**Solution methods**

In this paper we consider three types of disruptions: flight, aircraft and airport disruptions. Flight disruptions refer to flight delays or cancellations. Aircraft disruptions consist of time periods where an aircraft is unavailable, which is mainly caused by mechanical failures or industrial action. Airport disruptions refer to airport closures usually incurred by severe weather conditions. When these disruptions happen, the flights assigned to the disrupted aircraft cannot be operated.

Given the original aircraft schedule and a set of disruptions, the objective of the ARP 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 respective rotation. 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.

New flight creation algorithm for aircraft breakdown

Description

New flight creation algorithm for flight disruption

The next step consists in traversing the rotation to find out if the following constraints are being respected during the recovery procedure:

Departure and arrival airport capacity

Transit or turnround time between consecutive flights

Aircraft arrive on time for maintenance.

If a rotation is found feasible the departure and arrival airport capacity is updated and the algorithm and moves to the next aircraft. Otherwise, the algorithm returns the index of the first infeasible flight.

Initialization algorithm

Description

Starting at the first infeasible flight in the rotation the algorithm initiates a procedure 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. This search is done in increments of 60 minutes and it will allow to delay the aircraft’s flight. 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.

Add delay increments and max. delay

                    if all([airportDic[origin][int(dep/60)]['noDep'] + delta <= airportDic[origin][int(dep/60)]['capDep'],

                        airportDic[destination][int(arr/60)]['noArr'] + delta <= airportDic[destination][int(arr/60)]['capArr']]):

                        domain.append(t)

Algorithm to find flight domains

Description

Each flight 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:

Dictionaries

Size: 81,920

A 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.

Matrix

From 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, and the lower heuristic loops through every row of the of the matrix in order to find the optimal solutions that minimizes the number of cancelled flights and the total amount of delay, hence the lower heuristic solves to optimality any infeasible rotation.

Algo for the lower heuristic

As for the upper bound the initialization starts at 3\*10^9, however the upper heuristic decomposes the rotation into sub-rotations with a with a search space size lower than 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

Algo for the upper heuristic

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 algorithm, on any iteration the 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 and 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 by 10^4 and decrements the upper bound by 99.99 \* 10^6. 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.

Fig. of the pincer heuristic

In CSP it is common to find variables that have domain size 1, such variables are 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 aircrafts 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.

Algo Backtracking

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. To overcome this situation the algorithm tries to create a taxi flight when possible or cancels the flight from the recovered rotation.

Algo Taxi flights

Finally when all the constraints are complied with and the rotation is feasible the algorithm accepts recovered rotation and updates the airport capacity

Integrating the Block Time Fuel model