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

An aircraft rotation consists in assigning flights to aircraft in such a way that operational constraints, such as, maintenance, flight continuity, turn-round time and airport capacity, are satisfied. Aircraft rotations are often subject to disruption. The Aircraft Recovery Problem (ARP) consists in altering the planned flight schedule to compensate the disruptions that resulted in the temporary or permanent unavailability of aircraft. In this work, we propose a recovery procedure for each disrupted rotation that starts by finding the first infeasible flight. We split the rotation in two parts, the first one which is feasible and second one which is infeasible. For the infeasible part of the aircraft rotation, the algorithm finds if the flights can be cancelled or delayed during a given time window. The algorithm returns the flight domains and by calculating the cartesian product of these domains we derive the search space for the infeasible part of the rotation. Depending on the size of the search space, the algorithm uses the upper heuristic to find solutions above a certain size of the search space or the lower heuristic to find solutions below a certain size of the search space. For each iteration over the aircraft rotation the algorithm will perform a pincer movement by simultaneously decreasing the upper bound and increasing the lower bound, thus closing in on a feasible solution. During the recovery procedure we introduce new flights in the infeasible part of the rotation. If we find discontinuity when attempting to reconnect the recovered part of the rotation with the first part of the rotation, the algorithm also adds taxi flights. We present results for the solution cost and computing time for the Constructive Heuristic for the Aircraft Recovery Problem (CHARP) for various time windows and the introduction of new flights and taxi flights.

Key words: Constraint satisfaction, aircraft, disruption, recovery

1. Introduction

The objective for airline companies consists in maximize their profits while operating flights according to a schedule. Therefore, airlines create tight schedules 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 while complying with operational constraints such as, aircraft maintenance, flight

continuity, turn-round time and airport capacity for departure and arrival. Flight operations are on many occasions subject to irregularities that can complicate its execution.

A considerable 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 must be delayed or cancelled. Since airline companies are liable to pay compensation for the resulting passenger inconvenience in these situations, leading to a negative financial impact on an airline's profits, it thus becomes of the most importance to resume normal operations as quickly as possible.

In case of disruptions, it is necessary to recover the rotation during the period of time designated by the recovery time window (RTW). During the RTW a feasible rotation must be found that mitigates the impact of disruptions. This is 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 cases, it is possible to know a disruption in-advance), and a provisional plan must be provided quickly. Thus, it is necessary to construct efficient algorithms to have feasible solutions in minutes. It is also especially important to refer that when disruptions occur, they can affect more than just the directly disrupted flight and can send a shockwave of disruptions through the network.

It is during RTW's duration that actions should be implemented to re-plan usage of resources all over the entire network of flights. The objective that is intended 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, also known as the aircraft recovery problem (ARP). To achieve this objective, we will use the data sets for the ROADEF 2009 Challenge.

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 recovery problem is provided in Section 3; the CHARP will be described in detail in Section 4; the computational results will be presented in Section 5; the conclusions will be drawn in Section 6.

2. Literature Review

During the years several methodologies have been developed several approaches to tackle unpredicted irregularities that often cause disruptions in elaborately generated flight schedules.

As reviewed by (Teodorović and Guberinić 1984), the authors 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 considerable number of routings for even modest sized problems, it is doubtful that their approach could be materialized for practical problems. On the other hand, since 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 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. 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 the scope of this research to the operations of a single fleet as well as simplifying multi-leg flights by considering that the 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 limitations 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 neighbourhood search technique. In the procedure, the neighbours of an addressed solution are evaluated, and the most desirable are placed on a restructured candidate list. Neighbour 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 could produce in most cases optimal or near-optimal solutions.

(Thengvall et al. 2000) solves the aircraft recovery problem for a single fleet over the entire flight schedule with an objective function that accounts for flight revenues, delays, cancellations, and 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 modelled 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 aircraft 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 neighbourhood 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, significantly better solutions were not found. As for the SALS the authors mention that the

algorithm quickly finds 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 cancelling flights. The model proved to deliver 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 than 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. 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 departure time decision within the window of feasible departures for a specific flight.

Due to market deregulation, airlines started conceiving optimal flight schedules, with little slack capable to accommodate irregularities in the operations. (Kohl et al. 2007) discussed a system spanning multiple resources method integrating data sources. The authors claim the system improved the quality of decision-making by providing flexible tools that could add value to the business process of operations control at various airlines. An ulterior method used linear a programming model to improve robustness of aircraft schedules through flight re-timing and subject to a fixed aircraft routing to improve the solution's quality has been proposed by (AhmadBeygi et al. 2010). As an objective function, they considered the sum of the impacts of each individual primary delay (selected out of a discrete set of possible delay values) as it propagates downstream, weighting these delays by their relative probabilities. To make aircraft routes less vulnerable to disruptions Aloulou et al. (2013) developed a model that distributed slacks to connections where they are most needed operationally. To meet this end, the model assigns legs to aircraft and determines the flights departure times, while maintaining the designated time-slot assignments at airports and satisfying operational constraints. To find solutions for flight schedules subject to random delays (Ahmed et al. 2017) used a hybrid optimization-simulation approach based on a mixed-integer

non-linear programming model for the robust weekly aircraft maintenance routing problem. The recovery problem is solved with a full set of recovery options, including aircraft re-routing and flight delays and cancellations. In the event of a disruption, the aircraft routes in the planning stage solution may not be feasible for the disrupted flight schedules, forcing the creation of new routes. These new routes must comply with the origin and destination locations and also maintenance requirements. To improve the tractability and reduce solution runtime the authors used column generation and Benders' decomposition. The authors claim that on-time performance was improved and, the cumulative delays and passenger were reduced.

3. The Model for the Aircraft Recovery Problem

The ARP can be defined as the problem of modifying the aircraft rotation to compensate the presence of disruptions during operations that make the rotation infeasible. The strategy to overcome the disruptions, can only be implemented in a time window designated by recovery time window (RTW).

We will use the ROADEF 2009 Challenge data set to validate the CHARP. This data set has thirty two data instances which have the files that will be used in our model. In the following subsections we will describe each of the files, namely in Subsection 3.1 *airports.csv*, in Subsection 3.2 *dist.csv*, in Subsection 3.3, in Subsection 3.4 *flights.csv* and *rotations.csv*, in Subsection 3.5 *alt_flight.csv*, in Subsection 3.6 *alt_aircraft.csv*, and in Subsection 3.7 *alt_airport.csv*. Finally in Section 3.8 we will describe the constraint that aircraft are subject to.

3.1. 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).

3.2. Distance Between Airports

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.

3.3. Aircraft

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 remaining 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.

3.4. Flights and Rotations

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.

3.5. Flight Disruptions

Flight disruptions, consists of 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 $\langle p, i, t \rangle \in P \times \mathbb{N}^+ \times \mathbb{N}^+$ 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 $\langle p, i \rangle \in P \times \mathbb{N}^+$ 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.

3.6. Aircraft Disruptions

Aircraft disruptions, results 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 $\langle p, s, e \rangle \in P \times \mathbb{N}^+ \times \mathbb{N}^+$ 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.

3.7. Airport Disruptions

Airport disruptions results 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 $\langle a, h, z, c \rangle \in \mathcal{A} \times RTW \times \{t, l\} \times \mathbb{N}^+$ 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_airport.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. Aircraft Constraints

Since this model is a simplification of the real problem neither crew scheduling or passenger re-accommodation will 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. In expression 1 we have, in the interval from the first flight to the open end of the rotation size, the arrival airport of the current flight equals the departing airport of the next flight:

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

$|\sigma|$ 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. The Constructive Heuristic for the Aircraft Recovery Problem

The Constructive Heuristic for the Aircraft Recovery Problem (CHARP) is based in Constraint Satisfaction Problems (CSP) and consists of two heuristics, that perform a pincer movement over two subsets of the search space. We define two parameters consisting of an upper bound and lower bound. If the search space size is greater than the upper bound the model uses the upper heuristic, whereas if the search space is lower than the lower bound the model uses the lower heuristic. 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. In the next sections we describe in detail the algorithms that compose the CHARP, namely in Section 4.1 the algorithm that creates the flights in the presence of flight or aircraft disruption, in Section 4.2 the algorithm that checks the rotation's feasibility, in Section 4.3 the algorithm that computes the domains for each flight, in Section 4.4 the lower heuristic, in Section 4.5 the upper heuristic, in Section 4.6 the backtracking algorithm, in Section 4.7 the taxi flights algorithm, and finally in Section 4.8 the CHARP algorithm.

4.1. New Flights Algorithm

The algorithm starts by initializing the particular data instance 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 overshoots 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: σ_p, M

1 $\sigma_p^n \leftarrow$ List of available new flights in σ_p
2 $\sigma_p^c \leftarrow$ List of cancelled flights in σ_p
3 $start \leftarrow$ end time of aircraft disruption $\mathcal{B}_b$
4 **for** $\sigma_p^n(i), \sigma_p^c(i)$ *in* σ_p^n, σ_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 σ_p, M

4.2. Feasibility Verification Algorithm

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 added 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 $inf \leftarrow inf1 \cup inf2 \cup inf3 cup inf4 \cup inf5$
7 **if** $inf \neq \{\}$ **then**
8 $\quad index \leftarrow min(infList)$
9 $\quad$ return $\sigma_p, index$
10 **else**
11 $\quad$ return $[], -1$

4.3. Flight Domains Algorithm

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 except 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 nor 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.

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]}

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* σ_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 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.

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

4.4. Lower Heuristic Algorithm

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 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).

4.5. Upper Heuristic Algorithm

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

Algorithm 4: Lower heuristic

Input: $\sigma_p, index, matrix$

Output: σ'_p

```
1  bestSol ← {}
2  for row in matrix do
3      noCancel ← Σ_i row(i), if row(i) = −1;
4      totalDelay ← Σ_i row(i), if row(i) ≠ −1
5      newSol ← {noCancel, totalDelay, row}
6      if bestSol ≠ {} then
7          if newSol[0] < bestSol[0] then
8              continue
9          if (newSol[0] = bestSol[0]) ∧ (newSol[1] > bestSol[1]) then
10             continue
11     σ'_p ← σ_p(row)
12     if continuity(σ'_p) ≠ {} then
13         continue
14     if tt(σ'_p) ≠ {} then
15         continue
16     if maint(σ'_p) ≠ {} then
17         continue
18     bestSol ← newSol
19 σ'_p ← σ_p(bestSol[2])
20 return σ'_p
```

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.

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 overarching 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,

Algorithm 5: Upper heuristic

Input: $\sigma_p, index, flightDomains$

Output: σ_p

$lIndex \leftarrow index$
$uIndex, partialFlightDomains \leftarrow upperIndex(lIndex, \sigma_p, flightDomains)$
$matrix \leftarrow product(partialFlightDomains.values())$
$removeFlightDomains(flightDomains, \sigma_p(i) \forall i \in [uIndex, |\sigma_p|])$
while *True* **do**
  $bestSol \leftarrow \{\}$
  for row *in* $matrix$ **do**
    $noCancel \leftarrow \sum_i row(i)$, if $row(i) = -1$;
    $totalDelay \leftarrow \sum_i row(i)$, if $row(i) \neq -1$
    $newSol \leftarrow [noCancel, totalDelay, row]$
    if $bestSol \neq \{\}$ **then**
      if $newSol[0] < bestSol[0]$ **then**
        continue
      if $(newSol[0] = bestSol[0]) \wedge (newSol[1] > bestSol[1])$ **then**
        continue
    $\sigma'_p(i) \leftarrow \sigma_p(row)$
    if $continuity(\sigma'_p) \neq \{\}$ **then**
      continue
    if $tt(\sigma'_p) \neq \{\}$ **then**
      continue
    $bestSol \leftarrow newSol$
  $\sigma_p = newPartialRotation(bestSol[2], \sigma_p(i) \forall i \in [lIndex : uIndex])$
  if $uIndex = |\sigma_p|$ **then**
    return σ_p
  else
    $lIndex \leftarrow uIndex$
    $uIndex, partialFlightDomains \leftarrow upperIndex(lIndex, \sigma_p, flightDomains)$
    $matrix \leftarrow product(partialFlightDomains.values())$
    $removeFlightDomains(flightDomains, \sigma_p(i) \forall i \in [uIndex, |\sigma_p|])$

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.

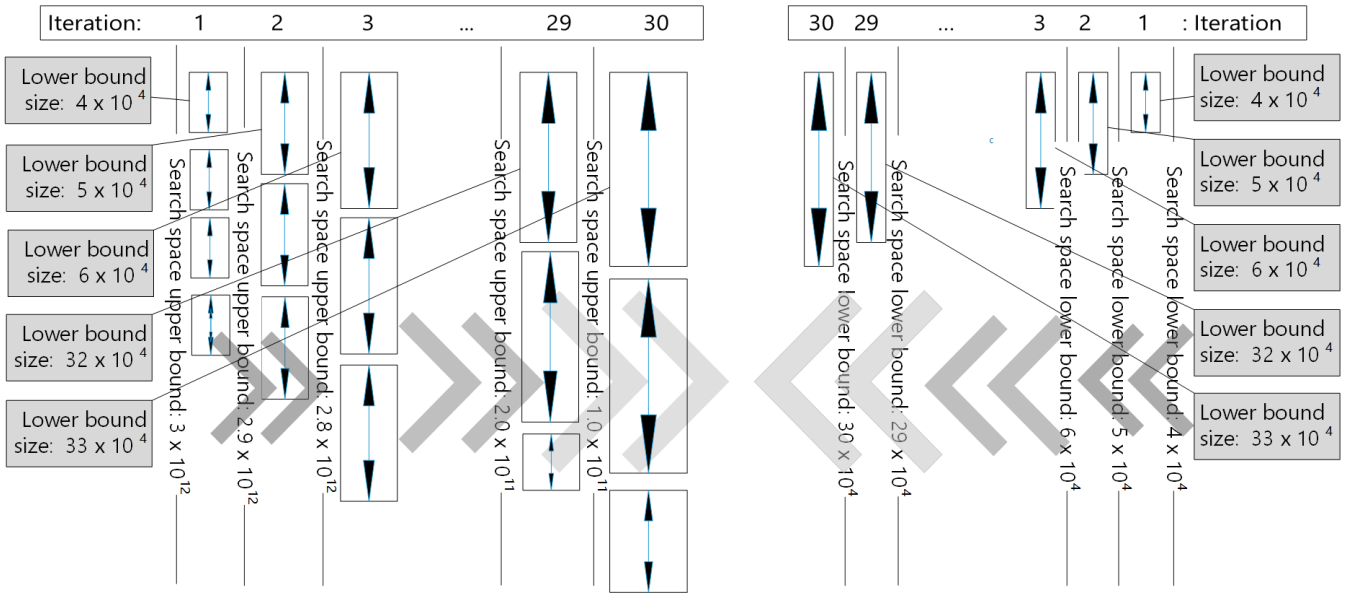

Figure 1 Pincer Heuristic

4.6. Backtracking Algorithm

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 and 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 solu-

tion. 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 ≠ {} do
2      if singletonList(0) =' dep' then
3          startInt ← 60 * int(singleton^d(0)/60)
4          endInt ← startInt + 60
5          origin ← singleton^o(0)
6          flight2Cancel ← solutionARP[(origin, startInt, endInt)]
7          airc2Cancel ← updateMulti(flight2Cancel, A, solutionARP])
8          aircraftSolList ← aircraftSolList − airc2Cancel
9          solutionARP.pop(airc2Cancel)
10         flightRanges, singletonList, totalCombos ← domainFlights(σ_p(i)∀i ∈
             [index, |σ_p|], A, index)
11     if singleton(0) =' arr' then
12         startInt ← 60 * int(singleton^a(0)/60)
13         endInt ← startInt + 60
14         destination ← singleton^f(0)
15         flight2Cancel ← solutionARP[(destination, startInt, endInt)]
16         airc2Cancel ← updateMulti(flight2Cancel, A, solutionARP)
17         aircraftSolList ← aircraftSolList − airc2Cancel
18         solutionARP.pop(airc2Cancel)
19         flightRanges, singletonList, totalCombos ← domainFlights(σ_p(i)∀i ∈
             [index, |σ_p|], A, index)
20 return solutionARP, aircraftSolList
```

4.7. Taxi Flights Algorithm

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 7 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 7 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

Algorithm 7: Taxi flights

Input: $\mathcal{B}, \Delta, originAirport, \sigma_p, \mathcal{A}, M$

Output: σ_p, M

$originSlots \leftarrow \mathcal{A}[originAirport]$ if $capDep > noDep$

for $\sigma_p(i)$ *in* σ_p **do**
  if $\sigma_p^o(i) = originAirport$ **then**
    return σ_p, M
  $\delta_{of} \leftarrow \Delta(originAirport, \sigma_p^o(i))$
  $originSlotsUpper \leftarrow originSlots$ if $endInt > \sigma_p^d(i) - \delta_{of} - t_{rp}$
  $originSlotsLower \leftarrow originSlots - originSlotsUpper - [\mathcal{B}_p^s, \mathcal{B}_p^e]$
  if $originSlotsLower = \{\}$ **then**
    cancel($\sigma_p(i)$)
    continue
  $destinationSlots \leftarrow \mathcal{A}[\sigma_p^o]$ if $capArr > noArr$
  $destinationSlotsUpper \leftarrow destinationSlots$ if $endInt > \sigma_p^d(i) - t_{rp}$
  $destinationSlotsLower \leftarrow destinationSlots - destinationSlotsUpper - [\mathcal{B}_p^s, \mathcal{B}_p^e]$
  if $destinationSlotsLower = \{\}$ **then**
    cancel($\sigma_p(i)$)
    continue
  $destIntervals \leftarrow \{\}$
  $i \leftarrow -1$
  $offset \leftarrow -1$
  for $x \in destinationSlotsLower$ **do**
    $destIntervals \leftarrow destIntervals \cup [x^s, x^e]$
  for $os(i)$ *in* $originSlotsLower$ **do**
    $obj \leftarrow interval(os^s, os^e) + distInitRot$
    $i, offset \leftarrow obj.findIntersection(destIntervals)$
    if $i \neq -1$ **then**
      $taxiFlight^o \leftarrow originAirport$
      $taxiFlight^d \leftarrow os['startInt'] + offset$
      $taxiFlight^f = \sigma_p^o(i)$
      $taxiFlight^a = taxiFlight^d + \delta_{of}$
      $taxiFlight['flight'] = M$
      $M \leftarrow M + 1$
      $\sigma_p \leftarrow \sigma_p + taxiFlight$
      return σ_p, M

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 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 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).

4.8. The CHARP

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 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.

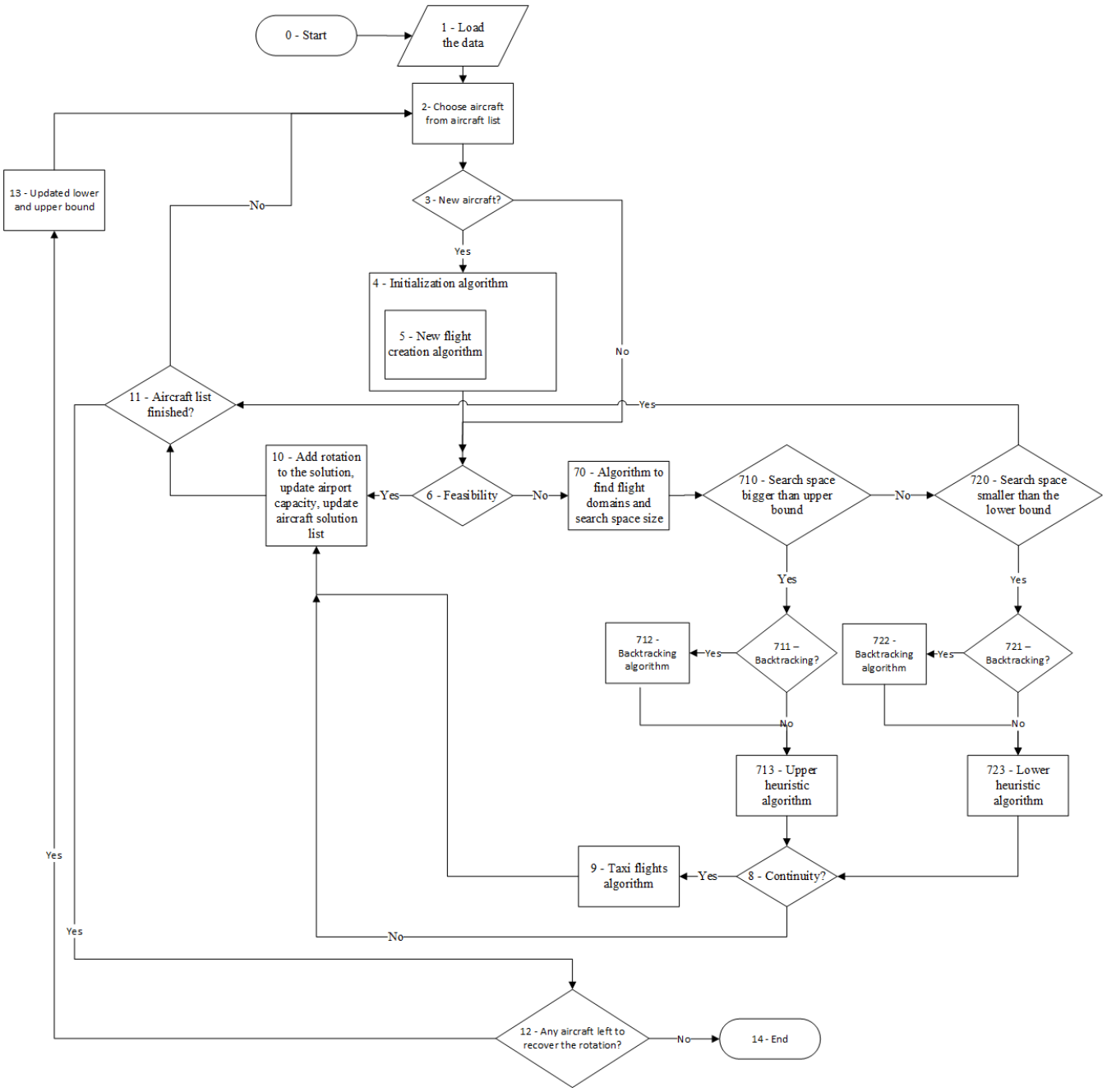

Figure 2 Constructive Heuristic

5. Computational Results

In this section, we aim at studying mainly the cost and the computing time of the CHARP for the overall thirty two data instances of the ROADEF 2009 Challenge. To obtain the computing time, we run all thirty two 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. To better understand the latter we will also compute the total amount of delay,

cancelled flights, new flights and, taxi flights. The remainder of this section consists of, Section 5.1 where we will demonstrate the impact of adding new flights and taxi flights. Since the heuristic performs a pincer movement in Section 5.2 we will determine which is the best speed by changing the decremental and incremental steps.

5.1. The Impact of New Flights and Taxi Flights

To study the impact of new flights we define the base scenario in table 1.

Table 1 Base Scenario

Domain step [min]	60
Upper bound	3.0×10^{12}
Upper bound step	1.0×10^{11}
Lower bound	4.0×10^{4}
Lower bound step	1.0×10^{4}

In figure 3, we can observe that the aggregate cost is lower for the heuristic that has new flights and taxi flights added, than for the one with none of them added. This result was expected since adding new flights or taxi flights consists of adding new arcs which in turn can increase the flow of the flight network. It is possible to observe a similar result for the maximum time to obtain a solution. The latter can be explained on a series of cascading effects: adding new flights will decrease the available airport capacity, which in turn will decrease the size of the domains hence resulting in the decrease of the size of search space to be traversed. We observe that extending the time window results increases the total amount of delay of the solution. As for the total number of cancellations we verify that it is lower for the heuristic that has new flights and taxi flights added, than for the one with none of them added. This observation is in line with the aggregate cost. Finally we point out that the total number of new flights is in the order of magnitude of 125 and the total number of taxi flights is in the order of magnitude of 200. Both values do not change substantially with the size of the time window.

5.2. The Impact of the Pincer Speed

To study the impact of the pincer speed we define the set of incremental and decremental steps in table 2.

In figure 4 we can observe that the minimum aggregate cost is obtained for speed 1 in the 960 minutes times window however the lowest maximum time is achieved for speed 0.5 and the 900 minutes time window. For the total amount of delay we observe that it increases with the time window and that the different speeds observe the same pattern of behaviour. In relation to the total number of flight cancellations we can observe that the values have a sharp decrease from

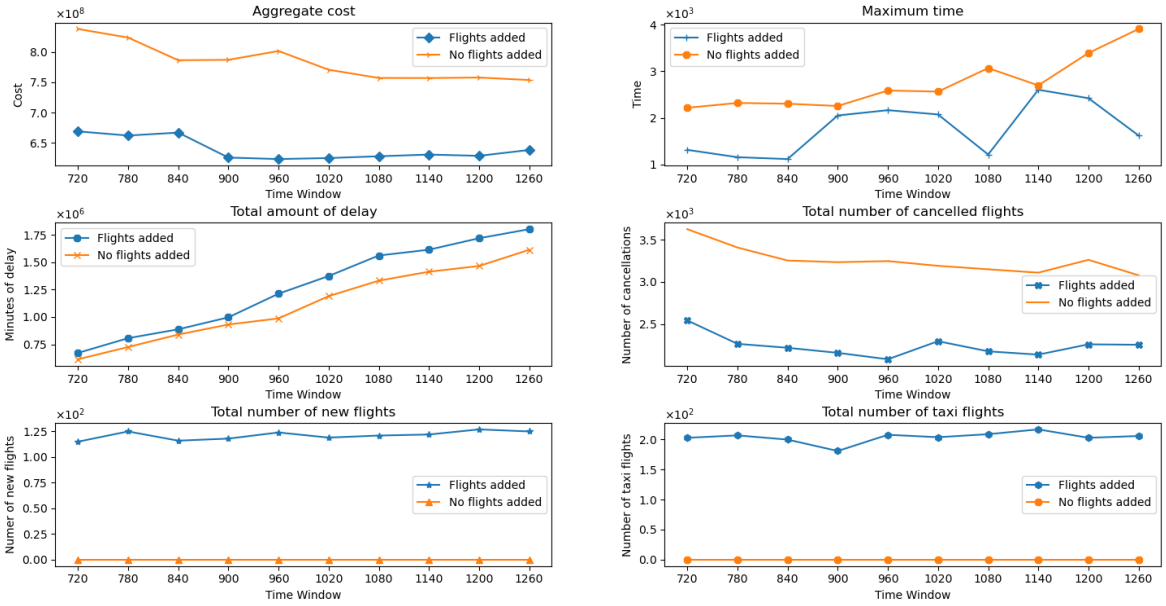

Figure 3 The Impact of New Flights and Taxi Flights

Table 2 Base Scenario

	Incremental step size	Decremental step size
Speed 0.5	0.5×10^4	0.5×10^{11}
Speed 1	1.0×10^4	1.0×10^{11}
Speed 2	2.0×10^4	2.0×10^{11}
Speed 3	3.0×10^4	3.0×10^{11}
Speed 4	4.0×10^4	4.0×10^{11}
Speed 5	5.0×10^4	5.0×10^{11}

720 to 780 minutes, after which they stabilize between a minimum of 2083 at speed 1 for the 960 minutes time window and a maximum of 2340 at speed 5 for the time window of 1200 minutes. We can conclude that the number of cancellations provides the solutions with the lowest cost, in fact this value is the same chose in (Bisaillon et al. 2011) for the Construction Phase algorithm. As for the number of new flights we can observe that they increase steadily with the size of the time window. Finally, in relation to the number of taxi flights other than a sharp decrease in value for the time window of 900 minutes, the remaining values distribute themselves between 195 at speed 4 for the 1200 minutes time window and 226 at speed 3 for the 1020 minutes time window.

To have a better understanding of the solution quality in figure 5 we draw the Pareto front considering the aggregate cost versus the maximum time for the different speeds and we conclude that increasing the speed generates lower quality solutions.

6. Conclusions

ARP 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. The CHARP

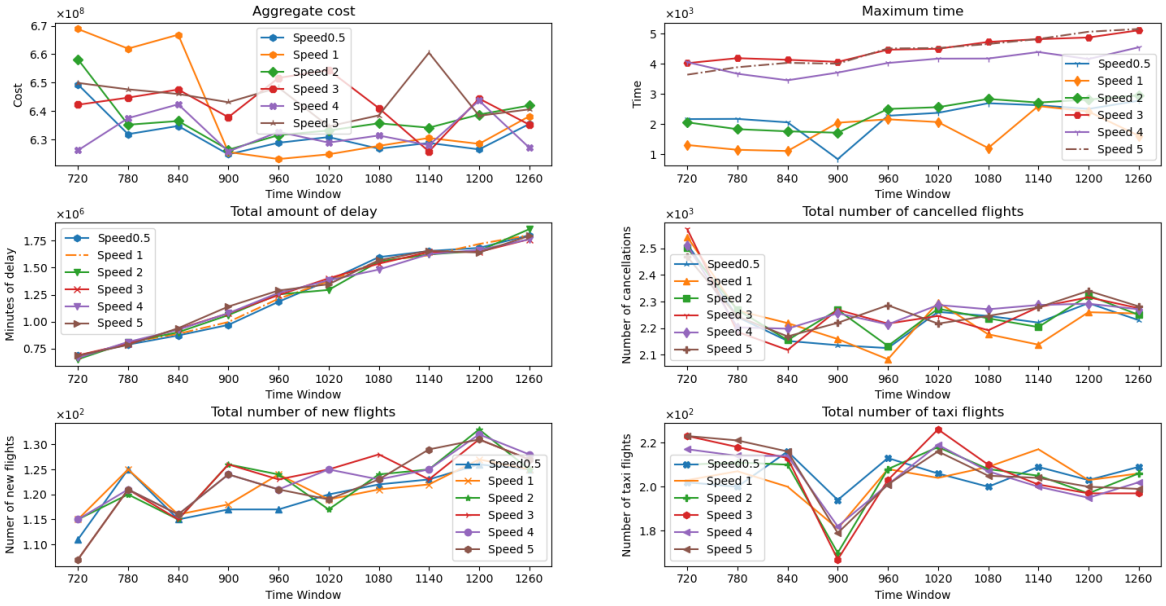

Figure 4 The Impact of the Pincer Speed

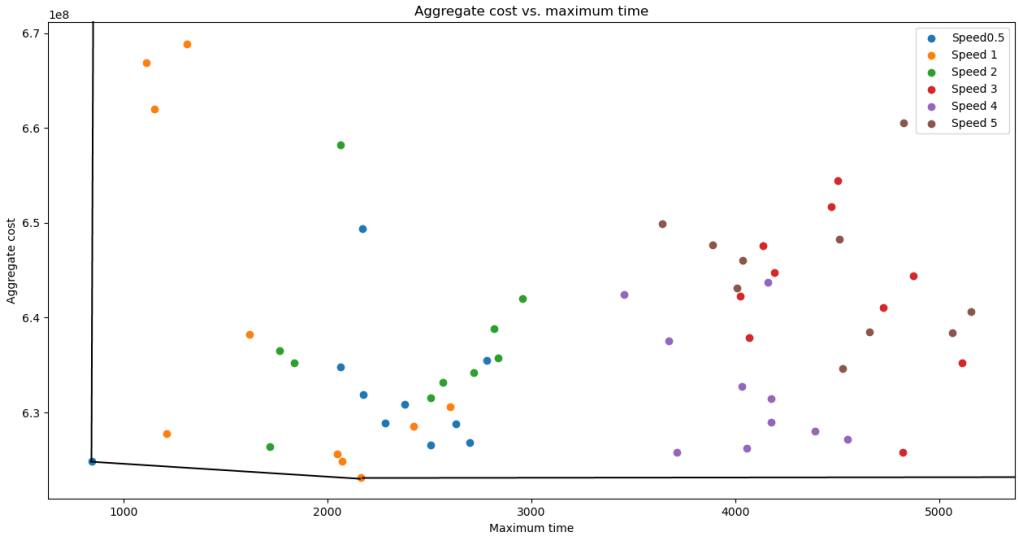

Figure 5 The Pareto Front

loads the respective data, introduces the disruptions and if necessary allocates times slots for new flights. We then select a sub-rotation starting in infeasible flight and proceed to recover it. The recovery procedure is done using an upper or a lower heuristic depending on the size of the search space. The recovery procedure will afterwards try to reconnect the first part of the rotation with the recovered one and for this purpose, if necessary, create taxi flights. In the impossibility of the latter the algorithm will cancel the flights in the recovered rotation until the continuity

constraint be satisfied.The data instance vary in size comprising a number of flights from 608 to 2178, aircraft from 85 to 618 and airport from 35 to 168. 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. In this work we achieve this goal with a novel heuristic based in CSP. One factor that facilitates these objectives is the greater availability of high-performance computing infrastructure. Our experiments were conducted in an Intel Xeon Gold CPU @ 2.3GHz box which allowed to run simultaneously 320 scenarios for every CHARP speed. We were able to find solutions, in less than 30 minutes, the best having a speed of 0.5 for a time window of 900 minutes.

References

AhmadBeygi S, Cohn A, Lapp M (2010) Decreasing airline delay propagation by re-allocating scheduled slack. *IIE Transactions* 42(7):478–489, URL `http://dx.doi.org/10.1080/07408170903468605`.

Ahmed MB, Ghroubi W, Haouari M, Sherali HD (2017) A hybrid optimization-simulation approach for robust weekly aircraft routing and retiming. *Transportation Research Part C: Emerging Technologies* 84:1–20, URL `http://dx.doi.org/10.1016/j.trc.2017.07.010`.

Aloulou MA, Haouari M, Mansour FZ (2013) A model for enhancing robustness of aircraft and passenger connections. *Transportation Research Part C: Emerging Technologies* 32:48–60, URL `http://dx.doi.org/10.1016/j.trc.2013.03.008`.

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.

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.

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.

Kohl N, Larsen A, Larsen J, Ross A, Tiourine S (2007) Airline disruption management—perspectives, experiences and outlook. *Journal of Air Transport Management* 13(3):149 – 162, ISSN 0969-6997.

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

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.