Integrated Aircraft Scheduling Problem: An Auto-Adapting Algorithm to Find Robust Aircraft Assignments for Large Flight Plans (Abstract)

Conference Paper · January 2012

DOI: 10.1109/HICSS.2012.330

CITATIONS
13

READS
21,815

4 authors:

Torsten Reiners
Curtin University
207 PUBLICATIONS **1,674** CITATIONS
SEE PROFILE

Julia Pahl
University of Southern Denmark
31 PUBLICATIONS **503** CITATIONS
SEE PROFILE

Michael Maroszek
Universität zu Lübeck
1 PUBLICATION **13** CITATIONS
SEE PROFILE

Cornelius Rettig
1 PUBLICATION **13** CITATIONS
SEE PROFILE

Integrated Aircraft Scheduling Problem: An Auto-Adapting Algorithm to Find Robust Aircraft Assignments for Large Flight Plans

Torsten Reiners
Curtin Business School, Curtin University, Perth, Aus.
treiners@curtin.edu.au

Julia Pahl, Michael Maroszek, Cornelius Rettig
Institute of Information Systems, University of Hamburg, Germany, pahl@econ.uni-hamburg.de, michael@maroszek.de, rettig.cornelius@gmail.com

Abstract

The overall airline scheduling process involves hierarchical steps starting with the network design and ending with crew assignment. Aircraft routing is especially important with respect to timing and costs for an airline. In this contribution, we focus on aircraft routing where aircraft are assigned to flight legs further considering maintenance requirements. We developed and implemented algorithms that extend the aircraft routing problem (ARP) by including profit and robustness. The latter objective is important as the dependencies of flights and airlines increases and deviations to the original time plan as unexpected events like volcano eruptions or heavy weather-related issues are difficult to handle. A robust aircraft routing ensures that unforeseen events have less impact. The results are compared to current state-of-the-art solutions. We developed a test instance-generator to create specific problems and build a library for future benchmarking tests.

1. Introduction

The aircraft (maintenance) rotation problem is part of the overall airline scheduling problem [21]. It takes place after the fleet assignment which is determined according to the flight schedule where destinations, flight dates, and departure times are established based on demand assumptions [2]. The objective is to assign an aircraft (identified by its *tail number*) of a given fleet to a feasible sequence of flight legs, so that maintenance constraints are satisfied. Generally, the planning steps are executed in a hierarchical order where the solution of one sub-problem is used as input for the subsequent problem [5, 16, 20, 21, 25].

In order to provide an idea of the planning constraints regarding the ARP, we first discuss the overall airline scheduling process as depicted in Figure 1. The overall process is divided into three parts, i.e., flight schedule generation, aircraft scheduling and crew scheduling all being further broken down into distinct planning steps.

Figure 1. Overall airline scheduling process

The first planning step is the flight schedule generation which generally covers a period of three to six months. In order to reduce complexity, cyclical patterns covering a day or week are constructed and repeated for the whole period. The flight schedule generation embraces *network design, frequency assignment,* and *flight schedule*. The network design considers the identification of origin-destination city pairs thus building the route network the airline wants to serve. The design is based on strategic and tactical decisions as well as traffic forecasts including demand variations. The number of flights to each origin-destination pair is assigned during the frequency assignment. Exact departure times are determined during the planning of flight schedules. The focus in this paper is set on the subsequent planning step *aircraft scheduling* including *fleet assignment* and *aircraft routing* with the later one being further subdivided in *through flight assignment* and *maintenance routing*. Fleet assignment concentrates on the assignment of aircraft types with respect to several aircraft characteristics, e.g., cruising speed, fuel consumption, capacity, and other time and cost relevant components such as maintenance requirements [15]. The minimum number of aircraft is determined in the fleet assignment. In case that a fixed time horizon is considered and deadhead flights (flights of aircraft without passengers) are excluded, the problem turns out to be rather easy and practical problem instances solvable in polynomial time using network flow technique [15]. Within aircraft routing, a rotation of an aircraft is denoted as a "sequence of aircraft routings that starts and ends at the same location and can be flown by one or more aircraft (in parallel)" [16]. Subsequently, the through flight assignment builds rotations in a way that a minimum of passengers have to change aircraft on connecting

flights, i.e., to avoid passenger or baggage transfers to other gates or aircraft, and, thus, potential causes for irritations. This is done by creating flight pairs that are operated by the same aircraft [15]. Generally, the first destination airport of such flight pairs is a hub in a hub-and-spoke network structure. The maintenance routing problem addresses the requirement of regular maintenance checks of aircraft with time intervals and durations depending on the aircraft type [15, 16]. These checks range from simple visual inspections (about 30 minutes) to major overhauling that employs 15 − 30 days [16]. Aviation authorities imply regular checks whose extend depend on the combination of flight hours, take-off cycles as well as the number of landings. They are usually labeled alphabetically ranging from visual examinations of important aircraft parts (A-checks) continuously extending to comprehensive overhauling actions (D-checks). Airlines target the execution of maintenance checks at night. As a result, maintenance checks constrain rotations and their determination significantly.

2. Literature review

Despite advances in computer hardware and optimization theory, the airline scheduling process is too complex to solve all stages simultaneously with existing optimization technique [16]. Nevertheless, advances enable applications of exact solution approaches with high(er) solution quality [23], higher degree of details as well as consideration of realistic problem sizes. We find a large number of exact approaches solving combined sub-problems of airline scheduling that might also be joined with heuristics or related elements. Deterministic formulations of airline scheduling (sub-) problems are predominant that assume fixed demand data, flight and turn times leading to tight schedules with short turn times due to objectives regarding profit maximization [16]. These schedules are easily affected by disruptions and delays [3]. Therefore, research concentrates on the integration of flexibility, e.g., buffer times in order to increase reliability of flight connections [19, 31]. With the aim to decrease complexity, it is common practice to determine flight schedules and fleet assignment for a single day that is repeated every other day of the week except for weekends that contain a subset of the flight schedule. The same is valid for the ARP (*daily routing model*) [12, 15].

Solution approaches to the ARP can be divided into those that use underlying *line-of-flight* (LOF) structures further employing heuristics to solve the problem and those that use mathematical programming formulations [15]. LOFs contain all flight legs of an aircraft defined in time units that may be given by the (first) departure airport and the (last) arrival airport of the assigned fleet type [16]. In a solution, the number of LOFs for each fleet type must equal the number of related available aircraft. Furthermore, all LOFs must contain all flights assigned to the specific fleet type. LOFs may be constructed by LIFO or FIFO rules [15, 16]. However, they may not contain valid routings regarding maintenance requirements [16]. Questions about the robustness of ARP are important due to their high impact on schedule reliability and relatively low impact on costs, e.g., crews or flights, or passenger revenues [21, 19]. Delays and disruptions are accounted for in [10, 12, 13]. A static as well as a dynamic ARP using fixed LOFs including maintenance for tail numbers in the static case and variable LOFs in the dynamic case are proposed by [15]. In the latter, the considered time horizon equals the number of aircraft. The authors propose a polynomial-time algorithm that generates LOFs and, subsequently, checks if maintenance stations are considered for every tail number, at most, every three days. They show that fixed LOFs allow for solving the problem in polynomial time whereas variable LOFs lead to an NP-complete model. Capacity limits of maintenance stations for C-checks are assumed by [15]. Four-day aircraft maintenance routing problems are solved in [29] using an exact approach in combination with heuristic; see [2, 26] for balanced utilization of aircraft.

Through flight revenues and maintenance constraints are examined in [10] with the aim of finding the most profitable solution. They formulate the problem as an asymmetric travelling salesman problem with side constraints [16]. The through flight problem and the maintenance routing problem are combined by employing an Eulerian di-graph. This means that the in-degree at each node equals the out-degree, so that the graph is strongly connected [15]. They use a time-based flight network determined subsequently to the fleet assignment. Maximizing through flight revenues is equal to finding the Hamilton circuit with maximal value in the related line graph. In case that this circuit includes long sequences of flights without maintenance stops, such sequences are cut and re-solved [10].

Advances in computational research allow for solving growing problem instances, so that parts of airline scheduling can be integrated to be simultaneously solved. Various integrated approaches are available; see [30] for an overview.

3. Problem analysis and data acquisition

Implementing an algorithm requires understanding of the problem domain and knowledge about the data. In this paper, we base all estimates and assumptions on analyzed real-world flight plans mainly on the freely available plans from Star Alliance [27], statistics from governmental institutions and other publications on the integrated aircraft scheduling and ARP. The most relevant data items for the considered problem and proposed algorithms are:

- Flight plans including the distribution of departure and arrival airports, flight slots, frequency of flights, types of aircraft, as well as estimated passenger demands;
- Airports including geographical position, IATA codes, capabilities of handling certain aircraft types, capacity, and restrictions of times for starts and landings;
- Maintenance including type of maintenance checks, frequency, duration, and airports that can perform the maintenance;
- Related cost data, i.e., average costs per flight leg, ground (parking), and maintenance.

These sources are used to create a test instance generator and to benchmark the developed algorithms to real-world problems.

4. Algorithm for ARP

We describe the basic optimization model for the ARP. Solutions are calculated, among others, using CPLEX. We develop greedy start heuristics for the ARP and compare their solutions to those of the network flow problem. The heuristics serve later as a starting point for the integrated aircraft scheduling problem in Section 5.

4.1. Mathematical model

The presented model for the ARP is a network flow problem formulation as given in [28] which does not consider costs in the objective function, but the number of rotations. The problem is acyclic due to the time progression which makes it solvable in polynomial time [28].

The objective function (5.1) minimizes the rotation count, (5.2) ensures that each flight – except for the start and end flights – has exactly one predecessor and one successor flight. Constraints (5.3) and (5.4) require that each flight is covered only once.

$$\min \sum_{(i,j)\in E} X_{i,j}, \qquad \forall i \in V, j = t \tag{5.1}$$

$$\text{s.t. } 0 \le X_{i,j} \le 1, \qquad \forall (i,j) \in E \setminus (t,s), \tag{5.2}$$

$$\sum_{(jj,i)\in R} X_{i,j} = 1, \qquad \forall i \in V, \forall j \in V \setminus (t,s), jj = j, \tag{5.3}$$

$$\sum_{(jj,i)\in E} X_{j,i} = 1, \qquad \forall i \in V, \forall j \in V \setminus (t,s), jj = j. \tag{5.4}$$

$s \in V$: source node

$t \in V$: end node

V = Airports $\cup$ $s \cup t$: vertices

E = flight legs $\cup$ connections from s to all airports and from all airports to t: edges

R = flight legs $\cup$ connections from t to all airports and from all airports to s: edges

$X_{i,j}$: vertex $j \in V$ is the direct predecessor of vertex $i \in V$ in one tour

Experiments using test instances are presented in 4.2 and compared with the developed greedy start heuristic described in the following section.

4.2. Greedy start heuristics

The developed greedy start heuristic named *random-start-fly-forth-back* (RSFFB) algorithm is based on an aircraft-centered algorithm where flights are selected by specific aircraft. All flights of a flight plan are included in a list from which a newly created aircraft randomly chooses a flight. While selecting suitable flights by checking location, planned arrival and departure time, the aircraft is moved forward in time until no possible flight is left. The algorithm terminates when every flight is assigned to an aircraft. Additionally, a similar algorithm searches backwards for possible sequences of flights. This is exemplified in Figure 2.

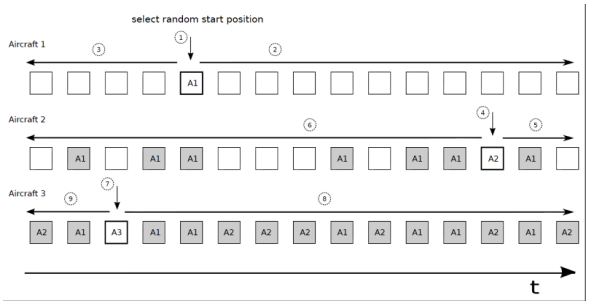

Figure 2. Example of the RSFFB

4.3. Results and Comparison

We translate the ARP given in 4.1 to an LP formulation in order to use the *GNU linear programming kit* (GLPK) software package, so that the model can be solved. Experiments are run on two systems using one core, respectively: (1) 2.93 GHz Dual-Xeon, Ubuntu x64 and (2) 1.2 GHz Core 2 Duo, Windows 7, x86. Different real-world test intances are used; more details are provided in Section 6. Results show that the proposed heuristics are very competitive in terms of calculation time,

used memory, and optimality gap in comparison to the optimization formulations represented by GLPK and CPLEX. Table 1 gives representative results.

Table 1. Results for large instances

solver	time	rotations	error	system	memory usage
GLPK	468s	467	0%	1	3GiB
CPLEX	25s	467	0%	1	2GiB
random-start-fly-forth-back	45s	467	0%	2	1MiB
aircraft-centered	$\leq$ 1s	468	0.21%	2	1MiB

Even though System 2 has the inferior computer system, the results clearly demonstrate that the heuristics can compete with the CPLEX solution, i.e., considering the time and memory to find a solution near optimality. Nevertheless, the results indicate that the ARP is too simple for current computer systems. Therefore, we extend the problem by a profit driven objective function and the consideration of robustness. Based on advanced experiments, we decide to use the *RSFFB*-algorithm as our starting point.

5. Integrated aircraft scheduling problem

The integrated aircraft scheduling problem is based on the ARP and considers profit maximization and increased robustness of solutions, e.g., by minimizing delays or cancelations in disruptive situations. The algorithm employed for finding solutions incorporates a fitness function that evaluates individual flights as well as whole flight paths taking into account certain criteria discussed in more details in Section 5.2. Applied validation criteria are the following:

- no open flights are left;
- each flight has to be assigned to exactly one aircraft;
- assigned flights do not exceed the range of the aircraft;
- the assigned flights are in chronological order and do not intersect or start on the arrival location of their respective predecessor;
- maintenance intervals are respected.

We consider maintenance checks according to [11] and allow for deadhead flights. They are generally not allowed in the classical ARP. Deadhead flights might be required to reduce the number of aircraft and allow for maintenance checks in case they are not feasible in a proposed flight sequence.

5.1. Control algorithm

The presented control algorithm determines a best set of parameters for the subsequently employed (parameterized) solution creation algorithm that is based on genetic algorithms. It consists of three core-components (Figure 3) that are (1) a multi-process including an auto-adapting control algorithm, (2) a fitness function to evaluate and compare solutions, and (3) a parameterized local search for solution generation. It contains four parameters which are a) the maximum population size c_p, b) the number of best solutions included in the next generation c_t, c) the number of random solutions considered in the next generation c_r in order to increase variety, and d) the number of unsuccessful generations before stopping denoted by c_i. An auto-adaptation is performed by the control algorithm where a crossover implies choosing a parameter from a parent for the search algorithm while the mutation randomly applies a 0.25 change in either direction within the allowed range.

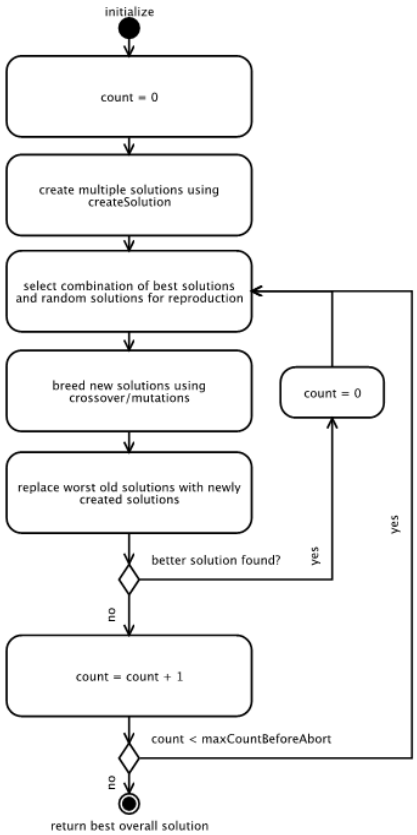

Figure 3. Aircraft scheduling problem solver control algorithm guiding the solution creation algorithm, similar to a local search algorithm, using varying parameters

5.2. Fitness Function

The fitness of a solution is measured regarding its profit and robustness. The following equation calculates the fitness using the profit weighted by the robustness of the solution:

$$Q_s = \begin{cases} P_s * (1 - w_r + R_s * w_r) & \text{if } P_s \geq 0 \\ P_s * (1 + w_r - R_s * w_r) & \text{if } P_s < 0 \end{cases}$$

with Q_S being the fitness of solution s, P_s the profit of solution s in *US Dollars* (USD), R_s the robustness of solution s expressed in %, and w_r the robustness evaluation value. The profit P_s is calculated regarding different revenues and expenses given as follows:

$$P_s = \sum_{f \in F_s} (r_f - e_{o_f} - e_{l_f}) - \sum_{g \in G_s} e_g - \sum_{m \in M_s} e_m$$

with r_f denoting the (ticket) revenue of flight f in the set of considered flights, thus $f \in F_s$ given in USD. Costs for operating the aircraft (e_{of}) and landing costs (e_{lf}) are subtracted for flight $f \in F_s$ as well as overall costs for parking events denoted by e_g with $g \in G_s$. The cost for landing and parking are based on the Exeter International Airport's schedule of fees and charges. Finally, overall costs for maintenance are considered and given by e_m for maintenance $m \in M_s$ where M_s denotes the set of maintenance checks. The above mentioned ticket revenue is calculated as follows:

$$r_f = \min\left(s_f^e, s_f^a\right) \cdot r_{sk} \cdot d_f$$

s_f^e: estimated seat demand for flight $f \in F_s$

s_f^a: seats available in the aircraft that executes flight $f \in F_s$

r_{sk}: average revenue per seat per km in USD

d_f: travel distance of flight $f \in F_s$ in km

In order to get an estimate on r_{sk}, we use the average values given in [7] and [1] for the ticket price (USD 340), the revenue per ticket (71.1%), and stage length (1813,73 km). Note that these values are average values and thus should be adapted in case better estimates or scenarios are available, e.g., if only one airline is involved or all flights are within a specific region.

We calculate maintenance costs e_m as follows:

$$e_m = t_m/3600 \cdot BHDOC_m^a \cdot e_{mp}$$

where t_m is the duration of the maintenance $m \in M_s$ in seconds, $BHDOC^a{}_m$ the block-hour direct operating costs of the aircraft a that executes the flight $f \in F_s$ measured in USD/h, and e_{mp} denoting proportions of maintenance cost in %.

We integrate robustness by assuming that robustness is increased if the probability of propagated delays is reduced. Propagated delays occur when a delay affects departures of successor flights. A common solution to increase robustness is to enhance planned parking and thus incorporate buffer times [3, 14, 18]. We consider the duration for the parking time as "perfectly robust" that is at least as long as the average late flight departure delay. In contrast, parking time durations up to the average total flight departure delay are regarded not robust. Such integration of robustness via parking duration counterbalances profits due to enhanced related costs.

5.3. Solution Creation Algorithm

The (parameterized) solution creation algorithm is based on genetic algorithms and employed to generate solutions. It is based on different sub-functions that interact regarding different parameters:

- c_{as}: the aircraft count used in sub-function *createAircraftGroupShifted(.)*
- c_{ag}: the aircraft count for the first execution of *createAircraftGroupGreedy(.)*
- t_{dg}: the minimum deadhead flight waiting time for overall usage
- t_{df}: the minimum deadhead flight waiting time for *fillSolutionSpaces* (.)

Figure 4 depicts the algorithm. The functions can be separated in three groups: (1) functions to create complete and valid solutions, (2) functions to optimize given solutions, (3) helper functions called by other functions to improve and repair the solution.

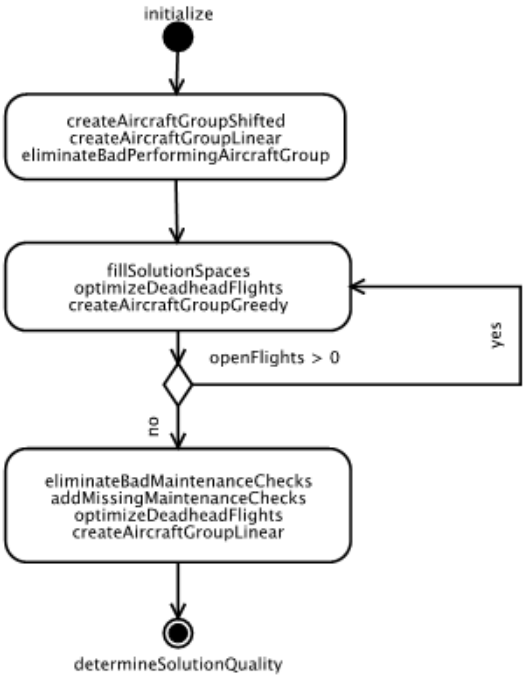

Figure 4. Solution creation algorithm

The sub-function *createAircraftGroupShifted*(.) assigns open flights to aircraft including deadhead flights if no aircraft can perform the related flight; see Figure 5. The function *tryToFly*(.) checks if a given flight can be added to the flight path of a specific aircraft also allowing for deadhead flights as long as a given minimum deadhead flight waiting time is not violated.

createAircraftGroupShifted

```
1: aircraftPool ← new list of aircraft
2: insert aircraftCount aircraft into aircraftPool each fitting to the chronologically
   next entry of openFlights
3: lastNum ← 0
4: for currentFlight ∈ openFlights do
5:   success ← false
6:   for currentAircraft ∈ aircraftPool, shifted by lastNum do
7:     success ← tryToFly(currentAircraft, currentFlight, 0)
8:     if success = true then
9:       break
10:    end if
11:  end for
12:  if success = false then
13:    for currentAircraft ∈ aircraftPool, shifted by lastNum do
14:      success ← tryToFly(currentAircraft, currentFlight, minimumDeadhead-
         FlightWaitTime)
15:      if success = true then
16:        break
17:      end if
18:    end for
19:  end if
20:  lastNum ← (lastNum +1)  mod  size(aircraftPool)
21: end for
22: append aircraftPool to usedAircraft
```

Figure 5. *createAircraftGroupShifted*(.)

The solution is evenly distributed over all aircraft, but does not guaranteed to be valid. If open flights remain, the function *createAircraftGroupLinear*(.) is called to produce a valid solution by creating deadhead flights or new aircraft that are assigned to remaining open flights; see Figure 6.

createAircraftGroupLinear

```
1: while size(openFlights) > 0 do
2:   flight ← first entry in openFlights
3:   aircraft ← generateAircraft(flight )
4:   for flight ∈ openFlights do
5:     tryToFly(aircraft, flight, minimumDeadheadFlightWaitTime)
6:   end for
7: end while
```

Figure 6. *createAircraftGroupLinear*(.)

Aircraft with a low number of assigned flights are eliminated by the function e*liminateBadPerformingAircraftGroup*(.); see Figure 7. The function *fillSolutionSpaces*(.) is applied to fill left-over open flights into aircraft schedules; see Figure 8. This is followed by *optimizeDeadheadFlights*(.) and *createAircraftGroupGreedy*(.) with a limited amount of aircraft until there are no open flights left; see Figure 10 and Figure 11. Up to this step, maintenance intervals are likely to be violated, so that the functions *eliminateBad-MaintenanceChecks*(.) and *addMissingMaintenance Checks*(.) are applied to either remove all invalid maintenance checks or insert necessary maintenance actions; see Figure 9, Figure 12, and Figure 13 for a visualization of the function *addMissing-Maintenance(.)*.

eliminateBadPerformingAircraftGroup

```
1: aircraftPool ← new list of aircraft and their corresponding effectiveUsageTime
2: for currentAircraft ∈ usedAircraft do
3:   effectiveUsageTime ← time used for scheduled flights and maintenance checks
     of currentAircraft's schedule
4:   add currentAircraft with effectiveUsageTime to aircraftPool
5:   maximumUsefulUsageTime ← max(maximumUsefulUsageTime, effec-
     tiveUsageTime)
6:   sort aircraftPool ascending by effectiveUsageTime
7: end for
8: for currentAircraft, effectiveUsageTime ∈ aircraftPool do
9:   if effectiveUsageTime < tolerance * maximumUsefulUsageTime then
10:    remove currentAircraft from usedAircraft
11:    add all scheduled flights that were scheduled for currentAircraft to open-
       Flights
12:    removedFlightCount ← removedFlightCount +1
13:  end if
14:  if removedFlightCount = maximumRemovalCount then
15:    break
16:  end if
17: end for
```

Figure 7. *eliminateBadPerforming-AircraftGroup*(.)

fillSolutionSpaces

```
1: for currentFlight ∈ openFlights do
2:   for currentAircraft ∈ usedAircraft in reverse order do
3:     for lastAction ∈ currentAircraft's schedule do
4:       if currentFlight can be inserted after lastAction, considering inserting
         deadheads flight before and after currentFlight if minimumDeadhead-
         FlightWaitTime is exceeded then
5:         insert currentFlight and eventual deadhead flights after lastAction
6:         break the two inner loops
7:       end if
8:     end for
9:   end for
10: end for
```

Figure 8. *fillSolutionSpaces*(.)

eliminateBadMaintenanceChecks

```
1: for D, C, B and A checks do
2:   for aircraft ∈ usedAircraft do
3:     checkDistance ← 0
4:     for action ∈ usedAircraft's schedule do
5:       increment checkDistance, if action is a scheduled flight
6:       reset checkDistance, after a maintenance check of the appropriate type
         has been executed
7:       if checkDistance > usedAircraft's corresponding maintenance interval
         then
8:         remove all future maintenance checks of the violated type from usedAir-
           craft's schedule
9:         break
10:      end if
11:    end for
12:  end for
13: end for
```

Figure 9. *eliminateBadMaintenanceChecks*(.)

```
optimizeDeadheadFlights
1: for aircraft ∈ usedAircraft do
2:   for action ∈ aircraft's schedule do
3:     remove action from the schedule, if it is a deadhead flight
4:   end for
5:   for action ∈ aircraft's schedule do
6:     insert deadhead flight sequences between the current flight and the next
       flight, if their arrival location and departure location are not matching
7:   end for
8: end for
```

Figure 10. *optimizeDeadheadFlights*(.)

```
createAircraftGroupGreedy
1: aircraftPool ← new list of aircraft
2: for currentFlight ∈ openFlights do
3:   potentialAircraft ← new list of aircraft
4:   fill potentialAircraft with entries from aircraftPool that are able to execute
     currentFlight without making a deadhead flight
5:   if size(potentialAircraft) = 0 then
6:     add all those aircraft to potentialAircraft from aircraftPool that can exe-
       cute the current flight by inserting one deadhead flight considering mini-
       mumDeadheadFlightWaitTime
7:   end if
8:   if size(potentialAircraft) > 0 then
9:     sort potentialAircraft by minimum parking time required
10:    selectedAircraft ← aircraft with minimum parking time required in poten-
       tialAircraft
11:  else if size(aircraftPool) < aircraftCount ∨ aircraftCount = 0 then
12:    selectedAircraft ← generateAircraft(currentFlight)
13:    add selectedAircraft to aircraftPool
14:  end if
15:  if selectedAircraft ≠ ∅ then
16:    tryToFly(selectedAircraft, currentFlight, minimumDeadheadFlightWait-
       Time)
17:  end if
18: end for
19: append aircraftPool to usedAircraft
```

Figure 11. *createAircraftGroupGreedy*(.)

```
addMissingMaintenanceChecks
1: for D, C, B and A checks do
2:   for aicraft ∈ usedAircraft do
3:     for action ∈ aircraft's schedule do
4:       if the corresponding maintenance interval has been violated then
5:         follow the aircraft's schedule backwards until a maintenance path can
           be inserted after a scheduled action without violating the maintenance
           interval
6:         follow the aircraft's schedule until an action has been found, that can
           be connected to the maintenance path with or without additional dead-
           head flights
7:         delete all flights that intersect with the maintenance check and the
           eventual deadhead flights to be introduced
8:         insert the maintenance and the eventual deadhead flights into the air-
           craft's schedule
9:       end if
10:    end for
11:  end for
12: end for
```

Figure 12. *addMissingMaintenanceChecks*(.)

Unnecessary deadhead flights are removed by *optimizeDeadheadFlights*(.). As the insertion of new maintenance checks is likely to have moved scheduled flights back into open flights, *createAircraftGroupGreedy*(.) is again employed with an unlimited aircraft count to render the solution valid.

5.4 Experiments

Extensive experiments are conducted to gain an understanding of parameters and their calibration as well as to verify the quality of solutions regarding real-world scenarios with respect to robustness and profit. We present the outcome of the calibration. Detailed descriptions for all runs including tables and graphs are provided in [4].

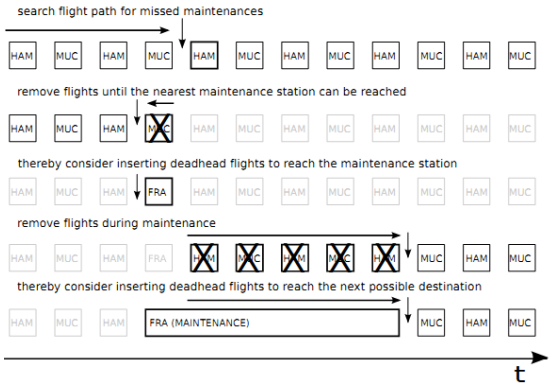

Figure 13. *addMissing Maintenance*(.)

Different parameter combinations are used for calibration of the control algorithm and for analyzing the behavior of the algorithm in order to determine good reference values. We replicate each parameter combination for each instance five times and record average values that give the best mixture of quality and runtime. The experiments indicate the following reference values (see Section 5.1) where the tested ranges are given in squared:

- c_p: 25 [15, …, 65; Step size: 5]
- c_t: 12 [8, …, 20; Step size: 2]
- c_r: 6 [2, …, 12; Step size: 2]
- c_i: 5 [1, …, 9; Step size: 2]

These reference parameters are used, again, on eight instances created by a problem generator described in Section 6; see Table 2 for information on the instance structure where the names associate that the problems represent existing scenarios and could be applied by airline operators.

name	ns	d	f	a	h	at	mat	fs	mcf
AustralianTwo	hts	55	13750	11	2	2	1	35	66
CanadaUSA	hts	100	10000	30	6	2	1	35	31
Europe	hts	150	30000	20	6	4	2	5	58
LittleFrenchConnection	hts	60	1200	10	4	2	2	0	7
AmericanDream	ptp	55	13750	40	x	3	4	15	101
DownUnder	ptp	56	13750	41	x	2	1	35	96
OneYear	ptp	365	73000	40	x	10	4	50	78
WorldTour	ptp	36	2450	30	x	1	3	20	40

Table 2. Problem instance summary (ns: network structure hub-and-spoke (hts) point-to-point (ptp); d: days; f: flights; a:airports; h: hubs; at: aircraft types; mat: maintenance per aircraft type; fs: fleet size; mcf: maximum concurrent flights)

Table 3. Experimental results

instance		statistics				results			
name	mcf	deadhead (in %)	demand served (in %)	seat load (in %)	used aircraft	time (s)	robustness	profit (USD)	quality
AustralianTwo	66	3.79	71.0	51.5	118	155	0.687	55,962,087	54,707,144
CanadaUSA	31	8.39	98.1	24.7	104	85	0.995	126,762,392	126,720,321
Europe	58	1.05	67.5	87.2	182	583	0.964	189,877,008	189,386,728
LittleFrenchConnection	7	56.70	97.1	76.6	10	1	0.962	-3,161,801	-3,170,329
AmericanDream	101	5.00	65.0	80.2	238	200	0.922	197,418,302	196,320,462
DownUnder	96	4.30	97.8	25.3	251	233	0.990	165,987,579	165,866,922
OneYear	78	7.22	99.1	12.2	185	2731	0.996	770,370,546	770,154,910
WorldTour	40	14.70	97.3	45.9	112	14	0.995	475,329,240	475,163,325
combined	477	101.15	692.9	403.6	1,200	4,002	7,511	1,978,545,353	1,975,149,483
average	60	12.64	86.6	50.5	150	500	0.939	247,318,169	246,893,685

All experiments are executed on an Intel Core i7-920 processor (2.66GHz, eight virtual CPU-cores) running Windows 7 x64. The algorithm is given as a single-threaded Java implementation. Used memory was below 1GB in all cases.

We analyzed the behavior of the algorithms, i.e. the development of the parameters c_{ag}, c_{as}, t_{dg} and t_{df}. Throughout all instances, we observe a quick increase for the solution quality which implies that the algorithm finds the good solutions at an early stage. Results in Table 3 show that the robustness is high with an average of 0.939 except for one case (*AustralianTwo*) with a value of 0.687. In comparison to problem instances *AmericanDream* and *Down-Under* that nearly include identical numbers of flights and days (see Table 2), the *maximum concurrent flights* (MCF) of *AustralianTwo* is equivalent to 66% of their average MCF which indicates a plausible connection between flight density for individual aircraft and robustness.

The algorithm prefers to serve passenger demand towards seat load. This can result from an inappropriate set of available aircraft types. The number of deadhead flights is rather low ranging from 1,05% to 14.70% except for the problem instance *LittleFrenchConnection* with 56.70% which further is the only instance showing a negative profit. In fact, alternative calculation of minimum departure expenses and maximum revenues employing the most efficient aircraft types demonstates that positive revenues are out of reach. Results show for this test instance a remarkably high percentage of deadhead flights (56.70%) which might explain the minor increase of aircraft count (10) in comparison to the MCF (7). The reason for this behavior might be the low flight density of the test instance. An increase in used aircraft would raise parking expenses to a greater extend than diminishing deadhead flight.

5.5. Comparison with other Algorithms

Over the last years, several authors proposed new algorithms to solve the combined fleet assignment and ARP. While robustness seems to be included in [22], crew pairing is often part of proposed algorithms [24, 17, 23]. Reported problem sizes (e.g., number of flights) are generally rather low with less than 2600 flights and less than 10 aircraft types. The same problem as considered in this paper is examined in [22] regarding a maximum size of 2558 flights and 9 aircraft types. Our experiments include large instances that are solved in a reasonable time (73,000 flights, 10 aircraft types). The suitability of the algorithm for such large instances results from using meta-heuristics. Other authors employing heuristics do not solve instances beyond 3,000 flights [22, 23]. In conclusion, the proposed algorithm is competitive and provides a valuable tool for airline companies constrained to handle large data sets, require short running times and positive profits to improve company results rather than finding (theoretical) optimum solutions for small data sets.

6. ARP Test Instance Generator and Visualization Suite (ARPViS)

One major contribution is the flight plan generator and visualization suite (ARPViS). For many established problems, there are libraries with test instances (e.g., OR-Library [6]), while libraries for the ARP as well as *integrated aircraft scheduling* problems are not available. Availability of such a library is of major interest in order to compare algorithms and define requirements for problem instances. We aim to fill this gap by creating a test instance generator that considers several parameters, viz. timeframe in days, flights per day, number of used airports, root network structure, connection density, maximum number of aircraft slots per airport, aircraft type count, number of maintenance stations per aircraft, fleet size, and maximum concurrent flights. We employ real-world input, if available, from different sources, e.g., flight plans, publications, and/or airline operators to create realistic flight plans. Nevertheless, the validity of the results depends on the parameter settings and requires individual verification. Currently, we do not consider

different weekdays or passenger demand distributions aligned with real-world demand. We used the generator to create eight instances regarding different specifications as an initial set for a library; see Figure 12 for a visualization of the flight plan *Europe* with 30,000 flights and 20 airports. The instances are visualized using ARPViS [4], a tool that allows for depicting special parameters of the flight plan, e.g., the density of flights for specific airports, connections, etc. as well as the solution. The understanding of the flight plan is enhanced by underlying maps as well as interactive elements to project results of individual flight legs. The planer is further supported, e.g., regarding overall expenses and profits of the best solution.

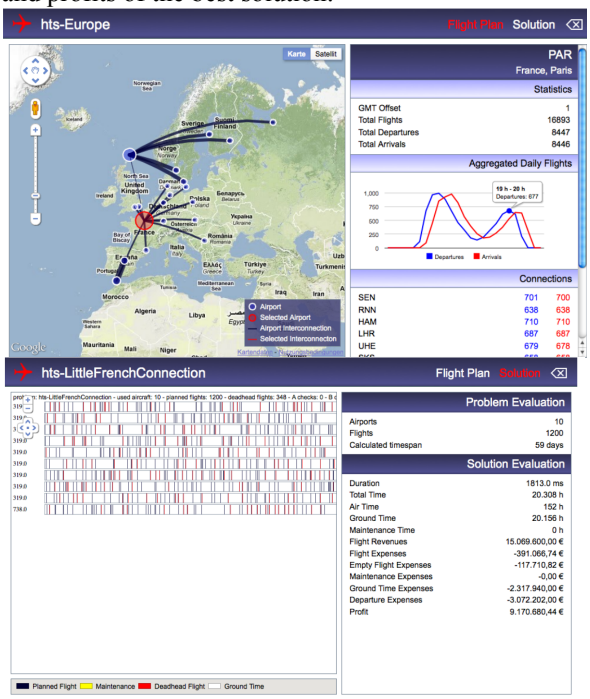

Figure 12. Flight plan and result presentation

7. Conclusions

Airline planners have to solve complex tasks. We discuss the current state-of-the-art for the (extended) ARP including profits, maintenance, and deadhead flights. We introduce an advanced algorithm to find robust and near optimal profitable solutions within real-world acceptable time intervals. We concentrate on the development of heuristics, thus leaving mathematical problem formulation to a later stage.

The major drawback of published algorithms is the limited size of considered test instances and the time required to find a solution. With instances containing less than 3,000 flights, only small airlines might use related algorithms and software. Otherwise, they need to reduce the time horizon to consider a few days to simplify the problem together with maintenance constraints that might not be an issue anymore. Our proposed algorithm solves test instances with more than 73,000 flights, more than 10 aircraft types, and deadhead flights to improve profits and to consider all levels of maintenance further regarding robustness of solutions. Robustness is of major importance as it allows airlines to fulfill plans even if delays occur, e. g., during boarding. The runtime for very large instances is still below an hour and, therefore, applicable during the planning stage in order to evaluate various flight plans. A further substantial contribution is the presented test instance generator that takes real-world statistics into account and generates flight plans based on various parameters that might also be specified by the user; see http://iwi.econ.uni-hamburg.de/IWIWeb/Uploads/Team/JP/AllFilesInternet.zip. Our intention of the generator and the eight instances used in this paper is the creation of a test instance library to allow for effective comparison of algorithms in the future. Currently, such a library does not exist. Finally, our visualization tool supports handling flight plans and interpretation of solutions.

8. References

[1] Airline Data Project, Massachusetts Institute of Technology. Average Stage Length Flown of Total Operating Fleet. Website, December 2009. URL http://web.mit.edu/airlinedata/www/2009%2012%20Month%20Documents/Aircraft%20and%20Related/Total%20Fleet/Average%20Stage%20Length%20Flown%20of%20Total%20Operating%20Fleet.htm. visited on January 20th 2011.

[2] H. M. Afsar, M.-L. Espinouse, and B. Penz. Buidling flight planning for an airline company under maintenance constraints. Journal of Quality in Maintenance Engineering, 15(4):430-443, 2009

[3] S. Ahmad-Beygi, A. Cohn, and M. Lapp. Decreasing Airline Delay Propagation By Re-Allocating Scheduled Slack. IIE Transactions, 42(7):478-489, 2010.

[4] T. Reiners, J. Pahl, M. Maroszek, C. Rettig. Optimization and Heuristics for Aircraft Planning and Scheduling: Survey and Implementation. Technical Report. https://iwi.econ.uni-hamburg.de/IWIWeb/Default.aspx?tabid=174, visited on August 26th 2011.

[5] C. Barnhart, N. L. Boland, L. W. Clarke, E. L. Johnson, G. L. Nemhauser, and R. G. Shenot. Flight string models for aircraft fleeting and routing. Transportation Science, 32(3): 208-220, 1998.

[6] J. E. Beasley. OR-Library: Distributing test problems by electronic mail. Journal of Operational Research Society, 41:1069-1072, 1990.

[7] Bureau of Transportation Statistics, U.S. Department of Transportation. 3rd-Quarter 2010, Domestic Air Fares Rose 10.7% from 3rd Quarter 2009. Press Release, 2/2/2011

[8] Bureau of Transportation Statistics, U.S. Department of Transportation. Average delay for the Newark International Airport and Continental Airlines between 01.01.2010 and 01.01.2011. Website (generated), March 2011b. URL http://transtats.bts.gov/. visited on March 26th 2011.

[9] Bureau of Transportation Statistics, U.S. Department of Transportation. On-Time Arrival Performance – National (January, 2011). Website (generated), March 2011c. URL http://transtats.bts.gov/. visited on March 26th 2011.

[10] L. Clarke, E. Johnson, G. Nemhauser, and Z. Zhu. The aircraft rotation problem. Annals of Operations Research, 69:33-46, 1997.

[11] A. J. Cook and G. Tanner. Innovative Cooperative Actions of R&D in EUROCONTROL Programme CARE INO III: Dynamic Cost Indexing: Aircraft maintenance marginal delay costs. Monograph, Transport Studies Group, University of Westminster, London, 2008.

[12] G. Desaulniers, J. Desrosiers, Y. Dumas, M. M. Solomon, and F. Soumis. Daily aircraft routing and scheduling. Management Science, 43(6):841-855, 1997b.

[13] S. Dozic, M. Kalic, O. Babic, and M. Cangalovic. Heuristic approach to the airline schedule disturbances problem: multi eet case. In J. Blazewicz, M. Drozdowski, G. Kendall, and B. McCollum, editors, Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2009), 10-12 Aug 2009, Dublin, Ireland, 311-320, 2009.

[14] M. Dunbar, G. Froyland, and C.-L. Wu. Robust Airline Schedule Planning: Minimizing Propagated Delay in an Integrated Routing and Crew Framework. Optimization Online, 2010.

[15] R. Gopalan and K.T. Talluri. The aircraft maintenance routing problem. Operations Research, 46(2):260-271, 1998.

[16] T. Grosche. Computational Intelligence in Integrated Airline Scheduling. PhD thesis, University of Mannheim, 2009.

[17] I. Ioachim, J. Desrosiers, F. Soumis, and N. Belanger. Fleet assignment and routing with schedule synchronization constraints. EJOR, 119(1): 75-90, 1999.

[18] S. Lan, J.-P. Clarke, and C. Barnhart. Planning for Robust Airline Operations: Optimizing Aircraft Routings and Flight Departure Times to Minimize Passenger Disruptions. Transportation Science, 40(1):15-28, February 2006.

[19] S. Lan, J.-P. Clarke, and C. Barnhart. Planning for robust airline operations: Optimizing aircraft routings and flight departure times to minimize passenger disruptions. Transportation Science, 40(1):15-28, 2006.

[20] M. Lohatepanont and C. Barnhart. Airline Schedule Planning: Integrated Models and Algorithms for Schedule Design and Fleet Assignment. Transportation Science, 38(1): 19-32, 2004

[21] L. Marla and C. Barnhart. Robust optimization: Lessons learned from aircraft routing, submitted to Transportation Science, Internet Source: http://www.mit.edu/ lavanya/, 2009.

[22] J. M. Rosenberger, E. L. Johnson, and G. L. Nemhauser. A Robust Fleet-Assignment Model with Hub Isolation and Short Cycles. Transportation Science, 38:357-368, 2004.

[23] R. A. Rushmeier and S. A. Kontogiorgis. Advances in the optimization of airline fleed assignment. Transportation Science, 31(2):159-169, 1997.

[24] R. Sandhu and D. Klabjan. Integrated Airline Fleeting and Crew-Pairing Decisions. Operations Research, 55(3):439-456, 2007.

[25] A. Sarac, R. Batta, and C. M. Rump. A branch-and-price approach for operational aircraft maintenance routing. EJOR, 175(3):1850-1869, 2006.

[26] C. Sriram and A. Haghani. An optimization model for aircraft maintenance scheduling and re-assignment. Transportation Research Part A: Policy and Practice, 37(1):29-48, January 2003

[27] Star Alliance Services GmbH. Timetable January 1st 2011, March 20th 2011. Website, January 2011. URL http://www.staralliance.com/assets/doc/en/services/tools-and-downloads/pdf/StarAlliance.pdf. Visited on January 20th 2011.

[28] L. Suhl and T. Mellouli, Optimierungssysteme: Modelle, Verfahren, Software, Anwendungen, Springer, Berlin, 2009.

[29] K. T. Talluri. The four-day aircraft maintenance routing problem. Transportation Science, 32(1):43-53, 1998.

[30] O. Weide. Robust and Integrated Airline Scheduling. PhD thesis, Department of Engineering Science, School of Engineering, University of Auckland, New Zealand, 2009.

[31] C.-L. Wu. Improving airline network robustness and operational reliability by sequential optimization algorithms. Networks and Spatial Economics, 6(3-4):235-251, 2006.