Air-Traffic Complexity Resolution in Multi-Sector Planning

Pierre Flener[*], Justin Pearson, and Magnus Ågren
Department of Information Technology
Uppsala University
Box 337, SE – 751 05 Uppsala, Sweden
`{pierreF,justin,agren}@it.uu.se`

Carlos Garcia Avello[1] and Mete Çeliktin[2]
[1] Directorate of ATM Programmes, ATC Applications and Systems Division
[2] Directorate of ATM Strategies, SESAR and ATM Strategy Division
European Organisation for the Safety of Air Navigation (EuroControl)
96 Rue de la Fusée, BE – 1130 Brussels, Belgium
`{carlos.garcia-avello,mete.celiktin}@eurocontrol.int`

Abstract

Using constraint programming, we effectively model and efficiently solve the problem of balancing and minimising the traffic complexities of an airspace of adjacent sectors. The traffic complexity of a sector is here defined in terms of the numbers of flights within it, near its border, and on non-level segments within it. The allowed forms of complexity resolution are the changing of the take-off times of not yet airborne flights, the changing of the remaining approach times into the chosen airspace of already airborne flights by slowing down and speeding up within the two layers of feeder sectors around that airspace, as well as the changing of the levels of passage over way-points in that airspace. Experiments with actual European flight profiles obtained from the Central Flow Management Unit (CFMU) show that these forms of complexity resolution can lead to significant complexity reductions and rebalancing.

1 Introduction

The mission of the European Organisation for the Safety of Air Navigation (EuroControl) is to promote the harmonisation of the different national air-traffic-management (ATM) systems and to lead the development of the next-generation pan-European ATM system capable of handling the foreseen increase in traffic demand. It is the counterpart of the Federal Aviation Administration (FAA) of the USA.

Today, the air-traffic control operation within any control centre rests on a division of the airspace into *sectors*, that is three-dimensional, possibly concave polygonal regions of airspace that are stacked at various altitudes. Because of this fragmented mode of operation, the capacity of a control centre is limited by its sector with the minimum capacity. Indeed, controlling capacity is lost and the total capacity of a control centre could be raised by an early identification of the traffic complexity bottleneck areas and a reorganisation of the traffic patterns such that the traffic complexity is more evenly balanced between its sectors. This has triggered the development of concepts dealing with *multi-sector planning* (MSP), which is based on *traffic complexity management* (TCM) [8], where tools are developed to predict the traffic complexity over several sectors and to manage their overall complexity by anticipating peaks and proposing alternate plans. The *traffic complexity* of a sector is an estimate of the air-traffic controller (ATC) workload of that sector. Here, it is defined in terms of the numbers of flights within it, near its border, and on non-level segments within it. Indeed, each of these positions or segments of a flight requires special attention and procedures to be followed by the ATC. See Section 2.1 for more details on traffic complexity. The initial MSP TCM research and development activities at EuroControl have focused on complexity measurement and complexity prediction [4, 5, 6].

[*]Currently on leave of absence at Sabancı University in İstanbul, Turkey.

This paper presents the first EuroControl research and development results on *complexity resolution*, that is the dynamic modification of flight profiles in order to reduce the predicted complexities over a given time interval of some sectors, thereby avoiding intolerable peaks of ATC workload, as well as, in this multi-sector framework, to balance the complexities of several adjacent sectors, thereby avoiding unacceptable dips of ATC workload and unfair discrepancies between ATC workloads in those sectors. We are only interested here in en-route flights in the upper airspace that follow standard routes, rather than performing free flight.

The tactical rolling-horizon scenario considered is as follows. At a given moment, suitably called *now* below, the (human) complexity manager queries the predicted complexities for an airspace of several adjacent sectors over a time interval that is some 20 to 90 minutes after *now*. Below a look-ahead of 20 minutes, there would not be enough time for the computation and implementation of the complexity-resolved flight profiles. Beyond a look-ahead of 90 minutes, there is too much uncertainty in trajectory prediction, and hence in complexity prediction. If there are ATC workload peaks, dips, or discrepancies over some time sub-interval $[m_1, \ldots, m_2]$ that warrant interference, then the complexity manager launches some complexity resolution process that suitably changes the current flight profiles over that sub-interval, whose length should be about 5 to 10 minutes. The average flight time through a European upper airspace sector is about 8 minutes, hence the proposed resolution window of 5 to 10 minutes is a trade-off between the calculation time required to find a reasonable resolution and the variability of complexity. Also, the implementation of a resolution strategy will have an impact on the evolution of the predicted complexity for later intervals. We thus came to the conclusion that a 5 to 10 minute interval may be a realistic value to start with. A minimum fraction *ff* of the number of flights planned to be in the chosen multi-sector airspace within $[m_1, \ldots, m_2]$ have to be there under the resolved flight profile as well. Indeed, complexity resolution would otherwise just re-plan a maximum of flights to be outside all those chosen sectors, at the expense of increased complexity in the adjacent sectors. This process is to be repeated around the clock approximately every 10 minutes. For this to work, the time spent on computing and implementing the complexity resolutions should not exceed those 10 minutes, and the implementation effort should be offset by the resulting complexity reductions and rebalancing among sectors.

In this work, the *allowed forms of complexity resolution* are as follows, for a flight that eventually enters the given airspace of adjacent sectors:

1. Changing the take-off time of a not yet airborne flight by an integer amount of minutes (not seconds), within the range $[-5, \ldots, +10]$. Today, if the ATM system is overloaded, the Central Flow Management Unit (CFMU) imposes slots on aircrafts of a 15 minute duration and with a $[-5, \ldots, +10]$ minute distribution along the slot time. However, a finer definition of the departure time within that slot allows a decrease in the predicted complexity peaks within the system. This is a powerful and economic means to manage traffic complexity whilst staying within flow control constraints.

2. Changing the remaining approach time into the chosen airspace of an already airborne flight by an integer amount of minutes, but only within the two layers of feeder sectors around that airspace, at a speed-up rate of maximum 1 minute per 20 minutes of approach time, and at a slow-down rate of maximum 2 minutes per 20 minutes of approach time. (Precise definitions of the notions of approach time and feeder sectors will be given in Section 2.2.) The present ATM system lets an aircraft fly its preferred speeds during the cruise phase and most of the descent phase. Typically, the first speed restriction for inbound flights happens below $10,000$ feet. There is quite some room on long flights to change the speed of some aircrafts in order to achieve a different future traffic distribution within sectors that will result in a lower traffic complexity. However, due to aircraft aerodynamics and airline cost index management, the speed control range during the cruise phase may be limited for long-haul flights but remains significant during the descent phase and interesting during the climb phase.

3. Changing the altitude of passage over a point in the chosen airspace by an integer amount of flight levels (hundreds of feet), within the range $[-30, \ldots, +10]$, such that the flight climbs no more than 10 levels per minute, or descends no more than 30 levels per minute if it is a jet, and 10 levels per minute if it is a turbo-prop. Aircrafts on crossing routes at the same level or in an overtake situation on the same route contribute significantly to traffic complexity. Separating the aircrafts vertically at an early stage may reduce drastically the traffic complexity perception of the controller by reducing the amount of time he needs to monitor a given pair of aircrafts. This technique is already used today to increase sector throughput and is named *level capping*. There are cost implications when

changing the cruise level of a flight, but this aspect can be taken into account by reducing the levels of inbound flights before affecting outbound flights and overflights.

Many other forms of complexity resolution can of course be imagined, such as the horizontal re-profiling along alternative routes (from a list of fixed or dynamically calculated routes), or the introduction of even more time variables than just on the entry into the considered multi-sector airspace. Such additional forms of complexity resolution should only be introduced if they are warranted by additional gains and by computational feasibility.

The *objective* of the present work can now initially be stated as follows: Given a set S of adjacent sectors, given moments $now < m_1 < m_2$, and given a fraction ff, return a modification (according to the allowed changes above) of the profiles of the N flights that are planned at *now* to be inside S within the time interval $[m_1, \ldots, m_2]$ such that minimum $\mathit{ff} \cdot N$ of these flights are now planned to be inside S within $[m_1, \ldots, m_2]$ and such that the complexities of the sectors in S are minimised (and ideally better balanced). In practice, an allocated amount *timeOut* of computation time is also given, and we want the best such flight profile changes that can be computed within *timeOut* seconds. This initial formulation of the objective will be refined as we proceed in this paper.

Our *assumptions* and their justifications for realism and impact are as follows:

- We assume that times can be controlled with an accuracy of one minute. Indeed, the resolved flight profiles may have new take-off times for some of the flights originally planned to take off after *now*, or new approach times into the chosen airspace for flights already airborne at *now*, and the rest of their profiles are shifted accordingly, but the computed optimal complexity only holds if these resolved flight profiles are adhered to by the minute, which is a realistic accuracy nowadays. Semantically, the two kinds of time changes amount to a change of the *entry time* of a flight into the chosen airspace, in whose sectors the traffic complexities are actually balanced and minimised. Outside that chosen airspace, the flight profiles are of course to be updated differently according to the actual kind of time change, but our purpose here is only to *measure* the impacts of time changes within that airspace. Consequently, we will from now on only talk about entry-time changes. Although entry times are controlled with an accuracy of one minute, there are no theoretical difficulties with switching to a more fine-grained control of entry times.

- We assume that the flight time along a segment does not change if we restrict the flight-level changes over its end points to be "small", as realistically constrained in the third form of resolution above. Otherwise, we cannot shift the flight profile according to the new entry time and many more time variables would be needed, leading to combinatorial explosion.

The main *contributions* of this paper are as follows:

- We use a more sophisticated notion of air-traffic *complexity* than just the number of flights in a sector at a given moment, so that limiting the capacity of a sector is not just about limiting the number of its flights.

- We introduce air-traffic complexity *resolution* (namely minimisation and balancing) in a *multi*-sector-planning framework.

- We present the probably first *constraint program* for such an airspace planning purpose, previous models being based on mathematical programming technology, such as [2, 10]. It is very efficient, highly readable, and is to serve as an early prototype for further experiments as well as additions and modifications of definitions and constraints. No claim is made that the designed program and the chosen underlying optimisation technology are optimal.

The rest of this paper is organised as follows. In Section 2, we introduce the necessary background information. In Section 3, we then design a constraint program modelling the complexity resolution problem in a multi-sector-planning framework. In Section 4, we report on the experiments we made with that program. Finally, in Section 5, we conclude, discuss related work, and outline future work.

2 Background

In Section 2.1, we define the air-traffic complexity of a sector as an estimate of the air-traffic-controller (ATC) workload of that sector. Next, in Section 2.2, we define various concepts arising for the considered forms of complexity resolution. In Section 2.3, we then recall the notion of Pareto optimisation. Finally, in Section 2.4, we briefly discuss the essence of the constraint programming approach to modelling and solving combinatorial problems.

2.1 Air-Traffic Complexity

Let us first summarise the findings of the previous research and development work at EuroControl (and without our involvement) on air-traffic complexity measurement [4, 5, 6].

Definition 1 *The complexity of a given sector s at a given moment m is based on the following terms:*

- Traffic volume*: Let N_{sec} be the number of flights in s at m.*
- Vertical state*: Let N_{cd} be the number of non-level (climbing or descending) flights in s at m. Indeed, such flights need more ATC monitoring than level flights.*
- Proximity to sector boundary*: Let N_{nsb} be the number of flights that are at most $h_{nsb} = 15$ nautical miles (nm) horizontally or $v_{nsb} = 40$ flight levels (FL) vertically beyond their entry to s, or before their exit from s, at m. Indeed, such flights require ATC interaction with the ATC of the previous or next sector.*

The moment complexity *of sector s at moment m is a normalised weighted sum of these terms:*

$$C(s, m) = (a_{sec} \cdot N_{sec} + a_{cd} \cdot N_{cd} + a_{nsb} \cdot N_{nsb}) \cdot S_{norm} \tag{1}$$

where the sector normalisation constant S_{norm} *characterises the airspace structure, equipment used, procedures followed, etc, of s. This is to ensure that complexity values have a relatively consistent meaning across a wide range of sectors.*

The other parameters initially identified in [5], called 'data-link equipage' (indicating whether the ground-air data-link is digital or not), 'time adjustment' (necessary if the specific flight has a time constraint that will require controller action), 'temporary restriction' (the proportion of the normal capacity of the sector that is predicted to be available, considering the weather, equipment malfunction, military use of shared airspace, etc), 'potentially interacting pairs' (the number of flights that will violate horizontal or vertical separation constraints within the considered sector), and 'aircraft type diversity' (as the diversity in aircraft types has an impact on the speeds they fly, the levels they use, and the rates at which they change altitude, and thus introduces additional complexity by requiring more monitoring), are not used here. Indeed, the 'data-link equipage', 'time adjustment', and 'temporary restriction' parameters, though probably relevant, were not used in the study [6] that determined the weights and normalisation constants for the sectors of the multi-sector airspace considered in our experiments of Section 4, and this simply because there were no data to quantify the weights affecting them. Also, the 'potentially interacting pairs' parameter was used in [6], but (somewhat surprisingly) did not show a good correlation with the complexity value, as estimated by the COCA metric [7]. This has been explained by the fact that the 'traffic volume' and 'vertical state' parameters already capture this impact [6]. Finally, the 'aircraft type diversity' parameter was also used in [6], but also showed a poor correlation with the COCA complexity value, but this may be due to the limited amount of data used in the determination of the weights [6]. The resulting air-traffic complexity measure is maybe simple, but it correlates with a model of (European) ATC workload and yet it has more parameters than other such metrics actually deployed so far for complexity resolution [2, 10]. See Section 5 for a comparison with this related work.

However, as already observed in [6], moment complexity follows has a large variance, with steep rises and falls within just seconds. See the curve for $k = 0$ in Figure 1 (the parameter k will be explained in Definition 2 below), where the x axis is the number of seconds after 11:10 on 23 June 2004, and the y axis is the planned complexity for sector *EBMALNL*. In order to reduce the probability that the complexity-resolved flight profile just falls into a dip of such a curve, we should define complexity so that its curve follows a smoother pattern. This can be achieved by a windows averaging technique, namely defining complexity over a given time interval rather than for a specific moment.

Definition 2 *The* interval complexity *of a given sector s over a given time interval $[m, \ldots, m + k \cdot L]$ is the average of the moment complexities of s at the $k + 1$ sampled moments $m + i \cdot L$, for $0 \leq i \leq k$:*

$$C(s, m, k, L) = \frac{\sum_{i=0}^{k} C(s, m + i \cdot L)}{k + 1} \tag{2}$$

where k is called the smoothing degree*, and L the* time step *between the sampled moments.*

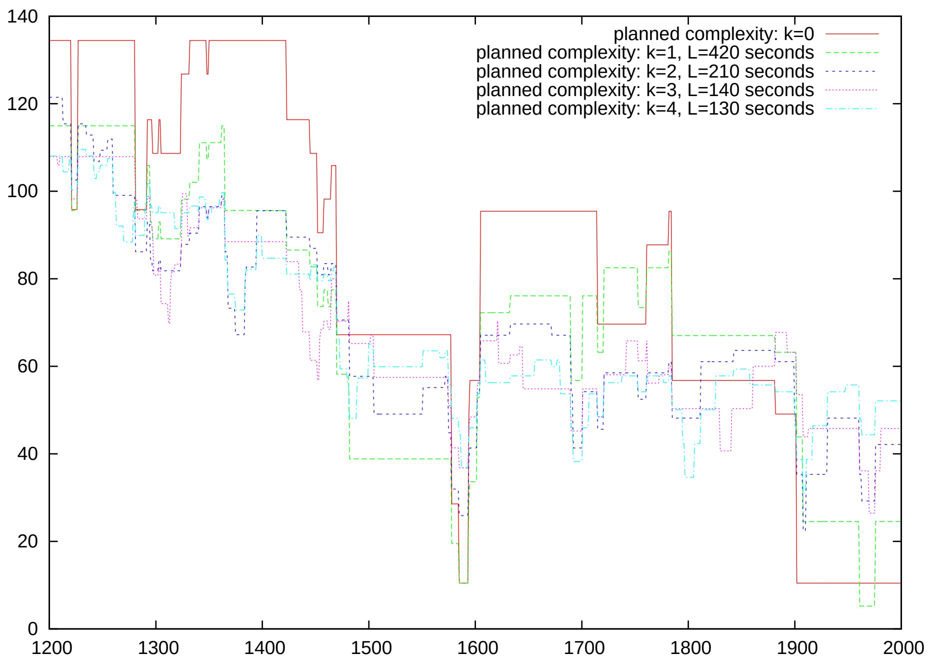

Figure 1: Planned complexity after 11:10 on 23 June 2004 in sector *EBMALNL*

Note that interval complexity reduces to moment complexity when $k = 0$, with the value of L being irrelevant in that case. From now on, all references to complexity are about interval complexity.

For the sake of complexity resolution, our experimentally determined good values of the parameters are $k = 2$ with $L = 210$ seconds, meaning that there are three sampled moments (namely m, $m + 210$, and $m + 420$), spanning a time interval of seven minutes. Under such values, interval complexity (plotted here for the start moment m) follows a much smoother curve than moment complexity, as seen in the curves for $k > 0$ in Figure 1. Any value $k > 2$ does not lead to any further significant smoothening (and would lead to impractically long computation times).

Any value of $k \cdot L$ very different from 420 seconds is not in tune with the average time flights spend on a given segment (within the airspace considered in our experiments of Section 4). Consider how the third form of complexity resolution works, in isolation from the other two forms, referring to Figure 2. For every point p_i on the planned route (drawn as a plain line) for a given flight f within the chosen airspace, there is a maximal range $[-30, \ldots, +10]$, denoted by a dashed vertical double arrow, of flight levels by which the altitude of passage of f over p_i can be changed, depending on the engine type of f and the distances to the previous point p_{i-1} and next point p_{i+1}, if any. Suppose, for $k = 2$, that the three sampled moments m, $m + L$, and $m + 2L$ fall as indicated on the horizontal time axis, namely on a climbing segment $[p_2, p_3]$, on a level segment $[p_3, p_4]$, and on another climbing segment $[p_4, p_5]$, respectively. Complexity resolution tries to level off the other two climbing segments, by making already $[p_1, p_2]$ reach the level h of $[p_2, p_3]$, or by making $[p_5, p_6]$ start only from h, if not by lowering or raising h for as many as possible of the points p_2 to p_5, depending on the lengths of $[p_1, p_2]$ and $[p_5, p_6]$. Indeed, the *sum* of the N_{cd} terms for the involved sectors would then decrease by the number of these steeper climbs that are compatible with the climbing performance of f. The dot-dashed alternative route in the figure assumes that both sampled climbing segments could be levelled off. Now, if L is too small, then several sampled moments might fall onto the same non-level segment, thereby making a single change achieve a lot of complexity reduction. Conversely, if L is too large, then the sampled complexity values concern segments that are too far apart for their average to be meaningful.

2.2 Feeder Sectors and Approach Times

The second considered form of complexity resolution is the slowing down and speeding up of already airborne flights, but only within the two layers of sectors around the chosen multi-sector airspace, so as to delay or advance their entries into that airspace. We call those surrounding sectors the feeder sectors.

Definition 3 *A* feeder sector *s of a flight f is a sector that is a neighbour of a sector of the chosen*

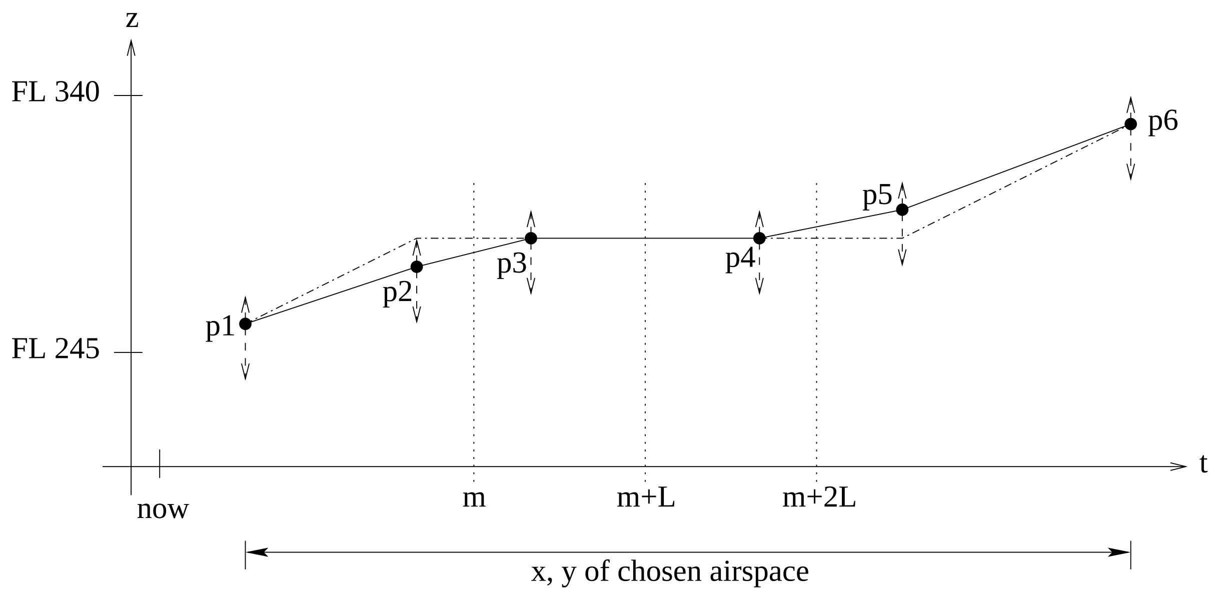

Figure 2: Planned profile (plain line) and resolved profile (dot-dashed line) that minimises the number of climbing segments for a flight at the three sampled moments m, m+L, and m+2L.

airspace, or a neighbour of a neighbour of such a sector, such that f is planned to fly through s before entering that chosen airspace.

Essentially, the time still to be spent in the feeder sectors is called the approach time, defined next.

Definition 4 *For a given moment, called now, the* approach time *of a flight f that has taken off at or before now but that has not yet entered the chosen airspace is the duration that f is planned to fly within its feeder sectors between now and its entry into the chosen airspace. If f has not yet entered its feeder sectors at time now, then the approach time of f is the entire planned duration of flight within its feeder sectors; otherwise, the approach time of f is the remaining duration of flight within its feeder sectors. Formally, if a flight f with take-off time $f.timeTakeOff$ is planned to enter the chosen airspace at time $t_C[f]$ and its first feeder sector at time $t_F[f] \leq t_C[f]$ such that $f.timeTakeOff \leq now < t_C[f]$, then the approach time of f is*

$$a[f] = t_C[f] - \max(now, t_F[f]). \tag{3}$$

The restriction to two layers of feeder sectors around the chosen airspace aims at propagating only a reasonable number of updates to the upstream air-traffic control centres for each updated flight profile, as well as at leaving sufficient time to implement those updates. Note that the actual complexity will be measured over a time interval $[m, \ldots, m + k \cdot L]$, with $now \leq m$. Furthermore, there is too much uncertainty involved to propagate changes more than two sectors upstream and still hope the new entry time into the chosen airspace will actually be as intended. Finally, at the moment, there is no practical way for an ATC centre to modify a flight long before it reaches its own airspace.

2.3 Pareto Optimisation

Essentially, we are dealing with a multi-objective minimisation problem. We are given a number of sectors $s_1, s_2, \ldots, s_n$ and wish to minimise the vector of their complexities with respect to a resolution R:

$$\langle C_R(s_1, m, k, L), \ldots, C_R(s_n, m, k, L) \rangle$$

Pareto efficiency is a concept that originates from economics but is now a widely used concept for multi-objective minimisation problems within engineering. A vector of complexities is said to be *Pareto optimal*, or *Pareto efficient*, if no element can be made less complex without making some other element more complex. More formally, a vector

$$\langle C_{R^*}(s_1, m, k, L), \ldots, C_{R^*}(s_n, m, k, L) \rangle$$

is *Pareto optimal* for a resolution R^* if and only if there is no resolution R such that for all j we have

$$C_R(s_j, m, k, L) \leq C_{R^*}(s_j, m, k, L)$$

but for some j we have

$$C_R(s_j, m, k, L) < C_{R^*}(s_j, m, k, L).$$

In general, there is more than one Pareto-optimal solution to a problem.

One of the standard techniques for solving Pareto optimisation problems is by combining the multiple objectives into a single objective using a weighted sum:

$$\sum_{i=1}^{n} \alpha_i \cdot C_R(s_i, m, k, L)$$

for some weights $\alpha_i > 0$. Minimising the weighted sum produces a solution that is Pareto optimal. Different weights α_i produce different Pareto-optimal solutions with different tradeoffs. In practice, one often takes $\alpha_i = 1$. Although the weighted sum only guarantees to minimise convex parts of the set of Pareto-optimal points, in practice non-convex sets are seldom found. Further, in this work, we are only interested in a resolution that reduces complexity, but not in the structure of the set of Pareto-optimal points, therefore any Pareto-minimal resolution is sufficient.

We have experimented with other Pareto weights and with other (non-Pareto) optimisation criteria as well, such as minimising the worst resolved complexity among the sectors of the chosen airspace, or minimising the worst discrepancy among the resolved complexities of the sectors of the chosen airspace. However, the results were not as good, including often a poorer balancing of the resolved complexities across those sectors, or even took much more computation time to get.

2.4 Constraint Programming

Constraint programming (CP) is a recent technology, offering a set of methods and tools for modelling and solving constraint problems [1]. Constraint programming originated in the late 1980s from artificial intelligence and computer science. Its solution methods are orthogonal but complementary to those of other technologies, such as mathematical programming (MP), which originated from operations research in the 1950s. In some cases CP has produced better solutions and has been quicker, while in other cases MP was better or a cooperation of CP and MP gave the best results [3, 12].

A *constraint problem* has a set of *variables*, each with a specified *domain* of possible values, together with a set of *constraints* (or relations over these variables) that restrict the possible combinations of values for these variables, and optionally an *objective function* over these variables whose value is to be optimised. Many problems can be modelled as constraint problems, such as problems in configuration, control, design, diagnostic, finance, logistics, resource allocation, scheduling, and the natural sciences.

The task of a *constraint solver* is to find, if possible, a value for each variable within its domain such that every constraint is satisfied, and, in the case of a constraint optimisation problem, such that the value of the objective function is optimised. A constraint solver works by a combination of propagation and *backtrack search*, while using *heuristics* to guide the search. *Propagation* is an operation that is local to a particular constraint and takes the current domains of its variables and removes (in polynomial time in the number of variables and their domain sizes) some, but not necessarily all, values that cannot possibly participate in any solution to the considered constraint. This achieves some desired form of local *consistency* on the variables of the considered constraint, such as *hyper-arc* consistency (the domain of the considered variable is a set and, for every value, there is a tuple of domain values for the other variables such that the considered constraint holds) or *bounds consistency* (the domain of the considered variable is an interval and, for the smallest and largest values only, there is a tuple of domain values for the other variables such that the considered constraint holds). The removed values are restored upon backtracking. When all initial propagation has been done, and until global consistency has been achieved on the conjunction of all the constraints, the solver iteratively searches the remaining search tree by first heuristically making a *decision*, such as assigning to some variable a value within its current domain, and then invoking propagation in the presence of this new constraint.

Industry-strength *CP environments* (such as the commercial CHIP, ILOG OPL, ILOG Solver, and SICStus Prolog, or the open-source Choco, ECLiPSe, and Gecode) offer very rich modelling languages and search languages. In particular, the constraints are not limited to some form, such as the linear inequalities of MP, and can be expressed in terms of powerful primitive constraints, including the so-called *global constraints*, for which effective polynomial-time propagation algorithms have been designed. The prototypical global constraint is *allDifferent*$(x_1, \ldots, x_n)$, which requires the variables $x_1, \ldots, x_n$ to take different values; achieving arc consistency for this global constraint is more efficient and prunes many more values than achieving arc consistency for the equivalent conjunction of $\frac{n(n-1)}{2}$ binary inequality constraints. A constraint problem is thus essentially modelled as a logical combination of (global) constraints, whose polynomial-time propagation algorithms combine in practice very well to address the usually NP-complete nature of the underlying problem. In consequence, constraint models are often very close to the intuitive statement of the problem, in the sense that many variables correspond directly to features of the problem and that the constraints naturally model the relationships between those features. This simplifies the design and maintenance of the model, the interpretation of solutions, and the choice of adequate heuristics, based on the modeller's domain knowledge or preferences. There is a clean separation of the problem model from the solution process, hence changes to either of those can easily be incorporated. Finally, modern CP environments offer *explanation* facilities, so as to explain why a particular solution, or no solution, was found.

3 The Constraint Program

A constraint optimisation program starts with the declarations of the inputs — called the *instance data*, as they give a particular instance of a general constraint problem — and outputs — called the *variables*, as a decision has to be made for each of them about which *value* it takes within its *domain*. Then comes the *objective*, which consists of an expression — called the *objective function* — that has to be optimised and some conditions — called the *constraints* — subject to which the optimisation takes place. Finally, there is an optional user-defined procedure — called the *search procedure* — that indicates under what *ordering heuristics* the search tree shall be explored when propagation has reached its fix-point and before the default search procedure of the constraint solver takes over.

3.1 The Instance Data

The parameters of the constraint program are as follows:

- *maxEarly* is the maximum non-negative-integer amount of minutes that a flight can take off before its planned time; a typical value is 5.
- *maxLate* is the maximum non-negative-integer amount of minutes that a flight can take off after its planned time; a typical value is 10.
- *maxSlowDown* is the maximum non-negative-integer amount of minutes per 20 minutes of flight that an airborne flight can be slowed down; a typical value is 2.
- *maxSpeedUp* is the maximum non-negative-integer amount of minutes per 20 minutes of flight that an airborne flight can be speeded up; a typical value is 1.
- *maxDownJet* is the maximum non-negative-integer amount of flight levels that a jet can descend per minute; a typical value is 30.
- *maxDownTurbo* is the maximum non-negative-integer amount of flight levels that a turbo-prop can descend per minute; a typical value is 10.
- *maxDown* is the maximum non-negative-integer amount of flight levels by which the altitude of a flight over a point can be decreased, even if a larger decrease is allowed by the previous two parameters, since we do not want to allow too radical changes of flight profiles; a typical value is 30.
- *maxUpJet* is the maximum non-negative-integer amount of flight levels that a jet can climb per minute; a typical value is 10.
- *maxUpTurbo* is the maximum non-negative-integer amount of flight levels that a turbo-prop can climb per minute; a typical value is 10.
- *maxUp* is the maximum non-negative-integer amount of flight levels by which the altitude of a flight over a point can be increased, even if a larger increase is allowed by the previous two parameters, since we do not want to allow too radical changes of flight profiles; a typical value is 10.

- *lookahead* is a non-negative-integer amount of minutes; a typical value is a multiple of 10 in the range $[20, \ldots, 90]$.
- *nowHM* is the time, given as an (hour, minute) pair, at which a resolved scenario is wanted with a forecast of *lookahead* minutes.
  Let
  $$now = nowHM$$
  and
  $$m = nowHM + lookahead$$
  be both expressed as the number of seconds since 00:00 of the first day of the traffic sample. Observe that $now \leq m$.
- k is the smoothing degree, that is the number of time steps into the future from m where the (original) complexity of the considered sector is measured; a good value is 2.
- L is the length, in seconds, of those time steps; a good value for complexity resolution is 200.
- *ff* is the minimum fraction of the sum of the numbers of flights planned to be in the chosen multi-sector airspace at the sampled moments $m + i \cdot L$, for all $0 \leq i \leq k$, that have to be there in the resolved flight profile as well.
- *timeOut* is the maximal non-negative-integer amount of seconds that should be spent on computations before returning the currently best feasible solution.

The constants used by the constraint program are as follows:

- *OurSectors* is the set of identifiers of the sectors of the chosen airspace.
- *AllSectors* is *OurSectors* plus the set of identifiers of the feeder sectors.
- *Engine* is the set of chosen engine types.
- *Sector* is a 1d array of $(bottomFL, topFL, a_{sec}, a_{cd}, a_{nsb}, S_{norm})$ tuples, indexed by *AllSectors*, such that the sector $Sector[s]$ stretches vertically between flight levels *bottomFL* and *topFL*, and has its complexity parameterised by a_{sec}, a_{cd}, a_{nsb}, and S_{norm}.
- $hv_{nsb} = 120$ is the maximal number of seconds of flight a flight must be horizontally or vertically beyond the entry to, or before the exit from, a sector s in order to qualify for being near the sector boundary of s. See the comments before the constraint (17) below for more information.

Raw CFMU (Central Flow Management Unit) flight profiles do not contain all the sector coordination (entry and exit) points of all the flights. As we would otherwise not be able to measure accurately any term of the moment-complexity expression (1), the raw CFMU flight profiles were first extended by EuroControl DAP/DIA and then processed into four data structures and three constants:

- *Flights* is the set of all *relevant flights*, given as $(flightId, engineType, timeTakeOff)$ triples, that is the flights that are planned to be in one of the chosen sectors at a sampled moment $m + i \cdot L$, for some $0 \leq i \leq k$.
- Let $n = |Flights|$. For the rest of the program to make sense, we must have $n > 0$.
- Let N be the sum of the numbers of flights that are planned to be in one of the chosen sectors at the sampled moments $m + i \cdot L$, for all $0 \leq i \leq k$.
- *SectorFlights* is a 1d array of subsets of *Flights*, indexed by *AllSectors*, such that $SectorFlights[s]$ is the set of relevant flights that are planned to be in s at time m.
- *FlightPoints* is the set of all *relevant flight-point pairs*, given as $(flightId, pointId, timeOver, level)$ tuples, that is the points flown over by one of the relevant flights in one of the chosen sectors. Note that the sector identifier is not part of the identifier of a flight-point pair; otherwise, some sector entry or exit points would appear twice in *FlightPoints*.
- Let $q = |FlightPoints|$. For the rest of the program to make sense, we must have $q > 0$.

- *Profile* is a 2d array of subsets of *FlightPoints*, indexed by *AllSectors* and *Flights*, such that *Profile*$[s, f]$ gives the point profile of relevant flight f in sector s. Observe that the first element in *Profile*$[s, f]$, denoted by *first*(*Profile*$[s, f]$), designates the entry of flight f into sector s, and that its last element, denoted by *last*(*Profile*$[s, f]$), designates the exit of f from s.

where *engineType* $\in$ *Engine*, *flightId* and *level* are integers, *pointId* is a string, and *timeTakeOff* and *timeOver* are not given as dd/mm/yyyy hh:mm:ss but expressed as the number of seconds since 00:00 of the first day of the traffic sample.

We use the dot notation from object-oriented programming for designating a field of a tuple. For instance, $f.\mathit{timeTakeOff}$ designates the planned take-off time of flight f, while *first*(*Profile*$[s, f]$).*timeOver* designates the entry time of flight f into sector s.

3.2 The Variables

Let

$$\Delta T = [-\mathit{maxEarly} \cdot 60, \ldots, +\mathit{maxLate} \cdot 60]$$

be the integer range of take-off-time changes, expressed in multiples of 60 seconds, i.e., minutes.
Let

$$\Delta H = [-\mathit{maxDown}, \ldots, +\mathit{maxUp}]$$

be the integer range of flight-level changes, expressed in flight levels.
Let

$$\mathit{Index} = [0, \ldots, k]$$

be the integer range of moment indices.
Let

$$\mathit{Fcount} = [0, \ldots, n]$$

be the integer range of numbers of flights.
Finally, let

$$\mathit{Bool} = [0, 1]$$

be the integer range of Booleans, such that 1 represents truth and 0 represents falsity.
The variables of our constraint program and their domains are as follows:

- δT is a 1d array of n integer variables for entry-time changes, indexed by *Flights*.

- δH is a 1d array of q integer variables for flight-level changes in ΔH, indexed by *FlightPoints*.

- N_{sec} is a 2d array of integer variables in *Fcount*, indexed by *Index* and *OurSectors*, such that $N_{sec}[i, s]$ is the number of flights in sector s at moment $m + i \cdot L$ when the entry-time changes are as in δT.

- N_{cd} is a 2d array of integer variables in *Fcount*, indexed by *Index* and *OurSectors*, such that $N_{cd}[i, s]$ is the number of flights on non-level segments in sector s at moment $m + i \cdot L$ when the entry-time and flight-level changes are as in δT and δH.

- N_{nsb} is a 2d array of integer variables in *Fcount*, indexed by *Index* and *OurSectors*, such that $N_{nsb}[i, s]$ is the number of flights near the boundary of sector s at moment $m + i \cdot L$ when the entry-time changes are as in δT.

- *complexity* is a 2d array of integer variables, indexed by *Index* and *OurSectors*, such that *complexity*$[i, s]$ is the moment complexity of sector s at moment $m + i \cdot L$.

- *complexitySum* is a 1d array of integer variables, indexed by *OurSectors*, such that *complexitySum*$[s]$ is the sum of the moment complexities of sector s at the moments $m + i \cdot L$, for each $i \in \mathit{Index}$. By formula (2) in Definition 2, it suffices to divide *complexitySum*$[s]$ by $k + 1$ to get the interval complexity of s over the time interval $[m, \ldots, m + k \cdot L]$. We postpone this division until display time, as constraint reasoning is faster on integers than floats. Indeed, the objective function to be minimised (see Section 3.4) is the sum of the interval complexities of all the chosen sectors, and it suffices to minimise the sum of those values before that division, as $k + 1$ is a constant.

3.3 The Constraints

All relevant flights planned to take off at or before *now* have taken off exactly according to their profile, but their approach times (within the feeder sectors) can be modified. All other relevant flights can necessarily only be changed to take off after *now*. To formalise all this, let $t_C[f]$ be the time flight f was originally planned to enter the chosen airspace:

$$t_C[f] = \mathrm{Min}_{s \in \mathit{OurSectors}:\mathit{Profile}[s,f]\neq\emptyset}\ \mathit{first}(\mathit{Profile}[s,f]).\mathit{timeOver}$$

Also, let $t_F[f]$ be the time flight f was originally planned to enter its first feeder sector:

$$t_F[f] = \mathrm{Min}_{s \in \mathit{AllSectors}:\mathit{Profile}[s,f]\neq\emptyset}\ \mathit{first}(\mathit{Profile}[s,f]).\mathit{timeOver}$$

Finally, let $a[f]$ be the approach time of a flight f with $f.\mathit{timeTakeOff} \leq \mathit{now} < t_C[f]$. As already stated by formula (3) in Definition 4, we have:

$$a[f] = t_C[f] - \max(\mathit{now}, t_F[f])$$

Formally, the constraint above now becomes:

$$\begin{array}{l} \forall f \in \mathit{Flights}\ . \\ \text{if } f.\mathit{timeTakeOff} \leq \mathit{now} \\ \text{then if } \mathit{now} < t_C[f] \\ \qquad \text{then } -\mathit{maxSpeedUp} \cdot \frac{a[f]}{20} \leq \delta T[f] \leq \mathit{maxSlowDown} \cdot \frac{a[f]}{20} \\ \qquad \text{else } \delta T[f] = 0 \\ \text{else } \delta T[f] \in \Delta T \wedge f.\mathit{timeTakeOff} + \delta T[f] > \mathit{now} \end{array} \tag{4}$$

Flight-point pairs flown over at or before *now* can (currently) not have their flight levels changed:

$$\forall p \in \mathit{FlightPoints} : p.\mathit{timeOver} \leq \mathit{now}\ .\ \delta H[p] = 0 \tag{5}$$

Changed flight levels stay within the bounds of the sector, as flights cannot be re-routed through a lower or higher sector:

$$\begin{array}{c} \forall s \in \mathit{OurSectors}\ .\ \forall f \in \mathit{SectorFlights}[s]\ .\ \forall p \in \mathit{Profile}[s,f]\ . \\ \mathit{Sector}[s].\mathit{bottomFL} \leq p.\mathit{level} + \delta H[p] \leq \mathit{Sector}[s].\mathit{topFL} \end{array} \tag{6}$$

The next three constraints define the $N_{sec}[i,s]$, $N_{cd}[i,s]$, and $N_{nsb}[i,s]$ variables. A flight segment is considered open at its target-point: a flight that is exactly over a point p is considered to be on the segment starting at p, rather than on the segment ending at p, or even on both. This implies that such a flight is only counted once within N_{cd} if both involved segments are non-level. Proximity to a sector boundary is approximated by being at most $hv_{nsb} = 120$ seconds of flight beyond the entry to, or before the exit from, the sector, while still being in that sector. This approximation [5] only holds for en-route airspace, which is the case here. Recall that the first element in $\mathit{Profile}[s,f]$ denotes the entry of flight f into sector s, and that its last element denotes the exit of f from s. Also, let p' designate $\mathit{Profile}[s,f].\mathit{next}(p)$, that is the element after p in the profile of flight f in sector s:

$$\begin{array}{l} \forall i \in \mathit{Index}\ .\ \forall s \in \mathit{OurSectors}\ . \\ N_{sec}[i,s] = \left| \left\{ f \in \mathit{SectorFlights}[s] \;\middle|\; \begin{array}{c} \mathit{first}(\mathit{Profile}[s,f]).\mathit{timeOver} \leq m + i \cdot L - \delta T[f] \\ < \mathit{last}(\mathit{Profile}[s,f]).\mathit{timeOver} \end{array} \right\} \right| \end{array} \tag{7}$$

$$\begin{array}{l} \forall i \in \mathit{Index}\ .\ \forall s \in \mathit{OurSectors}\ . \\ N_{cd}[i,s] = \left| \left\{ f \in \mathit{SectorFlights}[s] \;\middle|\; \begin{array}{c} \exists p \in \mathit{Profile}[s,f] : p \neq \mathit{last}(\mathit{Profile}[s,f])\ . \\ p.\mathit{timeOver} \leq m + i \cdot L - \delta T[f] < p'.\mathit{timeOver} \wedge \\ p.\mathit{level} + \delta H[p] \neq p'.\mathit{level} + \delta H[p'] \end{array} \right\} \right| \end{array} \tag{8}$$

$$\begin{array}{l} \forall i \in \mathit{Index}\ .\ \forall s \in \mathit{OurSectors}\ . \\ N_{nsb}[i,s] = \left| \left\{ f \in \mathit{SectorFlights}[s] \;\middle|\; \begin{array}{c} 0 \leq m + i \cdot L - (\mathit{first}(\mathit{Profile}[s,f]).\mathit{timeOver} + \delta T[f]) \leq hv_{nsb} \\ \wedge\ m + i \cdot L < \mathit{last}(\mathit{Profile}[s,f]).\mathit{timeOver} + \delta T[f] \\ \vee \\ 0 < \mathit{last}(\mathit{Profile}[s,f]).\mathit{timeOver} + \delta T[f] - (m + i \cdot L) \leq hv_{nsb} \\ \wedge\ \mathit{first}(\mathit{Profile}[s,f]).\mathit{timeOver} + \delta T[f] \leq m + i \cdot L \end{array} \right\} \right| \end{array} \tag{9}$$

As usual, logical conjunction ($\wedge$) has higher precedence than logical disjunction ($\vee$).

No flight has to climb more than *maxUpJet* or *maxUpTurbo* levels per minute or descend more than *maxDownJet* or *maxDownTurbo* levels per minute:

$$\begin{gathered}\forall s \in \mathit{OurSectors} \;.\; \forall f \in \mathit{SectorFlights}[s] \;.\; \forall p \in \mathit{Profile}[s,f] : f.\mathit{engineType} = \mathit{jet} \wedge p \neq \mathit{last}(\mathit{Profile}[s,f]) \;.\\ -(p'.\mathit{timeOver} - p.\mathit{timeOver}) \cdot \mathit{maxDownJet} \leq ((p'.\mathit{level} + \delta H[p']) - (p.\mathit{level} + \delta H[p])) \cdot 60\\ \leq (p'.\mathit{timeOver} - p.\mathit{timeOver}) \cdot \mathit{maxUpJet}\\ \wedge\\ \forall s \in \mathit{OurSectors} \;.\; \forall f \in \mathit{SectorFlights}[s] \;.\; \forall p \in \mathit{Profile}[s,f] : f.\mathit{engineType} = \mathit{turbo} \wedge p \neq \mathit{last}(\mathit{Profile}[s,f]) \;.\\ -(p'.\mathit{timeOver} - p.\mathit{timeOver}) \cdot \mathit{maxDownTurbo} \leq ((p'.\mathit{level} + \delta H[p']) - (p.\mathit{level} + \delta H[p])) \cdot 60\\ \leq (p'.\mathit{timeOver} - p.\mathit{timeOver}) \cdot \mathit{maxUpTurbo}\end{gathered} \tag{10}$$

A fraction of minimum *ff* of the sum N of the numbers of flights that are planned to be in one of the chosen sectors at the sampled moments $m + i \cdot L$ must remain in one of the chosen sectors:

$$\sum_{i \in \mathit{Index}} \sum_{s \in \mathit{OurSectors}} N_{sec}[i,s] \geq \lceil \mathit{ff} \cdot N \rceil \tag{11}$$

The next constraint defines the *complexity*$[i,s]$ values:

$$\begin{gathered}\forall i \in \mathit{Index} \;.\; \forall s \in \mathit{OurSectors} \;.\; \mathit{complexity}[i,s] =\\ (\mathit{Sector}[s].a_{sec} \cdot N_{sec}[i,s] + \mathit{Sector}[s].a_{cd} \cdot N_{cd}[i,s] + \mathit{Sector}[s].a_{nsb} \cdot N_{nsb}[i,s]) \cdot \mathit{Sector}[s].S_{norm}\end{gathered} \tag{12}$$

Finally, the last constraint defines the *complexitySum*$[s]$ values:

$$\forall s \in \mathit{OurSectors} \;.\; \mathit{complexitySum}[s] = \sum_{i \in \mathit{Index}} \mathit{complexity}[i,s] \tag{13}$$

3.4 The Objective Function

The chosen objective function corresponds to a Pareto optimisation with unit weights: we minimise the sum of the interval complexities of the chosen sectors. Formally, it suffices to minimise the following expression:

$$\sum_{s \in \mathit{OurSectors}} \mathit{complexitySum}[s] \tag{14}$$

if we divide its value by $k + 1$ before displaying it, as explained near the end of Section 3.2.

The discussed constraints and objective function are a faithful formalisation, in a first-order logic, of how to balance and minimise the complexities of a set of chosen sectors over a given time interval. Note that, as argued in Section 2.4, many variables correspond directly to features of the problem and that the constraints naturally model the relationships between those features.

3.5 Refinement of the Variables and Constraints

In practice, a much better propagation and thus a much faster computation are achieved if we introduce the following additional variables:[1]

- b_{sec} is a 3d array, indexed by *Index*, *OurSectors*, and *Flights*, of integer variables in *Bool*, such that $b_{sec}[i,s,f] = 1$ if flight f is in sector s at moment $m + i \cdot L$ when its entry-time change is $\delta T[f]$.
- b_{cd} is a 3d array, indexed by *Index*, *OurSectors*, and *Flights*, of integer variables in *Bool*, such that $b_{cd}[i,s,f] = 1$ if flight f is on a non-level segment in sector s at moment $m+i\cdot L$ when its entry-time change is $\delta T[f]$ and the flight-level changes are as in δH.
- b_{nsb} is a 3d array, indexed by *Index*, *OurSectors* and *Flights*, of integer variables in *Bool*, such that $b_{nsb}[i,s,f] = 1$ if flight f is near the boundary of sector s at moment $m + i \cdot L$ when its entry-time change is $\delta T[f]$.

[1]The reader may safely skip this sub-section.

The following three additional constraints then define these new variables:

$$\begin{aligned}
&\forall i \in \mathit{Index} \ . \ \forall s \in \mathit{OurSectors} \ . \ \forall f \in \mathit{Flights} \ . \\
&\text{if } f \in \mathit{SectorFlights}[s] \\
&\text{then } b_{sec}[i,s,f] = 1 \leftrightarrow \begin{pmatrix} \mathit{first}(\mathit{Profile}[s,f]).\mathit{timeOver} \leq m + i \cdot L - \delta T[f] \\ < \mathit{last}(\mathit{Profile}[s,f]).\mathit{timeOver} \end{pmatrix} \\
&\text{else } b_{sec}[i,s,f] = 0
\end{aligned} \tag{15}$$

$$\begin{aligned}
&\forall i \in \mathit{Index} \ . \ \forall s \in \mathit{OurSectors} \ . \ \forall f \in \mathit{Flights} \ . \\
&\text{if } f \in \mathit{SectorFlights}[s] \\
&\text{then } b_{cd}[i,s,f] = 1 \leftrightarrow \begin{pmatrix} \exists p \in \mathit{Profile}[s,f] : p \neq \mathit{last}(\mathit{Profile}[s,f]) \ . \\ p.\mathit{timeOver} \leq m + i \cdot L - \delta T[f] < p'.\mathit{timeOver} \wedge \\ p.\mathit{level} + \delta H[p] \neq p'.\mathit{level} + \delta H[p'] \end{pmatrix} \\
&\text{else } b_{cd}[i,s,f] = 0
\end{aligned} \tag{16}$$

$$\begin{aligned}
&\forall i \in \mathit{Index} \ . \ \forall s \in \mathit{OurSectors} \ . \ \forall f \in \mathit{Flights} \ . \\
&\text{if } f \in \mathit{SectorFlights}[s] \\
&\text{then } b_{nsb}[i,s,f] = 1 \leftrightarrow \begin{pmatrix} 0 \leq m + i \cdot L - (\mathit{first}(\mathit{Profile}[s,f]).\mathit{timeOver} + \delta T[f]) \leq hv_{nsb} \\ \wedge \ m + i \cdot L < \mathit{last}(\mathit{Profile}[s,f]).\mathit{timeOver} + \delta T[f] \\ \vee \\ 0 < \mathit{last}(\mathit{Profile}[s,f]).\mathit{timeOver} + \delta T[f] - (m + i \cdot L) \leq hv_{nsb} \\ \wedge \ \mathit{first}(\mathit{Profile}[s,f]).\mathit{timeOver} + \delta T[f] \leq m + i \cdot L \end{pmatrix} \\
&\text{else } b_{nsb}[i,s,f] = 0
\end{aligned} \tag{17}$$

while the next three constraints then replace the constraints (7) to (9) in order to re-define the $N_{sec}[i,s]$, $N_{cd}[i,s]$, and $N_{nsb}[i,s]$ variables in terms of these new variables:

$$\forall i \in \mathit{Index} \ . \ \forall s \in \mathit{OurSectors} \ . \ N_{sec}[i,s] = \sum_{f \in \mathit{SectorFlights}[s]} b_{sec}[i,s,f] \tag{18}$$

$$\forall i \in \mathit{Index} \ . \ \forall s \in \mathit{OurSectors} \ . \ N_{cd}[i,s] = \sum_{f \in \mathit{SectorFlights}[s]} b_{cd}[i,s,f] \tag{19}$$

$$\forall i \in \mathit{Index} \ . \ \forall s \in \mathit{OurSectors} \ . \ N_{nsb}[i,s] = \sum_{f \in \mathit{SectorFlights}[s]} b_{nsb}[i,s,f] \tag{20}$$

The next three additional constraints provide the increased propagation, as we can now express useful implied constraints. They respectively state that a climbing or descending flight in a sector is necessarily in that sector, that a flight near the boundary of a sector is necessarily in that sector, and that a flight in a sector cannot simultaneously be in any other sector:

$$\forall i \in \mathit{Index} \ . \ \forall s \in \mathit{OurSectors} \ . \ \forall f \in \mathit{SectorFlights}[s] \ . \ b_{cd}[i,s,f] = 1 \rightarrow b_{sec}[i,s,f] = 1 \tag{21}$$

$$\forall i \in \mathit{Index} \ . \ \forall s \in \mathit{OurSectors} \ . \ \forall f \in \mathit{SectorFlights}[s] \ . \ b_{nsb}[i,s,f] = 1 \rightarrow b_{sec}[i,s,f] = 1 \tag{22}$$

$$\forall i \in \mathit{Index} \ . \ \forall f \in \mathit{Flights} \ . \sum_{s \in \mathit{OurSectors}} b_{sec}[i,s,f] \leq 1 \tag{23}$$

For n relevant flights, q relevant flight-point pairs, s sectors, and $I = |\mathit{Index}| = k + 1$ sampled moments, there are then a total of $(3 \cdot s \cdot I + 1) \cdot n + q + (4 \cdot I + 1) \cdot s = \Theta(n + q)$ integer variables and at most $3 \cdot (s + 1) \cdot I \cdot n + 3 \cdot q + (4 \cdot I + 1) \cdot s + 1 = O(n + q)$ unquantified constraints. No global constraints are used in this model.

3.6 The Search Procedure and Heuristics

Our experimentally determined search procedure is given in Algorithm 1. First, it consecutively tries to label, for each chosen sector $s \in \mathit{OurSectors}$, for each sampled moment index $i \in \mathit{Index}$, and for each relevant flight $f \in \mathit{SectorFlights}[s]$, the $b_{sec}[i,s,f]$, $b_{cd}[i,s,f]$, and $b_{nsb}[i,s,f]$ variables, trying the value 1 before the value 0 (this is the right-to-left value ordering heuristic) for the former, and the value 0 before the value 1 (left-to-right value ordering) for the latter two. This means that the priority goes to placing f within s at $m + i \cdot L$, but neither on a non-level segment nor near the boundary of s. These value orderings make sense because *ff* is usually close to 100%, hence most of the flights need to be placed

Algorithm 1 Search procedure with variable and value ordering heuristics

```
for all $s \in \mathit{OurSectors}$, ordered by decreasing $|\mathit{SectorFlights}[s]|$ do
  for all $i \in \mathit{Index}$ do
    for all $f \in \mathit{SectorFlights}[s]$ do
      try $b_{sec}[i,s,f] = 1 \mid b_{sec}[i,s,f] = 0$ end try
      try $b_{cd}[i,s,f] = 0 \mid b_{cd}[i,s,f] = 1$ end try
      try $b_{nsb}[i,s,f] = 0 \mid b_{nsb}[i,s,f] = 1$ end try
    end for
  end for
end for
for all $f \in \mathit{Flights}$ do
  try all $d \in \Delta T$, ordered by increasing $|d|$ : $\delta T[f] = d$ end try all
end for
for all $p \in \mathit{FlightPoints}$ do
  try all $d \in \Delta H$, ordered by increasing $|d|$ : $\delta H[p] = d$ end try all
end for
```

somewhere in our multi-sector airspace. The order of consideration of the sectors s is by decreasing size of *SectorFlights*$[s]$, so as to begin with the sectors that are planned to be the most busy (this is a variable ordering heuristic). This propagates very well to the other $b_{sec}[i,s,f]$ variables, via the implied constraint (23), and from there to the other $b_{cd}[i,s,f]$ and $b_{nsb}[i,s,f]$ variables, via the contrapositives of the implied constraints (21) and (22), as well as to the δT variables (whose domains are already shrunk by the constraint (4)) and to the δH variables (whose domains are already shrunk by the constraints (5), (6), and (10)), via the constraints (15), (16), and (17).

Next, our search procedure labels the δT variables, using a value ordering by increasing *absolute value* among the remaining values within the ΔT range. Finally, it labels the δH variables, again using a value ordering by increasing *absolute value* among the remaining values within the ΔH range. This guarantees revised flight profiles that deviate as little as possible from the original ones.

Eventually, this fixes the remaining functionally dependent variables, via the constraints (18), (19), (20), (11), (12), and (13). The minimisation objective (14) is automatically performed by the constraint solver using a branch-and-bound scheme.

This search procedure turns out to be sufficiently efficient, even if stopped after just two minutes (see the experiments in Section 4 below). If allocated enough time, it (or any variation thereof) will eventually compute the optimal resolved flight plan, since complete search is performed, modulo the pruning achieved by propagation.

3.7 Implementation

The constraint program was implemented in OPL, the *Optimisation Programming Language* [11, 12]. See `http://www.ilog.com/products/oplstudio/` for more information.

Translating the designed constraint program into an OPL model is merely a matter of slight syntax changes, *without* remodelling any of the logical connectives used.[2] For instance, the constraints (15) and (17) become the following so-called *reification constraints*, which are both non-linear:

```
forall(i in Index, s in OurSectors, f in Flights)
  if f in SectorFlights[s]
    then b_sec[i,s,f] = (Profile[s,f].first().timeOver
                         <= m + i * L - deltaT[f]
                         < Profile[s,f].last().timeOver)
    else b_sec[i,s,f] = 0
  endif ;
```

and

```
forall(i in Index, s in OurSectors, f in Flights)
  if f in SectorFlights[s]
    then b_nsb[i,s,f] = (0 <= m + i * L - (Profile[s,f].first().timeOver + deltaT[f])
```

[2]The reader may safely skip the rest of this sub-section.

```
                                <= hv_nsb
                              &
                              m + i * L
                              < Profile[s,f].last().timeOver + deltaT[f]
                              \/
                              0 <  Profile[s,f].last().timeOver + deltaT[f] - (m + i * L)
                                <= hv_nsb
                              &
                              Profile[s,f].first().timeOver + deltaT[f]
                              <= m + i * L)
      else b_nsb[i,s,f] = 0
    endif ;
```

where the parentheses around the right-hand argument of the equality in the `then` block typecast that formula into the integer `1` if that formula is true, and into the integer `0` otherwise.

Note how the logical conjunctions and disjunctions are literally preserved. The same holds for logical implication. For instance, the constraint (21) becomes the following so-called *channelling constraint*, which is non-linear:

```
  forall(i in Index, s in OurSectors, f in SectorFlights[s])
    b_cd[i,s,f] = 1  =>  b_sec[i,s,f] = 1 ;
```

A linear, but less readable, formulation of this constraint is as follows, without any difference in performance:

```
  forall(i in Index, s in OurSectors, f in SectorFlights[s])
    b_cd[i,s,f] <= b_sec[i,s,f] ;
```

Note that "`=>`" stands for the left-to-right logical implication ($\rightarrow$), while "`<=`" stands for the less-than-or-equal comparison relation ($\leq$) and not for the right-to-left logical implication ($\leftarrow$).

Since there is no existential quantifier in OPL, an existential quantification of a formula β over a set S filtered by a condition α

$$\exists i \in S : \alpha(i) \;.\; \beta(i)$$

becomes a sum of integer truth values that is constrained to be positive:

`sum(i in S :` α`(i)) (`β`(i)) > 0`

where the parentheses around β`(i)` typecast that formula into the integer `1` if that formula is true, and into the integer `0` otherwise. For instance, the constraint (16) becomes another non-linear reification constraint:

```
  forall(i in Index, s in OurSectors, f in Flights)
    if f in SectorFlights[s]
      then b_cd[i,s,f] = (sum(p in Profile[s,f]: p <> last(Profile[s,f]))
                          (p.timeOver
                           <= m + i * L - deltaT[f]
                           <  Profile[s,f].next(p).timeOver
                           &
                           p.flightLevel + deltaH[p] <>
                           Profile[s,f].next(p).flightLevel + deltaH[Profile[s,f].next(p)]
                          ) > 0
                         )
      else b_cd[i,s,f] = 0
    endif ;
```

As the resulting OPL model has non-linear constraints, the OPL compiler translates the model into code for ILOG Solver, rather than for CPLEX, and constraint solving takes place at runtime.

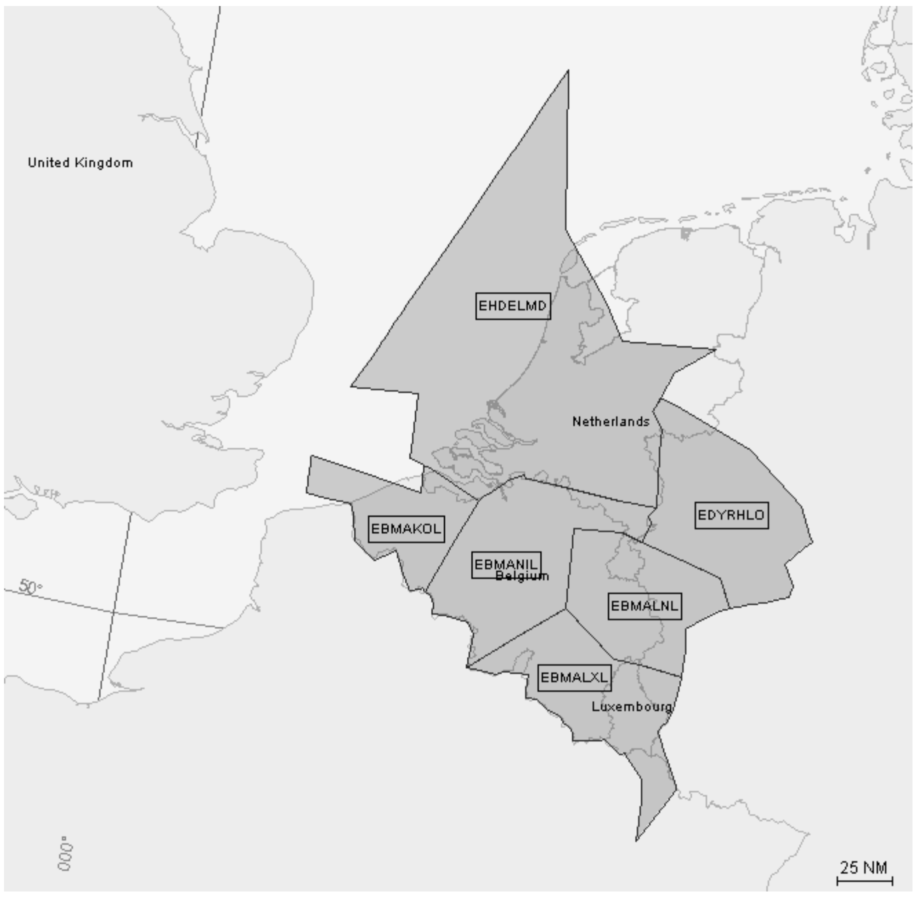

Figure 3: The chosen multi-sector airspace over Western Europe. On the chosen day, the sectors EBMAKOL and EBMANIL were collapsed into the sector EBMAWSL.

4 Experiments

For our experiments, the chosen air-traffic control centre is Maastricht, in the Netherlands. The chosen multi-sector airspace within the Maastricht airspace consists of five sectors, covering the upper airspace of the three BeNeLux countries and some airspace of northern Germany, depicted in Figure 3, and characterised in Table 1. They are all high-density, en-route, upper airspace sectors (above FL 245). The sector identified by *sectorId* stretches vertically between flight levels *bottomFL* and *topFL*. Unfortunately, none of these sectors is below any other one, so that our traffic complexity resolutions cannot consider re-routing a flight through a lower or higher sector in the chosen airspace, as required by constraint (6). The weights a_{sec}, a_{cd}, a_{nsb} and sector normalisation constants S_{norm} of the complexity metric are taken from [6]. There are an additional 34 feeder sectors (not listed here), for which we only need to know the *bottomFL* and *topFL* values. Hence:

$$\begin{aligned} \mathit{OurSectors} &= \{\mathit{EBMALNL}, \mathit{EBMALXL}, \mathit{EBMAWSL}, \mathit{EDYRHLO}, \mathit{EHDELMD}\} \\ \mathit{AllSectors} &= \{\mathit{EBMALNL}, \mathit{EBMALXL}, \mathit{EBMAWSL}, \mathit{EDYRHLO}, \mathit{EHDELMD}, \ldots\} \\ \mathit{Engine} &= \{\mathit{turbo}, \mathit{jet}\} \end{aligned}$$

and *Sector* encodes the corresponding superset of Table 1. Since constraint reasoning is faster on integers than floats, we actually multiplied all the a_{sec}, a_{cd}, a_{nsb}, and S_{norm} values by 100. As their original accuracy is 2 decimal positions, we will not lose any precision, as long as any obtained complexity is divided by $10,000$ before it is displayed.

The chosen day is 23 June 2004. At the moment (5 August 2004) of choosing this day, it was the busiest day that far of the year 2004. The chosen hours are the peak traffic hours, that is from 07:00 to 22:00 local time. The chosen flights follow standard routes (no free flight) and are of the turbo-prop or jet type. The chosen point profiles are so-called model-3 (radar-corrected) profiles, as these better reflect actual traffic than model-1 (filed) profiles. The Central Flow Management Unit (CFMU) provided

sectorId	*bottomFL*	*topFL*	a_{sec}	a_{cd}	a_{nsb}	S_{norm}
EBMALNL	245	340	7.74	15.20	5.69	1.35
EBMALXL	245	340	5.78	5.71	15.84	1.50
EBMAWSL	245	340	6.00	7.91	10.88	1.33
EDYRHLO	245	340	12.07	6.43	9.69	1.00
EHDELMD	245	340	4.42	10.59	14.72	1.11

Table 1: Characterisation of the chosen multi-sector airspace

lookahead	k	L	Average planned complexity	Average resolved complexity
20	2	210	87.92	47.69
20	3	180	86.55	50.17
45	2	210	87.20	45.27
45	3	180	85.67	47.81
90	2	210	87.29	44.67
90	3	180	85.64	47.13

Table 2: Average planned and resolved complexities in the chosen airspace

us with the flight profiles for these choices. There are $1,798$ flight profiles, after statically repairing 761 flight profiles that included impossibly steep climbs or descents (which would otherwise have to be repaired dynamically according to the constraint (10) during complexity resolution) and discarding 26 flights whose profiles were not repairable.

The maximum approach time is 78 minutes for this traffic sample. By formula (3), this means that approach times can be changed by integer amounts of minutes within the range $[-4, \ldots, +8]$. This is a sufficiently large range, almost of the magnitude of the range ΔT of take-off-time changes, so that the second form of complexity resolution can be expected to lead to significant changes in its own right.

Our experiments show that the described forms of complexity resolution lead to systematic reductions of the sum of the complexities of the sectors of the chosen airspace.

In Table 2, every line summarises the results on the 180 instances obtained by taking *now* at every 5 minutes between 07:00 and 22:00 on the chosen day. The average reduction in the average complexity over the five sectors is shown over these instances for various values of the smoothing degree k, the length L (in seconds) of the time steps, and *lookahead*. We kept *ff* $= 90\%$ of the planned flights in the chosen airspace, and used *timeOut* $= 120$ seconds. With larger values of *lookahead*, it is possible to get better complexity reductions, simply because more flights are not airborne yet and thus offer more opportunities for resolution. With larger values of k (and thus lower values of L), it is possible to get a slightly better reduction in complexity, but with $k = 2$, it is possible already to get nearly a 50% reduction in complexity. The experiments were performed with OPL 3.7 under Linux 2.6.4-52 on an Intel Pentium 4 CPU with 2.53GHz, a 512 KB cache, and a 1 GB RAM. Most of the computations finished before timing out, or were retrospectively seen (upon a larger value for *timeOut*) to have found (near-)optimal solutions at the moment of timing out, which means that the proofs of optimality were more time-consuming than finding the optima. Other experiments confirmed that reducing *ff* or increasing *timeOut* gives better results.

In practice, complexity resolution in an MSP context will not be a constraint optimisation problem (COP), as here, but rather a constraint satisfaction problem (CSP). There will be many additional constraints, which simply have to be satisfied, such as requiring the resolved complexities to be within prescribed intervals. Since CFMU flight profiles derived from the flight plans introduced by the airlines are not very accurate (witness the amount of necessary repairs we had to perform) and currently incorporate only an attempt at balancing the numbers of flights (the N_{sec} term of our traffic complexity metric) between sectors, we cannot impose such maximal or even minimal bounds on the resolved complexities, as feasible solutions might then not exist. Indeed, there are enormous discrepancies among the planned complexities, and even optimal complexity resolution can often not sufficiently reduce them. We ought to get better flight profiles for such additional constraints and experiments, so that we can then switch from a COP to a CSP. The reasons why we report here on optimisation experiments are that they give an upper bound on the runtime performance of the model and that this upper bound is already very good.

This is also why we do not illustrate the differences between some planned and resolved flight profiles, and rather just compiled our many experiments into the single two-dimensional Table 2. What that table does not show, however, is that the resolved complexities were much more evenly balanced among the chosen sectors than the planned complexities.

5 Conclusion

Constraint programming offers a very effective medium for *modelling* and efficiently *solving* the problem of minimising and balancing the traffic complexities of an airspace of adjacent sectors. The traffic complexity of a sector is here defined in terms of the numbers of flights within it, near its border, and on non-level segments within it. The allowed forms of complexity resolution are the changing of the take-off times of not yet airborne flights, the changing of the approach times into the chosen airspace of already airborne flights by slowing down and speeding up within the two layers of feeder sectors around that airspace, as well as the changing of the levels of passage over points in that airspace. Experiments with actual European flight profiles obtained from the Central Flow Management Unit (CFMU) show that these forms of complexity resolution can lead to significant complexity reductions and rebalancing.

A lot of related work is about dealing with potentially interacting pairs (PIPs) of flights. However, as the number of PIPs had a very low correlation with the traffic complexity, due to the fact that this element of moment complexity is already captured by the total number of flights in the sector and by the number of non-level flights in the sector [6], we did not have to resolve them away, nor even worry about their number in the resolved flight profiles.

There is also a large literature on complexity metrics for estimating workload, and we refer to [4] for a thorough recent survey thereof, as doing so here is beyond the scope of this paper.

The work closest to ours was done for the airspace of the USA in [10]. The main differences with our work are as follows. They have (at least initially) static lists of alternative routes to pick from for each flight and do not consider changing the time plans, whereas we dynamically construct (only vertically) alternative routes and new time plans. Their sector workload constraints limit the average number of flights in a sector over a given time interval (like the N_{sec} term of our complexity metric) and the number of PIPs (recall that our intended N_{pip} term was eliminated for lack of statistical significance [6]), but these terms are not part of a complexity metric. No multi-sector planning is performed to reduce and re-balance the workloads of selected sectors. However, a notion of airline equity is introduced toward a collaborative decision-making process between the FAA and the airlines.

Another important related work aims at minimising costs when holding flights (on the ground or in the air), if not re-routing them, in the face of dynamically changing weather conditions [2]. The main differences with our work are as follows. Their objective is cost reduction for airlines and airports, whereas our work is airspace oriented. They only consider ground holding and air holding, whereas we also consider the planning of earlier take-offs and the speeding-up of airborne flights. Their dynamic re-routing is on the projected two-dimensional plane, whereas ours is in the third dimension. Their sector workload constraints limit the number of flights in a sector at any given time, but there is no complexity metric and no multi-sector planning.

As this little overview of related work shows, the whole problem of optimal airspace and airport usage by the airlines is very rich, and only facets thereof are being explored in each project. Our work intends to reveal some new facets, such as complexity metrics and multi-sector planning.

Also, since the complexity resolution happens for a time interval $[m, \ldots, m + k \cdot L]$ in the future, constraints will be needed to make sure no unacceptable complexity is generated before m.

Another issue is the implementation of the calculated complexity resolutions. Additional constraints are needed to make sure that the proposed flight-profile updates can be implemented sufficiently quickly, and that doing so is still offset by the resulting complexity reductions and rebalancing among sectors. For instance, the number of flights affected by the changes may have to be kept under a given threshold.

There is a lot of other future work to do before an early prototype like ours can be deployed in a tactical context. Its main current objective is therefore strategic, namely to provide a platform where new definitions of complexity can readily be experimented with, and where constraints can readily be changed or added. This motivated the choice of constraint programming (CP) as implementation technology, since the maintenance of constraint programs is simplified. Furthermore, the likely addition of many more side-constraints will make the problem less and less purely combinatorial, and this is the typical scenario where CP is expected to be faster or to find better solutions than rival technologies [9].

Acknowledgements

Many thanks to Søren Dissing at EuroControl DAS/ATS, for supporting this work, and to Bernard Delmée, Jacques Lemaître, and Patrick Tasker at EuroControl DAP/DIA, for pre-processing the CFMU raw data into the extended data and format we needed. This research project has been funded by EuroControl grant C/1.246/HQ/JC/04 and its amendments 1/04 and 2/05.

References

[1] Apt, Krzysztof R. *Principles of Constraint Programming.* Cambridge University Press, 2003.

[2] Bertsimas, Dimitris and Stock Patterson, Sarah. The traffic flow management rerouting problem in air traffic control: A dynamic network flow approach. *Transportation Science*, 34(3):239–255, INFORMS, 2000.

[3] Darby-Dowman, Ken and Little, James. Properties of some combinatorial optimization problems and their effect on the performance of integer programming and constraint logic programming. *INFORMS Journal on Computing*, 10(3):276–286, 1998.

[4] EuroControl, Directorate of ATM Strategies, Air Traffic Services division (DAS/ATS). *Complexity algorithm development: Literature survey and parameter identification.* Edition 1.0, 9 February 2004.

[5] EuroControl, Directorate of ATM Strategies, Air Traffic Services division (DAS/ATS). *Complexity algorithm development: The algorithm.* Edition 1.0, 7 April 2004.

[6] EuroControl, Directorate of ATM Strategies, Air Traffic Services division (DAS/ATS). *Complexity algorithm development: Validation exercise.* Edition 0.3, 10 September 2004.

[7] EuroControl Experimental Centre (EEC). *Pessimistic sector capacity estimation.* EEC Note Number 21/03, 2003.

[8] Garcia Avello, Carlos. *Decomplexing traffic in a multi sector planning environment.* EuroControl DAS/ATS, 11 May 2004.

[9] Milano, Michela (editor). *Constraint and Integer Programming: Toward a Unified Methodology.* Kluwer Academic Publishers, 2004.

[10] Sherali, Hanif D.; Staats, Raymond W.; and Trani, Antonio A. An airspace planning and collaborative decision-making model: Part I — Probabilistic conflicts, workload, and equity considerations. *Transportation Science*, 37(4):434–456, INFORMS, 2003.

[11] Van Hentenryck, Pascal. *The OPL Optimization Programming Language.* The MIT Press, 1999.

[12] Van Hentenryck, Pascal. Constraint and integer programming in OPL. *INFORMS Journal on Computing*, 14(4):345–372, 2002.