Large Neighbourhood Search and Simulation for Disruption Management in the Airline Industry

Chapter · April 2015

DOI: 10.1007/978-3-319-15033-8_6

CITATIONS
7

READS
97

3 authors:

Daniel Guimarans
Monash University (Australia)
59 PUBLICATIONS **221** CITATIONS
SEE PROFILE

Pol Arias
Loughborough University
12 PUBLICATIONS **97** CITATIONS
SEE PROFILE

Miguel Mujica Mota
Amsterdam University of Applied Sciences
105 PUBLICATIONS **202** CITATIONS
SEE PROFILE

Large Neighbourhood Search and Simulation for Disruption Management in the Airline Industry

Daniel Guimarans, Pol Arias, and Miguel Mújica

Abstract The airline industry is one of the most affected by operational disruptions, defined as deviations from originally planned operations. Due to airlines network configuration, delays are rapidly propagated to connecting flights, substantially increasing unexpected costs for the airlines. The goal in these situations is therefore to minimise the impact of the disruption, reducing delays and the number of affected flights, crews and passengers. In this chapter, we describe a methodology that tackles the Stochastic Aircraft Recovery Problem, which considers the stochastic nature of air transportation systems. We define an optimisation approach based on the Large Neighbourhood Search metaheuristic, combined with simulation at different stages in order to ensure solutions' robustness. We test our approach on a set of instances with different characteristics, including some instances originating from real data provided by a Spanish airline. In all cases, our approach performs better than a deterministic approach when system's variability is considered.

1 Introduction

Operational disruptions are alterations of originally planned operations due to external events. The airline industry is notably one of the most affected industries regarding operational disruptions. The costs associated to them have gained more

Daniel Guimarans
Optimisation Research Group, National ICT Australia (NICTA), Sydney, Australia e-mail: daniel.guimarans@nicta.com.au

Pol Arias
Smart Logistics and Production Group, Internet Interdisciplinary Institute (IN3-UOC), Barcelona, Spain e-mail: pol.arias5@gmail.com

Miguel Mújica
Aviation Academy, Amsterdam University of Applied Sciences, Amsterdam, Netherlands e-mail: m.mujica.mota@hva.nl

and more importance with the increase of fuel costs and the punctuality policies that airlines have been forced to implement in order to maintain competitiveness [61]. Due to these and other emerging regulations that the aeronautical industry is facing nowadays [21], the optimisation of resources has become an important issue in the aeronautical agenda.

Flight plans are usually made several months prior to the actual day of operation. As a consequence, changes often occur in the period from the development of the schedule to the day of operation. Those changes may include unforeseen delays due to weather phenomena, air traffic control delays, reactionary delays, ground operations, etc. The fact that a single flight leg is, in general, a component of different flight schedules implies that a single perturbation may propagate to other elements in the network (Fig. 1). Thus, a delay usually leads to reactionary delays on several other flights. This effect is in many cases exacerbated by airlines network characteristics, which for most of the largest carriers relies on a hub-and-spoke configuration. Hence, a perturbation on the schedule in the hub may easily lead to many parts of the network being affected by the disruption.

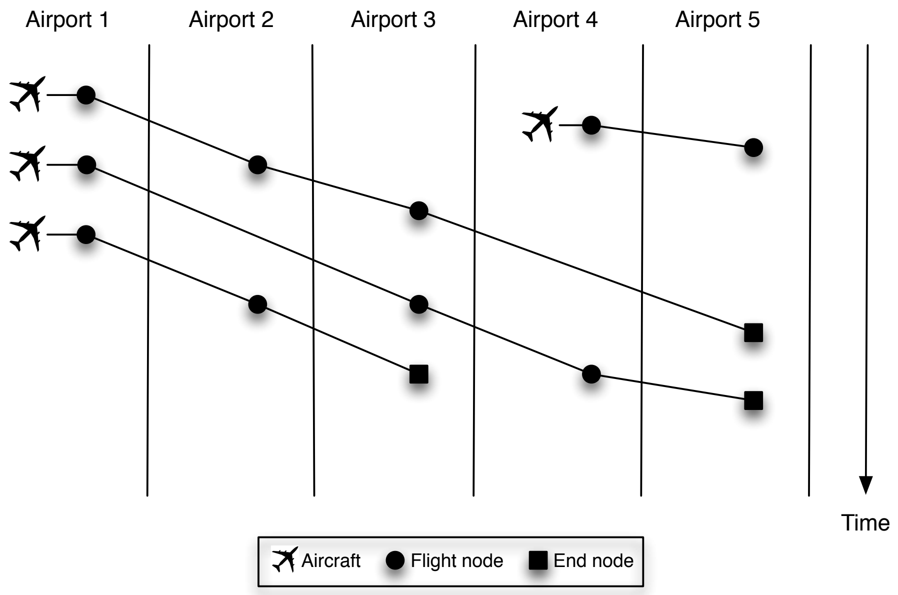

Fig. 1 Flight legs are, in general, a component of different flight schedules. Hence, a perturbation in one flight leg may propagate to other elements in the network, affecting aircraft, crew and passenger connections.

According to the Association of European Airlines [20], 22.4% of departures of intra-European flights were delayed more than 15 minutes in the first quarter of 2008, an increase from 20.5% in 2007. This figure is consistent with the observed average delay (22.74%) during the ten-year period 1998-2007. Breaking down the figures, the same study found that 7.3% of delayed departures were due to pre-flight preparation, i.e. the aircraft is not ready to leave on time because of late loading, crew availability, completion of paperwork, etc. Besides, 13.1% of delays were attributable to air traffic management (ATM) or airport management related issues. This figure increased from 11.5% in 2007 to 13.1% in 2008, suggesting that the European air space is getting more congested. Considering 27 major European airports, we observe an average of 23.1% of European departures delayed 15 minutes or more, with an average delay of 41.8 minutes. Arrival figures are similar, with an average of 25% of flights delayed 15 minutes or more, with an average delay of 41.3 minutes. Among major airports, London Heathrow is clearly the most affected, with 44.1% of European departures subject to delays of 15 minutes or more. Other studies [9, 48] suggest that USA figures are similar.

Estimating the cost airlines incur due to operational disruptions is difficult because of the many factors involved and some unquantifiable effects, e.g. passenger inconvenience. Shavell [48] reported that, for the year 1998, the total estimated costs of irregular operations incurred by the ten USA airlines that report performance data to the Department of Transportation was \$1.826 billion. Analysing this figure, \$858M were attributable to cancelled flights, \$909M to delays and \$59M to diversions, i.e. a flight that is directed to a different airport than the originally schedulled. From a case study of a heavy snowstorm in Boston, Shavell [48] concludes that the estimated cost per minute of delay over 15 minutes is \$13.35. In a more recent study [19], EUROCONTROL's Central Office for Delay Analysis (CODA) estimates the cost per minute of delay at €82 for delays in excess of 15 minutes. According to their statistics and extrapolating the percentage of delayed flights to all flights in Europe, CODA estimated the total cost of delays from all causes at more than €7 billion in 2008.

One of the major problems airlines face on a daily basis is that specific flight legs are not entirely predictable. It is normal that flights are subject to a certain level of variability in daily operations, due to the stochastic nature of air transport and the number of stakeholders involved. Moreover, due to network configurations, a late arrival of an aircraft often causes delays in the next departure of this aircraft and connecting flights. Traditionally, airlines cope with this problem in their schedules by introducing additional *buffer* time in between flights, aiming to absorbe potential delays in case they occur. However, buffer time increases operations cost significantly. EUROCONTROL estimates the cost of one minute of buffer time for an Airbus A320 at €49 per flight [18]. Hence, in the last years, airlines and researchers are addressing more efforts towards solving operational disruptions more efficiently (see Sect. 2). This way, airlines may configure tighter schedules and reduce operational costs, while still being able to respond to unforeseen events and disruptions. These problems make evident the need of decision support tools that help decision makers to cope with operative problems under urgent circumstances.

Among the different elements involved in a disrupted scenario (aircraft, crews and passengers), aircraft have received most attention from the research community (see Sect. 2), since it is normally considered the scarce resource and problem dimensions are relatively small. In general, the main objective is to restore the flight schedule as much as possible using the existing aircraft, i.e. minimise the number of cancellations and the total delay, in order to minimise the impact of the disruption. Given an original flight schedule and one or more disruptions (i.e. flight delays), the optimisation approach generates a solution by means of delaying flights, swapping aircraft to flight allocation, or even cancelling flights. Such plan considers all flights scheduled within a certain period of time by a given fleet, including the original departure, expected flight durations and connections between airports. This challenging problem is known as the Aircraft Recovery Problem (ARP) and regarded to be NP-Hard [8]. Introducing variability in the values associated to the problem, i.e. flights duration, turnaround times or delays, the Stochastic Aircraft Recovery Problem (SARP) arises. The SARP accounts for all characteristics inherited from the ARP, adding additional considerations present in realistic problems.

In this work, we propose an optimisation approach for the SARP that integrates simulation at different stages. On the one hand, optimisation-based methods have proved their efficiency to deal with operative problems in complex fields, e.g. logistics or manufacturing problems. However, many optimisation approaches lack the flexibility needed for tackling real operational problems. In most cases, the stochasticity inherent to these systems is not included in the developed models, thus reducing their applicability to real scenarios. On the other hand, simulation approaches have great flexibility and allow the modeller to face the problem under a different scope, including not only stochastic elements but also the dynamics of interest of the system. Nevertheless, in most simulation-based approaches the level of optimisation achieved depends on the number of scenarios evaluated, which in general is just a small fraction of the whole available configurations. Hence, optimality may not be guaranteed with the standalone simulation approach, likewise suitability is not ensured with the standalone optimisation approach. In the present approach, we combine both methodologies in order to obtain pseudo-optimal solutions which are robust enough to cope with system's stochasticity.

We present a methodology based on the *Large Neighbourhood Search* (LNS) metaheuristic (see Sect. 4.1). In our approach, we developed a *Constraint Programming* (CP) formulation to tackle the deterministic ARP (see Sect. 3.2). This model was combined with simulation and tested in previous works [6, 7]. In this case, we use the same model as a *repair* method in the LNS approach. It should be noticed that we do not consider cancellations in our approach. In these situations, connecting flights are generally cancelled so the aircraft may be assigned to a later flight from the same airport. In other cases, aircraft may be *ferried*, i.e. flying without passengers, to the destination airport in order to restore the original schedule. In both situations, we can represent a cancelled flight or aircraft shortage, e.g. due to temporary mechanical problems, as a long delay at flight's origin. We include simulation at different stages of the optimisation process, defining the SimLNS methodology (see Sect. 4). With this approach, we are able to increase solutions' robustness, as solutions are only accepted if they perform better than the incumbent in a variety of simulated scenarios. The proposed methodology has been assessed on a set of instances with different characteristics, some of them obtained from real data provided by a Spanish airline (see Sect. 5).

2 Literature Review

Traditionally, disruption management research is divided into aircraft recovery, crew recovery and passenger recovery. In this section we review previous academic research on aircraft recovery. For a more thorough review of academic research on different aspects of disruption management, including aircraft recovery, we refer the reader to the comprehensive reviews by Kohl et al. [32], Clausen et al. [14] and Le et al. [35].

The ARP has received most attention among operational recovery problems, due to aircraft are considered the scarce resource. In addition, rules applied for aircraft reallocation are often less complex than those governing crew or passenger recovery. However, in most cases crew recovery permits more flexibility by using standby crews at airline's bases. In the case of passenger recovery, it is not uncommon to reallocate heavily disrupted passengers in flights operated by other airlines. This solving flexibility is more difficult to achieve in aircraft recovery, since the number of spare aircraft is normally quite limited or inexistent. Furthermore, research on aircraft recovery to this date only deals with a single airline and does not support cooperation between different carriers.

Teodorović and Guberinić [50] are among the pioneers of the ARP. In their work, given one or more unavailable aircraft, their objective is to minimise the total passenger delay by allowing flight re-timing and aircraft swaps. The model is based on a network flow with side constraints, which is solved using a branch-and-bound method [34]. In a later work, Teodorović and Stojković [51] consider aircraft shortage and propose an improved approach. The authors solve the problem by a heuristic algorithm based on dynamic programming using an algorithm based on a lexicographic ordering of the flights. The constructed model allows cancellations, retiming and swaps. The main objective is to minimise the number of cancellations. If there are several solutions with the same number of cancelled flights, they use the total passenger delay as secondary objective.

The literature contains several works on different aspects of the ARP. Many of them are based on a multi-commodity flow problem solved on a time-band network. Jarrah et al. [30] develop two network flow models to cope with aircraft shortage. Both models –a delay model and a cancellation model– permit using standby aircraft. In both cases, the goal is to minimise the cost of the recovery, including not only cancellation and delay costs, but those associated to swapping or ferrying aircraft. The main drawback is that models are dissociated: one model handles retiming only, while the other handles cancellations only. Their work was deployed at United Airlines as part of their decision support system. Cao and Kanafani [12, 13] extended the delay model of Jarrah et al. [30] to include cancellations and multiple airports. However, Løve and Sørensen [36] proved that these models have serious deficiencies.

In a later work, Løve et al. [37] presented a heuristic for the ARP based on local search. The schedule is represented by the lines of work for each available aircraft. In order to solve the model, the consider cancellations, delays and aircraft reallocation, both within a single fleet or between fleets. The objective is to minimise the recovery costs related to delays, cancellations and swaps. It is even possible to assign costs on individual flights to weight the relative importance of different flights. The proposed approach was tested on the short-haul operation of British Airways (79 aircraft, 44 airports and 339 flights).

Granberg and Värbrand [24] proposed a mixed integer multi-commodity flow formulation with side constraints for the ARP, although they name the problem as *Flight Perturbation Problem*. They further reformulate the problem into a set packing model using the Dantzig-Wolfe decomposition [15]. Cancellations, delays and aircraft swaps are allowed in order to solve the perturbation. The authors propose two column generation strategies and test them on data from a Swedish domestic airline. Results show that the methods are capable of obtaining high quality solution, but running times increase drastically with instance size, e.g. 1139 seconds for one of the largest instances containing 215 flights.

Argüello et al. [4, 5] presented a method based on the Greedy Randomised Adaptive Search Procedure (GRASP) metaheuristic [43] to reschedule the aircraft routings if one or more aircraft are unavailable. The heuristic is capable of cancelling and retiming flights. As in the approach of Løve et al. [37], it also allows swaps between different fleet types. The goal is to produce a recovery plan so the original schedule is restored by the following day. The cost to be minimised includes measures of passenger inconvenience and lost flight revenue. Their approach was tested in a fleet of 16 aircraft, 42 flights and 13 airports, reporting results within 10% of optimality, according to the authors.

Yan and Yang [59] and Yan and Tu [58] developed four models to cope with temporary aircraft shortage. The models were specifically developed for small airlines. In the first model, it is possible to cancel flights to repair the disrupted schedule. In the second model, ferrying of spare aircraft is also considered together with flight cancellations. The third model considers cancelling and retiming flights. The last model incorporates all previous possibilities. In all models, swaps are allowed within a fleet. The objective in all models is to minimise the cost of repairing the schedule, including passenger revenue. The first two models are built as network flow models and can be solved to optimality very fast. The other two models contain side constraints and are solved using Lagrangian Relaxation and subgradient optimisation. In Yan and Tu [58], a multi-fleet version of these models is presented. In this case, a larger aircraft can be assigned to a flight that originally was serviced by a smaller aircraft. Yan and Young [60] also consider multiple aircraft types, but aircraft swaps between fleets are not allowed. The developed methods were tested on 534 different scenarios based on China Airlines data, solving all instances to optimality or at most 1% from optimality within 5.5 minutes. Yan and Lin [57] extend these models to solve the special situation when an airport is temporarily closed. The model presented allows swapping flights, retimings and cancellations, but not diverting flights. Therefore, flights arriving to or departing from the closed airport have to be cancelled or delayed.

Thengvall et al. [52] extend the models presented in [58, 59] by penalising deviations from the original schedule in the objective function. They use Linear Programming (LP) relaxation to solve the proposed network flow model with side constraints. In case an integer solution is not reached with this approach, the authors provide a rounding heuristic that finds feasible solutions within a small fraction of the optimum. Their approach is tested on data provided by Continental Airlines. In [53], the same authors extend their study by developing three multi-commodity network models for determining a recovery schedule following a hub closure.

Eggenberg et al. [16, 17] proposed a column generation scheme for the ARP with heterogeneous fleet, made of regular and reserve aircraft, and planned maintenance. The authors model the problem as a commodity flow problem on a dedicated network, one for each plane of the fleet. They report results on instances from Thomas Cook Airlines, ranging from 40 to 760 flights serviced with a fleet of 16 aircraft [17].

Wu and Le [54] also consider maintenance and regulations in their work. They base their model on flight strings, instead of individual flights, and transform it into a time-space network. The authors develop a heuristic, called *Iterative Tree Growing with Node Combination*, to solve this network model. Results are reported over a set of instances from China Airlines data consisting of 170 flights, 5 fleets, 35 aircraft and 51 airports.

Rosenberger et al. [45] were the first to use simulation when studying the ARP. The authors propose a problem decomposition, where the master problem is defined as a set partitioning problem (i.e. each flight is either cancelled or flown by an aircraft), and each subproblem is a route generation problem. The objective is to minimise the cost of cancellations and delays. In order to make the approach more computationally efficient, they define a heuristic to select only a subset of aircraft to be included in the set partitioning problem. The authors assess their results using the simulation environment *SimAir* [46]. They simulate 500 days of operations for 3 fleets, with datasets ranging from 32 to 96 aircraft and 139 to 407 flights. Nevertheless, Rosenberger et al. [45] do not use simulation in the optimisation phase in an integrated manner. Recovery procedures are invoked from *SimAir* any time a disruption is preventing the system to execute the flying schedule as planned. The disrupted scenario is then solved deterministically using data provided by the simulator, assessing the provided solution by resuming simulation with the recovery schedule.

In a more integrated approach, Wu [56] used simulation to calculate random ground operational delays and airborne delays in an airline network, instead of estimating delay propagation through the system. In a previous work [55], Wu showed that delays are inherent in airline operations due to stochastic delay causes. In [56], the author applies simulation to assess the robustness of airline schedules. This approach resembles the one proposed in our work. In a similar fashion, we use simulation to account for stochasticity in airborne and ground delays in order to obtain a more robust recovery plan (see Sect. 4).

3 Problem Formulation

The proposed formulation for the ARP is based on the *Constraint Programming* (CP) formalism. CP is a powerful paradigm for representing and solving combinatorial problems, whose nature provides easily adaptable problem representations. Moreover, constraints can be added or modified, even dynamically, without altering search procedures. A brief introduction to CP is provided in Sect. 3.1, whereas the proposed ARP formulation is described in Sect. 3.2.

3.1 Constraint Programming

Constraint Programming (CP) is a powerful paradigm for representing and solving a wide range of combinatorial problems [47]. In the last few decades, it has attracted much attention among researchers due to its flexibility and its potential for solving hard combinatorial problems in areas such as scheduling, planning, timetabling and routing. CP combines strong theoretical foundations (e.g. techniques originated in different areas such as Mathematics, Artificial Intelligence, and Operations Research) with a wide range of applications in the areas of modelling heterogeneous optimisation and satisfaction problems. Moreover, CP nature provides other important advantages such as fast program development, economic program maintenance and efficient runtime performance.

Problems are expressed in terms of three entities: *variables*, their corresponding *domains*, and *constraints* relating them. Constraints can be considered as the heart of CP. They are treated as logical relations among several *unknowns* (or *variables*), each taking a value from a set of accepted values called *domain*, which can be a range with lower and upper bounds or a discrete list of numbers. The representation of the problem, in terms of constraints, results in short and simple programs easily adaptable to future changing requirements.

Since CP is the study of computational systems based on constraints, its idea is to solve problems by stating constraints (requirements) about the problem area and, consequently, finding a solution satisfying all the constraints. This class of problems is usually termed *Constraint Satisfaction Problems* (CSP) and the core mechanism used in solving them is *constraint propagation*. Constraint propagation embeds any reasoning which consists in explicitly forbidding values, or combinations of values, for some variables of a problem because a given subset of its constraints cannot be satisfied otherwise [11]. In other words, constraint propagation is a way to produce the consequences of a decision. In general, when a variable belonging to a constraint is labelled, that value is propagated to the rest of variables involved in that constraint.

An important contribution of CP is to allow the end user to control the search. The topic of search comes from the heart of Artificial Intelligence, which has developed several algorithms to perform the search in a solution space. End user's search control is achieved by combining generic techniques, when the generation of the whole search tree is unfeasible, and problem-specific techniques, when there is an extra knowledge about special features of the problem. Thus, while mathematical programming is mainly based in the application of certain algorithms to a model, CP allows the user to take some decisions on the search stage like the order of instantiation of the variables and the order of selection of values from domains. This point represents one of the most important differences with Linear Programming (LP): when using LP, once the problem is modelled, the rest of the work is done by the solver. In the CP methodology, the order of variable labelling and value selection is essential to drive the search. However, it is important to notice that, although a search improved by these techniques can be useful to find a faster solution for a problem, it can significantly slow down the solution of a different problem. Depending on these choices the way decisions are made is totally different and the performance of the search algorithm can be highly affected.

Solutions to a CSP can be found by searching (systematically) through the possible assignments of values to variables, i.e. generating the whole search tree. From the theoretical point of view, solving a CSP is trivial using systematic exploration of the solution space. But that is not true from the practical point of view, where efficiency takes place. Search methods can be divided into two broad classes: those that traverse the space of partial solutions (or partial value assignments), and those which explore the space of complete value assignments (to all variables) stochastically.

The simplest algorithm that searches the space of complete labelings, is called *Generate-and-Test* (GT) [33]. The idea of GT is very simple: firstly, a complete labelling of variables is randomly generated and, consequently, if this labelling satisfies all the constraints then the solution is already found; otherwise, another labelling is tried. Its search space corresponds to the Cartesian product of all variables' domains. The GT algorithm is clearly a weak generic algorithm with poor efficiency for two reasons: it has a non-informed generator and there is a late discovery of inconsistence.

Backtracking (BT) [2] is a method used for solving CSPs by incrementally extending a partial solution that specifies consistent values for some of the variables, towards a complete solution, by repeatedly choosing a value for another variable consistent with the values in the current partial solution. BT is a merge of the generating and testing phases of GT. The variables are labelled sequentially and as soon as all the variables relevant to a constraint are instantiated, the validity of the constraint is checked. If a partial solution violates any of the constraints, backtracking is performed to the most recently instantiated variable that still has available alternatives. Clearly, whenever a partial instantiation violates a constraint, BT is able to eliminate a subspace from the Cartesian product of all variables' domains. Hence, BT is strictly better than GT. However, its running complexity for most non-trivial problems is still exponential.

BT still has as a major drawback the late detection of conflicts. *Consistency techniques* [11] are used to detect inconsistencies in partial solutions sooner in the search, and they are at the core of constraint propagation. These techniques are based on the idea of removing inconsistent values from variables' domains until a solution is found. It is very important to note that consistency techniques are deterministic. There exist several consistency techniques, but most of them are not complete [2].

For this reason, these techniques are rarely used alone to solve a CSP completely and normally are combined with search algorithms such as BT. Attention should be paid to the use of these consistency techniques. They provide a good mechanism to remove inconsistent values from variables' domains during search, but they often penalise with respect to efficiency terms. For this reason, they are often reduced to the most basic forms, i.e. *node-consistency* and *arc-consistency* [22].

To cope with *Constraint Optimisation Problems* (COP), one should take into account the cost function. The appropriate modification of the BT search schema is called *Branch-and-Bound* (BB) [34]. During the search, BB maintains the current best value of the cost function (*bound*) and, each time a solution with a smaller cost is found, its value is updated. There are many variations on the BB algorithm. One consideration is what to do after a solution with a new best cost is found. The simplest approach is to restart the computation with the *bound* variable initialised to this new best cost. A less naive approach is to continue the search for better solutions without restarting. In this case, the cost function upper bound is constrained to the *bound* variable value. Each time a solution with a new best cost is found, this cost is dynamically imposed through this constraint. The constraint propagation triggered by this constraint leads to a pruning of the search tree by identifying the nodes under which no solution with a smaller cost can be present.

Generic techniques for local search, such as *Genetic Algorithms* (GA) [42], *Simulated Annealing* (SA) [38] or *Tabu Search* (TS) [23], can also be used to aid CP to find quasi-optimal solutions when it is not feasible to generate the whole search tree (due to memory or CPU time problems). These methods are used when the size of the problem is huge and it is not possible to find the optimal solution. Usually, CP is used to find fast poor solutions which will be used as initial values for these techniques. A good solution is sought from these input values. If the best solution found by these techniques is not good enough, a new initial solution is generated by CP. To avoid the same values than in previous searches, either new constraints are added or some of the existing constraints are removed. Alternatively, CP may be embedded at different stages of the local search, either to quickly check feasibility [28], reduce neighbourhood size by using consistency techniques [27], or to repair partial solutions in *Large Neighbourhood Search* approaches (see Sect. 4)

3.2 Aircraft Recovery Problem Formulation

The proposed CP formulation for the ARP intends to enhance the use of constraint propagation, modelling the problem with two sets of variables: predecessors (P) and successors (S). These variables allow us modelling the same search space from two different perspectives, while redundant constraints propagate decisions made in any of the two sets to the other one. Thus, search is carried out simultaneously in both variable sets, increasing overall efficiency and speeding up problem solving. This formulation is inspired on the Vehicle Routing Problem formulation by Kilby and Shaw [39].

We consider a set of n flights and a fleet of m aircraft. Then, the variables used in this formulation are:

- $\Psi = \psi_1 ... \psi_n$ are the flights to be attended;
- $A = a_1 ... a_m$ are the available aircraft;
- $G = g_1 ... g_{n+2m}$ is the assignment set, with domain $G :: [1..m]$.

It should be noticed that there is one assignation for each flight and two special assignations per aircraft: the starting and ending airports for the aircraft. Thus, two subsets of G, F and L, are defined as the aircraft departure and arrival airports to ensure the closure of the cycle:

- $F = n+1 ... n+m$ is the set of first assignments;
- $L = n+m+1 ... n+2m$ is the set of last assignments.

Then, the predecessor and successor sets are defined as:

- $P = p_1 ... p_{n+m}$ is the predecessors set, with domain $P :: [1..n+m] :: (G-L)$;
- $S = s_1 ... s_{n+m}$ is the successors set, with domain $S :: [1..n, n+m+1..n+2m] :: (G-F)$.

A set of constraints is imposed to relate all the variables and define the problem. The predecessor and successor variables form a permutation of G and are therefore subject to the *difference constraints*.

$$p_i \neq p_j \quad \forall i,j \in G \wedge i < j \qquad s_i \neq s_j \quad \forall i,j \in G \wedge i < j \tag{1}$$

Equations (1) force predecessor and successor sets to contain no repetitions. Thus, one flight can have one and only one predecessor and successor. In practice, these constraints are implemented using the CP global constraint *alldifferent* [29] to enhance constraint propagation efficiency.

The successor variables are kept consistent with the predecessor variables via the following *coherence constraints*:

$$s_{p_i} = i \quad \forall i \in G-F \qquad p_{s_i} = i \quad \forall i \in G-L \tag{2}$$

Equations (2) connect the concepts successor and predecessor as follows: the former shows that i is the successor of its predecessor, and the latter indicates that i is the predecessor of its successor.

Along a set of connected flights, all assignations are performed by the same aircraft. This is maintained by the following *leg constraints*:

$$g_i = g_{p_i} \quad \forall i \in G-F \qquad g_i = g_{s_i} \quad \forall i \in G-L \tag{3}$$

Equations (3) are used to ensure that the aircraft assigned to i is the same as that assigned to its predecessor and successor.

Other sets of variables are defined to ensure the connections between origin and destination airports, as well as the times that aircraft are assigned to their flights:

- $O = o_1 ... o_n$ is the origin airport set;
- $D = d_1 ... d_n$ is the destination airport set;
- $\Delta = \delta_1 ... \delta_n$ is the flight duration list, including minimum turnaround times;
- $T = t_1 ... t_n$ is the departing times list, indicating the time when a flight departs;
- $\tau = \tau_1 ... \tau_n$ is the scheduled times list, indicating the time when a flight is originally scheduled to depart;
- $\Gamma = \gamma_1 ... \gamma_n$ is the list containing the initial delays that have disrupted the system;
- $\Lambda = \lambda_1 ... \lambda_n$ is the delays list, indicating the accumulated delays for each flight.

The actual departure time is calculated given the *departure time constraints*:

$$t_i \geq t_{p_i} + \delta_{p_i} \quad \forall i \in G - F \qquad t_i \leq t_{s_i} - \delta_i \quad \forall i \in G - L \tag{4}$$

Equations (4) bound the departure time of flight *i*. This time is, at least, the departure time plus duration time of its predecessor (δ_{p_i}). Equally, this time must be, at most, the departure time of its successor, minus the duration time of flight *i* (δ_i).

The connection between origin and destination airports is done by using the *connectivity constraints*:

$$o_i = d_{p_i} \quad \forall i \in G - F \qquad d_i = o_{s_i} \quad \forall i \in G - L \tag{5}$$

Equations (5) are used to narrow down the combinations of flights. The origin of flight *i* must be the destination of its predecessor. In the same way, the destination of flight *i* is the origin of its successor.

Equation (6) ensures that the departing time of flight *i* is greater than the scheduled time plus the initial delay.

$$t_i \geq \tau_i + \gamma_i \quad \forall i \in G - F - L \tag{6}$$

Equation (7) allows to calculate the total accumulated delay by obtaining the difference between the actual time of departure (t_i) and the scheduled time of departure (τ_i).

$$\lambda_i = t_i - \tau_i \quad \forall i \in G - F - L \tag{7}$$

Finally, the objective function (8) to be minimised is defined as the sum of accumulated delays for all flights.

$$\min \sum_{i=1}^{n} \lambda_i \tag{8}$$

4 SimLNS: Large Neighbourhood Search and Simulation

Aiming to solve the SARP, we propose an optimisation approach based on the *Large Neighbourhood Search* metaheuristic, which is described in Sect. 4.1. This local search method has proven to be specially successful when combined with CP. Therefore, we integrate our CP model for the ARP as part of the methodology, embedded in the so-called *operators* (see Sect. 4.2). In order to deal with the stochasticity present in the problem, we apply simulation to the obtained solutions in different phases of our approach (see Sect. 4.3). This way, we improve final solutions' robustness by only accepting those solutions which, on average, perform better than previous ones. Thus, each solution is evaluated in a set of simulated scenarios according to system's variability before being accepted or rejected. Rather than just testing the final solution, as most traditional approaches, we moved the evaluation to previous stages of the search aiming to detect earlier in the process undesired solutions' characteristics, e.g. extremely sensitive solutions due to connection tightness.

4.1 Large Neighbourhood Search

In *Large Neighbourhood Search* (LNS), proposed by Shaw [49], an initial solution is gradually improved by alternately destroying and repairing the solution. Over the years, LNS has proved to be competitive with other local search techniques, especially when combined with CP. It complements the CP framework as LNS benefits from improved propagation while CP benefits from this efficient, while simple, search framework [40]. A complete introduction to the subject can be found in [41].

The LNS metaheuristic belongs to the class of heuristics known as *Very Large Scale Neighbourhood search* (VLSN) algorithms [1]. A neighbourhood search algorithm is considered as belonging to the class of VLSN algorithms if the neighbourhood it searches grows exponentially with the instance size or if the neighbourhood is simply too large to be searched explicitly in practice. Although the concept of VLSN was not formalised until recently, algorithms based on similar principles have been used for decades [1].

All VLSN algorithms are based on the observation that searching a large neighbourhood results in finding local optima of high quality, and hence a VLSN algorithm may return better solutions. However, searching a large neighbourhood is time consuming, therefore various filtering techniques are used to limit the search. In VLSN algorithms, the neighbourhood is typically restricted to a subset of the solutions that can be searched efficiently.

Intuitively, searching a very large neighbourhood should lead to higher quality solutions than searching a small neighbourhood. Nevertheless, in practice, small neighbourhoods can provide similar or superior solution quality if embedded in a metaheuristic framework, because they typically can be searched more quickly. Large neighbourhoods generally lead to local solutions of better quality, but the search is more time-consuming. Thus, a natural idea is to gradually extend the size of the neighbourhood each time the search gets trapped in a local minimum (Fig. 2).

In the LNS metaheuristic, the neighbourhoods are implicitly defined by methods (often heuristics) which are used to destroy and repair an incumbent solution. A *destroy* method destructs part of the current solution while a *repair* method re-

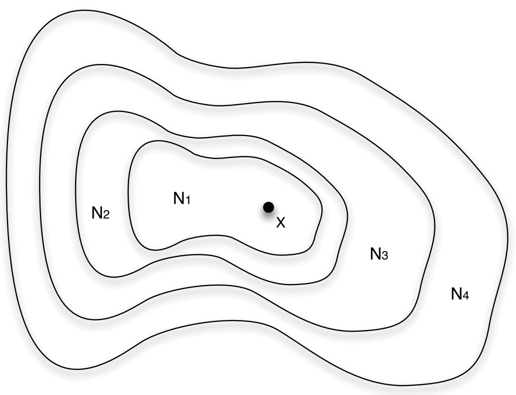

Fig. 2 Neighbourhood structure usually explored with Very Large Scale Neighbourhood (VLSN) search algorithms.

builds the destroyed solution. The destroy method typically contains an element of stochasticity such that different parts of the solution are destroyed in every invocation of the method. The neighbourhood $N(x)$ of a solution x is then defined as the set of solutions that can be reached by first applying the destroy method and then the repair method. Since the destroy method can destruct a large part of the solution, the neighbourhood contains a large amount of solutions, which explains the name of the heuristic. It should be noticed that the LNS metaheuristic does not search the entire neighbourhood of a solution, but merely samples it.

The steps of the LNS method are detailed in Alg. 1 and depicted in Fig. 3. Three variables are maintained by the algorithm: the variable x^b is the best solution observed so far during the search, x is the current solution, and x' is a temporary solution that can be discarded or promoted to the status of current solution. The function $d(\cdot)$ is the destroy method while $r(\cdot)$ is the repair method. More specifically, $d(x)$ returns a copy of x that is partially destroyed. Applying $r(\cdot)$ to a partly destroyed solution repairs it, i.e. it returns a feasible solution built from the destroyed one. Both destroy and repair methods can be implemented in different ways obeying different criteria. In step 4 the new solution is evaluated, and then the heuristic determines whether this solution should become the new current solution or whether it should be rejected. The *accept* function can be implemented in different ways. The simplest choice is to only accept improving solutions, as shown in Fig. 3. In this case, x^b corresponds to the current solution x at any time and steps 4 and 6 in Alg. 1 are merged. However, some works propose an acceptance criteria borrowed from SA [44], that is, accepting solutions that may be worse than the incumbent aiming to diversify the search.

The destroy method is an important part of the LNS heuristic. The most important choice when implementing the destroy method is the degree of destruction: if only a small part of the solution is destroyed then the heuristic may have troubles exploring the search space as the effect of a large neighbourhood is lost. If a very large part of the solution is destroyed, then the LNS heuristic almost degrades into repeated re-optimisation or a multi-start process. This can be time consuming

Algorithm 1: Large Neighbourhood Search (LNS)

```
1 $x^b \leftarrow$ find an initial solution $x$
2 repeat
3     $x' = r(d(x))$
4     if $accept(x', x)$ then
5         $x \leftarrow x'$
6     if $f(x') < f(x^b)$ then
7         $x^b \leftarrow x'$
8 until stopping condition is met
9 return $x^b$
```

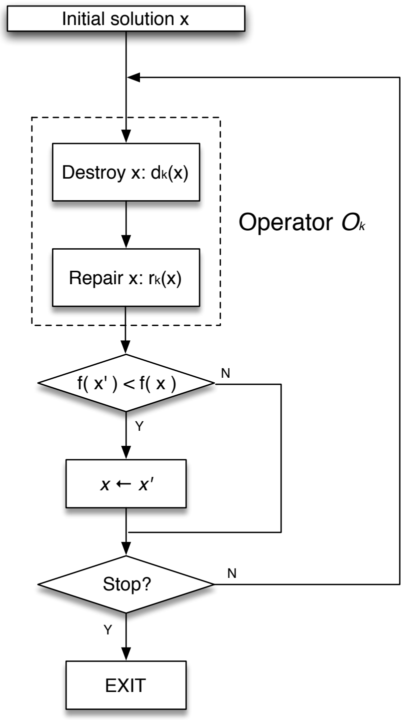

Fig. 3 Large Neighbourhood Search (LNS)

or yield poor quality solutions dependent on how the partial solution is repaired. Shaw [49] proposed to gradually increase the degree of destruction, while Ropke and Pisinger [44] choose the degree of destruction randomly at each iteration from a specific range dependent on the instance size. The destroy method must also be chosen such that the entire search space can be reached, or at least the interesting part of the search space where the global optimum is expected to be found. Therefore, it cannot focus on always destroying a particular component of the solution but must be possible to destroy every part of the solution.

Diversification and intensification for the destroy methods can be accomplished as follows: to diversify the search, one may randomly select the parts of the solution that should be destroyed (*random destroy* method). To intensify the search, one may try to remove a number of "critical" variables, i.e. variables having a large cost or variables spoiling the current structure of the solution (*worst destroy* or *critical destroy*, respectively). One may also choose a number of related variables that are easy to interchange while maintaining feasibility of the solution (*related destroy* method). Finally, one may use *history based destroy*, where a number of variables are chosen according to some historical information.

Choosing the repair method permits much more freedom when implementing a LNS heuristic. A first decision is whether the repair method should be optimal, in the sense that the best possible full solution is constructed from the partial solution, or whether it should be a heuristic, assuming that one is satisfied with a good solution constructed from the partial solution. An optimal repair operation will be slower than a heuristic one, but may potentially lead to high quality solutions in a few iterations. However, from a diversification point of view, an optimal repair operation may not be attractive: only improving or identical-cost solutions will be produced. Therefore, it can be difficult to leave valleys in the search space unless a significant part of the solution is destroyed at each iteration.

Finally, several destroy and repair methods may be combined to explore multiple neighbourhoods within the same search. Neighbourhood structures may be nested or cover different regions of the search space. In general, these neighbourhoods are explored in a systematic fashion, i.e. switching to the next neighbourhood whenever the current solution is not improved, or using different strategies to enhance the search, such as *Variable Neighbourhood Search* [26]. A more sophisticated LNS variant is the *Adaptive Large Neighbourhood Search* (ALNS) heuristic proposed by Ropke and Pisinger [44]. In this case, each destroy/repair method is assigned a weight that controls how often the particular method is attempted during the search. The weights are adjusted dynamically as the search progresses depending on the performance of each neighbourhood, so that the heuristic adapts to the instance at hand and to the state of the search.

4.2 LNS Operators for the ARP

In LNS, neighbourhoods are implicitly defined by the *destroy* and *repair* operators. The destroy method typically contains an element of stochasticity such that different parts of the solution are destroyed in every invocation of the method. Nevertheless, deterministic destroy methods can also be implemented.

As introduced in Sect. 3.1, CP search methods are mainly based on assigning values to variables, in such a way that constraints are satisfied and other variables' domains are reduced to their compatible values through constraint propagation. Therefore, CP-based destroy and repair methods will *unassign* and *assign* values to variables, respectively, at different stages of the search. It can be inferred then that a solution is a complete assignment (or complete *labelling*) to the variables of the problem, in such a way that all constraints are satisfied at once.

A CP-based destroy method unassigns some values from a solution, destroying it partially. For the ARP, we have defined two destroy methods, based on the idea of extending the size of neighbourhoods to be explored. In the *1-airport destroy method*, all aircraft-to-flight allocations are unassigned for all flights departing one airport (Fig. 4). As we are using the formulation introduced in Sect. 3.2, where redundant sets for *predecessor* and *successor* variables are used in order to enhance constraint propagation, aircraft allocations for previous and next flights should also be removed. Otherwise, the only feasible solution contained in the neighbourhood would be the preceding solution. The exploration of this neighbourhood permits swapping aircraft and delaying flights in one airport. We systematically use this operator to explore the corresponding neighbourhood for every airport present in the instance at hand. We only switch to the next operator with a higher degree of destruction when all airports have been explored and no improvement is found.

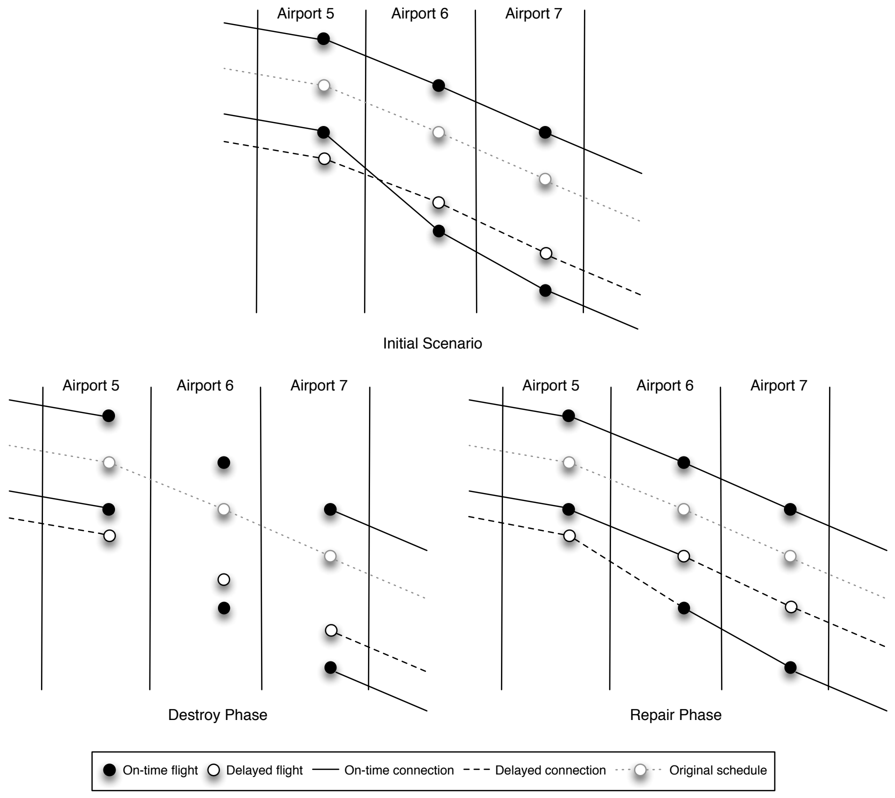

Fig. 4 Destroy/Repair phases of the 1-airport operator.

The second destroy method is based on the same principle of removing aircraft-to-flight allocations for particular airports. In this case, we unassign all variables corresponding to flights departing from three consecutive airports (*3-airport destroy method*). Therefore, we extend the size of the previously described neighbourhood by considering allocations of connected flight legs, rather than individual flights. In this case, the repair phase of the operator has more freedom to adjust flight departures (i.e. delaying flights) and swapping aircraft in particular legs.

It should be remarked that other destroy methods were considered. In particular, we explored neighbourhoods where the solution was partially destroyed for two consecutive airports. However, we found that all improvements obtained by means of this operator were also achieved and often exceeded by the 3-airport destroy method. All improvements obtained with the 1-airport operator can also be attained with the 3-airport operator, but the time required to explore the latter is higher. Therefore, we use 1-airport neighbourhoods to lead the algorithm to better solutions quicker. Starting closer to a local optimum reduces the search space for 3-airport operators, since previously explored feasible non-improving solutions are discarded. This way, these neighbourhoods can still be explored efficiently while reducing the required time. In terms of computational burden, we determined that larger neighbourhoods increase the required time excessively, due to the exact repair method we use.

We have chosen a CP-based repair method according to the formulation presented in Sect. 3.2. We use the same repair method for both 1-airport and 3-airport operators. Concretely, we apply a BB method (see Sect. 3.1) to repair the partially destroyed solution. Although slower than heuristic methods, we benefit from high quality solutions, while not being penalised with an excessive computational time by considering reasonably sized neighbourhoods. During search, the upper bound is set to the total delay of the best solution found so far. Aircraft-to-flight allocations may be forbidden if they take the lower bound on the total delay over the defined upper bound. We form the lower bound as the current total delay of the partial solution constructed during search, i.e. the lower bound is not computed separately by any other method. This makes the repair method faster, but the search tree is larger than it would be if we calculated an accurate lower bound. In its simplest form, the BB search explores the whole tree for the re-allocation of all flights to aircraft and re-scheduling all flights within their feasible time ranges in order to minimise the total delay.

4.3 Using Simulation: SimLNS

Large Neighbourhood Search has proved to be an efficient metaheuristic to deal with complex combinatorial optimisation problems [41]. However, LNS is designed to provide high-quality solutions under deterministic scenarios. In some real-life problems, like in the case of the ARP and many air transportation applications, uncertainty is present. In these cases, a deterministic approach may not be accurate enough, since it does not reflect the real stochastic nature of the system. Therefore, it is necessary to extend the deterministic framework to account for the variability of the system. A natural way is to integrate simulation within the optimisation process to cope with such stochastic combinatorial optimisation problems, e.g. the SARP.

Traditional simulation approaches allow the user to model the stochastic behaviour of the system, as well as all interactions between different elements. Nevertheless, in most simulation-based approaches the level of optimisation achieved depends on the number of evaluated scenarios. In general, this number is a small fraction of available configurations, not guaranteeing optimality and providing poor feedback regarding solution's quality. On the other hand, deterministic optimisation approaches do not take into account the uncertainty present in the system. Albeit using simulation to check the behaviour of different solutions is an extended practice, the decision on how to use the information provided by simulation is normally left to the final user. An example of this traditional approach applied to the ARP can be found in [45].

We propose two extensions of the LNS metaheuristic that integrates simulation at different stages of the search. The proposed approaches are similar to the *Iterated Local Search* extensions introduced by Grasas et al. [25]. They all fall within the *SimHeuristics* category [31].

In our first approach, which we call *SimLoop*, we include simulation as the final step to assess solution's robustness (Fig. 5). Up to this stage, the algorithm only accepts solutions that reduce the total delay with respect to the previous solution. Hence, either the initial solution or a solution with minor total delay is returned by the LNS framework previous to the simulation stage. However, this solution may not fulfil other desired characteristics, e.g. number of swaps, or may present an unacceptable degree of variance. We use simulation to test these attributes and provide feedback to the optimisation schema, extending the application of traditional simulation approaches.

Different criteria can be considered to determine whether a solution is robust or not. First, a solution may be considered to be robust if the standard deviation of the simulated solutions is proportional to the variation of the used probabilistic distributions and its expected propagation due to problems size. Second, a solution may be considered robust if the gap between the average of the simulated solutions and the deterministic solution falls within a tolerance interval. Third, we may define the criterion as the number of solutions whose gap to the deterministic solution is smaller than a given value. Finally, operational considerations such as the number of swapped flights / aircraft assignments may be introduced. In the practical case presented in this work, we use the first criterion to determine the robustness of the obtained solutions.

The proposed methodology for the SARP, which integrates SimLoop as the optimisation method, is structured in the following steps:

1. The stochastic problem is simplified to a deterministic instance by using the average values of the adjusted probability distributions of the different processes.
2. As the original flight schedule is known, we compute the total delay (the objective function) associated to this solution. This provides an initial value for the total delay, which is used as the initial upper bound for the objective function in

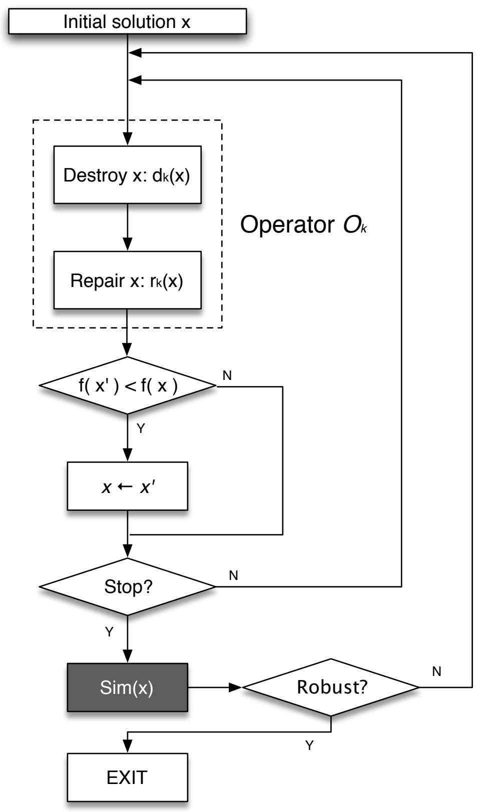

Fig. 5 SimLoop: Large Neighbourhood Search schema including simulation as the final step to assess solution's feasibility. Final solution's robustness is tested in this final step. In case the solution does not meet robustness criteria, results obtained from this process are used as feedback for further iterations of the local search process.

the local search process. As the original flight schedule is known to be feasible, we use it as the initial solution for our method. Intuitively, starting from the original schedule and applying local changes leads to solutions with a smaller number of swaps, a desired characteristic for most airlines.

3. A deterministic LNS framework is then used as a local search process to improve the initial solution, allowing flights to be delayed and enabling swaps. An improved flight schedule reducing the total delay is found as the result of this step.
4. The optimised solution is then checked using simulation to verify its robustness: a set of stochastic instances is generated using the probability distributions for the processes, namely flight and turnaround times. Maintaining the improved flight schedule returned by the LNS, we compute the total delay for each instance. This way, a single solution is evaluated in different scenarios. The results are analysed in order to determine the level of robustness of the obtained solution. If the solution is not robust, its objective function value is used as a lower bound and the optimisation process is repeated, i.e. the LNS is re-launched with a new lower bound. This way a worst solution may be found but with better robust-

ness characteristics. Otherwise, the solution is accepted and the algorithm ends. Thus, the methodology returns either the optimal deterministic solution, if it is robust enough, or a quasi-optimal one whose properties are more suitable for the stochastic problem at hand.

In our second approach, which we call *SimLNS*, we integrate simulation at an earlier stage in the process, as depicted in Fig. 6. In this case, simulation is embedded in the acceptance criterion instead of being used to evaluate the final result. The solution obtained after applying destroy and repair methods is then tested in a set of scenarios before deciding wether is accepted or discarded. Simulation results may be used to evaluate the solution in different ways. The most common approach consists of accepting solutions which, on average, are better than the incumbent. Hence, we are only accepting solutions whose average behaviour is better than previous solutions. A different criterion is borrowed from robust optimisation approaches [10], where we only accept solutions whose simulated worst case improves previous solution's worst case. Thus, we aim at accepting solutions able to perform reasonably well under challenging conditions.

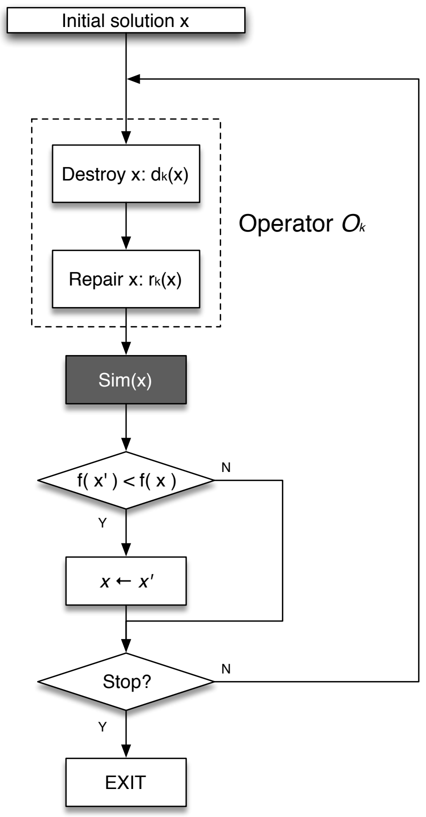

Fig. 6 SimLNS: Large Neighbourhood Search schema including simulation as part of the acceptance criterion. The solution obtained by operator O_k is accepted or discarded according to results obtained from its simulation.

By moving the simulation step to the acceptance criterion, our goal is to detect earlier in the process solutions with undesired attributes, e.g. extremely variable solutions due to tight connection times. If a solution has a lower deterministic total delay, but on average presents a worse behaviour due to solution's variability, it may be rejected and the local search process may proceed to evaluate the next neighbourhood. Using a more traditional approach or the SimLoop method, this solution would be accepted and would not be tested until the end of the local search process, therefore detecting late its unwanted aspects. On the other hand, the approach adopted in the SimLNS increases substantially the number of solutions to be evaluated. Although simulating a higher number of scenarios may increase the reliability of the final solution, so does the required computational time. Therefore, we need to find a trade-off between algorithm's execution time and the number of scenarios per solution to be tested.

5 Application

We have tested the methodologies described in the previous section on a set of instances with different characteristics. In all cases, tests have been done in a personal computer with an Intel Core i7 processor at 2.9GHz and 8Gb of RAM, running OS X 10.9. The different SimLNS variants have been implemented in Java, whereas the CP model (see Sect. 3.2) has been coded using the ECLiPSe 6.0 platform [3]. Simulation processes have also been implemented in Java.

Since most articles devoted to the ARP deal with specific instances (see Sect. 2), there are no accepted benchmarks for the ARP. Thus, we had to define a set of instances to assess our proposed methodologies. We have generated two separate sets with different attributes.

The first set contains scenarios with dense networks, i.e. networks with a higher density of flights and a larger connectivity degree for airports. Since we consider a dense network, the number of feasible swaps at each airport increases and so does the size of the neighbourhoods to be explored. These instances are purely theoretical and their goal is to push our methodology in more challenging and difficult scenarios. Details of the generated theoretical instances are as follow:

- Scenario 50 (denoted *50_*): consists of 49 flights, 3 airports, and 8 aircraft. In this scenario all the airports have approximately the same number of flights.
- Scenario 100 (denoted *100_*): consists of 98 flights, 6 airports, and 16 aircraft. Again, airports have approximately the same number of flights. In both scenarios 50 and 100 there is no visible hub.
- Scenario 200 (denoted *200_*): consists of 196 flights, 11 airports, and 32 aircraft. In this scenario, airport 1 has nearly twice the number of flights than other airports, therefore behaving like a hub. It also gives more opportunities for swapping aircraft.
- Scenario 300 (denoted *300_*): consists of 294 flights, 17 airports, and 48 aircraft. Again, we use the first airport as a hub to get more swapping opportunities.

The second set contains instances derived from real data provided by a Spanish airline (due to confidentiality agreements we cannot disclose the name of the airline). This airline relies on a hub-and-spoke network configuration. This kind of network provides fewer opportunities to swap aircraft between flights outside hub airports and generally propagates delays much faster to different parts of the network. We have generated several disrupted scenarios of different sizes from the original flight data provided by the airline. To keep consistency among instances, all of them have similar flights per airport and flights per aircraft ratios. Further details of real scenarios characteristics are described next:

- Real scenario 50 (denoted *real_50_*): consists of 49 flights, 17 airports, and 9 aircraft. In this scenario Madrid's airport works as a hub.
- Real scenario 100 (denoted *real_100_*): consists of 110 flights, 23 airports, and 23 aircraft. Madrid's airport works as a hub.
- Real scenario 150 (denoted *real_150_*): consists of 163 flights, 35 airports, and 40 aircraft. This scenario is a whole day of operation for the airline. Again, Madrid's airport works as a hub.

In the real scenarios, Madrid's airport is the main hub. Two more airports constitute secondary hubs for the airline: Palma de Mallorca and Barcelona. As it can be seen, the density of the real scenarios is substantially smaller than the theoretical scenarios, i.e. for a similar number of flights, the number of airports and aircraft is significantly higher for real scenarios.

For each described scenario, we generate four instances with different degrees of disruption. To do so, we introduce delays at selected flights and airports. These delays range from 30 to 120 minutes, increased in 30-minute intervals. The size of disruptions is denoted as a suffix in scenario's name, e.g. *50_30* corresponds to an instance of scenario 50 with 30-minute delays in some flights. In total, we generated 28 instances corresponding to disrupted scenarios: 16 with a dense network and 12 based on real data with a hub-and-spoke configuration.

Results for all instances are presented in Table 1. It shows the obtained total delay and computational times for all described methods: (i) deterministic LNS approach (*Det.*), simulating the best solution in the final step but not providing feedback to the optimisation process; (ii) SimLoop approach (*SimLoop*), using simulation to get feedback on solution's robustness; (iii) SimLNS approach (*SimLNS*), which uses simulation in the acceptance criterion optimising the average total delay; (iv) SimLNS approach optimising the worst solution (*SimLNS-W*), instead of the average total delay. Table 1 also contains the total delay of the original flight schedule (*Orig.*).

The original schedule is used as the initial solution for all approaches. Starting from the original schedule and performing local moves may lead to solutions with a minor total delay and few swaps. If a construction heuristic is used to obtain the initial solution, the algorithm may start at a region of the search space far from the original schedule. Although the obtained final solution may be better in terms of total delay, it may imply a large number of swaps regarding the original schedule, an unacceptable characteristic for most airlines.

Table 1 Results for 28 tested instances. Total delay (in minutes) and computational time are reported for the original schedule and solutions obtained by means of the different described methodologies.

Instance	Orig.	Det.	CPU (s)	SimLoop	CPU (s)	SimLNS	CPU (s)	SimLNS-W	CPU (s)
50_30	228.2	213.2	0.395	213.4	0.398	210.3	0.435	213.5	0.817
50_60	718.6	652.1	0.572	653.0	0.570	635.9	1.692	642.6	1.178
50_90	1458.4	1346.4	0.640	1395.1	0.624	1360.4	0.659	1369.9	1.270
50_120	2450.9	2346.1	0.623	2335.5	0.618	2290.4	0.871	2306.9	0.652
100_30	264.0	255.9	11.656	253.5	11.520	244.3	42.764	250.2	11.931
100_60	738.4	703.6	13.175	715.0	12.710	687.0	12.977	685.8	12.856
100_90	1577.3	1503.4	14.656	1513.5	13.752	1460.2	14.321	1460.1	14.312
100_120	2577.0	2476.9	14.216	2451.7	13.844	2430.5	27.908	2452.6	27.818
200_30	359.2	338.3	56.234	334.3	58.060	319.6	123.367	336.1	59.226
200_90	1692.5	1484.1	136.920	1470.4	145.809	1465.8	262.170	1478.2	130.145
200_60	959.9	813.6	68.088	815.0	70.195	792.1	69.624	809.1	87.114
200_120	2401.8	2023.1	204.585	2004.6	191.781	1993.5	340.803	2005.3	178.114
300_30	421.6	405.3	97.388	402.3	99.915	396.2	96.324	399.5	97.419
300_60	1010.6	950.1	101.819	935.0	104.345	922.1	282.811	931.7	100.527
300_90	1816.3	1649.4	97.697	1648.5	97.772	1609.1	102.590	1658.5	103.085
300_120	2605.4	2425.0	94.475	2417.6	96.248	2391.2	205.514	2427.6	100.611
real_50_30	120.6	121.0	6.618	121.5	6.518	120.1	6.748	120.0	6.610
real_50_60	241.2	241.2	6.411	241.9	6.616	240.1	6.364	240.3	6.828
real_50_90	385.4	387.0	7.231	385.9	8.510	383.5	7.318	384.2	6.971
real_50_120	535.8	536.6	6.642	536.2	6.716	533.5	13.347	533.9	7.017
real_100_30	450.6	451.3	8.386	451.5	8.356	450.1	8.198	450.4	8.086
real_100_60	906.0	907.3	8.416	905.4	8.446	904.0	8.433	903.9	8.133
real_100_90	1505.4	1494.7	10.722	1499.8	10.159	1490.1	30.471	1495.1	10.293
real_100_120	2322.2	2292.0	10.806	2290.2	10.931	2285.1	24.084	2285.7	21.691
real_150_30	98.4	97.9	36.160	102.5	38.536	96.3	81.135	98.9	37.036
real_150_60	196.4	194.2	37.341	193.1	36.820	188.4	42.760	188.4	46.805
real_150_90	336.2	309.5	55.739	304.8	43.308	303.6	85.841	304.0	87.155
real_150_120	513.5	425.2	38.368	427.2	40.655	424.0	226.505	425.4	38.444

In all approaches, we run 20 tests per solution whenever the simulation process is called. This means that the total number of run simulations may differ depending on the applied methodology. Probability distributions for flight durations and turnaround times are adjusted to reflect a similar behaviour to reported observations [18]. For the SimLoop approach, a 5% limit on the standard deviation has been imposed to consider a solution as robust.

As for operators, instances *100*, *200* and *300* are solved using only the 1-airport operator. Utilising the 3-airport operator for these instances increases the computational time dramatically. On the other hand, instances *50* and all real scenarios are solved using both 1- and 3-airport operators.

In general, we observe that SimLNS performs better than other approaches. For most instances, the largest reduction on total delay is achieved by means of this methodology, although computational time is slightly increased due to a higher number of simulations is required. These results are corroborated in Fig. 7. In this figure, the relative reduction of the total delay regarding the original schedule is presented for all approaches over all instances. We observe clearly that major re-

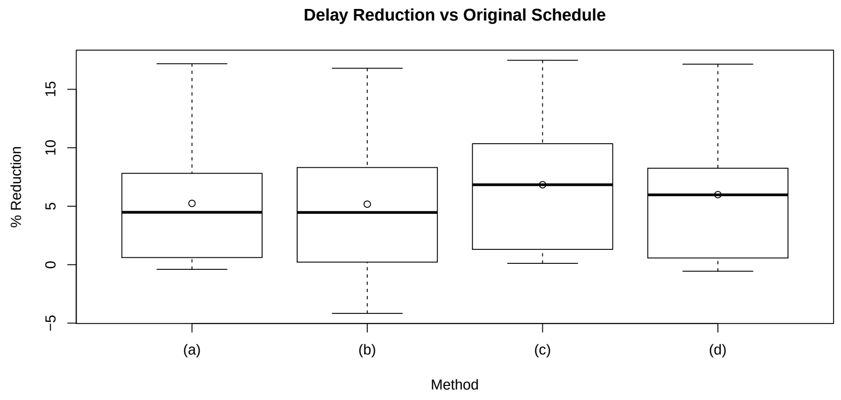

Fig. 7 Relative delay reduction with respect to the original schedule for all tested instances using different methods: (a) Deterministic approach using simulation at the end of the process; (b) SimLoop, using simulation to provide feedback on solution's robustness; (c) SimLNS using simulation integrated in the acceptance criterion; (d) SimLNS-W using simulation integrated in the acceptance criterion, but optimising for the worst obtained solution.

ductions are obtained by means of the SimLNS approach. This means that applying simulation during the search process may lead to better results than performing it as a final step.

It can be noticed that total delay reductions are lower for instances based on real data than for those based on a dense network. This can be easily explained by the network topology of real instances, since hub-and-spoke configurations provide less opportunities to perform swaps in order to reduce delays. In addition, initial delays account for a bigger proportion of total delay in instances based on real data. As a general practice to avoid consequences from unforeseen events, airlines introduce additional buffer time between flights. Since we are using real data for these instances, these oversized buffers are present and can absorb part of the initial delays, reducing their propagation through the network. We have defined tighter flight connections in the theoretical instances.

It is important to notice that total delay is not necessarily higher in large instances. This is due to the fact that for most instances initial delays are introduced in the first flights departing the hub airport, i.e. all flights up to one hour from the first departing flight are delayed. Some large scenarios have less flights departing from the hub during the first hour, therefore having a smaller initial delay.

As for efficiency terms, we observe that instances based on real data require more time to be solved. The cause may be attributed to two combined facts: we use the two defined airport-based operators and there is major number of airports present in these instances.

Although we account for total delay, airlines only consider delays larger than 15 minutes. Table 2 shows our results under this consideration. Equally, Fig. 8 presents a graphical interpretation on the relative reduction of delays in excess of 15 minutes.

Table 2 Results for 28 tested instances when only delays greater than 15 minutes are considered. Total delay (in minutes) is reported for the original schedule and solutions obtained by means of the different described methodologies.

Instance	Orig.	Det.	SimLoop	SimLNS	SimLNS-W
50_30	182.1	168.6	167.4	167.2	172.9
50_60	672.0	607.7	600.1	589.0	592.2
50_90	1421.7	1303.5	1348.1	1309.6	1323.0
50_120	2442.3	2332.1	2325.2	2275.6	2292.0
100_30	187.8	178.1	179.5	171.9	178.1
100_60	672.2	642.4	649.2	620.0	619.8
100_90	1520.5	1445.6	1454.5	1400.4	1393.0
100_120	2555.4	2453.3	2426.8	2406.2	2427.3
200_30	295.7	248.5	245.9	239.5	248.7
200_90	1664.9	1395.7	1394.9	1392.2	1396.0
200_60	901.7	703.3	701.8	691.2	703.5
200_120	2372.0	1964.9	1946.9	1928.5	1953.0
300_30	282.1	259.9	254.5	246.0	253.3
300_60	879.9	810.4	793.3	774.6	787.7
300_90	1716.3	1523.9	1515.3	1477.2	1531.8
300_120	2510.7	2312.3	2310.5	2277.2	2312.7
real_50_30	120.0	120.0	120.0	120.0	120.0
real_50_60	240.0	240.0	240.0	240.0	240.0
real_50_90	384.6	385.4	384.4	383.1	383.8
real_50_120	534.9	536.2	534.9	533.2	533.0
real_100_30	450.0	450.0	450.0	450.0	450.0
real_100_60	900.0	900.0	900.0	900.0	900.0
real_100_90	1471.0	1459.9	1463.1	1461.4	1462.3
real_100_120	2319.3	2290.4	2287.5	2281.6	2283.9
real_150_30	90.8	90.0	92.9	90.0	90.0
real_150_60	186.1	182.2	184.0	181.0	180.9
real_150_90	317.0	301.9	296.8	296.6	296.2
real_150_120	504.0	417.5	421.6	418.2	417.5

As expected from previous results, embedding simulation in the search yields better results than using simulation at the end of the process. This is clearly appreciated in Fig. 8, where we observe that SimLNS provides approximately a 10% average delay reduction and over 20% for some instances. We also see in this figure how both SimLNS methods outperform more traditional approaches.

As mentioned in Sect. 1, airlines face significant monetary costs when disruptions occur. If we consider the reported cost of delays beyond the first 15 minutes, €82 per minute according to CODA [19], we can estimate the associated cost of each solution. We report these results for instances based on real data in Table 3. It can be seen that consequences of a disruption affecting a reduced number of flights may escalate quickly to costs over €190,000 (e.g. instance *real_100_120*). This observation reinforces the fact that decision support tools can be extremely useful in disrupted scenarios in order to attain feasible solutions able to reduce incurred costs in reasonable times.

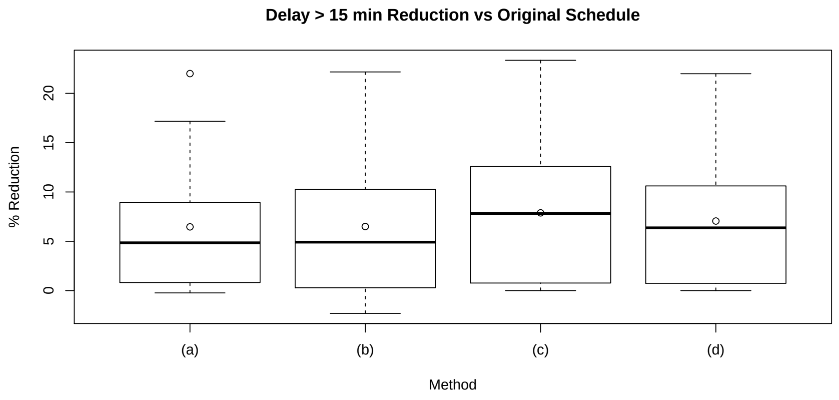

Fig. 8 Relative delay reduction for delays greater than 15 minutes with respect to the original schedule for all tested instances using different methods: (a) Deterministic approach using simulation at the end of the process; (b) SimLoop, using simulation to provide feedback on solution's robustness; (c) SimLNS using simulation integrated in the acceptance criterion; (d) SimLNS-W using simulation integrated in the acceptance criterion, but optimising for the worst obtained solution.

Table 3 Estimated cost of disruptions (in euros) for instances generated from real data. Results are reported for all described methodologies.

Instance	Orig.	Det.	SimLoop	SimLNS	SimLNS-W
real_50_30	9,840.00	9,840.00	9,840.00	9,840.00	9,840.00
real_50_60	19,680.00	19,680.00	19,680.00	19,680.00	19,680.00
real_50_90	31,533.10	31,598.70	31,520.80	31,414.20	31,471.60
real_50_120	43,861.80	43,964.30	43,861.80	43,718.30	43,706.00
real_100_30	36,900.00	36,900.00	36,900.00	36,900.00	36,900.00
real_100_60	73,800.00	73,800.00	73,800.00	73,800.00	73,800.00
real_100_90	120,617.90	119,707.70	119,970.10	119,830.70	119,904.50
real_100_120	190,178.50	187,812.80	187,570.90	187,087.10	187,279.80
real_150_30	7,445.60	7,380.00	7,617.80	7,380.00	7,380.00
real_150_60	15,256.10	14,936.30	15,088.00	14,837.90	14,829.70
real_150_90	25,994.00	24,755.80	24,333.50	24,321.20	24,284.30
real_150_120	41,328.00	34,235.00	34,571.20	34,292.40	34,235.00

6 Conclusions

Operational disruptions are deviations from originally planned operations. Airlines are among the most affected industries by this kind of events and have to face significant associated costs. For this reason, designing methods to deal with operational disruptions efficiently is getting an increasing attention from airlines and the research community.

Cancelled, delayed and diverted flights impact different elements involved in airlines' operations: aircraft, crews and passengers. Traditionally, aircraft are seen as the scarce resource, as well as a more tractable problem due to its size. In this work, we have tackled the Aircraft Recovery Problem, whose goal is to restore the original flight schedule as much as possible in a disrupted scenario by means of swapping aircraft-to-flight allocations and delaying flights when necessary. As real operations are subject to variability, we define an ARP variant which includes this inherent uncertainty: the Stochastic Aircraft Recovery Problem.

As a first step, we have developed a Constraint Programming model to solve the deterministic ARP. To the best of our knowledge, it is the first CP formulation presented for this problem. We integrate this model within a Large Neighbourhood Search framework, a metaheuristic which has proved to be very efficient when combined with CP to cope with a variety of combinatorial optimisation problems. In our approach, we embed the CP model in operators and take advantage of constraint propagation efficiency in the destroy and repair phases of the LNS. Concretely, we unassign a set of variables in the destroy phase and use an exact branch-and-bound method to re-optimise the partially destroyed solution. This method has shown to be efficient for solving the ARP.

In order to deal with the stochasticity present in the SARP, we have modified our LNS approach and included simulation at different stages of the search. In particular, we define two LNS-based frameworks that make extensive use of simulation. In our first schema, the SimLoop, simulation is performed at the end of the LNS method. This way, we simulate the solution obtained in the optimisation process to check its robustness. In this test, different criteria for robustness or solution's characteristics may be used. If the solution is rejected, a new lower bound is imposed and the LNS process is re-launched. In our second approach, the SimLNS, simulation is integrated at an earlier stage of the search. After the destroy/repair phase of the LNS, we test the obtained solution in several simulated scenarios and utilise these results in the acceptance criterion. Therefore, we check solutions' behaviour before accepting or rejecting them, allowing to detect undesirable attributes earlier in the process.

Results show that the proposed LNS variants constitute an efficient approach to tackle the SARP. Indeed, as they are defined as general methodologies, the presented LNS-based methods can be used to solve any combinatorial problem where stochasticity is an inherent characteristic of the system. In general, the SimLNS methodology provides the best solutions for most instances, although computational times are slightly higher because of a major number of simulated scenarios. Among defined instances, we have observed that scenarios based on real data provide few margin for improvement due to network configuration and oversized buffers. We believe that further developments on efficient tools to support decision making in disrupted scenarios may lead to schedules with reduced buffer times, as airlines may be able to respond more effectively to unforeseen events.

The work presented in this chapter leaves several lines open for future research. On the one hand, we aim at improving search efficiency and being able to increase neighbourhood's size. With this purpose, a more effective CP representation of the ARP is to be developed in order to enhance constraint propagation. This upgraded formulation may include other elements present in a disrupted scenario, i.e. crews and passengers. On the other hand, we contemplate integrating simulation within the CP search tree. This way, simulation is used to propagate most likely values in the domain of stochastic variables during search, instead of simulating complete scenarios after labelling variables in a deterministic fashion.

Acknowledgements NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

1. Ahuja, R., Ergun, O., Orlin, J., Punnen, A.: A survey of very large-scale neighborhood search techniques. Discrete Applied Mathematics **123**, 75–102 (2002)
2. Apt, K.: Principles of Constraint Programming. Cambridge University Press, Amsterdam (2003)
3. Apt, K., Wallace, M.: Constraint Logic Programming using ECLiPSe. Cambridge University Press, New York, USA (2007)
4. Argüello, M.F., Bard, J.F., Yu, G.: A GRASP for aircraft routing in response to grounding and delays. Journal of Combinatorial Optimization **5**, 211–228 (1997)
5. Argüello, M.F., Bard, J.F., Yu, G.: Models and methods for managing airline irregular operations. In: G. Yu (ed.) Operations Research in the Airline Industry, *International Series in Operations Research & Management Science*, vol. 9, pp. 1–45. Springer-Verlag (1998)
6. Arias, P., Guimarans, D., Mújica, M.A.: A new methodology to solve the stochastic aircraft recovery problem using optimization and simulation. In: International Conference on Interdisciplinary Science for Innovative Air Traffic Management (ISIATM). Toulouse, France (2013)
7. Arias, P., Guimarans, D., Mújica, M.A., Boosten, G.: A methodology combining optimization and simulation for real applications of the stochastic aircraft recovery problem. In: 8th EUROSIM Congress on Modelling and Simulation, pp. 265–270. Cardiff, UK (2013)
8. Arora, S., Barak, B.: Computational complexity: A modern approach. Cambridge University Press, New York, USA (2009)
9. Ball, M., Barnhart, C., Nemhauser, G., Odoni, A.: Air transportation: irregular operations and control, *Handbooks in Operations Research & Management Science*, vol. 14, chap. 1, pp. 1–67. Elsevier (2007)
10. Bertsimas, D., Nohadani, O., Teo, K.M.: Robust optimization for unconstrained simulation-based problems. Operations Research **58**(1), 161–178 (2010)
11. Bessiere, C.: Constraint Propagation, vol. Handbook of Constraint Programming, chap. 3, pp. 29–83. Elsevier, Amsterdam (2006)
12. Cao, J., Kanafani, A.: Real-time decision support for integration of airline flight cancellations and delays. Part I: Mathematical formulation. Transportation Planning and Technology **20**(3), 183–199 (1997)
13. Cao, J., Kanafani, A.: 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 (1997)
14. Clausen, J., Larsen, A., Larsen, J., Rezanova, N.J.: Disruption management in the airline industry – Concepts, models and methods. Computers & Operations Research **37**(5), 809–821 (2010)
15. Dantzig, G.B., Wolfe, P.: Decomposition principles for linear programs. Operations Research **8**(1), 101–111 (1960)
16. Eggenberg, N., Salani, M., Bierlaire, M.: A column generation algorithm for disrupted airline schedules. Tech. Rep. TRANSP-OR 071203, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland (2007)

17. Eggenberg, N., Salani, M., Bierlaire, M.: Constraint-specific recovery network for solving airline recovery problems. Computers & Operations Research **37**, 1014–1026 (2010)
18. EUROCONTROL: Report on punctuality drivers at major European airports. Tech. rep., EUROCONTROL (2005)
19. EUROCONTROL: Planning for delay: influence of flight scheduling on airline punctuality. Tech. Rep. Trends in Air Traffic - Vol. 7, EUROCONTROL (2010)
20. of European Airlines, A.: European airline punctuality in 1st quarter 2008. Tech. rep., Association of European Airlines (2008)
21. European Commission: Guide to European Community legislation in the field of civil aviation. Directorate-General for Energy and Transport (2007)
22. Freuder, E.C., Mackworth, A.K.: Constraint Satisfaction: an Emerging Paradigm, vol. Handbook of Constraint Programming, chap. 2, pp. 13–27. Elsevier, Amsterdam (2006)
23. Gendreau, M., Potvin, J.Y.: Tabu Search, vol. Handbook of Metaheuristics, chap. 2, pp. 41–60. Kluwer Academic, Boston, MA (2010)
24. Granberg, T.A., Värbrand, P.: The flight perturbation problem. Transportation Planning and Technology **27**(2), 91–117 (2004)
25. Grasas, A., Juan, A.A., Lourenço, H.R.: SimILS: a simulation-based extension of the Iterated Local Search metaheuristic for stochastic combinatorial optimization. Journal of Simulation (2014). DOI 10.1057/jos.2014.25
26. Guimarans, D.: Hybrid algorithms for solving routing problems. Ph.D. thesis, Universitat Autònoma de Barcelona, Barcelona (2012)
27. Guimarans, D., Herrero, R., Ramos, J.J., Padrón, S.: Solving vehicle routing problems using constraint programming and lagrangian relaxation in a metaheuristics framework. International Journal of Information Systems and Supply Chain Management **4**(2), 61–81 (2011)
28. Guimarans, D., Herrero, R., Riera, D., Juan, A.A., Ramos, J.J.: Combining probabilistic algorithms, constraint programming and lagrangian relaxation to solve the vehicle routing problem. Annals of Mathematics and Artificial Intelligence **62**(3-4), 299–315 (2011)
29. van Hoeve, W.J., Katriel, I.: Global constraints, vol. Handbook of Constraint Programming, chap. 6, pp. 169–208. Elsevier, Amsterdam (2006)
30. Jarrah, A.I.Z., Yu, G., Krishnamurthy, N., Rakshit, A.: A decision support framework for airline flight cancellations and delays. Transportation Science **27**(3), 266–280 (1993)
31. Juan, A.A., Grasman, S.E., Caceres-Cruz, J., Bektaş, T.: A simheuristic algorithm for the single-period stochastic inventory routing problem with stock-outs. Simulation Modelling Practice and Theory **46**, 40–52 (2014)
32. Kohl, N., Larsen, A., Larsen, J., Ross, A., Tiourine, S.: Airline disruption management – Perspectives, experiences and outlook. Journal of Air Transportation Management **13**(3), 149–162 (2007)
33. Kumar, V.: Algorithms for constraint-satisfaction problems: a survey. AI Magazine **13**(1), 32–44 (1992)
34. Lawler, E.L., Wood, D.E.: Branch-and-bound methods: A survey. Operations Research **14**(4), 699–719 (1966)
35. Le, M., Wu, C., Zhan, C., Sun, L.: Airline recovery optimization research: 30 years' march of mathematical programming – A classification and literature review. In: 2011 International Conference on Transportation, Mechanical, and Electrical Engineering (TMEE), pp. 113–117. IEEE, Changchun, China (2011)
36. Løve, M., Sørensen, K.R.: Disruption management in the airline industry. Master's thesis, Technical University of Denmark, Lyngby, Denmark (2001)
37. Løve, M., Sørensen, K.R., Larsen, J., Clausen, J.: Disruption management for an airline – Rescheduling of aircraft. In: S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, G.R. Raidl (eds.) Applications of Evolutionary Computing, *Lecture Notes in Computer Science*, vol. 2279, pp. 315–324. Springer-Verlag, Berlin Heidelberg (2002)
38. Nikolaev, A.G., Jacobson, S.H.: Simulated Annealing, vol. Handbook of Metaheuristics, chap. 1, pp. 1–40. Kluwer Academic, Boston, MA (2010)
39. amd P. Shaw, P.K.: Vehicle Routing, vol. Handbook of Constraint Programming, chap. 23, pp. 801–836. Elsevier, Amsterdam (2006)

40. Perron, L., Shaw, P., Furnon, V.: Propagation guided large neighborhood search. In: M. Wallace (ed.) 10th International Conference on Principles and Practice of Constraint Programming (CP-04), *Lecture Notes in Computer Science*, vol. 3258, pp. 468–481. Springer-Verlag, Berlin Heidelberg (2004)
41. Pisinger, D., Ropke, S.: Large Neighborhood Search, vol. Handbook of Metaheuristics, chap. 13, pp. 399–420. Kluwer Academic, Boston, MA (2010)
42. Reeves, C.R.: Genetic Algorithms, vol. Handbook of Metaheuristics, chap. 5, pp. 109–140. Kluwer Academic, Boston, MA (2010)
43. Resende, M.G.C., Ribeiro, C.C.: Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications, vol. Handbook of Metaheuristics, chap. 10, pp. 283–320. Kluwer Academic, Boston, MA (2010)
44. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science **40**(4), 455–472 (2006)
45. Rosenberger, J.M., Johnson, E.L., Nemhauser, G.L.: Rerouting aircraft for airline recovery. Transportation Science **37**(4), 408–421 (2003)
46. Rosenberger, J.M., Schaefer, A.J., Goldsman, D., Johnson, E.L., Kleywegt, A.J., Nemhauser, G.L.: A stochastic model of airline operations. Transportation Science **36**(4), 357–377 (2002)
47. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier, Amsterdam (2006)
48. Shavell, Z.A.: The effects of schedule disruptions on the economics of airline operations. In: 3rd USA/Europe Air Traffic Management R&D Seminar. Napoli, Italy (2000)
49. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: 4th International Conference on Principles and Practice of Constraint Programming (CP-98), *Lecture Notes in Computer Science*, vol. 1520, pp. 417–431. Springer-Verlag, Berlin Heidelberg (1998)
50. Teodorović, D., Guberinić, S.: Optimal dispatching strategy on an airline network after a schedule perturbation. European Journal of Operational Research **15**(2), 178–182 (1984)
51. Teodorović, D., Stojković, G.: Model for operational daily airline scheduling. Transportation Planning and Technology **14**(4), 273–285 (1990)
52. Thengvall, B.G., Bard, J.F., Yu, G.: Balancing user preferences for aircraft schedule recovery during irregular operations. IIIE Transactions **32**, 181–193 (2000)
53. Thengvall, B.G., Yu, G., Bard, J.F.: Multiple fleet aircraft schedule recovery following hub closures. Transportation Research A **35**, 289–308 (2001)
54. Wu, C., Le, M.: A new approach to solve aircraft recovery problem. In: INFOCOMP 2012: The Second International Conference on Advanced Communications and Computation, pp. 148–154. Venice, Italy (2012)
55. Wu, C.L.: Inherent delays and operational reliability of airline schedules. Journal of Air Transportation Management **11**(4), 273–282 (2005)
56. Wu, C.L.: Improving airline network robustness and operational reliability by sequential optimisation algorithms. Networks and Spatial Economics **6**(3–4), 235–251 (2006)
57. Yan, S., Lin, C.: Airline scheduling for the temporary closure of airports. Transportation Science **31**(1), 72–82 (1997)
58. Yan, S., Tu, Y.: Multifleet routing and multistop flight scheduling for schedule perturbation. European Journal of Operational Research **103**(1), 155–169 (1997)
59. Yan, S., Yang, D.: A decision support framework for handling schedule perturbation. Transportation Research B **30**(6), 405–419 (1996)
60. Yan, S., Young, H.: A decision support framework for multi-fleet routing and multi-stop flight scheduling. Transportation Research A **30**(5), 379–398 (1996)
61. Zhang, X., Zhao, M., Kuang, S.M., Du, Q.: Research on airline company fuel-saving model based on petri network. Advanced Materials Research **616–618**, 1107–1110 (2013)