states must be reached by a plane with enough resource potential.

We model the ARP as a set partitioning problem with additional constraints (MP) as follows:

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

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

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

$$\sum_{r \in \Omega} b_r^p x_r \leq 1 \qquad \forall p \in P \qquad (4)$$

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

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

Each recovery scheme r has a cost c_r and is associated with a binary variable x_r that is equal to one if it is taken into the solution, 0 otherwise. A recovery scheme is described by the binary constants b_r^f, b_r^s and b_r^p. Those constants take value one if the scheme r covers the flight f, ends with the final state s and is serviced by plane

p, respectively. A binary variable y_f is associated with each flight and it is equal to one if the flight f is canceled with cost c_f. Constraints (2) ensure that each flight is either serviced or canceled. The feasibility of the already planned schedule at the end of the recovery period is ensured by constraints (3). Constraints (4) ensure that an aircraft can be assigned at most to one recovery scheme. Constraints (5) and (6) enforce integrality on the variables.

Since the dimension of the set Ω is exponential in the dimension of the problem, we consider a subset of recovery schemes, $\Omega' \subseteq \Omega$ and we solve the linear relaxation of the so obtained restricted problem (RLMP). We then recourse to column generation either to prove the optimality of the linear problem or to generate new profitable recovery schemes to enter the formulation. If the optimal solution of the restricted master problem is not integral we recourse to an enumeration tree where, at each node, we take branching decisions.

Given the optimal solution of the linear restricted master problem, z^*_{RLMP}, the column generation algorithm solves a pricing problem to compute the recovery scheme r with minimum reduced cost for each aircraft p. The reduced cost is computed considering the dual variable λ_f associated with each flight f, the dual variable related to the final states η_s and the non-positive dual variable μ_p of the plane p as follows:

$$\tilde{c}^p_r = c^p_r - \sum_{f\in F} b^f_r \lambda_f - \sum_{s\in S} b^s_r \eta_s - \mu_p \qquad \forall p \in P. \tag{7}$$

If a column with $\tilde{c}^p_r < 0$ exists it is added to the RLMP, i.e. added to Ω', otherwise the LP optimality is proved. We thus have to compute, for each plane p, the recovery scheme minimizing the reduced cost given the dual multipliers λ_f, η_s and μ_p, i.e. find the feasible combination of the vector $(b^f_r, b^s_r, b^p_r)^T$ minimizing $\tilde{c}^p_r$.

We introduce in the next section the recovery network model that allows to compute, for each plane independently, the recovery scheme minimizing the reduced cost as a Resource Constrained Elementary Shortest Path Problem (RCESPP).

4.2 The Recovery Network

We introduce an extension of the time-space network model proposed by Argüello et al. (2001) that includes the plane maintenances and we describe a generation and a preprocessing algorithm to control the size of the network in terms of nodes and arcs. An independent recovery network associated with every plane and therefore, we consider a unique plane $p \in P$ and might omit the index p for simplicity of notation.

In the time-space network, a node $\{a, t\}$ corresponds to a point in space and time and it is labeled with a unique state $[a, t, \mathbf{H}]$.

A schedule of a plane in a time-space network is a set of *nodes* $\{a, t\}$ corresponding to the *earliest departure time* (edt) t at corresponding airport a, that are linked by *flight arcs*. The grounding time between the earliest departure time and the real take-off time at the origin airport a and the minimal turn around time $\mathtt{mtt}_{a'}$ at the destination airport a' are included in the flight arc, in order to avoid vertical arcs.

In the graphical representation of the network, we report time vertically and space horizontally.

Since the initial state of every plane is known, we introduce a (unique) *source node* $\{a_0, t_0\}$ corresponding to the location and first availability time of a plane and labeled by the initial state $[a_0, t_0, \mathbf{H}_0]$. The initial state records the information about the resource consumption associated with the plane.

We define a *sink node* $\{a_T, t_T\}$ associated for every final state $[a_T, t_T, \mathbf{H}_T]$ in the set of the coverable final states S_p of plane p. Recall that we do not restrict every plane to recover to it's initial schedule, we thus might have more than one sink node for each network. Information on the needed resource potential before the next maintenance is stored in $\mathbf{H}_T$, which is an upper bound on the consumed resource. A sink cannot be covered by a plane with to high resource consumption, as the plane would not be able to carry out the initial schedule after the recovery period until the next maintenance.

Intermediate nodes are time-locations where the plane is ready to take off after having performed some flights. The particularity of a node is that it is only a transition state that the plane can visit.

We have three type of nodes: *sources*, *sinks* and *nodes* and four arc types: *flight*, *maintenance*, *termination*, and *maintenance termination*.

A flight arc is associated with a flight and links two nodes corresponding to origin and destination airports at specific times. We represent several possibilities to delay a flight by several flight arcs connected to different nodes. A maintenance arc is similar to a flight arc, except that the maintenance is performed *before* proceeding the flight at the origin airport. Flight and maintenance arcs are not permitted to reach a sink node by convention.

Termination arcs link nodes (including eventually the source) to sinks. A termination arc is never associated with a flight, thus it is always vertical. A maintenance termination is the same as a termination arc but where a maintenance is performed before reaching the sink.

Figure 1 shows how the different nodes and arcs are represented.

Each arc affects the resource consumption $\mathbf{H}$ of the plane in a different way: flight arcs increase the resource consumption by a certain amount, whereas maintenance arcs reset it to zero before performing the flight.

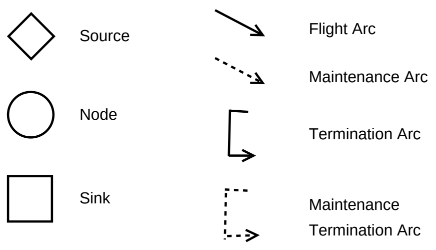

Figure 1: Representation of the different nodes and arcs

Finally we associate a cost with every arc type as follows:

- flight arcs: the flight cost plus the delay cost,
- maintenance arcs: the sum of the flight cost, the maintenance cost and the delay cost,
- termination arcs: no cost,
- maintenance termination: the maintenance cost.

The delay cost is incurred by a delayed departure. Usually, the cost of a delay is measured linearly with a time unit delay cost of 72 Euros per minute (Cook et al., 2004).

The algorithm for the recovery network generation is reported in Appendix A in Algorithm 1.

We consider the example of the schedule of Table 1 for plane p_2. We assume that p_2 can also cover the flights initially scheduled for p_1 and cover both final states. We thus have initial state [MIL,0740,10] and we create the source {MIL,0740} and apply Algorithm 1, with F_p being the set of flights $F_p = \{F1, F2, F3, F4, F5, F6\}$, and the set of sinks being $S_{p_2} = \{[GVA, 1800, 20], [BCA, 1800, 20]\}$. Airports are all in an activity slot with equal minimum turn around time of 30 minutes and there is only a maintenance slot in AMS, with maintenance duration $d_m = 1h$. The generated recovery network is shown in Figure 2 (we remove flights delayed by more than 4 hours).

The recovery network helps us to identify the different possible ways to recover the schedule. Each of them corresponds to a path from the source to a sink. In Figure 2, we clearly see the exponential behavior of the recovery network generation algorithm. We have 8 different possible paths from the source to a sink, each one corresponding to a feasible recovery scheme for plane p_2. The path corresponding to the recovery scheme of plane p_2 in Table 2 is the succession of the nodes {MIL,0740}, {AMS,1000}, {MIL,1200}, {BCN,1410}, {GVA,1620} and {GVA,1830} (including the source and the

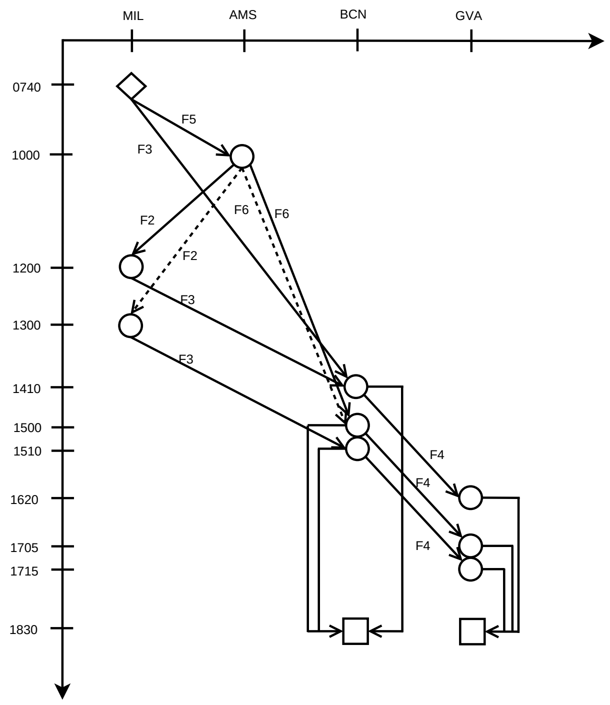

Figure 2: Recovery network of plane p_2 with initial schedule of Table 1 and initial state [BCN,0740,10].

sink). The succession of nodes is linked by the flight arcs corresponding to flight F5, F2, F3 and F4.

In this example, there is no maintenance termination arc, as there is no maintenance slot at GVA nor at BCN, the locations for the final states. Notice that there can be several arcs, both flight or maintenance, associated with the same flight but delayed.

In the next section we describe a generation algorithm for the recovery networks where we introduce a set of parameters that allow to control the exponential behavior.

4.3 Properties of the Recovery Network Generation Algorithm

The main idea of the generation algorithm is to extend dynamically the labels associated with every node (by increasing time) $\{a, t\}$ with all possible flights $f \in F_p$ departing at a to create a flight arc and a maintenance arc and corresponding destination nodes, and if there is a final state in airport a, to create a termination and maintenance termination arc. This approach has an exponential behavior in the number of flights, we thus introduce some parameters to control it.

Following the approach of Argüello et al. (2001), we use time discretization in order to control the size of the networks. The idea is to merge all nodes at same location within a time window, whose width is of size $\frac{T}{\Delta}$, where Δ is a parameter, into one single node. In order not to discard any feasible solution, we keep a time label corresponding to the earliest departure time edt from a node. We then use edt to determine whether a flight arc should be created or not. Notice that by doing so, we might underestimate the true delay cost, but the delay is evaluated exactly in the pricing algorithm. This parameter has been introduced for practical reasons: a high value for time discretization drastically reduces computing time and gives to the planner a qualitative feedback on the recoverability of the schedule and allows him to estimate reasonable values for the recovery period, T.

We then introduce two parameters to restrict the flight extension set. The first is the delay bound τ. We remove from the flight extension set all flights having a bigger delay than τ. The maximal waiting time bound ψ is similar to τ but in the reverse way. A flight is removed from the extension set of a node if $sdt_f - edt > \psi$, i.e. if the plane is grounded for a too long period before performing flight f.

The grounding time parameter Γ is the extension of ψ to termination arcs. If the node is too far in time from a sink, we do not create a termination nor a maintenance termination arc from this node.

The parameters ψ and Γ capture the fact that a long inactivity period is unlikely to be optimal.

The parameter $\rho \in [0, 1]$ corresponds to the minimal resource consumption proportion required before performing a maintenance and it captures the fact that it is unlikely to perform maintenance when only a low percentage of the resources are consumed.

In order to obtain an operational network model with a reasonable number of nodes and arcs, a preprocessing is necessary to decrease the complexity and remove useless arcs an nodes after the generation.

We first check feasibility with respect to resource consumption: for every node we compute upper and lower bounds ($\overline{h}^i$ and $\underline{h}^i$) for the consumption of resource h^i. Note that with that notation, parameter ρ allows a maintenance arc only if the upper bound $\overline{h}^i \geq \rho u^i$.

We illustrate the principle with the example of the number of flown hours since last maintenance. In a feasible schedule, a plane cannot perform more than u^{Flh} fly hours between two consecutive maintenances.

We compute the lower and upper bounds on the resource consumption using an unconstrained shortest and longest path algorithm, respectively, according to the resource consumptions. Given the capacity limit u^{Flh} and the corresponding lower bound $\underline{h}^{Flh}$ it is possible to erase an arc from the network if $\underline{h}^{Flh} + d_f > u^{Flh}$.

We also compute a shortest path with respect to the resource consumption from a sink to every node $\{a, t\}$, which corresponds to the minimal resource potential $\underline{h}^{\mathtt{Flh}}_{\mathtt{sink}}$ needed to reach the sink. Thus, if $\underline{h}^{\mathtt{Flh}}_{\mathtt{sink}} + \underline{h}^{\mathtt{Flh}} > u^{\mathtt{Flh}}$, there is no feasible path from the source to the sink going through node $\{a, t\}$. We maintain a list of reachable sinks for every node. If this list is empty at a node because of resource consumption we remove the node as well as all its ingoing and outgoing arcs.

Finally we remove all nodes (except the source) that have no predecessor and all nodes (except the sinks) that have no successor, as they are not leading to any feasible recovery scheme for plane p.

We thus have a recovery network for every plane that encodes all interesting recovery schemes as a path from the unique source to a sink. In order to solve the pricing problem, we thus have to compute the one minimizing the reduced cost $\tilde{c}^p_r$, given the dual multipliers λ_f, η_s and μ_p. We describe the RCESPP algorithm that solves the pricing problem in the next section.

4.4 Resource Constrained Elementary Shortest Path Problem

A pricing problem needs to be solved for each aircraft on its own recovery network, i.e. we need to solve a Resource Constrained Elementary Shortest Path Problem (RCESPP) on the recovery network for each plane. In the reminder we omit index p from the reduced cost formulation in (7).

Variable μ is a constant and it is not considered in the optimization but only to compute the final reduced cost. The dual variables λ_f and η_s can be taken into account in the recovery network by adding them to the flight and termination arcs, respectively, as follows:

- flight arcs: cost $-\lambda_f$
- maintenance arcs: cost $-\lambda_f$
- termination arcs: $-\eta_s$
- maintenance termination: cost $-\eta_s$

By consequence negative cost arcs could be present.

By updating the arc costs as described above, solving the pricing problem amounts to solve a resource constrained elementary shortest path problem (RCESPP) in each recovery network. The optimal column is the one having the minimum reduced cost of the $\mid P \mid$ columns obtained (one for each plane). Moreover, resource consumption ensures feasibility according to maintenance requirements, whereas elementarity is set on flights, ensuring one flight is covered at most once by a feasible column.

To solve the RCESPP for each recovery network, we use the algorithm proposed by Righini and Salani (2006). The idea of the algorithm is to create labels associated with nodes, which hold a feasible partial path to reach the node. If several labels are active at the same node, it is possible to eliminate some labels that are dominated, i.e. that we know that they cannot lead to the optimal path. In our case, one label at node n is given by the vector $(\mathbf{H}, C, n)$, where $\mathbf{H}$ is the vector of consumed resources since last maintenance at this stage of the partial path and C its reduced cost. We say that label $(\mathbf{H}', C', n)$ is dominated by label $(\mathbf{H}, C, n)$ at node n if and only if:

- $h^i \leq h^{i'}, \forall i = 1, \cdots \mid \mathbf{H} \mid$,
- $C \leq C'$,
- at least one of these equalities is strict.

If a label is not eliminated by domination, it will be extended through all feasible arcs (n, m) to a new label at node m. The optimal solution is the label with lowest cost at the sinks.

The network is acyclic by construction but, since multiple arcs could identify the same flight in order to model delay decisions, we must ensure that there are not two different arcs corresponding to the same flight traversed in the same path. Indeed, the elementarity of the flights corresponding to the traversed arcs is needed as we do not allow a flight to be covered more than once. This elementarity does not come for free though, the only cost minimization objective of a resource constrained shortest path cannot ensure it. To enforce elementarity, we use the idea introduced by Beasley and Christofides (1989), by adding a dummy resource vector L, where L_f is one if flight f is covered, and 0 otherwise. Thus, an arc (n, m) corresponding to a flight that has already been covered by the partial path $(L, \mathbf{H}, C, n)$ will not be feasible for label extension. The disadvantage is that the domination rules must be extended by adding the following rule for $(L, \mathbf{H}, C, n)$ to dominate $(L', \mathbf{H}', C', n)$:

- $L_f \leq L'_f, \forall f \in F$

To tackle the computational effort issued by the additional elementarity constraint we exploited the Decremental State Space Relaxation (DSSR) technique that has been recently introduced by Righini and Salani (to appear).

In order to control the number of non dominated labels, we discretize the resource consumption as we do for time: resource consumptions falling into the same interval are considered as equivalent. We introduce a logarithmic resource discretization, given as the parameter θ. It corresponds to the number of logarithmic intervals the resources are divided in, and the length of the intervals being proportional to $\log(\theta)$. The intervals for resource $h^i \in \mathbf{H}$ are denoted by I^i_j, $j = 1 \cdots \theta$. The idea behind this logarithmic discretization is that for low resource consumption maintenance is unlikely, and few precision is needed. In the RCESPP algorithm, the domination criteria of a label in the resource dimension compares the resource interval I^i_j rather than the real resource consumption h^i, i.e. if $h^i \in I^i_j$ and $h^{i'} \in I^i_{j'}$, domination occurs if $j \leq j'$. Therefore a label can now erroneously dominate another if they belong to the same discretization interval.

Notice that in a linear discretization, it might occur that two values of resource consumption fall again in the same discretization interval when taking more intervals. In the logarithmic case, for increasing θ, if two labels fall into different intervals once, they always stay in different intervals for bigger θ. The example in Figure 3 shows the behavior for increasing θ with linear interval lengths on the left hand side and with logarithmic lengths on the right hand side for a resource h^i and resource limit $u^i = 100$. We see that 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 $\theta = 1$, they are not for $\theta = 2$, they are again when $\theta = 3$ etc. Figure 3 also shows the saturation effect in logarithmic discretization when θ grows too large, i.e. that we get almost empty intervals for big θ.

4.5 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) project[1]. Test are run on a computer with a 2GHz processor and 2GB memory.

In our algorithm, we perform column generation only at the root node of the search tree, thus we solve the linear relaxation of the root node to optimality and we obtain a valid lower bound. Then we find an integral solution by branching on the column variables closest to 0.5. The algorithm we obtain is therefore an optimization based heuristic with a measure of the optimality gap.

[1]www.coin-or.org

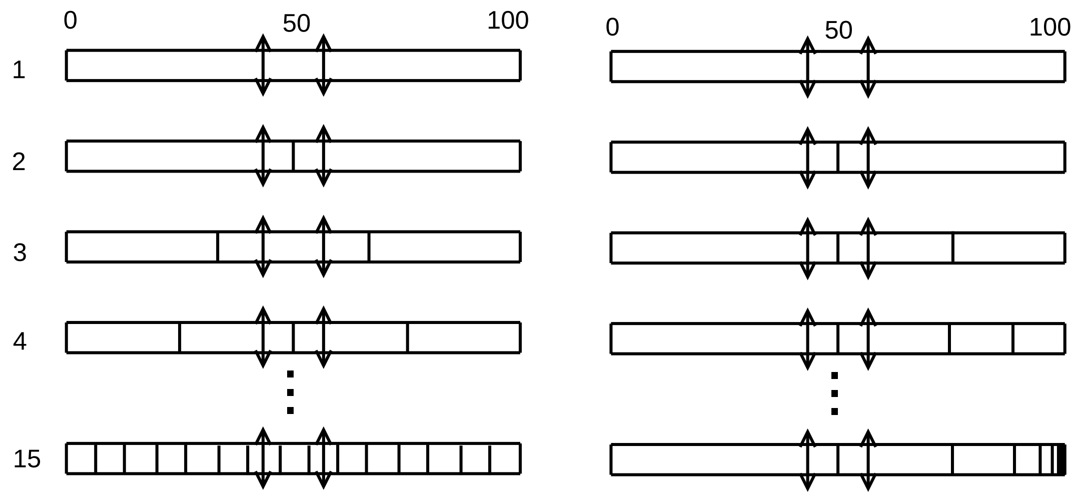

Figure 3: Linear on the left and logarithmic discretization on the right for increasing number θ of intervals ($\theta = 1, 2, 3, 4, 15$).

We devise a branching strategy where the structure of the pricing problem is not affected. First of all we search for flights covered by different planes and we branch on the flight-plane association. When no flight is covered by different planes we search for flights covered fractionally by several recovery schemes belonging to the same plane. We then branch on flight sequences following the scheme presented in Ryan and Foster (1981). The results we present in this paper are referred to the optimization based heuristic.

Column generation is known as a *primal method*, that is while the primal feasibility of the RLMP is guaranteed, the feasible dual vector is searched trough the addition of 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 column generation methods (see for example Vanderbeck, 2005), one needs to provide a relevant set of columns to obtain a good estimation of the dual vector and to prove its optimality in the end of the generation. As discussed in Section 4.4, a NP-hard pricing subproblem must be solved for each plane of the fleet to prove LP optimality.

Since the dual vector estimation during the early iterations of the method is poor, it is a common practice to solve the pricing problem heuristically in order to produce quickly negative reduced cost columns.

We obtain three pricing heuristics from the exact dynamic programming method using two relaxations of the problem. In the first relaxation we keep the elementarity constraint during the construction of the partial paths but we relax it in the domination test. In this way we enlarge the possibility of a good label, in terms of reduced cost and resource consumption, to dominate the others. The second method is to order the labels by reduced cost and to bound the number of active labels for each node by a constant k. Thus, thanks to the time discretization, the heuristic algorithm we obtain is polynomial in time and space since the number of nodes as well as the number of labels are bounded by a polynomial function.

We combine the two relaxations to obtain a third and fast heuristic we apply as the first choice. If this heuristic fails in finding new columns we apply the heuristic with fixed number of labels per node varying the bound on the number of active labels. Then we apply the heuristic in which we relax the elementarity dominance criteria. Finally if none of the heuristic methods returns a column with negative reduced cost, we resort to exact pricing.

Moreover, we add to the master problem all the new columns with negative reduced cost we find in the heuristic phase to accelerate the convergence of the column generation, useless columns are then removed from the LP by reduced cost fixing.

5 Computational Results

The data used in the instances comes from Thomas Cook Airlines (TC). TC is a medium size airline relying on a heterogeneous fleet of 30 aircrafts and operating around 500 flights a week. The size of the flight set in our instances varies from 40 to 250 flights but it is associated with a unique fleet. Thus, every flight is coverable by any plane, which increases the combinatorial complexity. We use the schedule implemented by Thomas Cook during May 2006 and we simulate some disruption scenarios using the following experimental setup:

- size of the fleet concerned by the disruptions: 5 and 10 aircrafts;
- recovery period T: from 1 to 7 days;
- delay of a plane: availability of plane is later than expected;
- grounding a plane: the plane is never available during the whole recovery period;
- close 1 airport: activity slots of an airport do not cover the whole recovery period;
- force maintenance: initial resource consumption of some aircrafts is set too high not to perform a maintenance.

Note that to get instances with larger number of flights, we need to consider a longer recovery period instead of considering a bigger fleet. In our experiments we notice that the computational complexity is affected more by the length of the recovery period than by the size of the fleet. This increase is due to the high number of leg copies (same origin and destination) that occurs more often with long recovery periods.

We extract initial schedules from TC's schedule of May 2006 and simulate some disruptions using delays or forced groundings. We derive more than 20 instances, combining grounding and delay for some of them. The name of the instance is related to its size: xD_yAC, where x is the number of days considered in the recovery period and y is the number of aircrafts. We denote the number n of grounded planes by ngrd and the number m of delayed planes by mdel.

We present the results in details for the original schedules and some small disruptions, i.e. plane delays, in a qualitative way in order to get a feeling of the solvable instances. We then present the results of a generated set of instances derived from an original schedule. We also discuss the impact of the parameters on the solution quality and present the added value of considering maintenances when solving the ARP. Finally, we illustrate the conflicting effect between cost and makespan optimization with an example.

5.1 Recovery Plans

Table 3 shows the size of the instances we are able to solve and the needed computation time. We see that the small disruption introduced in instance 2D_5AC_1del is recoverable within 2 days. For this reason, in this instance, considering more days or more planes will lead to the same recovery decisions, namely cancel two flights, and delay another four by the same amount. The results of the corresponding instances are not shown here, as the recovery decision remains the same and computational effort is equivalent to similar problem sizes.

When a schedule can be carried out almost as initially planned, the algorithm solves the problem within a second on the root node. Only bigger instances require branching, which drastically increases the computation time, as shown by the two last instances.

We also mention here that with the set of parameters we use to solve the instances, instance 7D_16AC fails because of too high memory consumption. The results presented for this instance are obtained with more restrictive parameters on delay and on inactivity time (τ and ψ).

We test the behavior of the algorithm on 12 different disrupted instances obtained from an instance with 10 planes and 36 flights during one day. The instance is a

Instance	2D_5AC	2D_5AC_1del	2D_10AC	2D_10AC_1del	2D_10AC_2del
# planes	5	5	10	10	10
# flights	38	38	75	75	75
# delayed planes	0	1	0	1	2
# canceled flts	0	2	0	2	2
# delayed flts	0	4	0	4	5
total delay [min]	0	969	0	969	989
max delay [min]	0	370	0	370	370
cost	380(*)	21175(*)	750(*)	21545(*)	21745(*)
tree size	1	1	1	1	1
run time [s]	< 0.1	< 0.1	0.7	0.7	1.0

Instance	3D_10AC	4D_10AC	5D_5AC	5D_10AC	7D_16AC
# planes	10	10	5	10	16
# flights	113	147	93	184	242
# delayed planes	0	0	0	0	0
# canceled flts	0	0	0	0	0
# delayed flts	0	0	0	0	11
total delay [min]	0	0	0	0	310
max delay [min]	0	0	0	0	45
cost	1130(*)	1470(*)	930(*)	1840(*)	5600
tree size	1	1	1	5	2033
run time [s]	3.0	6.5	1.0	29.1	3603

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

hub and spoke situation where all the planes start and end at Denver. Disruption scenarios consider either delayed planes only, grounded planes only, a mix of delayed and grounded planes or airport closure(s).

The instances `Den_3x100` and `Den_1x300` simulate a closure of the hub airport, i.e. Denver. In the first instance, Denver airport is closed during three 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 try to simulate a storm affecting several local airports. In instance `Den_Storm1`, four airports are closed for 300 minutes, and is instance `Den_Storm2`, the same airports are closed 500 minutes.

Table 4 shows the results of the algorithm applied to the different instances. The first two lines report the number of delayed and grounded planes, respectively. The third line reports the number of flights directly affected by the disruption without any forecast on the propagation of the disruption to other flights, thus it represents the minimum number of flights on which the planner must take a recovery decision. It is evident (see for example instance Den6grd) that the 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 equal to 16).

Instance	Den2del	Den2grd	Den4del	Den4grd	Den2del2grd	Den6del
# delayed planes	2	0	4	0	2	6
# grounded planes	0	2	0	4	2	0
# min. affected	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	Den6grd	Den3del3grd	Den_3x100	Den_1x300	Den_St1	Den_St2
# delayed planes	0	3	0	0	0	0
# grounded planes	6	3	0	0	0	0
# min. affected	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.

We see from Table 4 that a grounded plane incurs more often flight cancellation than a delayed plane. This follows intuition, as when a plane is grounded, 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 the grounded plane and the delays incured by the delayed plane are combined. This is a direct consequence of the network's density, meaning that if there are not enough available planes, the other plane's schedules do not permit to introduce 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 `Den_3x100` has more affected flights, the solution is better than for `Den_1x300`. The explanation is that the closure is splitted and covers more take-offs and landings at Denver, but the slots between closures allow planes to leave and start rotations from Denver to then land and take off at airports that are not affected by Denver's closure. This is not possible before the whole 300 minutes closure are over in `Den_1x300`. 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 could be covered but only by inducing huge delays.

These different instances allow us to derive some informations about the algorithm's behavior against increasingly severe disruptions.

The parameters may significantly influence the computation time but their actual impact on the solution depends strongly on the instance itself. We test several instances with different disruption types. The delay, maximum waiting time and maximum grounding time bounds (τ, ψ and Γ respectively) are drastically decreasing the computation time. The quality of the solution is not affected as long as the bounds are higher than a certain threshold corresponding to the highest delay of all the planes at the beginning of the recovery period for τ, and to the maximal grounding time between two flights for ψ.

One sensitive parameter is the estimated delay cost per minute c^D, which controls the delay limit before deciding to cancel a flight on the one hand. The lower the delay cost, the more the algorithm tries to cover all the flights regardless of the produced delay. On the other hand, if delay cost is high, the recovery plan will avoid as much as possible 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. The logarithmic resource discretization outperforms the linear one when using (the same) low number of intervals (up to $\theta = 10$). When increasing θ, the linear resource discretization performs globally better, but not necessarily homogeneously.

This is due to the saturation effect of the logarithmic resource discretization: when increasing the number θ of intervals, we reach a point where we get empty intervals, containing no realizable value of the resource consumption. Therefore, after a certain threshold, we do not gain any more precision. In opposition, increasing the number of linear intervals decreases linearly their length and thus we gain more precision.

However, the solutions do not improve homogeneously when increasing the number of linear intervals. This is due to the fact that labels do not always fall in the same interval for increasing θ: improvement can only happen when an erroneously dominated label falls outside its dominant label's interval, and the two labels might oscillate between same and different intervals for increasing values of θ with linear interval (see Figure 3), explaining the non homogeneous decrease of the solution cost.

The number of intervals plays a crucial role to control memory usage, thus it is more interesting to use a low number of intervals, making the logarithmic resource discretization more efficient.

We see from these results that our algorithm is able to solve to optimality instances of up to 184 flights in less than 30 seconds. Even if state of the art algorithms in the literature are able to address bigger instances they do not consider explicitly maintenance optimization. Moreover, notice that in the instances we address, the average number of flights per plane is higher than what we find in the literature.

We see that the introduced parameters are useful to accelerate computation and that they do not decrease the solution quality dramatically. The resulting recovery plans are indeed following intuition and behave as we would expect, deleting as few flights as possible by swapping planes or delaying flights. Finally, we see that the solution of the recovery algorithm depends on the initial schedule as much as on the actual disruption. The density of the schedule Denver instances of Table 4 shows propagation effect due to the low rest time of the planes between two flights. In the next section, we discuss the advantage of planning the maintenance and compare the results of our algorithm against simulated benchmark algorithms, and show the behavior of the solution when increasing the recovery period length.

5.2 Maintenance Scheduling

We want to show the added value of optimizing maintenances simultaneously with flight re-scheduling. To this extent we compare different recovery approaches that can be implemented at OCCs. The first approach (that probably no planner would use) is to use aircrafts up to their maximal resource consumption without scheduling a maintenance and eventually ask for a 5 to 10% limit extension, as mentioned in section 1. We refer to this approach as the *No maintenance* algorithm. The second approach, which is probably closer to human planner behavior, is to schedule a maintenance as soon as the resource consumption gets critical. We refer to it as the *Greedy maintenance* algorithm. This is achieved by fixing the minimal consumption before maintenance parameter ρ to a high value (it is set to 0.9 in our test, meaning maintenance can be performed when at least 90% of the resource is consumed). The third approach, called *Maintenance Optimization* is to let the algorithm schedule the maintenances in an optimal way ($\rho = 0$).

We consider two instances of 36 and 147 flights, respectively. For an illustration, we consider the first small instance with 10 planes and 36 flights used in the two previous sections where all the planes start and end at Denver. For this reason, we allow maintenance only at Denver, at any time in the recovery period. One plane, with ID P42, has a high resource consumption at the beginning of the recovery scheme (88%). The results for different algorithms are presented in Table 5.

Without allowing any maintenance, plane P42 cannot perform its schedule as expected. We thus need to delay 2 flights for a total delay of 210 minutes to get a feasible solution. However when allowing a 5% consumption excess for plane P42 the original schedule can be carried out as planned without performing any additional maintenance.

The solution given by the greedy maintenance algorithm is better than the one with no maintenance. This solution still has two delayed flights, but the total delay is reduced to only 30 minutes, which is a huge saving compared to the 210 minutes if no maintenance is possible.

Finally, our algorithm allows to forecast the maintenances and places them with more flexibility. Using the maintenance optimization algorithm, P42 has now the possibility to perform maintenance at the beginning of the recovery period and by doing so, no flight is delayed at all.

Instance	No maint.	No maint.+ 5%	Greedy maint.	Maint. Opt.
# canceled flts	0	0	0	0
# delayed flts	5	0	2	0
# uncovered final states	0	0	0	0
total delay [min]	210	0	30	0
max delay [min]	80	0	20	0
Additional costs	2100	0	800	500

Table 5: Results for maintenance optimization against three different simulated behaviors: no maintenance at all, no maintenance with a 5% consumption excess allowed and a greedy maintenance scheduling algorithm.

The presented Denver instance is made up artificially to show in a qualitative way the differences between our algorithm and the other algorithms. We generate a set of instances derived from the instance `4D_10AC` with 10 planes and 147 flights, allowing maintenances at half of the airports to get a quantitative comparison. We generate the initial resource consumptions for the 10 planes with different mean values and variances. Thus, we are in the situation where we know at the beginning of the recovery period which planes will have to perform a maintenance earlier than expected and try to recover from this situation. We show the results of our algorithm against the greedy algorithm and three simulations where no maintenance is allowed, but respectively 5,10 and 20% more resource is available. Table 6 shows the average results over the 10 instances.

Instance	No maint. + 5%	No maint. + 10%	No maint. + 20%
# canceled flts	52.7	46.7	33.2
# delayed flts	5	4.7	5.5
# uncovered final states	1.2	0.7	0.3
total delay [min]	851.3	635.7	712.5
max delay [min]	271.3	251.5	218.2
cost	289462	272067	144388
optimality gap [%]	0.61	0.54	1.27

Instance	Greedy maint.	Maint. Opt
# canceled flts	2.2	2
# delayed flts	2.7	1.5
# uncovered final states	0.1	0.1
total delay [min]	89.6	52.3
max delay [min]	37.7	37.1
cost	15881	14683
optimality gap [%]	0.73	0

Table 6: Average results for maintenance optimization on 10 randomly generated instances.

We see that even when allowing up to 20% more resource consumption, we still have a massive cancellation rate and hudge delays. We however mention that the 20% performs better than our algorithm for one of the ten instances, where actually this 20% increase is sufficient to perform the whole schedule, given the initial resource consumption, without any maintenance. In this instance, the only added costs in the solution of our algorithm are the maintenance costs, no delay nor flight cancellation is needed. Remarkably even with the 20% increase of the resource capacity, we get only seven feasible solutions out of the ten instances.

With the greedy algorithm, the solution performs much better than the no maintenance cases reducing the average cost by one order of magnitude. However, the greedy algorithm leads to solutions that are 7.5% higher than those given by our algorithm. The main savings are made thanks to delay reductions. The greedy algorithm finds the same solution as our algorithm for three of the ten instances but never leads to a better one.

We see from these results that considering maintenances is not only necessary in order to ensure feasibility of the recovery scheme, but the more freedom given to the maintenance scheduling, the better the solution.

These results show that the maintenance planing does improve the solution quality, it's main advantage being to reduce the delay. The results clearly show that allowing resource consumption excess is not performing well and this, without taking into account the negotiation and incurred costs for the airline to get the capacity limit extension.

5.3 Trade-off between cost and recovery period

We show in Table 7 the behavior of the solution when increasing the recovery period T. We solve the 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 7 shows the details of the solutions, where *additional costs* mean the aggregated delay and cancellation costs over the whole period, assuming schedule is recovered at T. Figure 4 shows the Pareto frontier, i.e. the additional costs against the recovery period length.

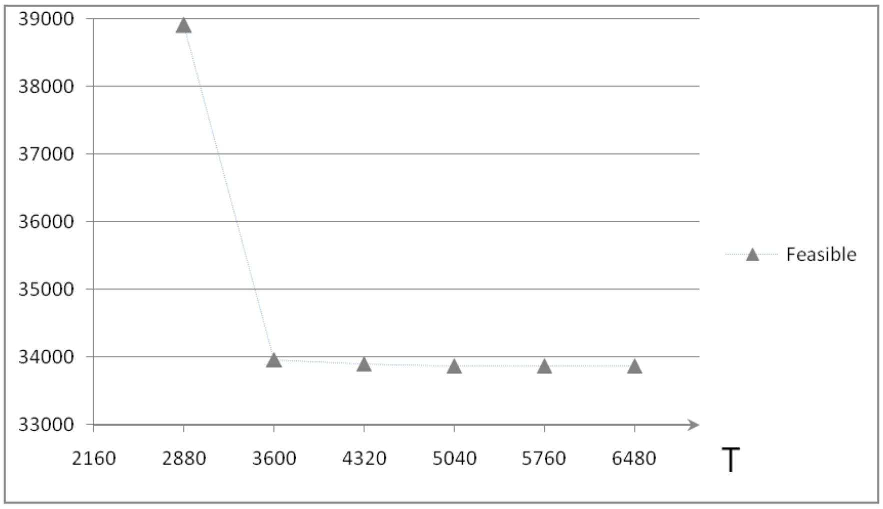

Figure 4: Pareto frontier: additional solutions costs against recovery period length T

First of all, we notice that for $T \leq 2160$, we do not get a feasible solution, which is evident, since one plane is grounded until time 2160 and one final state remains uncovered. Since the recovery costs after T are neglected by the algorithm, it is still

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

Recovery Period T	4320 min	5040 min	5760 min	6480 min
# canceled flts	5	5	5	5
# delayed flts	9	8	8	8
# uncovered final states	0	0	0	0
total delay [min]	630	627	627	627
max delay [min]	153	153	153	153
Additional costs	33900	33870	33870	33870
optimality gap [%]	0	0	0	0
tree size	9	3	1	1
run time [s]	25.0	81.8	73.6	183.6

Table 7: Results for the same instance with different recovery periods T. Solutions having cost followed by (#) are unfeasible.

useful to run it for T equal to 720, 1440, and 2160 minutes because we get an estimation of the recovery cost, which is more precise when T reaches the instant where a feasible recovery can be computed.

Once we find a feasible solution for a given T, we see that its increase leads quickly to a *stable* solution, i.e. we generate the same recovery plan even considering longer recovery periods. Thus, no additional recovery costs are incurred. We must mention here that due to computational complexity we have to restrict the delay bound to 800 minutes, ignoring therefore potential recovery solutions for longer recovery periods T with longer delays.

Finally, Table 7 and Figure 4 show the conflict between the two objectives when minimizing both T and the recovery costs.

The results here show that the choice to fix a recovery period and optimize the costs is reasonable. Thanks to the several parameters, namely delay bounds and time and resources discretization, we can quickly identify a reasonable value for T and then optimize the recovery costs around that value with less restrictive parameters, thus with more precision.

The numerical results we provide here give an intuition of the efficiency of the column generation scheme to solve the ARP problem. We also show how the recovery plans depend on the initial schedule and disruptions. We show that there is a clear benefit in planning the maintenances and the recovery decisions simultaneously. Moreover, this approach is saving the validation time as technical feasibility is ensured. Therefore interactions between the OCC and the technical department are no longer needed. Finally, we show how the algorithm can be exploited to optimize costs and recovery periods in a multi-objective optimization way. The valuable result of this approach is the possibility to evaluate alternative recovery plans of different cost and due date.

6 Conclusions and Future work

We present an airline schedule recovery algorithm based on column generation. The proposed algorithm arises from a collaboration, sponsored by the swiss government within the fund for technology transfer (CTI - Projet 8007.2 ESPP-ES), between EPFL and APM Technologies. We consider the aircraft recovery problem and we propose an algorithm where aircraft technical constraints (maintenances) are fulfilled and their placement within the aircraft schedule optimized. We detail a column generation scheme based on a commodity network flow model, where each plane has an associated recovery network, a dynamic programming algorithm to build the underlying recovery networks and a dynamic programming algorithm to solve the pricing problem.

The main contributions of this work are that we include maintenance planning, unconstrained delay management and plane swapping decisions within the aircraft recovery problem and that we introduce a multi-objective optimization algorithm based on column generation. This allows us to solve real instances for a medium sized airline efficiently by improving the existing network model of Argüello et al. (1997) thanks to the resource management we use to model maintenance constraints. Finally, we include some modern implementation issues to fasten up the computations.

Since this is an ongoing project, several issues should be refined and extended. In particular:

- the proposed algorithm must be validated against a wider set of instances, even though real-world cases are more difficult to obtain and to analyze, in particular when the set of disruptions must be collected during the day of operations.

- although most of the instances where solved to optimality at the root node, a branching scheme, thus a full Branch&Price algorithm, is needed to obtain a proved optimal solution. We intend to implement the branching decision described in Section 4.5.
- from a modeling point of view, the proposed algorithm does not consider all the possibilities a human planner does. We intend to add to the network generation algorithm the possibility to include positioning flights.

Acknowledgment

The authors would like to acknowledge the Swiss Center for Technology Transfer (CTI) [2] for supporting this project and Alberto De Min and Viet Dang, from APM Technologies [3], for their support.

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.

Argüello, M., Bard, J. and Yu, G. (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.

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

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

Cook, A., Tanner, G. and Anderson, S. (2004). Evaluating the true cost to airlines of one minute of airborne or ground delay, EUROCONTROL.
URL: *http://www.eurocontrol.int*

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

[2]www.bbt.admin.ch/kti/
[3]www.apmtechnologies.com

European airline punctuality report (2006). Association of European Airlines. **URL:** *www.aea.be*

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.

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. (to appear). New dynamic programming algorithms for the resource constrained shortest path problem, *Networks* .

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).

Ryan, D. and Foster, B. (1981). An integer programming approach to scheduling, *in* W. A. (ed.), *Computer Scheduling of Public Transportation Urban Passenger and Crew Scheduling*, North-Holland, pp. 269–280.

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

Scheidereit, H. C. (2006). The costs of delays & cancellations, m2p consulting, AGIFORS Operations Conference.

Shavell, Z. A. (2001). The effects of schedule disruptions on the economics of airline operations, *Progress in Astronautics and Aeronautics*, Vol. 193, American Institute of Aeronautics and Astronautics, Inc.

Sojkovic, G. (1998). *Gestion des Avions et des Equipages durant le Jour d'Opération*, PhD thesis, Université de Montréal.

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.

Vanderbeck, F. (2005). *Implementing Mixed Integer 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 Otpimization* **1**: 305–321.

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 pertubations, *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.

A Recovery Network Generation

For notational simplicity, we denote a final state $[a_T, t_T, \mathbf{H}_T]$ by s.

Algorithm 1 shows the dynamical structure of the generation algorithm, thus the networks' exponential behavior with respect to the size of the flight sets F_p. In the algorithm N represents the set of time-location nodes ordered by increasing time. The way nodes and arcs are created is described in the Table 8.

The notational detail are given in Section 3.1 and the parameters are introduced in Section 4.5.

Tables 9 and 10 give an overview of the different constraints that must be satisfied in each function (parametrized constraints are labeled by (P)).

`CreateFlight`$(\{a, t\}, f)$	Given depart node $\{a, t\}$, computes the destination node $\{a', t'\}$ and the flight arc $(\{a, t\}; \{a', t'\})$, where a' is destination airport, and t' is the earliest departure time at airport a'. 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 $t' = edt_f + d_f + mtt_{a'}$.
`CreateMaintenance`$(\{a, t\}, f)$	Similar to `CreateFlight`$(\{a, t\}, f)$, it computes the maintenance time and the cost of the maintenance arc.
`CreateTermination`$(\{a, t\}, s)$	Given depart node $\{a, t\}$, it creates the termination arc $(\{a, t\}; s)$.
`CreateMaintTermination`$(\{a, t\}, s)$	Given depart node $\{a, t\}$, it creates the maintenance termination arc $(\{a, t\}; s)$. By convention, the first available maintenance slot is used.

Table 8: Functions used is Algorithm 1

Algorithm 1 Recovery Network Generation for plane $p \in P$

```
Require: Set $P$ of planes, set $F_p$ of coverable flights, initial states $[a_0, t_0, \mathbf{H}_0]_p$ and set $S_p$ of final states
 1: for $p \in P$ do
 2:   INITIALIZATION: Create source node $\{a_0, t_0\}$, set $N = \{\{a_0, t_0\}\}$
 3:   while $N \neq \emptyset$ do
 4:     Select the first node $\{a, t\} \in N$
 5:     for $f \in F_p$ where $a$ is the departure of $f$ do
 6:       if FeasibleForFlightArc($\{a, t\}, f$) then
 7:         $\{a', t'\}$ = CreateFlight($\{a, t\}, f$)
 8:         set $N \leftarrow N \cup \{a', t'\}$
 9:       end if
10:       if FeasibleForMaintArc($\{a, t\}, f$) then
11:         $\{a', t'\}$ = CreateMaintenance($\{a, t\}, f$)
12:         set $N \leftarrow N \cup \{a', t'\}$
13:       end if
14:     end for
15:     for $s \in S_p$ where $a$ is the airport of $s$ do
16:       if FeasibleForTermArc($\{a, t\}, s$) then
17:         CreateTermination($\{a, t\}, s$)
18:       end if
19:       if FeasibleForMaintTermArc($\{a, t\}, s$) then
20:         CreateMaintTermination($\{a, t\}, s$)
21:       end if
22:     end for
23:     Set $(N \leftarrow N \setminus \{a, t\})$
24:     Sort $N$ by increasing time
25:   end while
26: end for
```

`FeasibleForFlightArc`$(\{a, t\}, f)$	The flight arc can only be created if flight is actually departing from airport a and if feasible departure and landing times are available at airports a and a'. The following constraints are checked: • $\exists\ edt \geq \max\{sdt_f, t\}$ such that 1. $\exists\ o_a \in O_a$ such that $edt \in o_a$ 2. $\exists\ o_{a'} \in O_{a'}$, such that $edt + d_f \in o_{a'}$ • $delay \leq \tau$ (P) • $edt_f - t \leq \psi$ (P) where τ and ψ are representing the maximal delay bound and the maximal waiting bound respectively.
`FeasibleForMaintArc`$(\{a, t\}, f)$	The maintenance arc can only be created if there is a maintenance slot available at airport a. 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 and landing in a'. The following constraints are checked: • $\exists\ t^M \geq t, m_a \in M_a$ such that $t^M \in m_a$ • $\exists\ edt \geq \max\{sdt_f, t^M + d_a^M\}$ such that 1. $\exists\ o_a \in O_a$ such that $edt \in o_a$ 2. $\exists\ o_{a'} \in O_{a'}$, such that $edt + d_f \in o_{a'}$ • $delay \leq \tau$ (P) • $edt_f - t - d_m \leq \psi$(P) • $h^i \geq \rho u^i$, $\forall i = 1 \cdots \mid \mathbf{H} \mid$ (P) where τ and ψ are the same as in `FeasibleForFlightArc`$(\{a, t\}, f)$ and ρ is the parameter of minimal resource consumption ratio before considering maintenance.

Table 9: Feasibility functions for flight and maintenance arcs used is Algorithm 1

`FeasibleForTermArc`$(\{a, t\}, s)$	A termination arc can be created between $\{a, t\}$ and the sink node s if the airports are matching and if the expected time t_T is not yet reached. A parameter Γ is used to bound the grounding time needed to reach the sink from $\{a, t\}$. • $t^M + d_a^M \leq t_T$ • $t_T - t \leq \Gamma$ (P) where Γ is the grounding time bound
`FeasibleForMaintTermArc`$(\{a, t\}, s)$	Similarly a maintenance termination arc can be created between $\{a, t\}$ and the sink node s if there is a maintenance slot available. • $\exists\ t^M \geq t,\ m_a \in M_a$ such that $t \in m_a$ • $t \leq t_T$ • $t_T - t \leq \Gamma$ (P) • $h^i \geq \rho u^i$, $\forall i = 1 \cdots \mid \mathbf{H} \mid$ (P) where Γ is the grounding time bound and ρ the resource consumption proportion.

Table 10: Feasibility functions for termination and maintenance termination arcs used is Algorithm 1