Journal of Aerospace Operations 1 (2012) 249–266
DOI 10.3233/AOP-2012-0013
IOS Press

Contingency plans for air traffic flow and capacity management using constraint programming

Karl Sundequist Blomdahl*,[1], Pierre Flener and Justin Pearson
Department of Information Technology, Uppsala University, Uppsala, Sweden

Abstract. We present a constraint-based local search heuristic that contributes to solving the problem of generating contingency plans for air traffic flow and capacity management, which are to be used in the case of a catastrophic infrastructure failure within EUROCONTROL, the European Organisation for the Safety of Air Navigation. Experiments with the heuristic, implemented in *Comet*, on real-world flight plans for the entire European airspace show that it is feasible to automate the development of contingency plans, which is currently done by human experts. This is desirable as the development time goes down from two person months per year to a few CPU hours, and as it allows contingency plans to be generated with an increased frequency.

Keywords: Contingency planning, air traffic flow and capacity management, constraint programming, constraint-based local search, tabu search

1. Introduction

1.1. Air traffic management

Air traffic management (ATM) is about managing and ensuring a safe, efficient, and fair flow of air traffic, assuming negligible interference from side-effects, such as adverse weather conditions or mechanical failures. During normal operation, the *Central Flow Management Unit* (CFMU) of the *European Organisation for the Safety of Air Navigation* (EUROCONTROL) has several planning stages, each in increasing detail, to satisfy these three conflicting operational goals:

1. A strategic stage, taking place several months before the day of operation.
2. A pre-tactical stage that starts six days before the day of operation.
3. An online tactical stage during the day of operation. This stage is called the *air traffic flow and capacity management* (ATFCM) stage [1], and has two main functions:

[1]This paper extends [5, 6].

*Corresponding author: Karl Sundequist Blomdahl, Department of Information Technology, Uppsala University, Box 337, SE–751 05 Uppsala, Sweden. E-mail: Karl.Sundequist@it.uu.se.

ISSN 2211-002X/12/$27.50

(a) Calculate the demand of each airspace volume using live flight plan information.
(b) Adjust the number of allocated departure slots of the involved aerodromes, such that they optimise the objectives defined in the pre-tactical stage. These objectives typically include, but are not limited to, minimising the total flight delay and the air volume overcapacity.

During an average day, the ATFCM unit currently handles approximately 30 000 flights spread over about 1 500 aerodromes.

1.2. Contingency planning

This study focuses on the special case of an ATFCM failure due to any reason, such as downtime of the *computer-assisted slot allocation* (CASA) system. In such a situation, where no timely updates from ATFCM are available and the air controllers of each aerodrome have no idea whether it is proper to release a flight or not, a safe alternative is necessary. EUROCONTROL addresses this by a *contingency plan*, which contains a pre-defined number of allocated departure slots for each major aerodrome in such a way that certain safety and efficiency objectives are satisfied, for a maximum duration of one day. During the last thirteen years, such a situation has occurred once, for a few hours. Nevertheless, EUROCONTROL requires the existence of such contingency plans, and they take time to develop.

An excerpt from such a contingency plan can be seen in Fig. 1. It defines the number of departure slots that the aerodrome with the *International Civil Aviation Organisation* (ICAO) identifier EBBR (Brussels National Airport, Belgium) is allowed to release for each hour to various destination aerodromes. For example, from 09:00 to 12:00, a maximum of 7 flights are allowed to take off in the flow EBBR1, which is defined by the departure aerodrome EBBR and a destination aerodrome whose ICAO identifier starts with C (Canada), EG (Great Britain), EI (Ireland), K (United States), or M (Central America and Mexico). Similarly, only 4 flights whose departure and destination aerodrome match the description of the flow EBBR2 are allowed to take off per hour from 06:00 to 17:00.

The current contingency plan can always be downloaded from the CFMU website https://www.cfmu.eurocontrol.int/.

The generation of ATM contingency plans within the *EUROCONTROL Experimental Centre* (EEC) and the CFMU is currently done by two human experts (using a process described in Section 2.1). They biannually (for the winter and summer timetables) develop a three-fold plan, namely one for weekdays, one for Saturdays, and one for Sundays.

Flow identifier	Flow description	Time span	Hourly rate
EBBR1	From: EBBR	00:00 – 06:00	2
	To: C EG EI K M	06:00 – 09:00	3
		09:00 – 12:00	7
		12:00 – 14:00	4
		14:00 – 22:00	8
		22:00 – 24:00	2
EBBR2	From: EBBR	00:00 – 06:00	1
	To: B EDDH EDDW EE	06:00 – 17:00	4
	EF EH EK EN ES	17:00 – 21:00	6
		21:00 – 24:00	2

Fig. 1. A contingency plan excerpt, which describes the hourly take-off rates of two flows originating from the aerodrome EBBR (Brussels National Airport, Belgium).

The total contingency planning time is two person-months per year, hence automated contingency planning is desirable. Another benefit with automating the process is that it could be done at the tactical level instead of the strategic level, which would increase the quality of the generated contingency plans.

1.3. Contributions and organisation of this paper

This paper presents a local search [2] heuristic that solves in just a few CPU hours the problem of finding (near-)optimal time spans and hourly rates for *given* flows (which typically do not change much between contingency plans anyway) for the *entire* European airspace. It is intended as a feasibility study about replacing the human experts with constraint programming (CP) technology [3]. To our knowledge, this is the first time that contingency planning has been at least partially automated.

We here outline the model and the best of the two local search heuristics in our paper [5] at a specialist conference on CP.

The rest of this paper is split into four parts, dealing with the contingency planning problem in increasingly concrete terms: a formal definition of the problem as a constraint model (Section 2), a local search heuristic that operates on the constraint model (Section 3), experimental results with an implementation of the heuristic (Section 4), and a conclusion (Section 5).

2. The contingency planning problem

Informally, we address the following subproblem in contingency planning. We are given a set of flight plans and a set of flows. Our objective is to determine (near-)optimal hourly departure rates and time spans for these flows, such that efficiency and safety of the global air traffic flow are maximised, under a fair allocation of departure slots. We measure efficiency as the total delay cost of all flights, under a first-submitted, first-served allocation. We measure safety as the total overcapacity cost of all air volumes. We minimise the weighted sum of these two terms and ensure fairness on the fly.

We now give the current state of the art algorithm, and give a formal description of this combinatorial optimisation problem as a constraint model.

2.1. Current state of the art

The current state of the art, and the only known algorithm, to solve the contingency planning problem is the unpublished process used by the CFMU and EEC human experts. It has been described to us in the following high-level steps:

1. A statistical analysis is performed in order to point out the airspace volumes with a high demand. The duration and capacity of each air volume are recorded (there may be several durations per air volume).
2. An analysis of departing flows is made:
   - For the major European airports (i.e. with more than two arrivals or departures per hour on average), the traffic needs to be divided into main flows, where several destinations are grouped into each flow.
   - For the other airports, the flows are mainly divided into two categories: domestic flights and international flights. If the number of domestic flights is low, it seems better that a local flow manager handles this traffic.

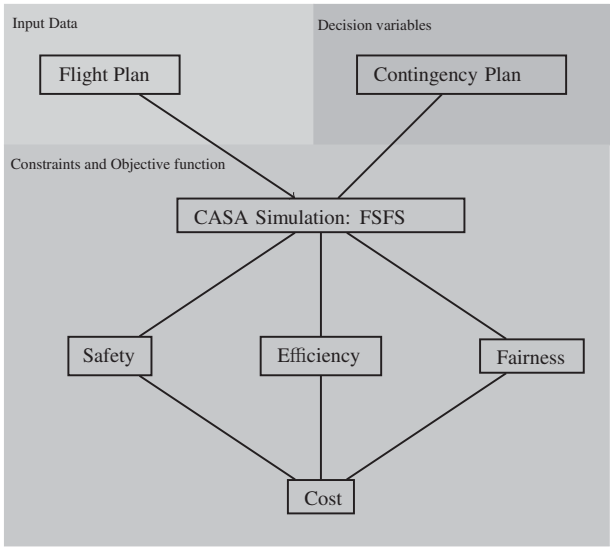

Fig. 2. Overview of our constraint model.

Recall that it takes one person-month each for two senior human experts to perform this algorithm, and that all this is done twice a year (once for the summer timetable and once for the winter timetable), for weekdays, Saturdays, and Sundays.

2.2. Constraint model

Our constraint model is implemented in *Comet* [4], an object-oriented constraint programming language for the modelling of combinatorial problems. It has back-end solvers for (global) tree search interleaved with constraint propagation, for constraint-based local search, and for mixed integer linear programming. *Comet* is available at http://dynadec.com/.

Comet offers a very-high-level *modelling language* for fully declaratively specifying a combinatorial optimisation problem by (1) identifying the decisions that need to be made, namely the so-called *decision variables* (or *unknowns*) and their sets of possible values, called *domains*, (2) stating the *constraints* that are to be satisfied, and (3) defining the expression (called the *objective function*) that is to be minimised or maximised. Such a constraint model is (in principle) independent of the back-end solver.

An overview of our constraint model is given in Fig. 2. The inputs are a set of flight plans, and the main decision variables denote the hourly rates and the time spans of the output contingency plan. Through constraints that simulate the slot allocation process of ATM, the hourly rate decision variables are connected to overcapacity decision variables and take-off delay decision variables, from which the safety and efficiency terms of the objective function are respectively determined. The fairness term of the objective function is obtained through the search heuristic rather than through constraints.

2.2.1. Constraints and decision variables

An instance of the contingency planning problem is defined by the following input and output data, where identifiers starting with capital letters denote given sets, subscripted identifiers denote given constants, identifiers with indices within square brackets denote decision variables (outputs), identifiers that

are Greek letters denote given parameters, and all time moments are measured in seconds since some fixed origin:

- A set of flights $F = \{f_1, \ldots, f_m\}$, where each flight f_ℓ has a departure aerodrome $adep_\ell$, a destination aerodrome $ades_\ell$, an expected take-off time $etot_\ell$, a calculated take-off time $ctot[\ell]$, an expected landing time $eldt_\ell$, and a take-off delay $delay[\ell]$. All later specified sets of flights are subsets of F.
- A set of air volumes $AV = \{av_1, \ldots, av_p\}$, where each air volume $av_a \in AV$ has a capacity cap_a that limits the hourly number of flights that can enter it for the duration dur_a. There is also a flight set $F_a \subseteq F$ for each air volume av_a that contains all flights that pass through av_a, where each flight $f_\ell \in F_a$ has an expected entering time $enter_{a,\ell}$, an expected exit time $exit_{a,\ell}$, a calculated entering time $cnter[a, \ell]$, and a calculated exit time $cxit[a, \ell]$. In the real-world, an air volume can represent either a part of the airspace or an aerodrome. Note that a dynamic airspace, where air volumes can change over time, can be modelled by splitting each air volume into its unique states and making sure the flights enter the correct one. For example, consider a dynamic air volume av_d that has the capacity 2 between 00:00 and 12:00 o'clock and the capacity 4 between 13:00 and 15:00 o'clock. This air volume av_d can be modelled by splitting it into two air volumes av_{d_1} with a capacity of 2 and av_{d_2} with a capacity of 4, and then partitioning F_d such that all flights entering av_d between 00:00 and 12:00 o'clock are stored in F_{d_1} and all flights entering av_d between 13:00 and 15:00 o'clock are stored in F_{d_2}.
- A set of flows $\mathcal{F} = \{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$, where each flow $\mathcal{F}_f$ consists of a set of flights F_f and a set of span-rate pairs $\mathcal{R}_f = \{r_1, \ldots, r_{o_f}\}$, where each span-rate pair r_i consists of a time span $span[i]$ denoting when it is active, and an hourly rate of allocated departure slots $rate[i]$ in the closed integer interval $[1, demand[f, i]]$, where $demand[f, i]$ is the maximum number of flights that are re-scheduled to depart in flow $\mathcal{F}_f$ during the time span $span[i]$:

  $$demand[f, i] = \max_{t \in T[i]} |\{f_\ell \in F_f : t \leq ctot[\ell] < t + 3600\}|$$

  where set $T[i]$ contains the beginning times of all one-hour-long time intervals that fit inside the time span $span[i]$ of the span-rate pair r_i, with a five minute step:

  $$T[i] = \{t \in span[i] : t + 3600 \in span[i] \wedge t \bmod 300 = 0\}$$

  Further, for any two span-rate pairs r_i and r_j, where $i \neq j$, their spans must not overlap, but for each r_i its lower bound must either be zero or immediately follow the upper bound of another span-rate (which can be accomplished by inserting dummy spans with infinite rates during a pre-processing stage); note that the union of all spans for a given flow need not be 00:00 – 24:00 since no constraint on the upper bound of the span has been given. To ease the manipulation of the spans, and to avoid violations of the given constraints, a span length $len[i] \in [0, 24]$ is defined as the difference between the lower and the upper bound of $span[i]$; note that since we know there is no free space between each span we can, given some fixed ordering among the span-rates within each flow, say that the lower bound of each span-rate pair must be at the end of another (or 0 if it is the first). This means that one can fully determine the span bounds in a flow given their lengths:

  $$span[i] = \begin{cases} [0, len[0]) & \text{if } i = 0 \\ [\max(span[i-1]), & \\ \max(span[i-1]) + len[i]) & \text{otherwise} \end{cases}$$

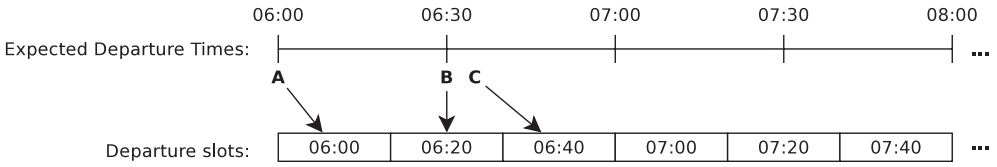

Fig. 3. An example showing how the calculated take-off times are calculated for three fights, *A*, *B*, and *C* for the flow EBBR1 (defined in Fig. 1) between 06:00 and 08:00 o'clock with an hourly departure rate of 3.

where the notation $[x, y)$ designates the right open integer interval $\{x, x+1, x+2, \ldots, y-1\}$. There is also a flight set $F_f \subseteq F$ for each flow $\mathcal{F}_f$ that contains all flights matching the flow description. For example, Fig. 1 defines two flows EBBR1 and EBBR2, where the flights are defined by a subset of F that matches the flow description, and the spans and rates are defined by the two right-most columns.

Additional decision variables used in the objective function will be defined in the following paragraphs.

Recall that ATM has three conflicting operational goals: ensure an efficient flow of air traffic (by minimising the total delay), ensure a safe flow of air traffic (by minimising the total overcapacity), and ensure a fair flow of air traffic. During a crisis situation, safety is especially important. Before giving the objective function, we first discuss the constraints induced by these operational goals.

2.2.2. Constraints on air traffic efficiency

The take-off delay $delay[\ell]$ of any flight f_ℓ is the difference between its calculated take-off time $ctot[\ell]$ and its expected take-off time $etot_\ell$:

$$delay[\ell] = ctot[\ell] - etot_\ell$$

where $ctot[\ell]$ is calculated using the allocated departure slots as defined by the span-rate pairs for each flow. These slots are assigned to flights using the first-submitted, first-served principle (FSFS) [7]. For example, consider Fig. 3 which shows the flow EBBR1 (defined in Fig. 1), where there are three departure slots allocated for each hour between 06:00 and 09:00: if three flights with expected take-off times 06:00 (A), 06:30 (B), and 06:35 (C) were available, then they would get the calculated take-off times 06:00 (A), 06:30 (B), and 06:40 (C), and delays of 0, 0, and 300 seconds, respectively; note that flight C was delayed since its slot had already been allocated to B. Another example, which shows the domino effect of delaying a flight, is Fig. 4, which shows the same flow EBBR1, where there are two departure slots allocated for each hour between 00:00 and 06:00: if three flights with expected take-off times 04:05 (A), 04:20 (B), and 04:30 (C) were available, then they would get the calculated take-off times 04:05 (A), 04:30 (B), and 05:00 (C); notice that the last flight was given a departure slot in the hour after its planned departure hour and might therefore delay flights that were planned to depart during that hour.

Similarly, the take-off delay $delay[\ell]$ of any flight f_ℓ also is the difference between its calculated entering time $cnter[a, \ell]$ into any air volume av_a and its expected entering time $enter_{a,\ell}$ into that air volume; the same applies to a flight's calculated exit time:

$$delay[\ell] = cnter[a, \ell] - enter_{a,\ell}$$

$$delay[\ell] = cxit[a, \ell] - exit_{a,\ell}$$

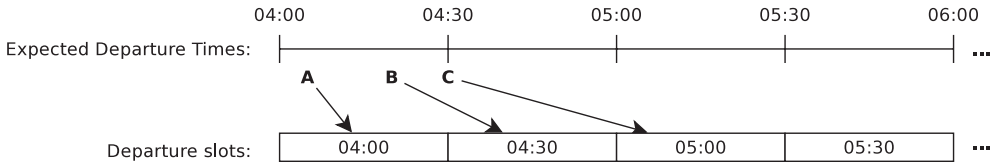

Fig. 4. An example showing how the calculated take-off times are calculated for three fights, *A*, *B*, and *C* for the flow EBBR1 (defined in Fig. 1) between 04:00 and 06:00 o'clock with an hourly departure rate of 2.

The *delay cost* of any flight f_ℓ is as defined by the dynamic delay cost function of [8], which assigns a different cost (constructed from real-world measurements) to each aircraft based on maintenance cost, number of passengers, roster rotations, etc. Our model is slightly simplified in the sense that we define a *global* cost function instead of using one per aircraft; however if such granularity is desired then our model could be adopted without any major work. The delay cost $delayCost[t]$, where t is the number of seconds of delay, is presented as a weight function, however it is recommended that interpolation is used in order to make the transition between the weights smoother:

$$delayCost[t] = \begin{cases} 0.00 & \text{if } t = 0\,\text{sec} \\ 0.10 & \text{if } 0 < t \leq 300\,\text{sec} \\ 0.50 & \text{if } 300 < t \leq 900\,\text{sec} \\ 1.52 & \text{if } 900 < t \leq 1800\,\text{sec} \\ 5.00 & \text{if } 1800 < t \leq 2700\,\text{sec} \\ 9.85 & \text{if } 2700 < t \leq 3600\,\text{sec} \\ 15.61 & \text{if } 3600 < t \leq 5400\,\text{sec} \\ 30.86 & \text{if } 5400 < t \leq 7200\,\text{sec} \\ 50.00 & \text{if } 7200 < t\,\text{sec} \end{cases}$$

Notice that the exact shape of this function can be changed, as the only assumption our heuristic makes is that it is monotonically non-decreasing. The *total delay cost* is the sum of the delay costs of all the flights:

$$\sum_{f_\ell \in F} delayCost[delay[\ell]]$$

2.2.3. Constraints on air traffic safety

The *safety* of air traffic is determined by how crowded the air volumes are. The air volume av_a is capable of handling up to cap_a flights entering per hour, so any flight above this capacity creates an additional risk and any flight below this capacity means unused resources (in the form of underused controllers) that could be better utilised elsewhere. Hence, safety is here defined by the difference between each air volume's hourly capacity and its demand.

For each air volume av_a, a set $T[a]$ is defined that contains the beginning times of all one-hour-long time intervals that fit inside the air volume's capacity duration dur_a, with a five minute step, as depicted by the x axis in Fig. 5:

$$T[a] = \{t \in dur_a : t + 3600 \in dur_a \wedge t \bmod 300 = 0\}$$

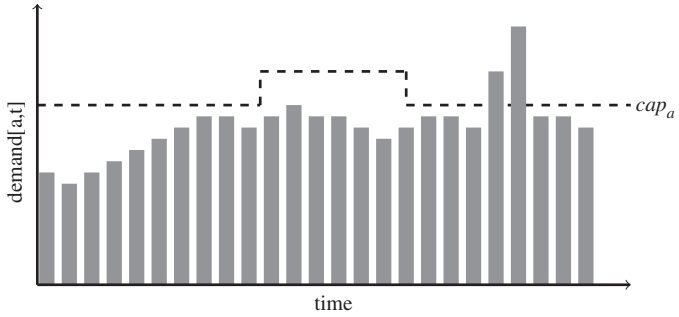

Fig. 5. Demand for an air volume a over time: the vertical bars denote overlapping one-hour-intervals that start every five minutes, and the height of a bar for start time t indicates the number of flights scheduled to enter a during the hour following t, so that any excess of a bar over the capacity of a denotes an overcapacity.

The *overcapacity* of each air volume av_a and beginning time $t \in T[a]$ is the number of flights, above the capacity of av_a, that enter av_a during the right-open time interval $[t, t + 3600)$:

$$overcapacity[a, t] = |\{f_\ell \in F_a : cnter[a, \ell] \in [t, t + 3600)\}| - cap_a$$

Note that a negative overcapacity designates an undercapacity.

Alternatively one can use a load constraint that controls the number of flights inside the air volume at any given time (one can simply substitute $overcapacity[a, t]$ for $overload[a, t]$ and cap_a for $load_a$ in all consecutive equations to get the desired result):

$$overload[a, t] = |\{f_\ell \in F_a : cnter[a, \ell] \leq t \wedge cxit[a, \ell] \geq t\}| - load_a$$

Which of the two formulations is most suitable depends on the regulations of the particular country hosting the air volumes. In this particular study we have used only capacity constraints, but for example the Netherlands and the United Kingdom use load constraints in the air volumes they control. It would be a trivial change, only to the model but not to the search, to use a suitable mixture of capacity constraints and load constraints.

The *overcapacity cost* of air volume av_a and beginning time $t \in T[a]$, denoted by $overcapacityCost[a, t]$, is defined by a piecewise linear function of the overcapacity percentage $\frac{overcapacity[a,t]}{cap_a}$, where a suitable slope is defined for the overcapacity percentage breakpoints -30%, -20%, -10%, 0%, 10%, 20%, and 30%. An illustration of our chosen function can be seen in Figure 6. Note that the exact shape of this function can be changed, the only property that we require is that it is monotonically non-decreasing on the distance from zero. The cost scales exponentially, because a small overcapacity will likely only increase the workload of the affected ATM personnel slightly, while a large overcapacity might result in a mistake by the ATM personnel.

The *total overcapacity cost* is the sum of the overcapacity costs of all the air volumes and beginning times:

$$\sum_{av_a \in AV} \sum_{t \in T[a]} overcapacityCost[a, t]$$

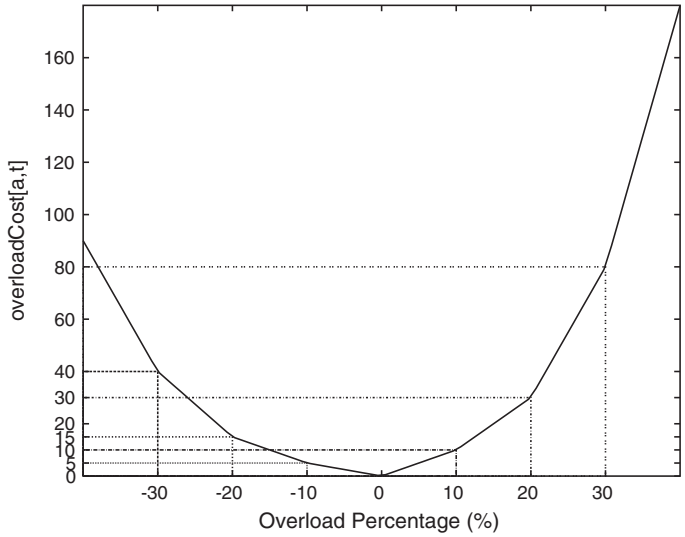

Fig. 6. Piecewise linear function giving the overcapacity cost *overcapacityCost*[*a*, *t*] in terms of the overcapacity percentage.

2.2.4. Constraints on air traffic fairness

The *fairness* of air traffic is here defined by how fairly the departure slots are allocated among the flows. No formal definition of fairness will be given at this point, as fairness is ensured by the search heuristic rather than by the constraint model, so we defer its discussion to Section 3.4.

2.2.5. The objective function

The objective function, to be minimised, is a linear combination of the total delay cost and the total overcapacity cost, where α and β are parameters that can be chosen by the user:

$$\begin{aligned} cost = {} & \alpha \cdot \sum_{f_\ell \in F} delayCost[\ell] \\ & + \beta \cdot \sum_{av_a \in AV} \sum_{t \in T[a]} overcapacityCost[a, t] \end{aligned} \tag{1}$$

Experiment results and feedback from our research partners at the EEC suggest that $\alpha = 6$ and $\beta = 1$ are good values, because there are currently about six times fewer flights than air volumes and time steps. However, they can be changed to reflect any desired balance between a low delay and a low overcapacity.

3. Local search heuristic

Comet [4] also offers a very-high-level *search language* for expressing a search procedure including its heuristics and meta-heuristics. The *Comet* constraint solving architecture takes care of all low-level, tedious, and error-prone computational details, and thereby significantly accelerates the development of effective and efficient search procedures, as well as enormously eases the experimentation with alternative constraints or (meta-)heuristics. Often, this convenience is achieved at no additional cost in run-time compared to a hand-crafted program written in a low-level language. High-level *Comet* programs have

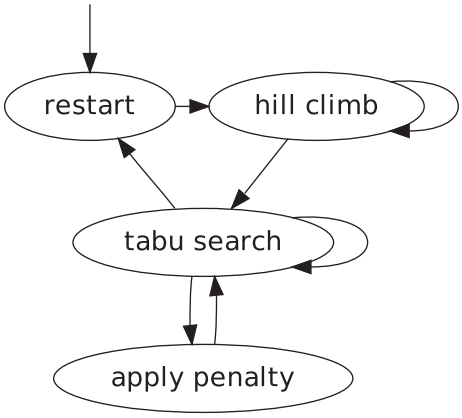

Fig. 7. The generalised local search machine (GLSM) [2] of our tabu search heuristic.

even been reported to out-perform low-level programs that were hand-crafted by experts. Achievements based on *Comet* are listed at the *Comet* web-site and are easily found on the internet.

We here report on using the local search [2] back-end solver of *Comet*, which performs constraint-based local search (CBLS) [4]. This backend was chosen because of the sheer size of the data sets we have to handle. Trying the global search back-ends (tree search interleaved with propagation, and mixed integer linear programming) is considered future work.

In CBLS, constraints are used not only to state the problem but also to control the search. Search heuristics are guided by measures of constraint violation and variable violation. *Constraint violation* measures how close a constraint is to being satisfied. *Variable violation* measures for each decision variable in a constraint the variation of the constraint violation that could be achieved if that variable was suitably modified. Although these terms are not formally defined here, it is possible, for a large number of constraints, to come up with heuristically useful definitions of constraint and variable violations, and to compute them quickly.

Given an *initial assignment* of domain values to all the decision variables, a CBLS *heuristic* iteratively tries to find a better assignment that decreases the amount of constraint violation, by making a *move* to an assignment within the *neighbourhood* of the current assignment, that is a set of assignments that do not differ much from the current one. An assignment with zero (or minimal) constraint violation and an optimal value of the objective function is to be found. *Meta-heuristics* are used to escape local minima, that is when the neighbourhood contains no better assignments than the current one. Since the constraint and variable violations might thus need to be calculated thousands of times so as to pick the best move, and since thousands of moves might be needed, the algorithms and data structures implementing these violation calculations must be very efficient and, where possible, incremental.

We have developed two CBLS heuristics that operate on our constraint model. We here outline the better one of the two heuristics, which performs *tabu search*, and refer the reader to our paper [5] for a detailed description of the other heuristic, which performs *large neighbourhood search*.

The *generalised local search machine* (GLSM) of our tabu search heuristic can be seen in Fig. 7. A GLSM [2] is a finite-state machine that describes a local search heuristic by breaking it down into smaller algorithms, such that each state represents an individual algorithm and the edges represent the conditions for switching between these algorithms.

Our heuristic uses a tabu [9] meta-heuristic for escaping local minima. It uses a slightly modified objective function, which adds a penalty term to (1) in order to guide the heuristic toward a fair traffic flow, where *Penalty* is a set of values maintained by integer invariants (discussed in Section 3.4 below):

$$\begin{aligned} cost = {} & \alpha \cdot \sum_{\ell \in F} delayCost[\ell] \\ & + \beta \cdot \sum_{a \in AV} \sum_{t \in T[a]} overcapacityCost[a, t] \\ & + \sum_{p \in Penalty} p \end{aligned}$$

The heuristic can be summarised in the following steps, where each step and new terminology will be described in further details below:

1. (Re)start the search by assigning each flow rate variable $rate[i]$ a random value in its domain.
2. Hill-climb the current solution, until a local minimum has been reached.
3. Do a single run of tabu search, and then pick a random real number $u \in [0, 1]$. If $u < 0.05$, then pick a flow rate variable $rate[i]$ with an unfair value, add its penalty to the set *Penalty*, and repeat Step 3. Otherwise, if more than 200 iterations have gone by since the last improvement, then go to Step 1, else repeat Step 3.

The heuristic terminates once *maxIter* iterations have been completed, where *maxIter* is initialised to 1 000 and is set to the number of the current iteration plus 500 whenever a new best solution is found, unless this sum is smaller than 1 000.

The main source of diversification (directing the search toward another region of the search space) is Step 1, the main source of intensification (focussing the search on promising regions of the search space) is Step 2, while Step 3 performs a mix of both diversification and intensification.

3.1. The restart mechanism

The restart mechanism is the main source of diversification in our heuristic. It completely (re)starts the search by assigning each flow rate variable $rate[i]$ a random value in its domain, and the $len[i]$ variable of each flow i is reassigned a new value $v_{i+1} - v_i$ where v is a list of sorted random variables in the range [0, 24]. It also clears the *tabu list*, which is the list of most recently visited assignments, stored for the sake of avoiding an untimely return to them.

3.2. Hill-climbing

The hill climbing algorithm is a *non*-greedy algorithm: during each iteration, it picks the *first* move $rate[i] := v$ or $len[i] := v$ such that the objective function is decreased. It does so until no such move can be found, that is until a local minimum has been reached. The method used to find this assignment is through the use of a meta-neighbourhood, which is a circular list of neighbourhoods $\{N_1, \ldots, N_q\}$ (where q is the total number of flow rate and span length variables) that are searched in successive order until an improving assignment is found. Each neighbourhood N_i consists of all moves on flow rate variable $rate[i]$ or a span length variable $len[i]$. The method terminates once a cycle has been completed with no improving assignment found.

3.3. Tabu search

Tabu search [9] is the core of the heuristic. While it is the main contributor of neither intensification nor diversification, it ensures that the neighbourhood of a local minimum has been properly explored so that no improvements have been missed. During each iteration, it searches a neighbourhood (to be defined in the next paragraph) for a best non-taboo move $rate[i] := v$ or $len[i] := v$ and, after making the inverse move taboo for the number of iterations defined by the *tabu tenure*, it performs the assignment $rate[i] := v$ or $len[i] := v$. The only exception to this process is the *aspiration criterion*, which kicks in if the candidate solution is better than any solution found so far. If this is the case, then a move is performed even if it is in the tabu list. Our experiments were made with a tabu tenure $\tau = 20$, which seems to work well on our data sets.

The tabu search uses a specialised neighbourhood (outlined below) that is designed to reduce the most severe air volume overcapacities. It does so using a four-step algorithm that begins by looking at which flights are responsible for said overcapacities, and then finds their respective $rate[i]$ and $len[i]$ variables:

1. The minimum and maximum one-hour time intervals t of $overcapacity[a, t]$ for each air volume a are found. The algorithm then chooses, at random, one of the time intervals (across all air volumes), where each time interval t has a probability $P(a, t)$ proportional to its value compared to its minimum (in the case of a maximum one-hour time interval) or maximum (in the case of a minimum one-hour time interval) value in its domain:
$$P(a,t) = \begin{cases} \max(overcapacity[a,t]) - overcapacity[a,t] & \text{if minimum} \\ overcapacity[a,t] - \min(overcapacity[a,t]) & \text{otherwise} \end{cases}$$
2. All flights in F that contribute (in the case of a maximum) or do not contribute (in the case of a minimum) are collected. Note that flights that always contribute, and never contribute should be omitted in order to avoid creating moves with a very low effect on the objective function.
3. The $rate[i]$ and $len[i]$ decision variables belonging to a flow containing at least one of the collected flights are gathered. Note that decision variables that cannot affect the objective function should be ignored, such as the rate of a span with a length of zero.
4. The neighbourhood consists of all single assignment moves on the gathered decision variables.

The reason for using this type of neighbourhood is because of two facts: (1) We always start the tabu search in a local minimum, and (2) Our model contains a lot of moves that has a very small, or no, effect on the objective function, such as changing the rate of a span with a length of zero. Hence this choice of neighbourhood limits the available moves to a subset of all moves that have a large probability of working on variables that are relevant to the whole network rather than working with variables that change only a very small portion of the objective function while still retaining (with a comparatively low probability) the ability to change any decision variable hence making any part of the solution space reachable.

3.4. Penalty invariant

The apply-penalty state is the part of the heuristic that tries to ensure a high level of fairness of the air traffic flow. It does so by suitably modifying the value of the cost function under a fixed probability after each run of tabu search, such that the flow rate variable $rate[i]$ with the minimum $\frac{rate[i]}{demand[f,i]}$ quotient

is deemed *unfair* and an expression that tries to guide $rate[i]$ toward a fairer value is added to the set *Penalty*. It is an exponential expression that decreases the higher the value of $rate[i]$:

$$\gamma \cdot e^{-8 \cdot \frac{rate[i]}{demand[f,i]}}$$

where γ is a user-definable parameter that controls how aggressively the heuristic should be guided toward fairness. In our experiments, we used $\gamma = 200$, which is only slightly aggressive.

4. Experiment results

Three real-life flight plans, which are comparable to those used by EUROCONTROL when generating the official contingency plans, have been provided by the EEC, and have been used as *training* flight plans:

- A weekday (Friday 2008-06-27), with 261 flows (320 rates), 36 161 flights, and 348 air volumes.
- A Saturday (2008-08-30), with 256 flows (387 rates), 29 842 flights, and 348 air volumes.
- A Sunday (2008-08-31), with 259 flows (397 rates), 31 024 flights, and 348 air volumes.

When translated into a constrained optimisation problem, each instance yields approximately 150 000 constraints and 50 000 decision variables.

All experiments were done on a Linux x86-64 dual-core laptop with 4GB of primary memory, 2MB of L2 cache, and a CPU frequency of 2.2GHz. Under *Comet* version 2.1.1, the tabu search usually terminated after approximately three CPU hours. Keep in mind that this is a proof-of-concept implementation and that significant improvements can be made to improve this speed if it were to be deployed in practice. Similarly, if speed in some cases is more desirable than quality, then the termination criteria could be rewritten to suit such needs.

An overview of the results of our tabu search heuristic for different values of α and β for the weekday flight plan can be seen in Fig. 8. As indicated earlier, the average delay of delayed flights decreases as α grows in comparison to β, and the average overcapacity decreases as β grows in comparison to α. The minor artefacts that do not follow this pattern can be explained by the fact that our heuristic only converges to a local minimum, and therefore some runs might get luckier than others. For the rest of this section we will use $\langle \alpha, \beta \rangle = \langle 6, 1 \rangle$, which provides a reasonable mix of low delay and low overcapacity, however any other assignment to said parameters is equally feasible depending on what properties are desirable in the final solution. Note that the reason why the average overcapacity is still greater than zero, even at very high values of β compared to α, is because it was impossible to fix all overcapacities on the data set we were provided with, as some sectors (usually with capacities less than 2) were overcapacitated by flights that are not contained in any flow (presumably originating from outside Europe) and are therefore unaffected by any change to the decision variables.

Our own comparison between contingency plans generated by our heuristic (denoted by Tabu) and contingency plans generated by the EEC and CFMU human experts (denoted by EEC) can be seen in Table 1, giving the expected take-off delay (in seconds, only for the *delayed* flights) and the 95th percentile thereof, as well as the expected air volume overcapacity percentage (only for the *overcapacitated* sectors and overcapacities greater than zero) and the 95th percentile thereof. We observe that our heuristic outperforms the EEC algorithm on both measures.

This good performance of our heuristic has been validated independently by the EEC and CFMU human experts, using their internal simulation tool COSAAC. They compared our contingency plans and

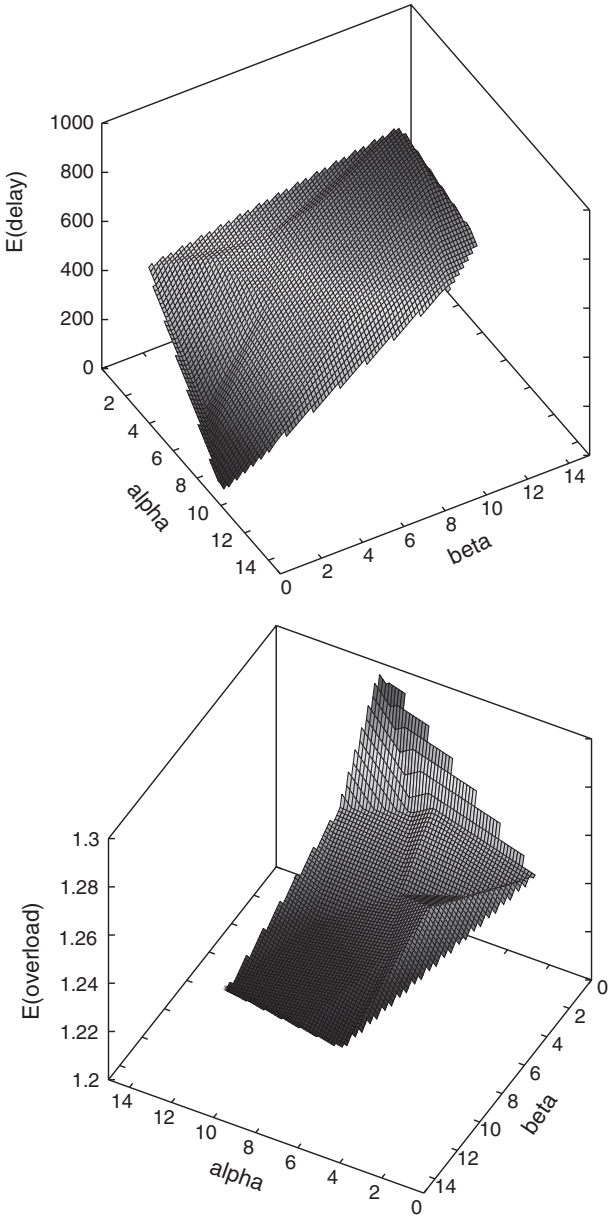

Fig. 8. The resulting average *delay*$[\ell]$ of all delayed flights (above) and average *overcapacity*$[a, t]$ of all overcapacitated air volumes (below) for different values of α and β. Note that the α and β axes are not oriented the same way in both figures.

their contingency plans on realistic *test* flight plans (which were not given to us), though *not* according to the objective function we used in our optimisation, but more realistically according to a CASA-style slot allocation, as if CASA was actually *not* down. Indeed, our objective function and constraints only *simulate* (our understanding of) CASA, as calls to CASA itself for every candidate move in the neighbourhood of every iteration of local search would be prohibitively expensive.

Table 1
Our analysis: Expected take-off delay (in seconds, only for the *delayed* flights) and the 95th percentile thereof, as well as the expected air volume overcapacity percentage (only for the *overcapacitated* air volumes and overcapacities greater than zero) and the 95th percentile thereof, on a weekday, a Saturday, and a Sunday in the European summer 2008 timetable according to the contingency plans generated by the algorithm of the EUROCONTROL Experimental Centre (EEC) and our tabu search heuristic (Tabu)

Contingency plan	Delayed flights%	E(delay)	p_{95}(delay)	Overcapacitated volumes%	E(overcapacity)%	p_{95}(overcapacity)%
EEC 2008-06-27	35	645.6 sec	2340.0 sec	28	29	100
Tabu 2008-06-27	26	310.2 sec	1200.0 sec	27	27	72
EEC 2008-08-30	38	528.1 sec	1800.0 sec	31	23	61
Tabu 2008-08-30	31	316.1 sec	1200.0 sec	30	22	56
EEC 2008-08-31	35	407.0 sec	1500.0 sec	28	29	68
Tabu 2008-08-31	31	345.9 sec	1264.5 sec	26	24	57

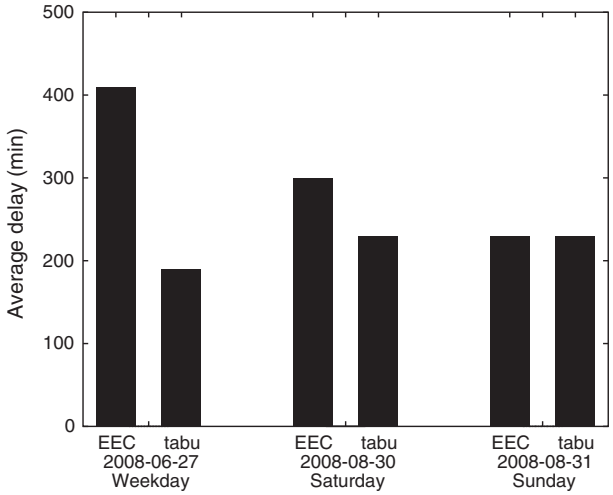

Fig. 9. EEC/CFMU analysis: Average take-off delay (in seconds), only for the *delayed* flights, on a weekday, a Saturday, and a Sunday in the European summer 2008 timetable according to the contingency plans generated by the algorithm of the EUROCONTROL Experimental Centre (EEC), and our tabu search heuristic (tabu).

Figures 9 and 10 respectively give the average take-off delay (in seconds, only for the *delayed* flights) and the average overcapacity percentage (only for the overcapacitated air volumes and overcapacities greater than zero), on a weekday, a Saturday, and a Sunday in the European summer 2008 timetable according to the contingency plans generated by the algorithm of the EUROCONTROL Experimental Centre (EEC) and our tabu search heuristic (tabu). We observe that our heuristic significantly decreases both the average take-off delay (among the delayed flights) and the overcapacity (among the overcapacitated air volumes) of the contingency plans generated by the human experts.

Our tabu search heuristic not only decreases the take-off delay of delayed flights (as seen in Fig. 9) but also delays significantly fewer flights compared to the EEC algorithm (as seen in column 2 of Table 1).

Finally, our tabu search heuristic not only decreases the overcapacity of overcapacitated air volumes (as seen in Fig. 10) but also overcapacitates fewer air volumes compared to the EEC algorithm (as seen in

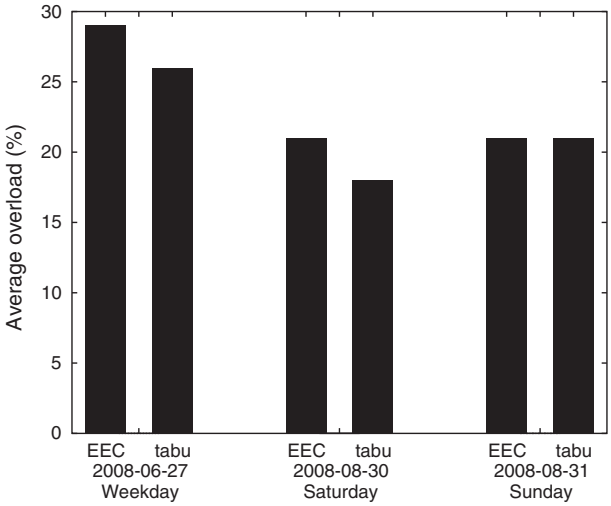

Fig. 10. EEC/CFMU analysis: Average overcapacity percentage, only for the *overcapacitated* air volumes and overcapacities greater than zero, on a weekday, a Saturday, and a Sunday in the European summer 2008 timetable according to the contingency plans generated by the algorithm of the EUROCONTROL Experimental Centre (EEC) and our tabu search heuristic (tabu).

column 5 of Table 1). This is also visible when looking at the histograms of overcapacity in Fig. 11, which shows the number of time-windows with a specific overcapacity for three different contingency plans, our tabu search heuristic (tabu), a plan where all flights depart on time (init), and the contingency plans developed by the EEC (EEC). Notice that our tabu search heuristic significantly decreases the number of overcapacities (which is good), but increases the number of undercapacities (something that we want to avoid if possible). Another, slightly more subtle, difference between the contingency plans is that our tabu search heuristic actually has more flights in the histogram than any of the others, that is the sum of all the bars is higher for our tabu search heuristic. The reason for this is because our tabu search heuristic improves some of the more severe undercapacities ($<70\%$) that are not visible in the figure. The other reason our tabu search heuristic generates more undercapacities than the other two contingency planners is because the total number of capacities, including the ones not visible in the figure, must remain constant, and therefore fewer overcapacities imply more undercapacities. Overall, these figures show that our tabu search heuristic performs better than the human planners in terms of trying to keep all capacities as close to 100% as possible.

5. Conclusion

This work was part of a feasibility study about whether it is possible to automate the development of contingency plans for EUROCONTROL, the *European Organisation for the Safety of Air Navigation*. Our positive results were expected, due to the similarities between the contingency planning problem and scheduling problems, which have been solved successfully using constraint programming technology for a couple decades. It thus seems to be possible to automate contingency planning efficiently enough with constraint programming technology.

Regarding future work, compared to our old heuristic in [6], the new heuristic in this paper still assumes that it is given a set of flows, however this is less of an issue as currently the flows are derived based on geographical location and will likely remain so in order to ease readability of the contingency plan by

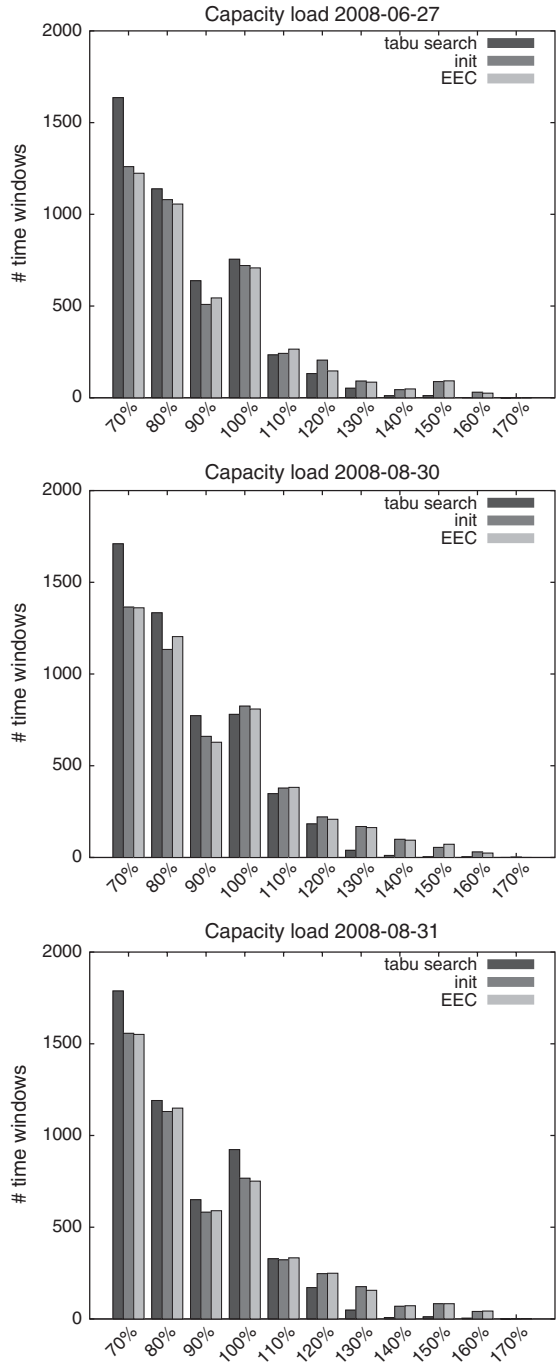

Fig. 11. The capacity balance $\frac{demand[a,t]}{cap_a}$, with values outside the closed interval [70%, 170%] omitted, on the training flight plans for three different contingency plans, an empty contingency plan (init), a contingency plan generated by our tabu heuristic (tabu), and a contingency plan generated by EUROCONTROL (EEC). A favourable distribution would be heavy around 100% and decrease with distance, faster on the right-side (overcapacities) than on the left-side (undercapacities). The reason this is not the case for undercapacities is because during non-rush hours there are not enough flights to fully utilise the airspace, hence undercapacities are inevitable.

humans. For example, the flows in Fig. 1 are split based on whether the destination is west (EBBR1), north or east (EBBR2) of the departure aerodrome; there is also a southbound flow not included in the figure. Some experimentation (not reported here) showed that if inference of flows is desired, then an approach based on both bin-packing and clustering produces acceptable results, but this has not been fully explored by the authors and is therefore future work.

Acknowledgements

We thank Serge Manchon, Elisabeth Petit, Bernard Kerstenne, Leïla Zerrouki, and Marc Dalichampt at the EUROCONTROL Experimental Centre for their feedback on our progress. This work has been financed by the European Organisation for the Safety or Air Navigation (EUROCONTROL) under its Care INO III programme (grant 08-121447-C). The content of the paper does not necessarily reflect the official position of EUROCONTROL on the matter.

References

[1] EUROCONTROL, *Air Traffic Flow & Capacity Management Users Manual*, 15th ed. EUROCONTROL CFMU, March 2011, available at http://www.cfmu.eurocontrol.int/j_nip/cfmu/public/standard_page/library_handbook_supplements.html

[2] H.H. Hoos and T. Stützle, *Stochastic Local Search: Foundations & Applications*. Elsevier/Morgan Kaufmann, 2004.

[3] F. Rossi, P. van Beek and T. Walsh, Eds., *Handbook of Constraint Programming*. Elsevier, 2006.

[4] P. Van Hentenryck and L. Michel, *Constraint-Based Local Search*. The MIT Press, 2005.

[5] K. Sundequist Blomdahl, P. Flener and J. Pearson, Contingency plans for air traffic management, in: *Proceedings of CP'10, the 16th international conference on Constraint Programming*, ser. Lecture Notes in Computer Science, vol. 6308, D. Cohen, Ed. Springer-Verlag, 2010, pp. 643–657.

[6] K. Sundequist Blomdahl, P. Flener and J. Pearson. Contingency plans for air traffic flow and capacity management. In: D. Schaefer (editor), *Proceedings of INO'10, the 9th EUROCONTROL Innovative Research Workshop & Exhibition*, EUROCONTROL Experimental Centre, 2010. ISBN 978-2-87497-021-4.

[7] EUROCONTROL, *General & CFMU Systems*, 15th ed. EUROCONTROL CFMU, March 2011, available at http://www.cfmu.eurocontrol.int/j_nip/cfmu/public/standard_page/library_handbook_supplements.html

[8] A. Cook, G. Tanner, and A. Lawes, The hidden cost of airline unpunctuality, *Journal of Transport Economics and Policy*, January 2011, available at http://www.ingentaconnect.com/content/lse/jtep/pre-prints

[9] F. Glover and M. Laguna, Tabu search, in *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, 1993, pp. 70–150.