Submitted to *INFORMS Journal on Optimization*
manuscript MS-0001-1922.65

A Constructive Heuristic for the Aircraft Recovery Problem

Francisco de Lemos
School of Electronic Engineering and Computer Science, Queen Mary University of London, 10 Godward Square, Mile End Rd, Mile End, London E1 4FZ, f.j.r.lemos@qmul.ac.uk

John Woodward
School of Electronic Engineering and Computer Science, Queen Mary University of London, 10 Godward Square, Mile End Rd, Mile End, London E1 4FZ, j.woodward@qmul.ac.uk

In this paper we present a Constructive Heuristic for the Aircraft Recovery Problem (CHARP) using constraint satisfaction programming. Aircraft rotations in commercial aviation are often subject to disruption making them infeasible. The recovery procedure that we propose consists of an upper and lower heuristic that execute a pincer movement over the upper and lower bounds of the search space. Both heuristics use Constraint Satisfaction Programming, however the upper bound heuristic iteratively recovers the rotation to near optimality whereas the lower bound recovers recovers it to optimality. We Validate the CHARP using the ROADEF 2009 Challenge rotations. We extend our validation by re-schedule the latter using the Block Time and Fuel model.

Key words: Constraint satisfaction, aircraft, disruption, recovery

1. Introduction

The objective for airline companies consist in maximize their profits while operating flights according to a schedule. Therefore, airlines generally create tight schedules in order to increase their profitability. These tight schedules, have a minimum slack between flights legs, because they rely on the assumption that the flight legs will be operated as planned. A rotation consists in assigning an aircraft to a flight schedule. This process is in many occasions subject to irregularities that can complicate its execution.

A significant number of factors, such as mechanical failures, crew delays, problems with ground operations, industrial action, inclement weather, congestion at airports can cause disruptions such as flight delays or cancellations. As a result the initial rotation becomes inefficient in the sense that flights have to be delayed or canceled. Since airline companies are liable to pay compensation for the resulting passenger inconvenience in these situations, leading ultimately to a negative financial impact on an airline company's profits, it thus becomes of the most importance to resume normal operations as quickly as possible. In conclusion, in case of disruptions, a feasible rotation must be determined for a period of time. The period of time is designated by the recovery time window (RTW). During the RTW a feasible rotation must be found to minimize the costs that result form flight delay or cancellation. The latter consists of a process called disruption recovery.

Disruption recovery is a real-time practice that requires finding a fast solution when irregularities occur. Often, disruptions take place during operations (*i. e.*, only in few particular cases, it is possible to know a disruption in-advance), and a provisional plan has to be provided quickly. Thus, it is necessary to construct efficient algorithms to have good solution in minutes. It is also very important to refer that when disruptions occur they can affect more than just the directly disrupted flight, and can send a shock-wave of disruptions through the network. To resume normal operations it is necessary to implement certain actions during the RTW. It is during its duration that actions should be implemented to re-plan usage of resources all over the entire network of flights. The objective that is intend to meet is to find a solution with minimal flight cancellation and delay, while meeting the constraints associated with flight time, aircraft or airport capacity and passenger itineraries. In this paper we will focus solely in recovering the aircraft rotation, which is a specific part of the broader problem known as the aircraft recovery problem (ARP). To achieve this objective we will use the data sets for the ROADEF 2009 Challenge.

Every two year the *French society of Operational Research and Decision Making* releases the ROADEF Challenge, which consists on a competition to solve a complex optimization problem that occurs in industry. In the ROADEF 2009 Challenge there is a step-wise simplification of a model, for disruption management in commercial aviation that aims at finding recovery planning of flights, aircraft assignments and passengers (including flight leg cancellation) on a given maximal horizon, so that a sum of penalties corresponding to various costs or discomforts is minimized. The present work is based on ROADEF 2009 Challengee and intends to develop a series of methodologies that will result in the generation of recovery processes for the disruptions induced by flight delays or cancellations, aircraft availability shortage or airport capacity decrease.

The remainder of the paper is organized as follows: an overview of the relevant literature is provided in Section 2; the model for the aircraft rotation recovery problem (ARRP) is provided in Section 3; the heuristic for the ARRP will be described in detail in Section 4; the results obtained using the solution methods; the final conclusions.

2. Literature Review

Various methodologies have been developed for the aircraft recovery problem in the literature. A detailed overview of the relevant studies is presented in this section.

2.1. Aircraft Recovery

(Teodorović and Guberinić 1984) developed a network model that minimizes the total passenger delay on an airline network and solved the problem of finding new routing and scheduling plan to optimality using a branch and bound heuristic. The authors model passenger delays explicitly but they assume that all passenger itineraries contain only a single flight leg. Given the large number of possible routings for even modest sized problems, it is doubtful that their approach could be materialized for practical problems. On the other hand due to the fact that companies manage their operations using hub and spoke flights, this study would also have limited scope of application.

(Jarrah et al. 1993) present an overview of a decision support system for the aircraft recovery problem with the attempt of conceptualizing the problem. The authors used a minimum cost network models, one delay model and one cancellation model and implemented an algorithm that solves the shortest path problem repeatedly in order to determine the necessary flows. One of the major drawbacks of this work consists of not having solutions that combine both flight delay and cancellation. (Yan and Lin 1997) developed a framework to help carriers handle schedule perturbation, due to temporary closure of airports, and resume their normal services as possible. This framework is based in a time-space network flow model, where the arcs in the network represent airborne flights, grounded flights and overnight connections. The research was conducted over a set of scenarios, mathematically formulated either as pure network flow problems or network flow problems with side constraints, the former solved using simplex method and the latter using a Lagrangian relaxation-base algorithm, respectively. These models minimize the schedule-perturbed time after incidents so that carriers can resume their normal services as soon as possible in order to maintain their levels of service. The salient features of the models consist in combining systematically flight cancellations, flight delays, the modification of multi-stop flights, the ferrying of idle aircraft and the swapping of aircraft. This procedure aims at adjusting effectively a schedule following incidents, so that a carrier can maintain its profitability. The authors confine scope of this research to the operations of a single fleet as well as simplifying multi-leg flights by considering the that time block is from the beginning of the first leg to the end of the last leg.

(Cao and Kanafani 1997a) and (Cao and Kanafani 1997b) extended the work developed by Jarrah et al. (1993) evaluating solutions with flight cancellations and delays. The authors used a linear programming approximation algorithm based on a quadratic programming model that included both flight delays and cancellations. One of the major limitation of this work relates to the approach being limited to aircraft assigned to routes containing at most two flight legs.

The research led by (Arguello et al. 1997) used a greedy randomized adaptive search procedure (GRASP) meta-heuristic to minimize flight cancellation and delay costs associated with a recovery aircraft routing, as a response to groundings and delays. The GRASP described in the study is adapted for use as a randomized neighborhood search technique. In the procedure, the neighbors of an addressed solution are evaluated, and the most desirable are placed on a restructured candidate list. Neighbor generation operations are performed on pairs of aircraft routes. The authors introduced a series of constraints to enforce the model to share operational practices namely that, every flight in each aircraft route must depart from the airport where its previous flight arrived (balance constraint), a minimum turn-round time must be enforced between each flight arrival and subsequent departure, the recovery period extends to the end of the current day, aircraft must be positioned at the end of the recovery period in a specific airport so that the flight schedule can be resumed next day, airport departures are restricted in the curfew period and aircraft with planned maintenance will not have their original route altered. The approach was tested on data supplied by *Continental Airlines* and the results proved that the GRASP can produce in most cases optimal or near-optimal solutions.

(Thengvall et al. 2000) 2000) solves the aircraft recovery problem for a single fleet over the the entire flight schedule with an objective function that accounts for flight revenues, delays, cancellations and also an incentive to minimize deviations from the original schedule. The objective function consists in minimizing total operating costs, but crew and passenger disruptions are not considered. The problem is modeled using a time-space network and solved as an integer linear program using an optimization software. To represent the delays for a particular flight leg, a series of delay arcs are introduced to consider the available options for later flights. To ensure that only a single flight is chosen, a side constraint must be added requiring that the sum for all arcs representing the same flight is less or equal to one. The model was able to accommodate recovery periods of arbitrary length that begin and end at arbitrary periods of the day. In addition, a rounding heuristic is introduced to obtain feasible integer solutions from the linear programming relaxation of the mixed-integer programming formulation. As mentioned by the authors one weakness of the model is that it does not track individual passengers and thus does not consider passenger connections. As for aircraft maintenance requisites the authors chose to remove aircraft due for maintenance from delays or cancellations to assure they reach their proper destination. This work was extended in (Thengvall et al. 2001) by solving the multi-fleet aircraft schedule recovery, using multi-commodity network-type models during a hub closure.

(Løve et al. 2001) defined the dedicated aircraft recovery problem (DARP) as *given an original flight schedule and one or more disruptions, the DARP consists of changing the flight assignments* *of the aircrafts in order to produce a feasible and more preferable revised flight schedule*". The authors compared the results obtained using various versions of a heuristic, namely the iterated local search (ILS) with variable neighborhood search (VNS) incorporated and a simple steepest ascent local search (SALS) along with a repeated steepest ascent local search (RSALS), which performs entirely like SALS but is repeated for different initial conditions. According to the authors ILS algorithms were able to find solutions within the first 10 second, that prove to be robust, after 24-hour test runs without significantly better solutions were not found. As for the SALS the authors mention that the algorithm finds quickly a local optimum and that once reached it is not easy to escape from. The latter proved not being a drawback when the local optimum is close to the value of the global optimum which was confirmed as being the case. Since the results achieved were very auspicious *British Airways* (BA), provided some of their flight data for realistic testing of the RSALS heuristic. After a few simplifications 10 BA flight schedules were extracted, that had an average of 80 active aircraft, 44 airports and 340 flights. Each flight schedule was disrupted and afterwards the RSALS heuristic was used to improve the flight schedules by delaying flights, swapping aircraft and canceling flights. The model proved to deliver reasonable good recovered flight schedules, by solving a SALS heuristic using the subjacent network representation of the problem. The authors conclude that further advancements should be performed to include crew scheduling and passenger itineraries. As for the initial claim made by the authors that the time for a tool to find solutions for such problem, should be less that three minutes, (Andersson 2006) refers that this benchmark varies depending on the envelopment of the problem, and also on quality of the solutions delivered. In (Andersson 2006) tabu search (TS) and simulated annealing SA methods were used to solve the aircraft schedule recovery problem. After evaluating the quality of the solutions and the robustness of the method TS proved to be the preferable solution strategy.

2.2. Integrated Recovery

When scheduling flights, passenger travel demand is clearly a key consideration for commercial airlines. The efficient flow of passengers is critical, especially when regular operations are affected by disruptions, however in practice, passenger recovery is observed as the last stage of the sequential recovery approach. Due to the high dependency of passenger schedules on aircraft and crew schedules, passenger recovery is either integrated with aircraft recovery or with both aircraft and crew recovery.

In (Bratu and Barnhart 2006) the authors focus on passenger recovery while incorporating rules and regulations on aircraft and crew. They propose models for integrated recovery, using a flight schedule network representing flight legs with flight arcs. Several copies of the flight arcs are made to account for each possible departure time decision within the window of feasible departures for a specific flight. Making copies of arcs to represent flight leg departure decisions has been widely used in previous schedule recovery problems (Yan and Lin 1997), (Thengvall et al. 2000) and (Thengvall et al. 2001). The objective either minimizes the sum of operating costs and disrupted passenger costs, or the sum of operating costs and total passenger delay costs. To test the models the authors developed an *Airline Control Center Simulator*, to simulate domestic operations of a major *United States* airline. The airline's data consisted of 83,869 passengers on 9,925 different passengers' itineraries per day, operated by 302 aircrafts divided into 4 fleets, 74 airports and 3 hubs. For all scenarios solutions are generated that resulted in reductions in passenger delay and disruptions.

The ROADEF 2009 Challenge introduced a competition that intended to obtain a recovery plan that was able to integrate schedule, aircraft, and passengers. In the context of the ROADEF 2009 Challenge, different algorithms, have been proposed.

(Bisaillon et al. 2011) proposed a very efficient large neighborhood search model consisting in three phases, construction, repair and improvement, respectively. The construction phase consists of constructive heuristic that tries to create feasible solutions, by removing flight sequences until all constraints, with the possible exception of airport capacity constraints, are satisfied. The repair phase makes use of a heuristic that proceeds in three steps. First, by delaying flights to respect all airport capacity constraints. Second, by re-inserting flights that were removed during the construction phase, between two successive flights whose time interval is long enough to accommodate them. The authors argument that this approach yields expressive cost improvements in few iterations. Third, by accommodating passengers whose itineraries have been canceled by repeatedly solving shortest path problems for a network of flights. The source node represents the origin airport of the itinerary at the departure time and the sink node represents the destination airport of the itinerary at the arrival time. This process attempts to assign to the path the largest number of passengers satisfying the aircraft seating capacity and is iterated as long as new passenger can be accommodated. Although the solutions obtained are feasible, they possibly are sub-optimal since many itineraries may have been cancelled. The improvement phase consists of a procedure that attempts to extend the repair phase by delaying some flights in the hope of accommodating additional passengers. Just like in the repair phase passengers are re-assigned by repeatedly solving shortest path problems. The global process is iterated introducing diversification in the construction phase, by randomly sorting the aircraft so as to treat them in a different order each time the construction phase is performed. Finally, whenever improved cost is found, the corresponding solution replaces the current one. On the overall, the algorithm executes a very large number of simple and fast actions. By doing so, not only it finds quickly feasible solutions, but also does not rely on any knowledge of the current flight network.

(Mansi et al. 2012) presented a model that combines a math-heuristic with an oscillation heuristic to improve the solutions obtained. The math-heuristic consists of mixed integer programs that intends to maximize the number of aircraft that satisfy the maintenance constraint and the number of passengers that arrive at their final destination while minimizing the total delay. However this procedure, may not render a feasible solution, meaning that at least one aircraft will not be able to reach the airport assigned for maintenance on time without canceling flights. To make the solution feasible, a repair heuristic is applied to try to satisfy hard maintenance constraints, and also to maximize the number of passengers arriving at destination. After generating a feasible solution the algorithm makes improvements alternating between constructive and destructive phases, by adding or removing some parts of the aircraft routes.

(Acuña-Agost et al. 2009) addresses the *Disruption Management for Commercial Aviation* problem using a mixed integer programming formulation that integrates two multi-commodity flow problems. The first one regards to aircraft flowing through airports, and the second one to passengers flowing through flights. Due to its complexity the authors choose to solve the problem using statistical analysis of propagation of incidents (SAPI). This method assumes that every schedule event could be affected (or not) by some disruption with a certain probability, hence the goal of using (SAPI) is to determine the probability that a flight has of being cancelled in the optimal solution. Afterwards it is possible to divide the flights in two sets: flights expected to be cancelled, and the remaining flights. The idea is to study the probability that disruptions affect future flights, and use this information to intensify search in the affected areas while the rest is kept unperturbed. The second one consists of a post-optimization procedure designed to improve the solutions obtained, by assigning cancelled passenger to flights that can accommodate them.

(Jozefowiez et al. 2013) presented a three-phase heuristic, which the authors designated by new connections and flights. During the first phase, after the disruptions are integrated in the system, the heuristic algorithm to searches for feasible solutions, by means of canceling or truncating itineraries to respect rotation connectivity or airport capacities. In the second phase the goal consists in reassigning to the existing set of rotations as many passenger groups as possible. The search for an itinerary for a given group of passengers is modeled as a shortest path problem. This procedure may cause some groups to be split in order to respect the aircraft seat capacities. In the third phase the algorithm tries to insert new sub-rotations to existing aircraft rotations to allow the transportation of passengers. Passenger re-accommodation is again based on shortest path solving on a time-space graph for each group of passengers. Although the results of this method, for smaller instances (data sets A and B respectively) look promising, the results proved intractable for larger instances (data sets X), because of the large number of variables and memory amount that was necessary.

In (Eggermont et al. 2009) the authors decompose the problem into smaller and more tractable ones. The sub-problems are chained in the sense that each one attempts to improve a certain aspect in the schedule, and its current output consists of the next sub-problem's input. The first considered sub-problem attempts to fix aircraft rotation by canceling for an unavailable aircraft all remaining flights originally scheduled within its unavailability period. The second considered sub-problem takes as input the solution of the first sub-problem and aims at respecting airport capacities. However while respecting airport capacities, maintenance might not be carried out. In order to fix this the pre-maintenance rotations are scheduled first. The third solved sub-problem lies in delay fine tuning. Additional delays may allow for more passengers to be re-accommodated. Finally in the fourth problem, since many itineraries become infeasible, passengers are re-routed using the cheapest feasible alternative routes where there is available capacity in the operated flights.

In (Darlay et al. 2009) the authors proposed a heuristic that alternates between an aircraft rotations construction phase and a passenger reassignment phase. For the first phase, aircraft allocation is solved via a minimum cost multi-flow in a time-space network with additional constraints modelling airport capacities and aircraft shortages or maintenance. In the second phase, a set of possible paths is created for each of the passenger's itineraries. Every path has an associated cost that accounts for delaying and cabin class downgrading. This phase is modelled using a capacitated multi-flow with a path formulation.

In (Peekstok and Kuipers 2009) the authors propose a simulated annealing method with a modified objective function to account for constraint violation. After the introduction of disruptions in the data sets corresponding to the original schedule, the authors choose to accept infeasible solutions by allowing constraint violation for airport hourly capacities for arrivals and departures, airport connectivity, aircraft minimum transit and turn-round time, aircraft maintenance periods, aircraft seating capacity, minimum passenger connection time and passenger schedule. The authors showed a preference for this procedure since in their opinion, restricting the local search to feasible solutions would not render good results. During testing of the data instances the authors found that it is harder to fix airport and aircraft constraints than passenger constraints. Therefore after achieving airport and aircraft feasibility, the algorithm does not allow to violate them again. The algorithm then iterates between attempting to lower the objective function and attempting to reach zero passenger constraint violation, by using a local search that changes flight time, operating aircraft of an existing flight or the flight of a passenger's itinerary.

(Dickson et al. 2009) proposes a two phase process. The first phase uses a mixed integer linear integer program connection network that aims to re-route aircraft, re-time and/or cancel flights so as to minimize the disruption experienced by passengers in their itineraries. The second phase uses a network multi-commodity network flow model to re-optimize passenger itineraries base on the flight schedule determined in phase in the first phase. The goal is to maximize the value of the itineraries flowing through the flight network, within the given flight capacity and passenger demand.

(Eggenberg and Salani 2009) address the problem proposing a modeling framework that allows the consideration of operational constraints within column generation. The authors approach the problem modeling the aircraft recovery problem (ARP) with the passenger recovery problem (PRP). For the ARP each recovery scheme has a cost, which can be uniquely determined by operational and delay costs, and is associated with a binary variable that is equal to one if it is considered in the solution, or zero otherwise. A recovery scheme is described using binary coefficients to depict if a route covers a certain flight, a route reaches a certain final state and if a route is assigned to certain aircraft. 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.

2.3. Flight Speed Control

Although in the airline industry there is a realization that the choice of cruise speed has a critical impact on the trade-off between reducing delays versus reducing fuel cost, the airlines do not take full advantage of this alternative. In the work of (Aktürk et al. 2014) the authors formulated the problem to recover a flight schedule, subject to a single (or multiple) disruption(s), while minimizing the sum of tardiness cost for all flight, the cost which is incurred if two aircraft are swapped and if they end up at different airports than originally planned in the initial schedule, the additional fuel and carbon emission cost due to increased cruise speed in the new schedule. To repair the disrupted airline schedule the authors use three approaches, delay propagation, cruise speed control and the third approach is swap and cruise speed control. The problem is solve using conic mixed integer programming and the authors claimed at the time that this was the first implementation of a conic quadratic optimization approach to solve an aircraft recovery problem in an optimal manner.

The work of (Arikan et al. 2016) uses a mathematical formulation for the integrated aircraft and passenger recovery problem that considers aircraft and passenger related costs simultaneously. The method uses a realistic fuel cost function, which was developed respecting the technical report of Airbus 2004. The authors mention that airlines tend to operate their flights at maximum range cruise (MRC) speeds to save fuel and that the relation of fuel cost with deviation from MRC speed is increasing and convex. The authors formulated the problem using conic quadratic mixed integer programming model and claim that the computational experiments can handle several simultaneous disruptions optimally on a four-hub network of a major U.S. airline within less than a minute on the average. They conclude that proposed approach is able to find optimal trade-off between cruise time controllability and passenger-related costs in real time.

In more recent work (Marla et al. 2017) the authors introduce flight planning to explore the trade-off between delays and fuel burn. Flight planning consists of process to specify the route of a flight, its speed and fuel burn. This work makes use of the cost index (CI) for a flight to capture both the flight time and fuel burn. The CI is the ratio of the time-related cost of an aircraft operation and the cost of fuel. The value of the CI reflects the relative effects of fuel cost on overall trip cost as compared to time-related direct operating costs.The authors evaluated, for each possible flight speed , the fuel cost and the passenger-related delay costs to the airline, using JetPlan, a flight planning tool developed by Jeppesen Commercial and Military Aviation and the impacts on passengers using an airline disruption management simulator. At CI 500, there is a sharp reduction in the passenger cost function as several passengers can make their planned connections, while at lower CI values these passengers misconnected. However the authors mention that the specific airline from which the example was extracted, typically operates at CI 30 and allows its dispatchers and pilots to speed up to a maximum of CI 300. This standard of operation would not have any significant impact in minimizing the total costs, however it leads to the conclusion that it is possible to optimize total costs by increasing aircraft speeds relative to those initially planned.

3. Problem Statement

The problem is build up upon a time series of data files that provide information for a specific period where flights are scheduled. By this it is meant the planning horizon. During this planning period several disruptions can occur. The strategy to overcome them, can only be implemented in a time window designated by recovery time window (RTW).

3.1. Sets

Table 1 ARRP sets

A	Set of all airports, indexed by a
P	Set of all aircraft, indexed by p
$\mathcal{D}$	Set of flight disruptions, indexed by d
$\mathcal{B}$	Set of aircraft disruption, indexed by b
$\mathcal{R}$	Set of airport disruptions, indexed by r

Table 2 ARP parameters

RTW_s	Start of the recovery time window
RTW_e	End of the recovery time window
c_a^{lh}	Maximum number of arrivals l during time window h at airport a
c_a^{th}	Maximum number of departures l during time window h at airport a
$d_{a_1a_2}$	Distance measured in flight minutes between airports a_1 and a_2
σ_p	Rotation of aircraft p
$\sigma_p^o(i)$	i^th flight leg's origin airport o for the rotation of aircraft p
$\sigma_p^f(i)$	i^th flight leg's destination airport f for the rotation of aircraft p
$\sigma_p^d(i)$	i^th flight leg's origin departure time d for the rotation of aircraft p
$\sigma_p^a(i)$	i^th flight leg's destination arrival time a for the rotation of aircraft p
O_p	Origin airport of aircraft p
t_{rp}	Turn-round time for aircraft p

3.2. Parameters

The start (respectively, the end) of the recovery time window is denoted RTW_s (respectively, RTW_e). The recovery time window is divided into smaller time windows equal to one hour, denoted $h \subseteq RTW$.

The information regarding the starting and ending of the RTW is set in the first line of the *config.csv* file *e.g.*:

```
01/03/08 16:00 03/03/08 04:00
```

3.3. Airports

The key infrastructure in commercial aviation consists of airports. In this problem, they have also an essential role since their capacity consists of a hard constraint that bounds the maximum number of arrivals and departures per hour. The airports form a set A. For each $a \in A$ and for each window $h \subseteq RTW$, the value c_a^{lh} is the maximum number of arrivals that can occur during the time window h at airport a and c_a^{th} the maximum number of departures. It is necessary to discretize these hourly capacities to incorporate them within the model's formulation. For example, the recovery time window between $[12{:}00, 16:00[$ comprises four unit time windows, and as an illustration, if the departure capacity of an airport between $[14:00, 15:00[$ is equal to 3, this indicates there can be at most three flights departing form this airport between 14:00 and 14:59. The data file *airports.csv* provides the departure and arrival airport capacities, which correspond to a maximum number of operations allowed per one-hour interval, for a typical day. These thresholds depend upon the time of day (peak time, normal time, night time, possible curfew). Each line of the file contains the three-letter code of the airport (type "airport") followed by a series of quadruples specifying the capacities associated with each time period. Capacities are nonnegative integers and the time periods are characterized by two entries of type "time" corresponding to the start time and end time of the period *e.g.*:

```
NCE 0 0 00:00 03:00 5 0 03:00 07:00 20 20 07:00 19:00 5 10 19:00 21:00 0 0 21:00 00:00
```

The above example describes a typical day (in GMT) for Nice airport:

- neither departures nor arrivals between 21:00 and 3:00
- maximum five departures per one-hour interval [H, H + 1[between 3:00 and 7:00 (and no arrivals)
- maximum 20 departures and 20 arrivals per one-hour interval [H, H + 1[between 7:00 and 19:00
- maximum five departures and 10 arrivals per one-hour interval [H, H + 1[between 19:00 and 21:00

For simplicity, some real constraints are not taken into account in the problem (*e.g.* the maximum number of aircraft in the airport surface).

3.4. Distance

For each airport pair $a_1, a_2 \in A$, $d_{a_1 a_2}$ is the distance measured in flight minutes between a_1 and a_2. The *dist.csv* provides the typical flight times between each airport pair, as well as the flight type. Note that, for a given airport pair, the flight times may depend on the direction of the flight. Each line of the file contains the three-letter codes of the origin and destination airports, the flight time and the flight type *e.g.*:

```
CDG NCE 95 D
NCE CDG 95 D
```

The above example provides the flight times between Nice airport and Paris-Charles de Gaulle (95 minutes in both directions) and specifies that the flight is domestic.

3.5. Aircrafts

Let P denote the set of aircraft, where each aircraft $p \in P$ has the following set of operational characteristics:

- aircraft type: it defines the family, model and configuration; subsets of aircraft with common characteristics are grouped within families (*e. g., A318, A319, A320,* and *A321* in the *AirbusSmall* family). Operational characteristics are common to all aircraft of a given model: turn-round time, transit time, range, and set of possible configurations. The configuration provides the number of seats for each cabin class, economic (E), business (B) and first (F).

• transit time: corresponds to the minimum time between the arrival and departure of multi-leg flight operated by the same aircraft. The advantage of this configuration is that it allows for a reduction in the time necessary to prepare the aircraft for the second leg.

• turn-round time: defines the minimum idle time between two different consecutive flights operated by the same aircraft;

• maintenance: an aircraft needs to undergo scheduled maintenance on a specified airport during a period of time, in which it is unavailable for flight duty, and a maximum allowable flying range before the maintenance is due.

The *aircraft.csv* file provides the aircraft characteristics *e.g.*:

```
A320#1 A320 AirbusSmall 0/20/150 480 1500.0 30 30 CDG CDG{10/01/08{14:00{10/01/08{20:00{900   A320#2
A320 AirbusSmall 10/30/110 480 1500.0 30 30 NCE NULL
TranspCom#1 TranspCom TranspCom -1/-1/-1 60 0.0 5 5 CDG NULL
TranspCom#2 TranspCom TranspCom -1/-1/-1 60 0.0 10 10 ORY NULL
```

The above example provides the characteristics of the first two Airbus A320s in the fleet. Note that they have several common characteristics, but their configurations are different. The first one is located in Paris-Charles de Gaulle at the beginning of the recovery period, and must undergo maintenance there on 10/01/08 between 14:00 and 20:00 (it cannot fly more than 15 hours between two consecutive maintenance actions). The second one is located in Nice and has no planned maintenance until the end of the recovery period. The remining two consist of surface transportation vehicles 1 and 2. Both of them belong to the family TranspCom, have infinite seating capacities, and operating costs of zero. Vehicle 1 is located in in Paris-Charles de Gaulle at the beginning of the period, whereas vehicle 2 is in Paris-Orly.

3.6. Flights and rotation

An aircraft performs a rotation which is a sequence σ_p of flight legs starting from an origin airport O_p. The i^{th} flight leg in the rotation $\sigma_p(i)$ is defined as direct flight connecting an origin $\sigma_p^o(i)$ to a destination airport $\sigma_p^f(i)$ without any stop in between. Each flight is also characterized by a scheduled departure time $\sigma_p^d(i)$ and an arrival time $\sigma_p^a(i)$. A rotation can be extended for several days and the alterations can be made only on the part of the rotation σ_p of an aircraft p taking place during the recovery time window.

The *flight.csv* file provides information regarding the flights operated by the airline on a typical day of the recovery period. For each flight, the following data is provided: unique identification number (strictly positive integer), origin and destination airports, departure and arrival times, and the flight number of the preceding leg (strictly positive in the case of multi-leg flights, 0 otherwise) *e.g.*:

```
1 NCE CDG 14:00 15:35 0

2 SIN LHR 15:20 05:30+1 0

3 LHR CDG 06:30 07:45 2

4 ORY CDG 08:30 09:00 0
```

The above example describes flight number 1 (domestic, see file dist.csv), leaving from Nice at 14:00 GMT and arriving at Paris-Charles de Gaulle at 15:35 GMT. It also describes the multi-leg flight from Singapore to Paris-Charles de Gaulle via London Heathrow, composed of flights numbered 2 and 3. The first leg (intercontinental) leaves from Singapore at 15:20 GMT and arrives at London Heathrow at 5:30 GMT the following day; the second flight (continental) departs London-Heathrow at 6:30 GMT and arrives in Paris at 7:45 GMT. The last line describes flight number 4 (proximity) from Paris-Orly to Paris-Charles de Gaulle, corresponding to a surface transportation "flight".

The *rotation.csv* file describes rotations for all aircraft throughout the recovery period. Each flight is uniquely defined by a flight number (strictly positive integer) and a departure date. The aircraft operating the flight is also provided (type "aircraft"). The lines are grouped by aircraft and sorted chronologically for each aircraft *e.g.*:

```
2 20/01/08 B747#5

3 21/01/08 B747#5

2 21/01/08 A340#2

3 22/01/08 A340#2

4 21/01/08 TranspCom#2
```

The above example describes the rotations of the Boeing 747 number 5, the Airbus A340 number 2, and the surface vehicle number 4 throughout the recovery period, which is from 20/01/08 to 21/01/08. These rotations consist of flights number 2 and 3 (multi-leg flights from Singapore to Paris-Charles de Gaulle) on 20/01 and 21/01, and of the surface trip from Paris-Orly to Paris-Charles de Gaulle on 21/01, respectively. Note that flight number 3 on 22/01 is not within the recovery period; however, since this flight is linked to a flight within the recovery period (flight number 2 on 21/01), it is included in the problem.

3.7. Disruptions

The possible disruptions can be grouped in three categories:

• flight disruptions, consisting in multiple flight delays or cancellations. The set of flight delays $\mathcal{D}$ is such that each delay $d \in \mathcal{D}$ can be defined by a triplet $(p,i,t) \in P \times \mathbb{N}^+ \times \mathbb{N}^+$ with p the affected aircraft, i the index of the affected leg in σ_p, and t the delay in minutes. The set of flight cancellations is defined by a couple $(p,i) \in P \times \mathbb{N}^+$ with p the affected aircraft and i the index of the affected leg in σ_p. The $alt_flight.csv$ file provides the disruptions happening to flights operated by the considered airline, namely delays and cancellations. Each impacted flight is uniquely identified by a flight number and a departure date. Information about the disruption is also provided: the length of the delay in case of delay, and -1 in case of cancellation *e.g.*:

```
2 20/01/08 45

1 21/01/08 -1
```

The above example specifies a delay of 45 minutes on flight number 2 (Singapore-London Heathrow) on 20/01/08 and the cancellation of flight number 1 (Nice to Paris-Charles de Gaulle) on 21/01/08. Flights mentioned in this file, as well as passengers travelling on them, must be taken into account for the calculation of costs in the objective function.

• aircraft disruptions, resulting in unavailability during a certain period due to mechanical failures or maintenance constraints. The set of aircraft disruptions $\mathcal{B}$ is such that each aircraft disruption $b \in \mathcal{B}$ forms a triplet $(p,s,e) \in P \times \mathbb{N}^+ \times \mathbb{N}^+$ with p the affected aircraft, s the start of the aircraft's period of unavailability and e the end time of the aircraft's unavailability period. The $alt_aircraft.csv$ file provides the periods of unavailability of aircraft. Each line contains the aircraft ID of the unavailable aircraft, the date and time of the beginning of the period of unavailability, and the date and time of the end of the period of unavailability *e.g.*:

```
A320#1 20/01/08 04:00 20/01/08 20:00
```

The above example mentions that the Airbus A320 number 1 will not be available between 4:00 and 20:00 on 20/01/08.

• airport disruptions resulting in capacity reduction in the number of departures or arrivals per hour, caused by inclement weather or industrial action. The set of airport capacity reductions $\mathcal{R}$

is such that each reduction $r \in \mathcal{R}$ is defined by a quadruplet $(a,h,z,c) \in \mathcal{A} \times RTW \times \{t,l\} \times \mathbb{N}^+$ with a the affected airport, h the time window during which the capacity reduction occurs, z the affected activity (departure or arrival), and c the new capacity. The $alt_a irport.csv$ file provides the periods of temporary reductions in airport capacities.Each line contains the three-letter code of the airport where the reduction occurs, the date and time of the start of the period of reduction, the date and time of the end of the period of reduction, and the applicable departure and arrival capacities during the period of reduction. The capacities are non-negative integers *e.g.*:

```
LHR 20/01/08 04:00 20/01/08 10:00 0 2
```

The above example corresponds to dense fog in London: no departures and only two arrivals are allowed per one-hour interval [H, H + 1[from 4:00 to 10:00.

3.8. Constraints

Since this model is a simplification of the real problem crew scheduling will not be considered. However, an aircraft change can only be done within the same family of aircraft.

3.8.1. Rotation continuity A rotation σ_p starting at the origin airport $\sigma_p^o(1) = O_p$, must be connected:

$$\forall i \in [1, |\sigma|[, \sigma_p^f(i) = \sigma_p^o(i+1) \tag{1}$$

$|\sigma|$ being the number of flights in the rotation

3.8.2. Turn-round time The aircraft p must respect the turn-round duration t_r between the consecutive legs:

$$\forall i \in [1, |\sigma_p|[, \sigma_p^a(i) + t_{rp} \leq \sigma_p^d(i+1) \tag{2}$$

The same applies for transit time between multi leg flights.

3.8.3. Maintenance The maintenance constraints are hard constraints. For a subset $P_m \subset P$, each aircraft $p \in P_m$ must undergo maintenance in a certain airport during a period of time.

4. Solution Methods

Constraint Satisfaction Problems (CSPs) have been a subject of research in artificial intelligence for many years. It has been recognised that CSPs have practical significance because many problems arising in operations research, in particular scheduling, timetabling and other combinatorial problems, can be represented as CSPs.

A constraint is simply a logical relation among several unknowns (or variables), each taking a value in a given domain. A constraint thus restricts the possible values that variables can take, it represents some partial information about the variables of interest. Constraint programming (CP) as the method of optimizing a function subject to logical, arithmetic, or functional constraints over discrete variables or interval variables.

A CSP $\mathcal{P}$ is defined by a triple $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ where $\mathcal{X}$ is an n-tuple of variables $\mathcal{X} = \langle x_1, x_2, ..., x_n \rangle$ $\mathcal{D}$ is a corresponding n-tuple of domains $\mathcal{D} = \langle D_1, D_2, ..., D_n \rangle$ such that $x_i \in Di, C$ is a t-tuple of constraints $\mathcal{C} = \langle C_1, C_2, ..., C_t \rangle$. A constraint C_j is a pair $\langle R_{S_j}, S_j \rangle$ where R_{S_j} is a relation on the variables in $S_i = scope(C_i)$. In other words, R_i is a subset of the Cartesian product of the domains of the variables in S_i (Rossi et al. 2006).

A solution to the CSP $\mathcal{P}$ is an n-tuple $A = \langle a_1, a_2, ..., a_n \rangle$ where $a_i \in D_i$ and each C_j is satisfied in that R_{S_j} holds on the projection of A onto the scope S_j . In a given task one may be required to find the set of all solutions, $sol(\mathcal{P})$, to determine if that set is non-empty or just to find any solution, if one exists. If the set of solutions is empty the CSP is unsatisfiable.

The Constructive Heurist for the Aircraft Recovery Problem (CHARP) is based in CSP and consists of two heuristics, that perform a pincer movement over the lower and upper bounds of the search space. Given the original aircraft rotation and a set of disruptions, the objective of the CHARP is to create a new combination of aircraft routings during the RTW to minimize the total cost of recovery.

The algorithm starts by initializing all the data sets after which it will loop through each aircraft and their rotations. After initializing the rotation the algorithm checks the rotation for cancelled flights or aircraft breakdown periods that also lead to flight cancellation. If these disruptions exist the algorithm will then create new flights.

Algorithm 1 describes the creation of new flights. The algorithm receives as inputs the aircraft rotation, the distance set, the maximum flight number, the aircraft breakdown period and the end time of the recovery window. The algorithm will return the aircraft rotation with the new flight and the updated maximum flight number. This algorithm starts by initializing two sub-rotations containing available flight slots, the cancelled flights and the starting time from which new flights can be created. The algorithm will afterwards loop through the new flight slots and cancelled flights and calculate the departure time and the flight time (line 6). If the departure time added to the flight time of the new flight over shoots the end time of the recovery time window the loop breaks and the algorithm returns the rotation with the new flights and the number of the last flight. Else, the flight number is incremented a and the new flight slot is updated (lines 11 to 15). Finally the algorithm checks if the new flight's arrival time added with the transit time overshoots the end time of the recovery time window. If true the algorithm breaks the loop and returns the rotation with the new flights and the number of the last flight. An identical algorithm can be derived from 1 by simply allocating the new flight slots to those that were cancelled due to flight disruption.

Algorithm 1: New flights from aircraft disruption

```
   Input: $\sigma_p, \Delta, M, \mathcal{B}_b, RTW_e$
   Output: $\sigma_p, M$
1  $\sigma_p^n \leftarrow$ List of available new flights in $\sigma_p$
2  $\sigma_p^c \leftarrow$ List of cancelled flights in $\sigma_p$
3  $start \leftarrow$ end time of aircraft disruption $\mathcal{B}_b$
4  for $\sigma_p^n(i), \sigma_p^c(i)$ $in$ $\sigma_p^n, \sigma_p^c$ do
5      if $i \neq 0$ then
6          $start \leftarrow \sigma_p^{na}(i-1) + t_{rp}$
7      $\delta_{of} \leftarrow \Delta(\sigma_p^{co}(i), \sigma_p^{cf}(i))$
8      if $start + \delta_{of} > RTW_e$ then
9          break
10     $M \leftarrow M + 1$
11     $\sigma_p^n(i) = M$
12     $\sigma_p^{nd}(i) \leftarrow start$
13     $\sigma_p^{no}(i) \leftarrow \sigma_p^{co}(i)$
14     $\sigma_p^{na}(i) \leftarrow start + \delta_{of}$
15     $\sigma_p^{nf}(i) \leftarrow \sigma_p^{cf}(i)$
16     if $\sigma_p^{na}(i) + t_{rp} > RTW_e$ then
17         break
18 return $\sigma_p, M$
```

The next step uses algorithm 2 consists in traversing the rotation to find if the following five constraints are being respected:

- Flight schedule continuity.
- Transit or turnround time between consecutive flights.
- Departure and arrival airport capacity.
- Aircraft arrive on time for maintenance.

The algorithm receives as inputs the aircraft rotation and the airport set. The algorithm will verify the infeasibilities in the rotation regarding transit time, airport departure and arrival capacity, and maintenance (lines 1 to 5). In line 6 the algorithm concatenates all the infeasibility sets in a single set and if the latter is not empty it returns the rotation and the index of the first infeasibility. Else, if the rotation is feasible the algorithm returns an empty set and -1. Consequently, the rotation will be add to the recovery solution, the departure and arrival airport capacity is updated and the algorithm and moves to the next aircraft.

Algorithm 2: Feasibility verification

```
   Input: $\sigma_p, \mathcal{A}$
   Output: $\sigma_p, index$
1  $inf1 \leftarrow continuity(\sigma_p)$
2  $inf2 \leftarrow tt(\sigma_p)$
3  $inf3 \leftarrow dep(\sigma_p, \mathcal{A})$
4  $inf4 \leftarrow arr(\sigma_p, \mathcal{A})$
5  $inf5 \leftarrow maint(\sigma_p)$
6  $inf \leftarrow inf1 \cup inf2 \cup inf3 cupinf4 \cup inf5$
7  if $inf \neq \{\}$ then
8      $index \leftarrow min(infList)$
9      return $\sigma_p, index$
10 else
11     return $[], -1$
```

Starting at the first infeasible flight in the infeasible rotation the next step of the recovery procedure consists of an algorithm that searches for each flight, of this part of the rotation, the domain where it is possible to depart and land without breaching airport departure and arrival capacity constraints. This search is done incrementally and it will allow to delay the rotation's flights. Added to the latter, and with the exception of those flights that are already subject to a disruptive delay, the algorithm also adds the option to cancel the flight as a delay valued -1. To implement this procedure, algorithm 3 receives as inputs the infeasible rotation, the airport capacity, the maximum amount of delay and the delay increment. After initializing the variables to save the rotation's flight domains, singletons and size of the search space (line 1 to 3) the algorithm loops through the rotation to compute and retrieve their respective values. The flight domain is initialized in line 5 and if it has been disrupted its domain assumes a single value of zero thus becoming a singleton (line 7). If the singleton makes the recovery infeasible by breaking the airport capacity constraint it will be added to the singleton list and the algorithm loops to the next flight in the rotation (line 8 and 9). If the flight departs outside the recovery the algorithm will simply return the flight ranges, the singleton list and the total number of combinations. If the flight is neither disrupted or departs outside the recovery time window then the algorithm adds the cancellation option. Afterwards it will loop incrementally through the range of delay values starting at zero and add any of them that make the flight feasible (lines 14 to 20). In lines 21 and 22 the algorithm adds the domain to the flight domains dictionary and updates the number of combinations.

Algorithm 3: Find flight domains

```
Input: $\sigma_p, \mathcal{A}, maxDelay, delayIncrement$
Output: $flightDomains, singletonList, totalCombos$
$flightRanges \leftarrow \{\}$
$singletonList \leftarrow \{\}$
$totalCombos \leftarrow 1$
for $\sigma_p(i)$ $in$ $\sigma_p$ do
    $domain \leftarrow \{\}$
    if $\sigma_p(i) \cap \mathcal{D}_d \neq \{\}$ then
        $domain \leftarrow domain \cup \{0\}$
        $checkSingleton(\sigma_p(i), \mathcal{A}, singletonList)$
        continue
    if $(\sigma_p^d(i) < RTW_s) \vee (\sigma_p^d(i) > RTW_e)$ then
        return $flightRanges, singletonList, totalCombos$
    $domain \leftarrow domain \cup \{-1\}$
    for $delay$ $in$ $range(0, maxDelay, delayIncrement)$ do
        $dep \leftarrow \sigma_p^d(i)$
        $arr \leftarrow \sigma_p^a(i)$
        $dep \leftarrow dep + delay$
        $arr \leftarrow arr + delay$
        $\sigma_p'(i) \leftarrow \sigma_p(i, dep, arr)$
        if $(dep(\sigma_p'(i), \mathcal{A}) = []) \wedge (arr(\sigma_p'(i), \mathcal{A}) = [])$ then
            $domain \leftarrow domain \cup delay$
    $flightDomains[\sigma_p(i)] \leftarrow domain$
    $totalCombos \leftarrow totalCombos * |domain|$
return $flightRanges, singletonList, totalCombos$
```

Each flight domain will be coded in a dictionary, using the flight number and date as the key, and a vector with all the possible delay as the value:

```
{'286802/03/08': [-1, 0], '720201/03/08': [-1, 60, 120, 180, 240, 360, 420], '720701/03/08': [-1,
0, 120, 180, 360, 420], '730801/03/08': [-1, 0, 240, 300], '737502/03/08': [-1, 0, 60, 120, 180,
240], '742002/03/08': [-1, 0, 120], '744002/03/08': [-1, 0]}
```

Each vector consists of the domain values that the flight can assume, and the search space is obtained by computing the Cartesian product between all the vectors. Any infeasible rotation has a search space that consists of a matrix whose columns are the flights and the rows are the possible values each flight can have.

$$\begin{bmatrix} -1 & -1 & -1 & ... & -1 & -1 & -1 \\ -1 & -1 & -1 & ... & -1 & -1 & 0 \\ -1 & -1 & -1 & ... & -1 & 0 & -1 \\ ... & & & & & & \\ 420 & 420 & 300 & ... & 120 & 180 & 0 \\ 420 & 420 & 300 & ... & 120 & 240 & -1 \\ 420 & 420 & 300 & ... & 120 & 240 & 0 \end{bmatrix}$$

In our experiments we found that the number of rows varies between the order of magnitude 10^3 to 10^{12}, hence it is not possible a unique approach to find feasible solutions. To tackle the two distinct situations the algorithm uses a lower heuristic for the lower bound of the search space and an upper heuristic to handle the upper bound of the search space.

The lower bound is initialized at 4×10^4 by default, and the lower heuristic loops through every row of the of the matrix in order to find the optimal solutions that minimize the number of cancelled flights and the total amount of delay. Algorithm 4 receives as inputs the rotation, the index of the first infeasibility and the search space in the form of a matrix. The algorithm will recover the infeasible rotation and will output the recovered one. It starts by initializing the best solution in line 1 and henceforward it will loop through the search space to find its value. In lines 3 and 4 for the algorithm sums the the cancelled flights and the total amount of delay for each row of the matrix and in line 5 it assigns them to the new solution. The algorithm compares the new solution with the best one and if the new one is not better it will iterate (lines 6 to 10). Based on the values coded in the row, in line 11 the algorithm creates the new rotation by cancelling or delaying flights. Afterwards if the row creates a new feasible solution the best solution is updated (lines 12 to 18). After traversing the entire search space the algorithm updates the aircraft rotation with the optimal solution and returns it (lines 19 and 20).

Algorithm 4: Lower heuristic

```
Input: $\sigma_p, index, matrix$
Output: $\sigma'_p$
1  $bestSol \leftarrow \{\}$
2  for $row$ $in$ $matrix$ do
3      $noCancel \leftarrow \sum_i row(i)$, if $row(i) = -1$;
4      $totalDelay \leftarrow \sum_i row(i)$, if $row(i) \neq -1$
5      $newSol \leftarrow \{noCancel, totalDelay, row\}$
6      if $bestSol \neq \{\}$ then
7          if $newSol[0] < bestSol[0]$ then
8              continue
9          if $(newSol[0] = bestSol[0]) \wedge (newSol[1] > bestSol[1])$ then
10             continue
11     $\sigma'_p \leftarrow \sigma_p(row)$
12     if $continuity(\sigma'_p) \neq \{\}$ then
13         continue
14     if $tt(\sigma'_p) \neq \{\}$ then
15         continue
16     if $maint(\sigma'_p) \neq \{\}$ then
17         continue
18     $bestSol \leftarrow newSol$
19 $\sigma'_p \leftarrow \sigma_p(bestSol[2])$
20 return $\sigma'_p$
```

For the initialization starts The upper heuristic's upper bound is initialized at 3×10^{12} by default. This heuristic decomposes the rotation into sub-rotations with a search space size lower than the lower bound defined for the lower heuristic. The algorithm loops through every sub-rotation until it finds a feasible solution for the entire rotation. Although this procedure does not return an optimal solution, it can find feasible solutions in a reasonable computing time. Algorithm 5 receives as inputs the infeasible rotation, the index of the first infeasibility and flight domains. On line 1, the lower index of the sub-rotation is initialized with the value of the index of the first infeasibility. I line 2 the algorithm uses a sub-routine that computes the upper index for the sub-rotation that will be recovered and extracts the corresponding flight domains. In line 3 the algorithm computes the search space and in line 4 reduces the flight domains to the un-recovered part of the inputted rotation. From line 5 to 29 the algorithm will loop through every sub-rotation, recovering each one of them. With the exception of maintenance check, the algorithm finds the best solution to recover the sub-rotation using a similar method as in the lower heuristic (lines 7 to 21). In line 22 the part between the lower and the upper index of the inputted rotation is updated with the best solution. If the upper index equals the size of the inputted rotation this means the recovery procedure is over and it returns the recovered rotation (lines 23 and 24). Else, the algorithm assigns the upper index to the lower index, computes the new upper index and new partial flight domains, computes the search space and the remaining flight domains.

Algorithm 5: Upper heuristic

```
Input: $\sigma_p, index, flightDomains$
Output: $\sigma_p$
1  $lIndex \leftarrow index$
2  $uIndex, partialFlightDomains \leftarrow upperIndex(lIndex, \sigma_p, flightDomains)$
3  $matrix \leftarrow product(partialFlightDomains.values())$
4  $removeFlightDomains(flightDomains, \sigma_p(i) \forall i \in [uIndex, |\sigma_p|])$
5  while $True$ do
6      $bestSol \leftarrow \{\}$
7      for $row\ in\ matrix$ do
8          $noCancel \leftarrow \sum_i row(i)$, if $row(i) = -1$;
9          $totalDelay \leftarrow \sum_i row(i)$, if $row(i) \neq -1$
10         $newSol \leftarrow [noCancel, totalDelay, row]$
11         if $bestSol \neq \{\}$ then
12             if $newSol[0] < bestSol[0]$ then
13                 continue
14             if $(newSol[0] = bestSol[0]) \wedge (newSol[1] > bestSol[1])$ then
15                 continue
16         $\sigma'_p(i) \leftarrow \sigma_p(row)$
17         if $continuity(\sigma'_p) \neq \{\}$ then
18             continue
19         if $tt(\sigma'_p) \neq \{\}$ then
20             continue
21         $bestSol \leftarrow newSol$
22     $\sigma_p = newPartialRotation(bestSol[2], \sigma_p(i) \forall i \in [lIndex : uIndex])$
23     if $uIndex = |\sigma_p|$ then
24         return $\sigma_p$
25     else
26         $lIndex \leftarrow uIndex$
27         $uIndex, partialFlightDomains \leftarrow upperIndex(lIndex, \sigma_p, flightDomains)$
28         $matrix \leftarrow product(partialFlightDomains.values())$
29         $removeFlightDomains(flightDomains, \sigma_p(i) \forall i \in [uIndex, |\sigma_p|])$
```

It is important to notice that in order to optimize the looping through the search space, the heuristics compare the current solution with the new one and if the latter is not better they will not proceed to test feasibility, thus saving significant computation time. As for the over arching algorithm, in every iteration it loops through the aircraft list and based on the size of the search space decides which heuristic will find solutions to recover the infeasible rotations, minimizing the number of cancelled flights and the total amount of delay. However, the search space size may be above the lower bound or below the upper bound, hence the infeasible rotation is not recovered. To overcome this situation the algorithm iterates the aircraft loop using the list of aircraft left with infeasible rotations, increments the lower bound and decrements the upper bound.This procedure, combining the movement of the lower and upper bounds, results in a pincer movement that will entrap the entire search space, making sure that every infeasible rotation recovers.

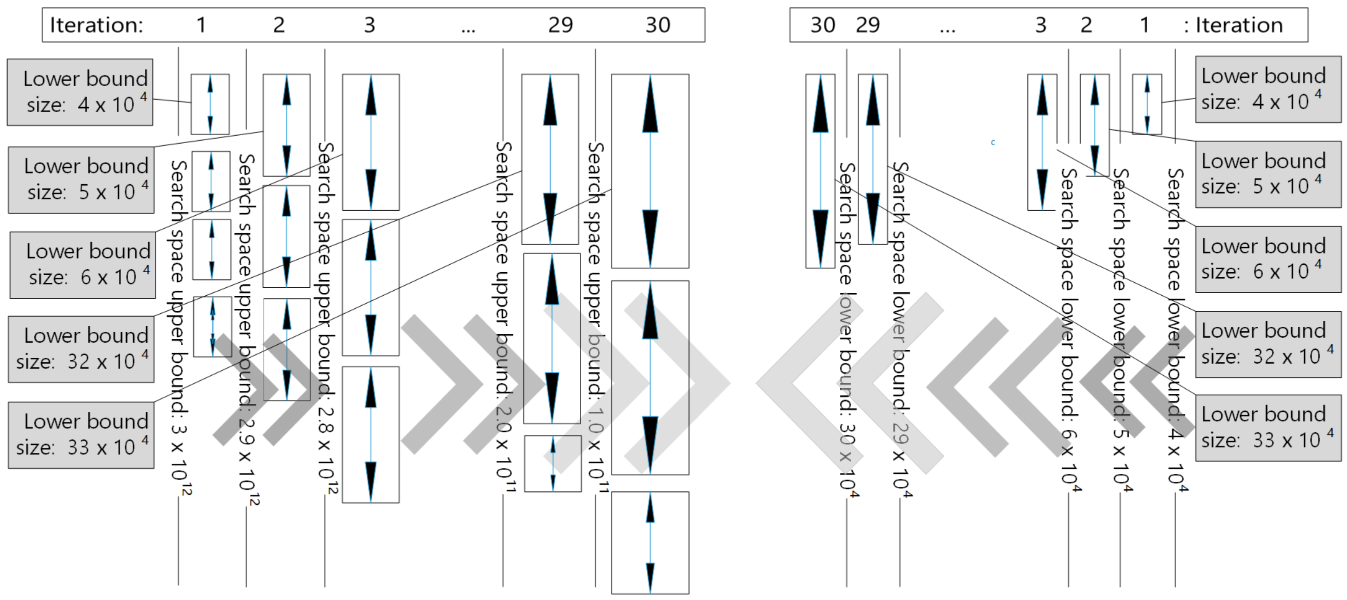
Iteration: 1 2 3 ... 29 30
Lower bound size: 4 x 10^4
Lower bound size: 5 x 10^4
Lower bound size: 6 x 10^4
Lower bound size: 32 x 10^4
Lower bound size: 33 x 10^4
Search space upper bound: 3 x 10^12
Search space upper bound: 2.9 x 10^12
Search space upper bound: 2.8 x 10^12
Search space upper bound: 2.0 x 10^11
Search space upper bound: 1.0 x 10^11
30 29 ... 3 2 1 : Iteration
Search space lower bound: 30 x 10^4
Search space lower bound: 29 x 10^4
Search space lower bound: 6 x 10^4
Search space lower bound: 5 x 10^4
Search space lower bound: 4 x 10^4

Figure 1 **Pincer heuristic**

In CSP it is common to find variables that have domain size 1, such variables are designated by singletons. Since both heuristics are based in constraint satisfaction programming, the rotations that we know in advance having variables with domain size 1, are handled first. In case of rotations with schedule maintenance, the algorithm treats them as a flight without turn round time and with the same origin and destination. Thus when the algorithm creates the aircraft list, the first aircraft have scheduled maintenance. The flights that are disrupted with delays are designated by fixed flights and they too cannot be moved. In this case the domain is a singleton consisting of value $\{0\}$ and if this value is infeasible, because there are no available departure and/or arrival airport capacity, the algorithm backtracks by removing the rotation of an aircraft that can release the necessary airport capacity.

Algorithm 6 receives as inputs the singleton list, the airport capacity, the ARP current solution, the rotation nad the index of the first infeasibility. The algorithm will return the ARP solution without the rotation that will allow the singleton to become feasible and, the updated aircraft list. This algorithm will loop while the singleton list is not empty and will verify if the first singleton's infeasibility is on the departure and/or in the arrival airport capacity. Depending on the situation the algorithm computes the airport time slot for the departure/arrival, and based on the origin/destination airport searches the aircraft to cancel. The algorithm will then remove the aircraft from the solution list, and the respective rotation from the ARP solution. Finally, we will use algorithm 3 to determine there are any more infeasible singletons.

Algorithm 6: Backtracking

```
Input: $singletonList, \mathcal{A}, aircraftSolList, solutionARP, \sigma_p, index$
Output: $solutionARP, aircraftSolList$
1  while $singletonList \neq \{\}$ do
2      if $singletonList(0) =' dep'$ then
3          $startInt \leftarrow 60 * int(singleton^d(0)/60)$
4          $endInt \leftarrow startInt + 60$
5          $origin \leftarrow singleton^o(0)$
6          $flight2Cancel \leftarrow solutionARP[(origin, startInt, endInt)]$
7          $airc2Cancel \leftarrow updateMulti(flight2Cancel, \mathcal{A}, solutionARP])$
8          $aircraftSolList \leftarrow aircraftSolList - airc2Cancel$
9          $solutionARP.pop(airc2Cancel)$
10         $flightRanges, singletonList, totalCombos \leftarrow domainFlights(\sigma_p(i) \forall i \in$
             $[index, |\sigma_p|], \mathcal{A}, index)$
11     if $singleton(0) =' arr'$ then
12         $startInt \leftarrow 60 * int(singleton^a(0)/60)$
13         $endInt \leftarrow startInt + 60$
14         $destination \leftarrow singleton^f(0)$
15         $flight2Cancel \leftarrow solutionARP[(destination, startInt, endInt)]$
16         $airc2Cancel \leftarrow updateMulti(flight2Cancel, \mathcal{A}, solutionARP)$
17         $aircraftSolList \leftarrow aircraftSolList - airc2Cancel$
18         $solutionARP.pop(airc2Cancel)$
19         $flightRanges, singletonList, totalCombos \leftarrow domainFlights(\sigma_p(i) \forall i \in$
             $[index, |\sigma_p|], \mathcal{A}, index)$
20 return $solutionARP, aircraftSolList$
```

After finding a feasible solution for the part of the rotation that was initially infeasible the algorithm tries to reconnect both parts but on occasions it is possible to find discontinuities between them. Algorithm 33 receives as inputs the set of aircraft disruptions, the set of distances, the origin airport, the recovered rotation the maximum flight number and returns the updated recovered rotation and the maximum flight number. Algorithm 33 will either create a taxi flight to connect the first part of the rotation and the recovered one, or cancel flights in the recovered rotation until the continuity infeasibility is removed.

It starts in line 1 initializing the origin slots where there is available departure capacity. Afterwards the algorithm will loop through the flights of the recovered rotation and if their origin is the same as origin airport it returns the updated recovered rotation and the maximum flight number (lines 2 to 4). In line 5 the algorithm computes the distance from the origin airport and the origin of the flight in the recovered rotation. In line 6 the algorithm extracts the upper slots from the origin slots by subtracting to the flight's departure time in the recovered rotation the distance and the turn round time. In line 7 the upper slots and the aircraft breakdown period are subtracted to the origin slots in order to retrieve the lower slots from where the aircraft can depart.

If there are no available departure slots the algorithm cancels the flight in the recovered rotation and continues to the next flight (lines 8 to 10). If there are available departure slots at the origin, the algorithm tries to find destination slots with available airport arrival capacity. To achieve the latter the algorithm finds the destination slots with available capacity, the upper destination slots, and by subtracting the latter to the former and aircraft breakdown period determines the lower destination slots (lines 11 to 13). If there are no lower destination slots, the the algorithm cancels the flight in the recovered rotation and continues to the next flight. Since $\mathcal{A}$ is a dictionary it is necessary to extract the destination intervals and initialize the destination index i and the the time offset from which the flight departs and arrives in feasible airport slots (lines 17 to 21). The algorithm will then loop through the destination slots and in line 23 it will initialize the object obj with the starting and end time of the origin lower slots plus the distance and, in line 24 determine the index of intersection in the destination slot and the offset. If the index i is different from -1 the algorithm will the taxi flight and add it to the recovered rotation (lines 25 to 33).

In figure 2 we provide the complete flowchart of the CHARP. Step 1 of the algorithm consists in loading the data set, after which it starts looping through the aircraft list. In step 2 the algorithm selects an aircraft and it verifies if the rotation has been initialized, if it has not in step 4 and 5 it lists the infeasibilities and adds new flights. If the aircraft's rotation has been initialized in step 6 the algorithm checks if there are infeasibilities and if there are none, in step 10 it updates the ARP's solution with the aircraft's rotation, the airport capacity and the aircraft's solution list. If there are infeasibilities the algorithm will, in step 70, find the flight domains and compute the search space size, after which it will choose solution method (steps 710 and 720) and if necessary the algorithm backtracks. If the size of the search space is lower than the lower bound the algorithm uses the lower heuristic algorithm (step 723), if the search space size is bigger than the upper bound it will use the upper heuristic algorithm (step 713). If neither of the previous apply the algorithm verifies if the loop has finished and if it has not it will move to the next aircraft. After recovering the infeasible part of the rotation the algorithm checks, in step 8, if there is a continuity infeasibility when it tries to reconnect both parts. If there is, in step 9 the taxi flights algorithm reconnects both parts and afterwards adds the recovered rotation to the solution, updates the airport capacity and updates the aircraft solution list (step 10). In step 11 the algorithm checks if the loop has finished aircraft list and if it has it checks if there are any aircraft left to recover (step 12). If there are the algorithm updates the lower and the upper bound and iterates. If there are no more aircraft the algorithm ends.

Algorithm 7: Taxi flights

```
   Input: $\mathcal{B}, \Delta, originAirport, \sigma_p, \mathcal{A}, M$
   Output: $\sigma_p, M$
1  $originSlots \leftarrow \mathcal{A}[originAirport]$ if $capDep > noDep$
2  for $\sigma_p(i)$ in $\sigma_p$ do
3      if $\sigma_p^o(i) = originAirport$ then
4          return $\sigma_p$, M
5      $\delta_{of} \leftarrow \Delta(originAirport, \sigma_p^o(i))$
6      $originSlotsUpper \leftarrow originSlots$ if $endInt > \sigma_p^d(i) - \delta_{of} - t_{rp}$
7      $originSlotsLower \leftarrow originSlots - originSlotsUpper - [\mathcal{B}_p^s, \mathcal{B}_p^e]$
8      if $originSlotsLower = \{\}$ then
9          cancel($\sigma_p(i)$)
10         continue
11     $destinationSlots \leftarrow \mathcal{A}[\sigma_p^o]$ if $capArr > noArr$
12     $destinationSlotsUpper \leftarrow destinationSlots$ if $endInt > \sigma_p^d(i) - t_{rp}$
13     $destinationSlotsLower \leftarrow destinationSlots - destinationSlotsUpper - [\mathcal{B}_p^s, \mathcal{B}_p^e]$
14     if $destinationSlotsLower = \{\}$ then
15         cancel($\sigma_p(i)$)
16         continue
17     $destIntervals \leftarrow \{\}$
18     $i \leftarrow -1$
19     $offset \leftarrow -1$
20     for $x \in destinationSlotsLower$ do
21         $destIntervals \leftarrow destIntervals \cup [x^s, x^e]$
22     for $os(i)$ in $originSlotsLower$ do
23         $obj \leftarrow interval(os^s, os^e) + distInitRot$
24         $i, offset \leftarrow obj.findIntersection(destIntervals)$
25         if $i \neq -1$ then
26             $taxiFlight^o \leftarrow originAirport$
27             $taxiFlight^d \leftarrow os['startInt'] + offset$
28             $taxiFlight^f = \sigma_p^o(i)$
29             $taxiFlight^a = taxiFlight^d + \delta_{of}$
30             $taxiFlight['flight'] = M$
31             $M \leftarrow M + 1$
32             $\sigma_p \leftarrow \sigma_p + taxiFlight$
33             return $\sigma_p, M$
```

5. Computational Results

In this section, five test scenarios were proposed to validate the effectiveness and the efficiency of the CHARP. All the models and algorithms were written in Python language. The scripts were implemented on a computer with 4-core processors running at 2.3 GHz and 16 GB memory.

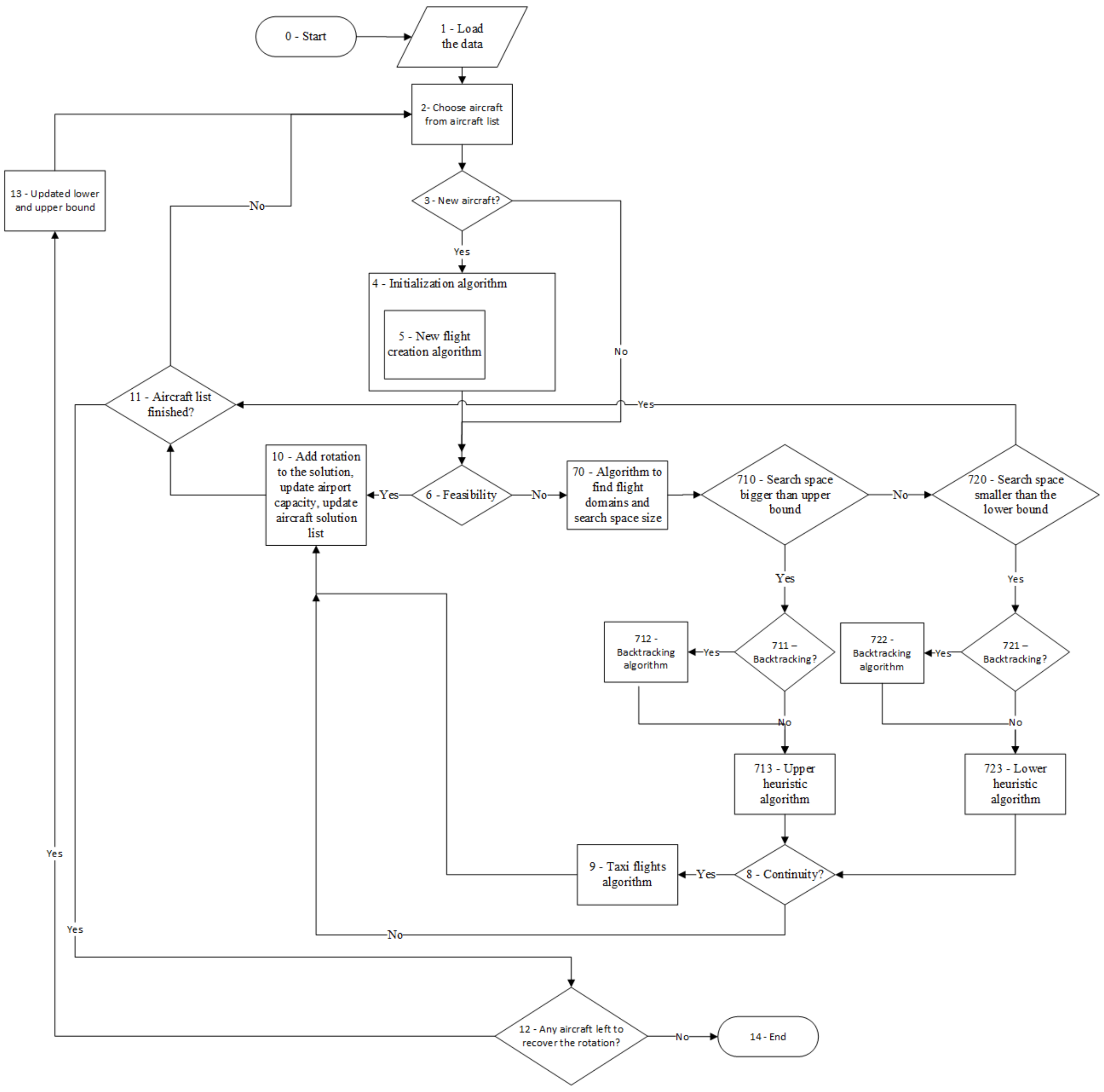
0 - Start
1 - Load the data
2- Choose aircraft from aircraft list
3 - New aircraft?
Yes
No
4 - Initialization algorithm
5 - New flight creation algorithm
13 - Updated lower and upper bound
11 - Aircraft list finished?
Yes
10 - Add rotation to the solution, update airport capacity, update aircraft solution list
6 - Feasibility
70 - Algorithm to find flight domains and search space size
710 - Search space bigger than upper bound
720 - Search space smaller than the lower bound
711 – Backtracking?
712 - Backtracking algorithm
721 – Backtracking?
722 - Backtracking algorithm
713 - Upper heuristic algorithm
723 - Lower heuristic algorithm
8 - Continuity?
9 - Taxi flights algorithm
12 - Any aircraft left to recover the rotation?
14 - End

Figure 2 **Constructive heuristic**

5.1. Research Scenarios

The scenarios that are proposed aim at determining the computing time,the total amount of delay, number of cancelled flights and cost. Since the pincer heuristic is a constructive heuristic we aim at understanding its results for the overall 32 data instances of the ROADEF 2009 Challenge, namely the effect of the size of the:

- delay window for 900, 960, 1020, 1080, 1140 and 1200 minutes

- domain step for 20 and 60 minutes
- decremental step for the upper heuristic for 1×10^{11} and 2×10^{11}
- incremental step for the lower heuristic for 1×10^{4} and 2×10^{4}

For all the scenarios the upper heuristic starts at 3×10^{12} and the lower heuristic starts at 4×10^{4}. To obtain the computing time, we run all 32 instances simultaneously and we retrieve the computing time of the last one being solved. As for the cost we use the cost checker application provided by the ROADEF 2009 Challenge. The first scenario being tested has the domain step of 60 minutes, decremental step for the upper heuristic for 1×10^{11}, and incremental step for the lower heuristic for 1×10^{4}. In table 3 we show the results for time windows of 900, 960 and 1200 minutes:

Table 3 Scenario 1 results

Time window [min.]	Cost [EURO]	Solution time [sec.]	Total delay [min.]	Number of cancelled flights
900	624,720,102.75	1,160.78	997,380	2,221
960	627,075,226.70	1,451.42	1,212,900	2,145
1200	1,254,846,959.00	1,422.62	2,716,860	4,543

6. Conclusions

It turns out that recovery is a practical problem that needs to be solved during operations, therefore, the efficiency of the method in terms of computational time is a very important characteristic. Most of the works in the literature reach computational times of less than thirty minutes, however it is important that this limit is possible for large instances, that is, those with more than 400 flights. These two conditions are achieved in a few publications, which shows that there is room for advances in research. One factor that facilitates these objectives is the greater availability of high-performance computing infrastructure in recent years. Another feature that has increased its presence in the research is the models' ability to reflect the flight's block time. This is important so that companies can, in fact, enjoy the benefits promised by these methods. Otherwise, the professional practice of disruption recovery will continue to be done manually in the operational centres. The constructive heuristic proposed in this work seeks to achieve these two objectives.

Acknowledgments

The authors gratefully acknowledge the existence of the Journal of Irreproducible Results and the support of the Society for the Preservation of Inane Research.

References

Acuña-Agost R, Michelon P, Feillet D, Gueye S (2009) Statistical analysis of propagation of incidents for rescheduling simultaneously flights and passengers under disturbed operations. *ROADEF 2009. 10éme Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision.*

Aktürk MS, Atamtürk A, Gürel S (2014) Aircraft rescheduling with cruise speed control. *Operations Research* 62(4):829–845, URL `http://dx.doi.org/10.1287/opre.2014.1279`.

Andersson T (2006) Solving the flight perturbation problem with meta heuristics. *Journal of Heuristics* 12(1-2):37–53, ISSN 1381-1231.

Arguello MF, Bard JF, Yu G (1997) A grasp for aircraft routing in response to groundings and delays. *Journal of Combinatorial Optimization* 1(3):211–228.

Arikan U, Gurel S, Akturk MS (2016) Integrated aircraft and passenger recovery with cruise time controllability. *Annals of Operations Research* 236(2):295–317, ISSN 1572-9338, URL `http://dx.doi.org/10.1007/s10479-013-1424-2`.

Bisaillon S, Cordeau JF, Laporte G, Pasin F (2011) A large neighbourhood search heuristic for the aircraft and passenger recovery problem. *4OR: A Quarterly Journal of Operations Research* 9(2):139–157.

Bratu S, Barnhart C (2006) Flight operations recovery: New approaches considering passenger recovery. *J. of Scheduling* 9(3):279–298, ISSN 1094-6136.

Cao J, Kanafani A (1997a) Real-time decision support for integration of airline flight cancellations and delays part i: mathematical formulation. *Transportation Planning and Technology* 20(3):183–199.

Cao J, Kanafani A (1997b) Real-time decision support for integration of airline flight cancellations and delays part ii: algorithm and computational experiments. *Transportation Planning and Technology* 20(3):201–217.

Darlay J, Kronek LP, Schrenk S, Zaourar L (2009) Une approche heuristique pour la gestion de perturbation dans le domaine aérien, challenge roadef 2009. *Roadef 2009, 10éme congrès de la société Française de Recherche Opérationnelle et d'Aide à la Désicion.*

Dickson S, Smith O, Li W (2009) Challenge roadef 2009: Disruption management for commercial aviation, a mixed integer programming approach. *ROADEF 2009* 401.

Eggenberg N, Salani M (2009) Challenge roadef 2009. airline disruption recovery roadef challenge 2009. *ROADEF 2009* 403.

Eggermont C, Firat M, Hurkens C, Modelski M (2009) Roadef 2009 challenge: Description of the tue solution method.

Jarrah AIZ, Yu G, Krishnamurthy N, Rakshit A (1993) A decision support framework for airline flight cancellations and delays. *Transportation Science* 27(3):266–280, ISSN 00411655, 15265447.

Jozefowiez N, Mancel C, Mora-Camino F (2013) A heuristic approach based on shortest path problems for integrated flight, aircraft, and passenger rescheduling under disruptions. *Journal of the Operational Research Society* 64(3):384–395.

Løve M, Sørensen KR, Larsen J, Clausen J (2001) Using heuristics to solve the dedicated aircraft recovery problem .

Mansi R, Hanafi S, Wilbaut C, Clautiaux F (2012) Disruptions in the airline industry: math-heuristics for re-assigning aircraft and passengers simultaneously. *European Journal of Industrial Engineering 10* 6(6):690–712.

Marla L, Vaaben B, Barnhart C (2017) Integrated disruption management and flight planning to trade off delays and fuel burn. *Transportation Science* 51(1):88–111, ISSN 1526-5447, URL `http://dx.doi.org/10.1287/trsc.2015.0609`.

Peekstok J, Kuipers E (2009) Roadef 2009 challenge: Use of a simulated annealing-based algorithm in disruption management for commercial aviation. *)ˆ(Eds.):‘Book Roadef* .

Rossi F, Beek Pv, Walsh T (2006) *Handbook of Constraint Programming (Foundations of Artificial Intelligence)* (New York, NY, USA: Elsevier Science Inc.), ISBN 0444527265.

Teodorović D, Guberinić S (1984) Optimal dispatching strategy on an airline network after a schedule perturbation. *European Journal of Operational Research* 15(2):178 – 182, ISSN 0377-2217.

Thengvall BG, Bard JF, Yu G (2000) Balancing user preferences for aircraft schedule recovery during irregular operations. *IIE Transactions* 32(3):181–193.

Thengvall BG, Yu G, Bard JF (2001) Multiple fleet aircraft schedule recovery following hub closures. *Transportation Research Part A: Policy and Practice* 35(4):289 – 308, ISSN 0965-8564.

Yan S, Lin CG (1997) Airline scheduling for the temporary closure of airports. *Transportation Science* 31(1):72–82.