# Slot Allocation with Constraint Programming: Models and Results

*Nicolas Barnier, Pascal Brisset and Thomas Rivière*
ENAC - CENA, Toulouse, France
barnier,brisset,riviere@recherche.enac.fr
http://www.recherche.enac.fr/opti

**Abstract**

Current European Air Traffic Control system is far exceeded by the demand and the resulting delays are a financial and psychological burden for airlines and passengers. The Central Flow Management Unit, in charge of regulating the flights to respect en-route capacity constraints of Air Traffic Control Centres, uses a greedy algorithm to allocate departure slots which features several drawbacks concerning soundness, interpretation of the constraints and optimization. A new model taking advantage of Constraint Programming (CP) has been proposed within the SHAMAN system (from CENA) but still suffers from unevenly distributed workload and capacity violation. This paper presents two alternative models, also implemented using CP technology, of the slot allocation problem focused on the controllers workload: an extension of the SHAMAN model with a standard formulation, and a novel approach involving the *sort* constraint. Both of them can maintain workload constantly below a given capacity and the latter also provides efficient failure proof on over-constrained instances. The behaviours of the different models are discussed and results are presented with partial and full instances from real French air traffic data set. We eventually describe the potential operational improvement supplied by these *continuous* models.

**Keywords:** Slot Allocation, Constraint Programming, Combinatorial Optimization, Modelling

## 1 Introduction

Airspace congestion is today the most critical issue European Air Traffic Management (ATM) has to face. French Air Traffic Control Centres (ATCC) capacities are far exceeded by a constant growth in air traffic demand, resulting in ever increasing flight delays. These time and management costs are such a nuisance for all airlines and passengers that the European Commission has released a special statement [IP/99/924, 1999] acknowledging that current ATFM systems are unable to support high traffic loads and unscalable for the predicted growth.

The Central Flow Management Unit (CFMU) in Brussels is in charge of reducing these congestion costs by, among several other strategic or tactical measures, delaying departure slots for the flights involved in overloaded sectors. The purpose of delaying is to respect the en-route capacity constraints provided by each ATCC as a number of planes per hour according to their daily schedule.

The CFMU currently solves this problem in two stages [CFMU, 2000]:

- A "pre-tactical" tool (PREDICT) identifies overloaded sectors and helps (an experienced human operator) to place regulations by simulating their impact.

- Then a "tactical" system (TACT) including the CASA (Computer Assisted Slot Allocation) tool uses a greedy algorithm to dynamically allocate regulated departure slots on a "first planned first served" basis while flight plans data are received.

Basically, CASA computes a list of slots by dividing the time length of the regulated period by its capacity and tries to fill each slot according to the estimated arrival time of the flights in the concerned sector. Several revision processes occur when new flight plans are received that may shift all the flights already scheduled during the same period to respect the "first planned first served" principle.

The solution obtained obviously depends very much on the resolution order of the constraints

and no optimization is possible[1]. Moreover, the regulation for a given flight is computed with respect to the sector inducing the greatest delay (in case of multiple regulations), which may lead to capacity constraints violation for the other sectors crossed by the same flight. Eventually, a former study [Gotteland et al., 2000] estimates that CFMU local regulations have little effects on ATCCs overall overload. However, CASA is an operational tool which must comply with other additional constraints (not taken into account in the models presented in this paper) and well integrated in a complex ATM process.

To overcome the drawbacks and lack of efficiency of CASA, the CENA (French Air Navigation Study Centre) has developped a slot allocation module based on Constraint Programming within the SHAMAN[2] platform [Plusquellec and Manchon, 1998]. This tool states capacity constraints on all of the open sectors by dividing their overall lengths in contiguous 30 min long slices and constraining the number of flights arriving in the sectors over each slice (e.g. suppose sector AIX is open from 12h00 until 13h30 with capacity 32 flights per hour, three constraints ensuring that no more than 16 flights will enter the sector between respectively 12h00 and 12h30, 12h30 and 13h00, 13h00 and 13h30, are posted). In this approach, the use of Constraint technology provides a global view of all the capacity constraints such that solutions (if found) are consistent (all the constraints are satisfied).

However, the SHAMAN model does not prevent traffic peaks exceeding the sectors capacities during periods which do not begin at multiples of 30 min from the start time of their constraints, such that controllers have to face too high workloads. Furthermore, even if SHAMAN does not perform any optimization on its solution[3], the search strategy is guided by the minimization of the sum of the delays. Hence delayed flights have a chronic tendancy to gather at the beginning of the 30 min periods, yielding traffic peaks above the requested capacities.

As the statements of capacity constraints are obviously not very satisfactorily interpreted by current ATFM systems from an operational point of view, a better model of these requirements would be to maintain controllers workload below the specified capacities continuously. We provide for this aim several new models endowed with a more realistic, thus harder, satisfaction of capacity constraints:

- Sliding discrete windows which allow to smooth the profile of controllers workload from the weakest constrained model (the SHAMAN model) to the hardest (the satisfaction of capacity constraints over any time period of a given length);
- The "sorting" model which states continuously the constraints over the ranks of the entry time of flights within a given sector.

Constraint Programming allows to express a very straightforward formulation of these two latter models, as well as for a "twin" of the SHAMAN model presented for comparison purpose. The versatility of CP technology eases fast implementation and alternatives testing. An all-purpose constraint library, FaCiLe[4] [Barnier and Brisset, 2001] written in Objective Caml [Leroy, 2000] has been used to experiment with the various models, leading to drastically different allocation schemes and abilities to obtain proof of optimality or unfeasibility.

The paper is divided as follows: we first give a precise description of the problem and some hints about the size and complexity of the instances we are interested in. The next section introduces briefly Constraint Programming technology, thereafter we present several models for the capacity constraints, starting with standard ones and refining them with a more continuous formulation. Eventually, section 5.2 is devoted to the results for simplified and real instances where behaviour of the different models are compared. We conclude with an overview and some indications about possible future work.

# 2 Slot Allocation Problem Description

Air traffic flow management is a daily pre-tactical filter intended to regulate scheduled flights across controlled airspace. Its main aim is to limit the number of planes in a given space over a given period. This planning will be precisely scheduled by the air-traffic controllers in real-time.

The ATFM problem is described in terms of

- *Flights*: A flight starts from one airport at a specified time, follows a predefined route at a fixed speed and arrives at another airport.

[1] Except at the very final stage when the slots allocated to lately cancelled flight may be redistributed.
[2] System to Help Analysis and Monitoring of Acc resources and Air route Network
[3] Usually done in Constraint programs with a standard Branch & Bound algorithm.
[4] Functional Constraint Library

- *Sectors*: The airspace is divided into sectors crossed by the routes of the flights. A sector is a 3D polyhedra, usually a vertical cylinder endowed with a *capacity* expressed as the maximum number of flights entering the sector during a time period (usually one hour). The partition of the airspace, i.e. the number and the shape of the sectors, varies during the day. The capacity of the same sector may also changes at given times. We call *sector-period* a sector for particular period of time and capacity.

The constraints of the problem are the capacities of the sectors. There are several degrees of freedom to satisfy these constraints: choosing different routes for flights, delaying departures, changing speed of the aircrafts during flight, asking aircrafts to hold their position... In this paper, we focus only on the ground-holding policy: each flight may be delayed at departure. ATFM is in charge of solving the problem on a day-by-day basis.

The difficulty of the problem does not come from the complexity of its constraints but from its size. We have used real data archived by the French civil aviation tool COURAGE to solve the ATFM problem. E.g. for May 20th, 1999:

- The French airspace is concerned by 7375 flights entering between 0h00 and 23h59.
- 140 sectors are active during the day, some of them with various capacities (up to 6 different).
- More than 700 flights may enter a single sector-period.
- Capacities vary from 19 to 52 flights per hour.

The objective of the slot allocation problem is to reduce the delays. There are several ways to achieve this purpose: reducing the total sum of the delays (utility), the maximum delay (equity), the average delay, etc. [Maugis, 1996] gives an extensive description of what could and should be a cost function and finally concludes with results for the simplest one, the total sum of delays. It is also the choice of [Bertsimas and Stock, 1995] with a slightly different model. In this study, we choose to minimize the maximum of all delays, but we are more interested in the qualitative properties of the solution than in its numerical cost.

# 3 Constraint Programming

Constraint Programming (CP) is an emergent modelling technology for declarative description and effective solving of large combinatorial optimization problems. The idea is to solve problems by stating constraints (conditions, properties on the decision variables) which must be satisfied by all the solutions while avoiding the tedious and error-prone task of explicitely maintaining the consistency of these constraints as search goes on.

Constraint solvers efficient enough to compare well with Operational Research technology emerged in the early 90s, based on the CLP (Constraint Logic Programming) paradigm and integrated in Prolog systems [Van Hentenryck, 1989]. Nowadays, CP is a challenging technology addressing a wide range of combinatorial academic and industrial hard problems in many fields (planning, scheduling, crew rostering, configuration, circuit design, DNA sequencing etc). State-of-the-art constraints solvers like Ilog Solver [Solver, 1999], Eclipse [ECL, 2001], CHIP [Dincbas et al., 1988], Prolog IV, SICStus Prolog or Mozart are used by more and more important companies (British Airways, British Telecom, SNCF, Hong Kong International Terminals, Michelin etc [Barták, 1998]).

CP is a very active research field and solvers technology evolves very quickly. One of the most promissing areas is the integration in constraints solvers of other paradigms of combinatorial optimization technologies, e.g. local search, stochastic search, linear programming. A current working version of a constraint program solving the slot allocation problem written with FaCiLe uses a generic *invariants* library [Michel and Hentenryck, 1997] typically designed towards the implementation of local search algorithms [Rivière, 2001][5]. The next three sections attempt to describe the main underlying ideas of CP.

## 3.1 Constraints

A constraint is a logical relation among several variables (or unknowns), each taking a value in a given domain (e.g. an integer interval). A constraint thus restricts the possible values that variables can take by maintaining its *consistency*. Basically, constraints remove from domains values which cannot appear in solutions: for instance an inequality constraint $x < y$ between two variables $x$ and $y$ with respective domains $[8..16]$ and $[6..15]$ will reduce the

[5] In our application, it helps to maintain efficiently complex data structures involved in a dynamic search heuristic.

variables domains to respectively $[8..14]$ and $[9..15]$. Each time a constraint is posted to the *constraint store* of a solver, the involved variables domains are refined and trigger the awakening of other constraints involved in one of the formerly reduced variables. This process is repeated until a fix point is reached where no more reduction can be deduced.

Constraints are declarative statements, close to natural description of combinatorial problems, which describe what relationship must hold without specifying a computational procedure to enforce it. This feature confers to CP solvers the status of modelling tools. They allow to express straightforwardly practical problem thanks to a rich language of constraints from linear or non-linear arithmetic to global constraints on sets of variables (difference, cardinality, sequence etc) and higher-order constraints (or logical constraints on constraints[6]). Modern solvers integrates a wide variety of built-in constraints whose consistency schemes are borrowed from state-of-the-art OR techniques, thus offering the versatility of CP modelling features while preserving the efficiency of specialized and well-proven algorithms.

## 3.2 Search

Posting constraints alone does not reduce domains enough to find a solution (at least in non-trivial problems or when the problem has several solutions) and search has to be performed. Basically, a variable is chosen as well as an instantiation value in its current domain, then related constraints are waken and the propagation process starts, reducing the domains to prune the search tree as much as possible. Thereafter another variable is chosen and instantiated until all variables are bound to a value and a solution is found or a failure occurs. In the latter case, the search algorithm backtracks to reconsider the last choice point. An optimization process can be elegantly integrated to this branching mechanism by dynamically adding a constraint that enforces the cost of the problem to be strictly less than the one of the last solution found.

CP languages are very well suited to express readily complex search strategies based on dynamic ordering (i.e. which depends on the state of the system) of variables and instantiation values. Such non-deterministic search procedures are implemented with Prolog-like *goals*. E.g. one of the most popular strategy, called the "first fail principle"[7], is to choose the uninstantiated variable with the smallest domain and/or involved in the greatest number of constraints. Most constraint solvers provide built-in goals to implement such heuristics.

The orthogonal strengths of a rich declarative constraint language and ability to specify search strategies with a high level of abstraction help to separate the problem modelling from the search for solutions. Hence they improve the correction of programs and allow fast prototyping, developpement and alternatives testing.

## 3.3 Formal Description

A Constraint Satisfaction Problem (CSP) consists of:

- A set of *variables* $\mathcal{X} = \{x_1, ..., x_n\}$.
- For each variable $x_i$, a finite set $\mathcal{D}_i$[8] of possible values, called its *domain*; we note $\mathcal{D} = \mathcal{D}_1 \times ... \times \mathcal{D}_n$. An *assignement* $V = (v_1, ..., v_n)$ of $\mathcal{X}$ is a valuation of each variable of $\mathcal{X}$ from a value of its domain: $V \in \mathcal{D}$.
- A set of constraints $\mathcal{C} = \{c_1, ..., c_m\}$.
- Each constraint $c_i$ restricts the values that a subset of variables $\{x_{i_1}, .., x_{i_k}\} \subset \mathcal{X}$ can simultaneously take. Therefore, constraint $c_i$ can be represented as a subset $\mathcal{D}_{c_i}$ of the cartesian product $\mathcal{D}_{i_1} \times ... \times \mathcal{D}_{i_k}$ and a function:

$$c_i : \mathcal{D} \longrightarrow \{\text{true}, \text{false}\} \quad \text{such that}$$
$$\forall\, V = (v_1, ..., v_n) \in \mathcal{D},$$
$$(v_{i_1}, .., v_{i_k}) \in \mathcal{D}_{c_i} \iff c_i(V)$$

A solution to a CSP is an assignment $V$ of $\mathcal{X}$ such that every constraint is satisfied, i.e. $\bigwedge_{i=1}^{m} c_i(V)$. Depending on the problem, to solve a CSP means to find:

- a proof of failure, i.e. there is no solution;
- just one solution, with no preference as to which one;
- all solutions;

[6] Like implication e.g. $x > y \Rightarrow y \neq z$ or disjunction e.g. $x + d_x < y \vee y + d_y < x$.

[7] *"To succeed, try first where you are most likely to fail."* [Barták, 1998]

[8] As formerly said, CP main contribution field is combinatorial problems, most of the time expressed with variables ranging over integer domains. However, some commercial and research solvers integrate efficient constraints over real, rational, finite set, boolean and even tree variables. This paper and our constraints librairy FaCiLe only consider integer domains.

- an optimal, or at least a good solution, given some objective function defined in terms of some or all of the variables.

Actually, real-life problems (huge size and/or high complexity) sometimes cannot yet be solved optimaly. In such cases, "good solutions" obtained with a limited amount of optimization (e.g. with a computation time bound) or satisfying a subset of the constraints are searched for.

# 4 Three Models

In this section, we describe three models for the slot allocation problem. These models are not equivalent and differ according to the interpretation of the sector load constraints.

The models are described with the following data:

- $\mathcal{S}$: the set of sector-periods, each with a $start$ and an $end$;
- $\mathcal{F}$: the set of flights;
- $t_i^s$: the time the flight $i$ enters the sector $s$ according to the original timetable;
- $capa^s$: the capacity of the sector-period $s$ (flights/hour);
- $\delta$: time base for the capacity constraint (minutes).

All the models use the main decision variables

- $D_i$: departure delay for flight $i$.

We first present the `Basic` model implemented within SHAMAN in section 4.1 [Maugis, 1996, Plusquellec and Manchon, 1998]. Then two novel models are introduced in section 4.2.

## 4.1 Non Overlapping Windows

The SHAMAN model (refered afterward as the `Basic` model) discretizes the sectors with $\delta = 30$ min periods to enforce the capacity constraints and attempts to reduce the sum of all delays. Several unsuitable side-effects arise from this model, leading to irregular workload profiles.

### 4.1.1 The Basic model

We consider the load constraint in successive contiguous periods, i.e. non overlapping time windows:

- $\mathcal{P}^s = \{p_0^s, p_1^s, ...\}$: successive periods of length $\delta$, each with a $start$ and a $end$. The first period starts with the sector-period: $start(p_0^s) = start(s)$.

In an other way:

$$\mathcal{P}^s = \{[start(s), start(s) + \delta[, \\ [start(s) + \delta, start(s) + 2\delta[, ...\}$$

The `Basic` model is written with auxiliary boolean variables:

- $B_{i,p_j^s}$: flight $i$ enters the sector $s$ during the period $p_j^s$.

A first constraint relates the delay variables with the boolean variables. A second one sets the sector load capacity.

$\forall s \in \mathcal{S}\ \forall p_j^s \in \mathcal{P}^s$:

- $B_{i,p_j^s}$ iff
  $start(p_j^s) \leq t_i^s + D_i < start(p_j^s) + \delta$
- $\sum_{i \in \mathcal{F}} B_{i,p_j^s} \leq capa^s$

This model needs a huge amount of boolean variables:

$$|\{B_{i,p_j^s}\}| \quad = \quad |\mathcal{F}| \sum_{s \in \mathcal{S}} \frac{end(s) - start(s)}{\delta}$$

### 4.1.2 Drawbacks

The `Basic` model suffers from strong discontinuity. It only states that the sector load limit must be respected at the beginning of each period, usually 48 times a day (for 30 min windows). Such that workload may climb much higher for time windows that do not start at multiples of $\delta$ min from the beginning of the constraint, as illustrated in figure 1 (with $\delta = 30$ min): here, for an overall amount of flights that does not exceed the capacity either in the first period or in the second one, an in-between period of the same length may have almost twice as much flights.

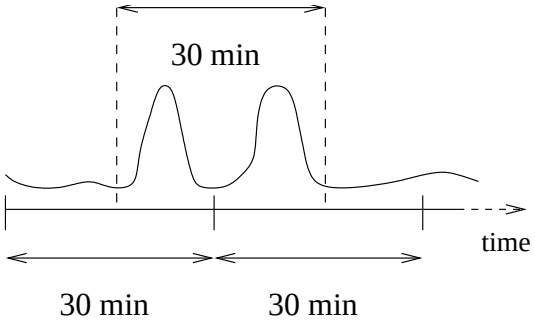


Figure 1: Overloaded in-between period with the `Basic` model.

Furthermore, the constraints of this model does not prevent traffic peaks for time windows shorter than the $\delta$-length periods, as shown in figure 2: the two workload profiles correspond (roughly) to the same amount of flights (supposedly below the capacity for the duration of the period), though the first one (allowed by the `Basic` model) is obviously not well suited to ease the task of controllers.

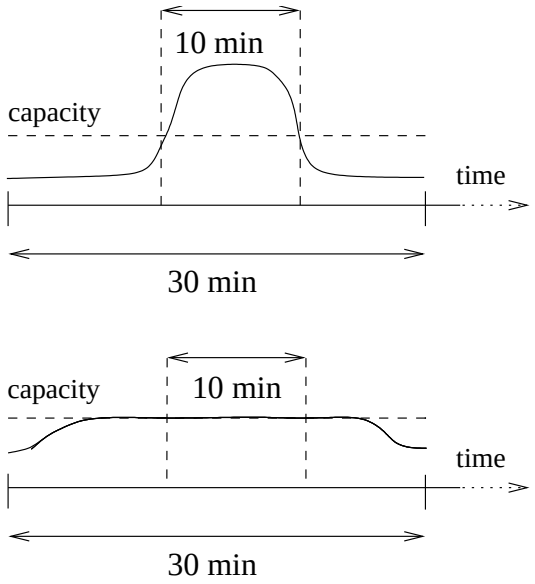


Figure 2: The `Basic` model does not prevent the first workload profile: peaks may arise for short time periods.

Because the main objective of the cost function is to reduce the delays, the expected side-effect of this model is a concentration of flights entering at the beginning of the periods: postponed flights are scheduled as early as possible in the next time window with the `Basic` model. Figure 3 illustrates this behaviour: the dashed curve is the workload profile before any regulation, then the `Basic` model scheme tends to shift the traffic peak of the first time window at the beginning of the second one. This chronic side-effect is confirmed by the experiments (see section 5.2.1).

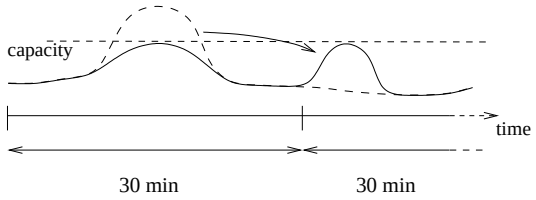


Figure 3: Traffic peaks at the beginning of the period with the `Basic` model.

## 4.2 Continuous Models

We propose two new formulations which attempt to smooth the workload with a more constrained modelling of capacity constraints. The first model is an extension of the `Basic` one. The second one relies on a *sorting* global constraint.

### 4.2.1 Sliding Periods

Keeping the same idea of the `Basic` model, it is possible to get a more continuous model with overlapping periods, illustrated in figure 4.

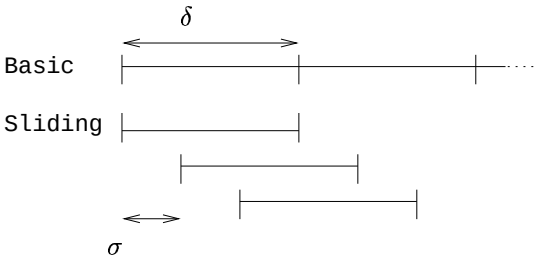


Figure 4: Sliding windows

Here we use another parameter ($\sigma$) which is the time step between periods:

$$\mathcal{P}^s = \{[start(s), start(s) + \delta[, \\ [start(s) + \sigma, start(s) + \sigma + \delta[, ...\}$$

Of course this model supersedes the `Basic` one and is equivalent for $\sigma = \delta$. It can be implemented with the same kind of boolean variables.

The workload profile is expected to become smoother as $\sigma$ decreases, such that more $\delta$-length time periods are taken into account.

#### 4.2.2 The Sorting Model

**The Sorting Constraint** Let $D$ be a totally ordered set. The sort constraint is the relation associated with the standard sort function. A sort function takes a sequence of length $n$ containing elements in $D$ and returns another sequence containing the same elements ordered. The sort constraint "relates" two sequences containing finite domain variables taking values in $D$.

For example, let

$$X = \{[0-13]; [6-10]; [10-11]; [4-16]; [4-6]\}$$

and

$$Y = \{[1-3]; [5-10]; [6-9]; [11-17]; [10-15]\}$$

be two sequences of interval variables. Posting the constraint

$$sort(X, Y)$$

must lead to the following refinements:

$$X = \{[1-3]; [6-9]; 11; [11-15]; [5-6]\}$$
$$Y = \{[1-3]; [5-6]; [6-9]; 11; [11-15]\}$$

[Guernalec and Colmerauer, 1997] proposed an efficient filtering algorithm for this constraint. Surprisingly enough, the complexity of this complete (i.e. the propagation refines the domains as much as possible) narrowing algorithm has an optimal complexity of $\mathcal{O}(n \log n)$, the same than classic sort algorithms. However, the narrowing is done only on the bounds of the variables such that no propagation is performed if "holes" appear in the domains.

**A Dual Model** For our problem, we use one sorting constraint for each sector-period. The constraint is set on auxiliary variables corresponding to the entry times and the sorted entry times of the flights:

- $T_i^s$: actual entry time of flight $i$ in sector-period $s$;
- $S_j^s$: entry time of the $j$th flight entering the sector-period $s$.

The constraints relate the auxiliary variables to the main delay variables and set the sector load limit with a *dual* model of the standard one. We no longer count the number of flights within a given period, but we rather consider the rank of the entry times and specify that two flights must be separated by a given duration whenever their ranks are too close.

Two flights with entry times $S_i$ and $S_j$ in the ordered sequence such that $i + capa \leq j$ must be separated by at least $\delta$ minutes from each other (c.f. figure 5):

$$\begin{aligned} \forall s \forall i \in \mathcal{F} \quad & T_i^s = t_i^s + D_i \\ \forall s \quad & sort(T^s, S^s) \\ \forall s \forall j \in \{1, .., |\mathcal{F}| - capa^s\} \quad & S_j^s + \delta \leq S_{j+capa^s}^s \end{aligned}$$

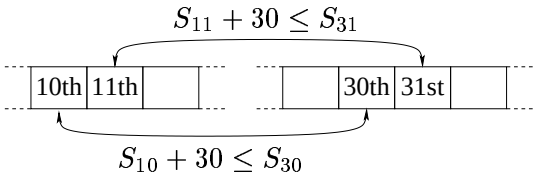


Figure 5: Capacity constraint applied to sorted flights ($\delta = 30$, $capa = 20$).

The previous constraint for the load limit is not totally correct because it is set even if one of the two concerned flights enters the sector before or after the corresponding period. A solution is to relax the constraint using auxiliary boolean variables:

$$\begin{aligned} &\forall s \in \mathcal{C}\ \forall j \quad B_j^s \text{ iff } start(s) \leq S_j^s < end(s) \\ &\forall s \in \mathcal{C}\ \forall j \in \{1, ..., |\mathcal{F}| - capa^s\} \\ &\qquad B_j^s \wedge B_{j+capa^s}^s \Longrightarrow S_j^s + \delta \leq S_{j+capa^s}^s \end{aligned}$$

The boolean variable $B_j^s$ states that the sector-period $s$ is concerned by the flight $j$. If one of the two flights is out of the period, the constraint is not posted.

The `Sorting` model states the capacity constraints continuously over a given sector and thus is expected to maintain the workload constantly below the fixed limit. The `Sliding` model should be equivalent with $\sigma$ reduced down to the time unit, i.e. all possible $\delta$-length time windows are constrained.

# 5 Experiments

## 5.1 Implementation

The three models have been implemented with FaCiLe, a finite domain library we have written using the Objective Caml system [Leroy, 2000]. This strongly typed functional language provides well documented libraries for various data-structures. A fast compiler, ported on various OS and processors, produces efficient native code.

Our constraint library includes standard integer finite domain variables, linear and non-linear

arithmetic constraints, reification, higher-oder constraints and some global constraints: difference, cardinality, sorting, etc. The search is controlled with goals in a Prolog way.

## 5.2 Results

We give in this section some results obtained for the different models applied on real data, for May 20th, 1999.

All the experiments are done with a precision $\epsilon$ (time unit) of 5 min and a maximum delay of 60 min, except where mentioned otherwise.

### 5.2.1 A Single Sector

In order to analyse the solutions of the different models, we first looked at a very simplified problem with a single sector and no variation of capacity. We chose the sector (of May 20th, 1999) concerned by the maximum number of flights (644). The hourly capacity is 40. The flights are expected to enter in the sector between 0h52 and 23h39.

It was first possible to check the equivalence and order between models (`M1` $\leq$ `M2` means that `M2` is more constrained than `M1`, i.e. that a solution for `M2` is a solution for `M1`):

$$\begin{aligned} \texttt{Basic} &\leq \texttt{Sorting} \\ \texttt{Basic} &\equiv \texttt{Sliding} \text{ with } \sigma = \delta \\ \texttt{Sliding} &\leq \texttt{Sorting} \\ \texttt{Sliding} \text{ with } \sigma = \epsilon &\equiv \texttt{Sorting} \end{aligned}$$

Table 1 reveals the corresponding numerical results: a more constrained model gets a higher cost. For various models and parameters settings are presented respectively in the last three rows the sum of all delays, the number of on-time flights (no delay) and the number of flights with delays less than 15 min. We see in this table that most of the flights are not delayed flights or get a small delay (compared to 644 concerned flights). Note that the average delay for all the solutions is smaller than the time precision (5 min).

| Model | $\delta$ | $\sigma$ | $\sum D_i$ | $\lvert\{D_i = 0\}\rvert$ | $\lvert\{D_i \leq 15\}\rvert$ |
|---|---|---|---|---|---|
| Basic | 60 | | 690 | 602 | 624 |
| Basic | 30 | | 1960 | 532 | 595 |
| Sliding | 60 | 30 | 1760 | 549 | 597 |
| Sliding | 60 | 15 | 2480 | 504 | 565 |
| Sorting, Sliding | 60 | $\epsilon$ | 3660 | 467 | 553 |

Table 1: Delays for different models and parameters

Numerical cost is a poor indicator about a solution. The expected differences between the three models are qualitative. In figures 6 and 7 is plotted the instantaneous load of the sector, i.e. the number of flights which will arrive during the next $\delta$ period. Dots correspond to the unregulated flow (the initial data) while solid curves show the solutions.

At first glance the `Basic` model does not seem to regulate much. Actually, it only insures that the curve goes under the capacity bound every $\delta$ minutes (every hour starting from 0).

The figure 7 corresponding to the `Sorting` model shows more interesting and expected results. The solution gives an almost constant load from 5h00 to 20h00.

Figure 8 shows the influence of the $\sigma$ parameter on the `Sliding` model. With $\sigma = \delta = 60$, we get the solution of the `Basic` model. With a smaller $\sigma$ (15), we observe that the load goes under the capacity every $\sigma$ minutes but has still chances to go up 10% over the capacity in between.

Figure 9 shows the side-effects of the `Basic` model. In this experiment, we reduced the capacity (10%) and augmented the max delay (120 min) to force more flights to be delayed. In figure 9 is plotted the instant number of flights inside the sector (one can notice that a plane does not stay for a long time in a sector). Peaks occur at the beginning of this periods: a delayed flight has to be scheduled during the next non-full period; then a minimum delay corresponds to the beginning of the period. The `Sorting` model does not suffer from this side-effect and ensures an even load.

| Model | Capacity | Result |
|---|---|---|
| `Basic` | 36 | $\infty$ |
| `Sliding` $\epsilon$ | 36 | $\infty$ |
| `Sorting` | 36 | Failure proved |
| `Sorting` | 37 | 19775 |

Table 2: Proof of infeasibility

Table 2 gives some indication of the ability of the models to prove that a problem has no solution. With a reduced capacity of 36, the failure could not be proved in a "finite time" ($\infty$ in the table) except using the `Sorting` model. Note the problem seems to have a phase transition since a solution is easily found with a slightly greater capacity (37).

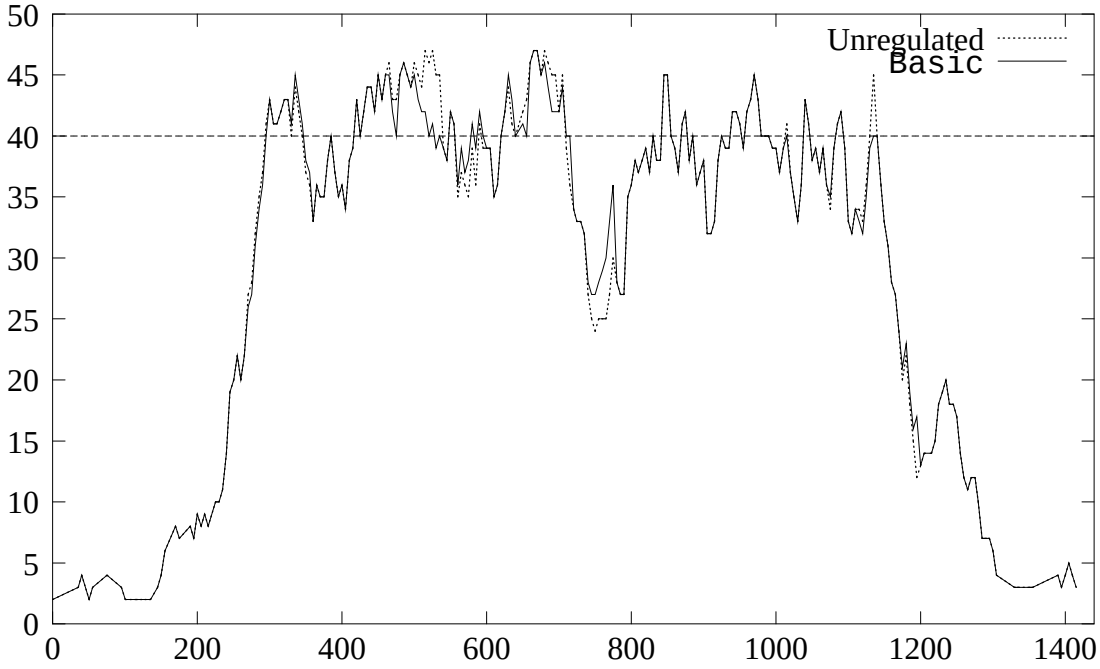


Figure 6: Regulation with the `Basic` model (flights entering during the next $\delta$ min). After regulation (solid curve), the load falls below the capacity every $\delta(=60)$ min.

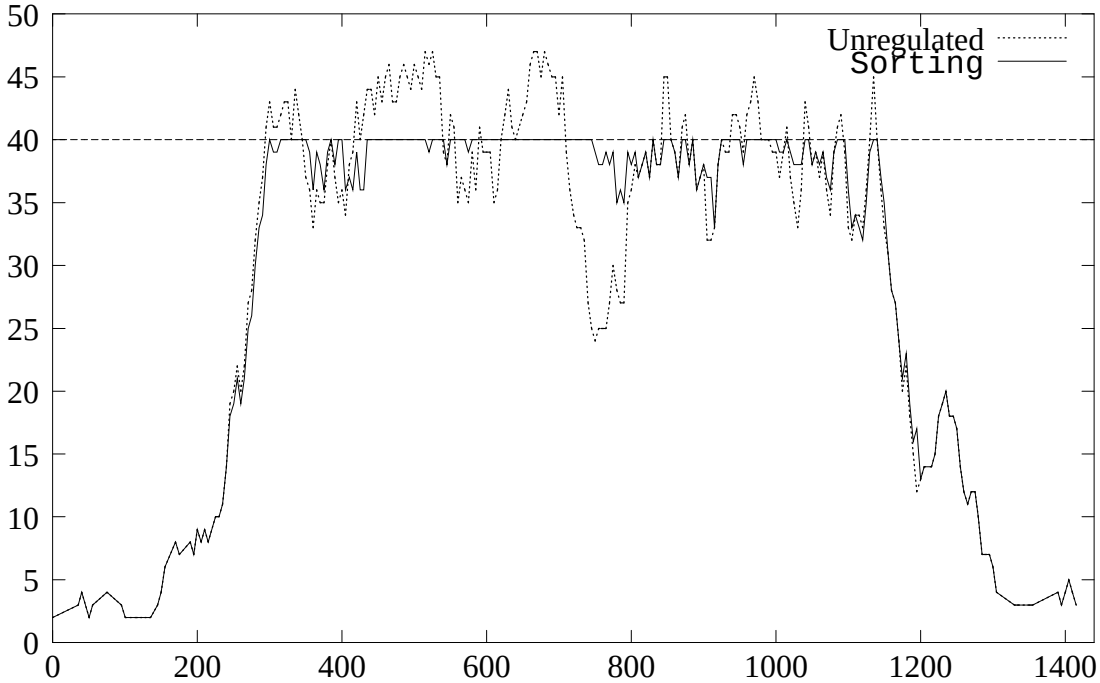


Figure 7: Regulation with the `Sorting` model (flights entering during the next $\delta$ min). The `Sorting` model ensures that the load remains constantly below the capacity.

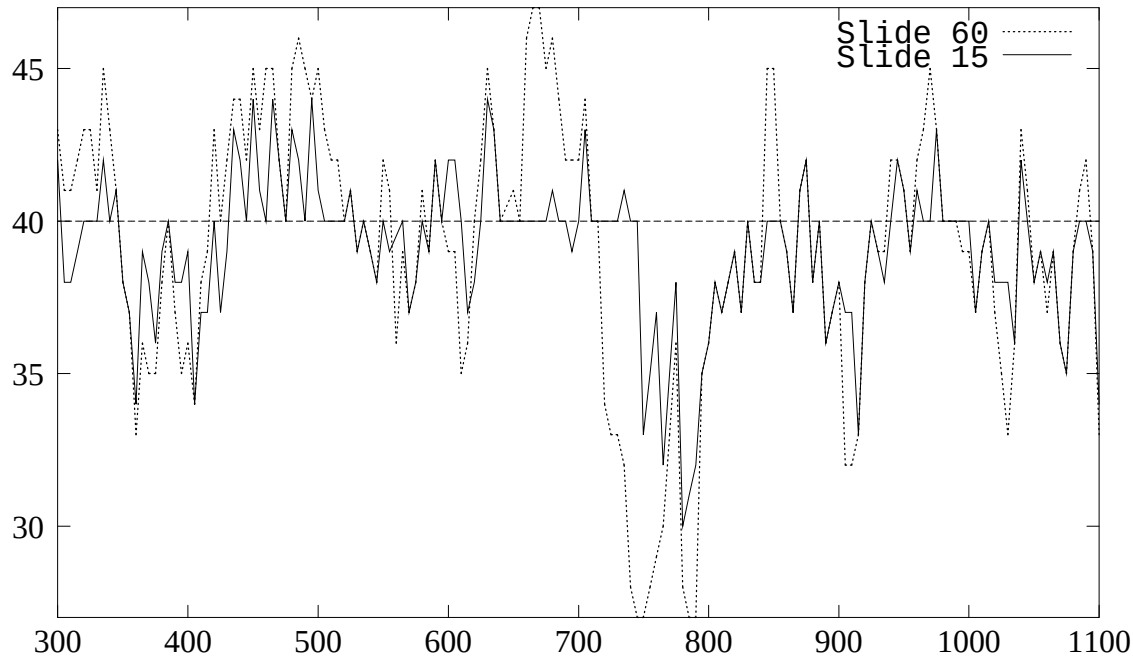


Figure 8: Influence of $\sigma$ with the `Sliding` model (flights entering during the next $\delta$ min; zoom between 5h and 19h20). The curves fall below the capacity every $\sigma$ min (dashed curve: $\sigma = 60$; solid curve $\sigma = 15$).

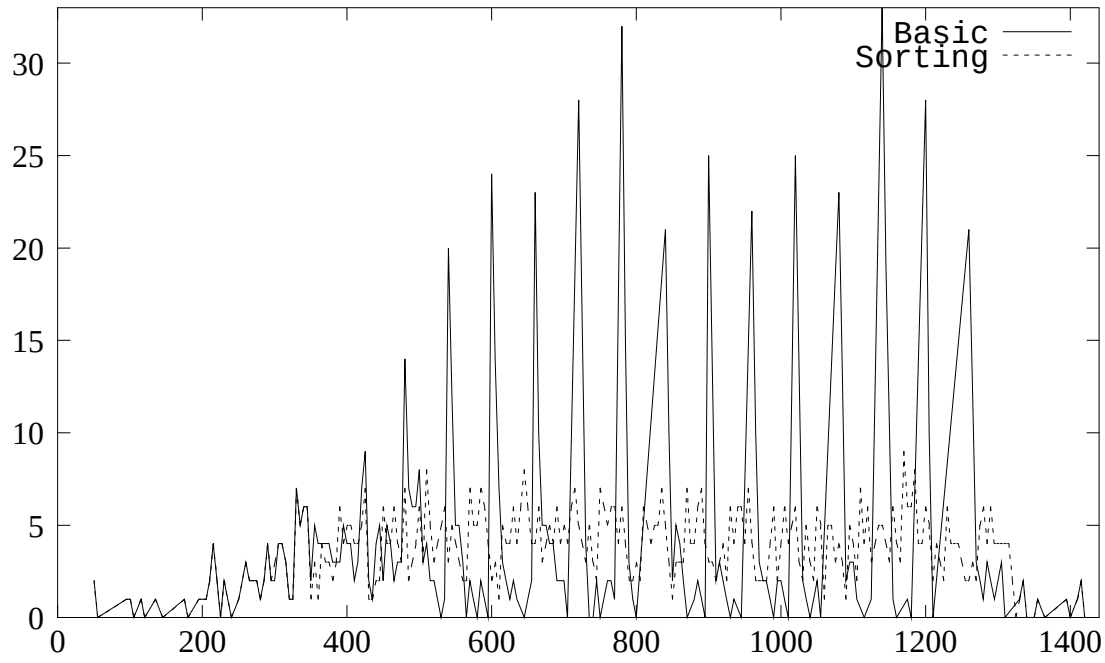


Figure 9: Instantaneous load (number of flights in the sector) for the `Basic` and the `Sorting` models. The `Basic` model tends to yield traffic peaks at the beginning of each period, whereas the `Sorting` model keeps a low average load.

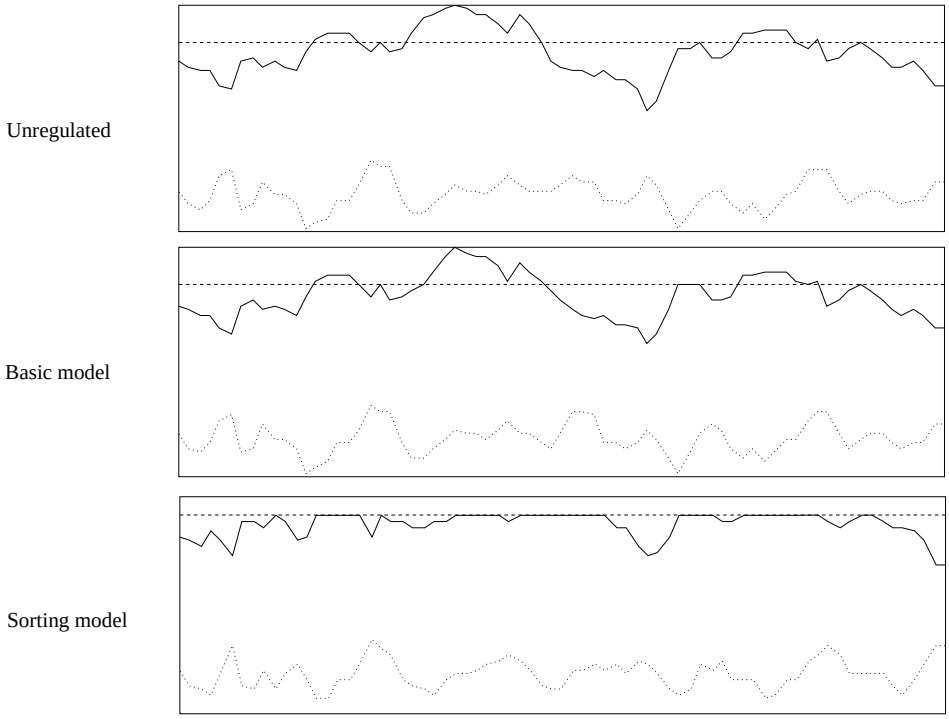


Figure 10: Brest Airspace Control Center; May 20th, 1999; sector J around noon. The `Basic` model has little effect on the load of crowded sector compared to the `Sorting` model.

#### 5.2.2 Full Problem

We were able to prove that there is no solution for the `Sorting` model with the given capacities. But solutions can be obtained with allowing an *overload* of the capacity of each sector. We found a solution with a capacity overload of 25%.

The regulation has no spectacular qualitative effect like in the simplified case; the whole problem is too much complex to have simple local properties. However, in figure 10 are compared the effects of the `Basic` and `Sorting` model on sector J around noon. The solid curves correspond to the flights entering the sector during the next 60 min and the dotted ones to the instantaneous number of flights present in the sector. This sector is very crowded and the regulation produced by the `Basic` model has little effect on the overload, whereas the `Sorting` model yields a load that levels out along the capacity line. This confirms the expected regular behaviour of the `Sorting` model.

## 6 Conclusion

Time slot allocation in ATFM is a hard combinatorial problem yet hardly solved by current CFMU systems. Capacity constraints are subject to interpretations and Constraint Programming provides an efficient and straightforward way to implement various models addressing control workload with different operational points of view. Non overlapping windows hardly cope with a realistic semantics of capacity constraints and would probably lead Control Centres to underestimate their capacities because of periodic traffic peaks whereas $\epsilon$-sliding windows and sort models ensure an even regulation inducing a much more constrained CSP with possibly higher overall delay costs. The sort model however needs fewer control parameters than its time windows counterpart. It is also far more efficient on fail proofs because of very powerful constraint propagation while offering similar computation time performances.

Large size and high dimensionality of air traffic input data make the slot allocation problem hard to optimize with respect to the cumulated delays criterion and future work should address this issue more efficiently. But the integration in the objective function of other factors (which may also be stated at the constraint modelling stage) like the regularity of workload distribution seems to be an important operational requirement and would penalize the basic

models.

## Authors

**Nicolas Barnier** is a PhD student in Computer Science at the University of Toulouse and works at CENA's Global Optimization Laboratory. He is also a graduate assistant at the ENAC (the French Civil Aviation University). He obtained his MSc in Computer Science from the University of Toulouse and graduated ENAC in 1997. His research interests focus on Constraint Programming, Combinatorial Optimization and Genetic Algorithms. He is involved in the design and implementation of FaCiLe, an open source Functional Constraint Library.

**Dr Pascal Brisset** is a lecturer at ENAC and research assistant at CENA's Global Optimization Laboratory since 1994. His research interests focus on Constraint Programming, Combinatorial Optimization, Modelling Languages and hybrid solvers applied to ATM problems. He is an alumnus of the École Normale Supérieure and received a PhD in Computer Science from the University of Rennes. He was involved in the design and implementation of the ECL$^i$PS$^e$ constraint solver and is the main designer of Prolog-MALI (a $\lambda$Prolog compiler) and the FaCiLe constraint solver.

**Thomas Rivière** is a MSc student in Computer Science at the University of Toulouse and student at ENAC. His research interests focus on Constraint Programming, Scheduling and hybrid solvers.

# References


[Barnier and Brisset, 2001] Barnier, N. and Brisset, P. (2001). *FaCiLe : A Functional Constraint Library*. ENAC/CENA, (`http://www.recherche.enac.fr/opti/facile`).

[Barták, 1998] Barták, R. (1998). On-line guide to constraint programming. `http://ktiml.mff.cuni.cz/~bartak/constraints`.

[Bertsimas and Stock, 1995] Bertsimas, D. and Stock, S. (1995). The air traffic flow management problem with enroute capacities. Technical report, MIT.

[CFMU, 2000] CFMU (2000). *Basic CFMU Handbook - General & CFMU Systems*. Eurocontrol CFMU, Brussels, 6.0 edition.

[Dincbas et al., 1988] Dincbas, M., Hentenryck, P. V., Simonis, H., Aggoun, A., and Graf, T. (1988). The constraint logic programming language CHIP. In *Int. Conf. Fifth Generation Computer Systems*, volume 1, pages 693–702, Tokyo, Japan.

[ECL, 2001] ECL (2001). ECL$^i$PS$^e$ user manual (`http://www.icparc.ic.ac.uk/eclipse`).

[Gotteland et al., 2000] Gotteland, J.-B., Kerlirzin, P., Manchon, S., and Plusquellec, C. (2000). Building and evaluating a minimal regulation scheme. In *3rd USA/Europe Air Traffic Management R&D Seminar*, Napoli.

[Guernalec and Colmerauer, 1997] Guernalec, N. B. and Colmerauer, A. (1997). Narrowing a $2n$-block of sorting in $O(n \log n)$. In *Principles and Practice of Constraint Programming*. Springer-Verlag.

[IP/99/924, 1999] IP/99/924 (1999). European commission: Fifteen countries, a single European sky, (`http://europa.eu.int/comm/pr_en.htm`).

[Leroy, 2000] Leroy, X. (2000). The Objective Caml System: User's and reference manual (`http://caml.inria.fr`).

[Maugis, 1996] Maugis, L. (1996). Mathematical programming for the air traffic flow management problem with en-route capacities. In *in Proceedings of the 14th Triennial World Conference of the International Federation of Operational Research Societies*.

[Michel and Hentenryck, 1997] Michel, L. and Hentenryck, P. V. (1997). Localizer: A modeling language for local search. In *Proceedings of the Third Conference on Principles and Practice of Constraint Programming*.

[Plusquellec and Manchon, 1998] Plusquellec, C. and Manchon, S. (1998). Description du module d'allocation de crneaux utilisant la programmation par contraintes implant dans shaman. Technical report, CENA/RFM/NT97.105.

[Rivière, 2001] Rivière, T. (2001). Allocation de créneaux à la SHAMAN avec FaCiLe. Technical report, CENA.

[Solver, 1999] Solver (1999). ILOG Solver 4.4 user's manual (`http://www.ilog.fr`).

[Van Hentenryck, 1989] Van Hentenryck, P. (1989). *Constraint Satisfaction in Logic Programming*. MIT Press, Cambridge, MA.