Using clp(FD) to Support Air Traffic Flow Management

Denise Chemla[1,3], Daniel Diaz[2], Philippe Kerlirzin[1], Serge Manchon[1]

[1] CENA, Orly Sud 205, 94542 Orly Aérogare Cedex, France
[2] INRIA, Domaine de Voluceau, 78153 Le Chesnay, France
[3] SYSECA, 315, bureaux de la Colline, 92213 Saint-Cloud, France

Abstract. In this paper, a Constraint Logic Programming (CLP) approach is used to solve an Air Traffic Flow Management (ATFM) problem, the aircraft departure slot allocation. Moreover, our purpose is to show that CLP, combining the declarativity of logic programming with the efficiency of constraint solving, is well suited to model many combinatorial optimization problems involved in the ATFM domain. clp(FD), a Constraint Logic Programming language with Finite Domain constraints has been chosen to implement our practical application.

1 Introduction

The density of traffic over Europe has been steadily increasing for several years. This growth is difficult to manage and causes delays for passengers and work overloads for controllers. ATFM aims at adapting a variable demand (the airplanes which want to fly) to the variable available capacity of the system of control so as to use this capacity at best. It has significant safety and economic consequences as well.

Our research is pursued in the French Air Navigation Research Center (CENA), that is involved in the development of the future Air Traffic Control system. This work will be integrated into the SPORT decision support system for traffic flow management. This system helps flow managers in analyzing traffic data and in preparing flow management measures. It is operational in the six French Air Control Centers and at the Eurocontrol Central Flow Management Unit located in Brussels. Figure 1 is a view of the SPORT system showing the French sectors and the most congested routes.

In this paper, a CLP approach is used to solve the ATFM problem of departure slot allocation. clp(FD), a CLP language with Finite Domain constraints has been chosen to implement this practical application. The departure slot allocation is done manually until now, so we couldn't compare our approach with linear methods that could have been yet used. Such a comparison (linear versus CLP methods to solve ATFM problems will be done in our next research).

The structure of this paper is as follows: Sect. 2 gives a brief description of ATFM; the third one presents the clp(FD) language features and a new

constraint developed for our needs; Sect. 4 shows how `clp(FD)` can be used to solve the departure slot allocation problem under capacity and/or flow rate constraints.

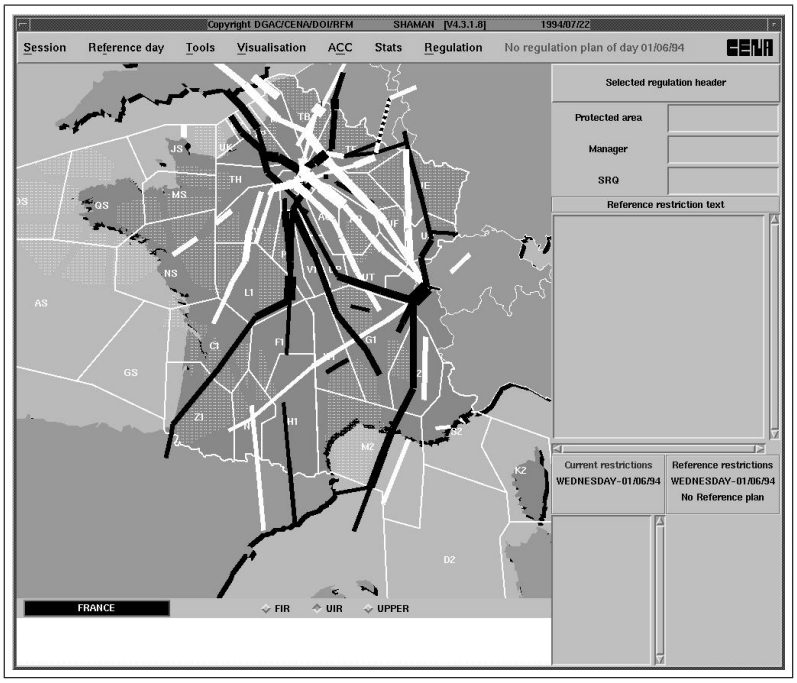

Fig. 1. Display of SPORT system

2 Problem Context

2.1 Air Traffic Flow Management Overview

ATFM aims at adapting a variable demand to the variable capacity of the system of control. Its first objective is to assure, by smoothing the flow of aircraft, that unacceptable levels of traffic congestion do not develop. Its second goal is to perform this task without imposing unnecessary flow restrictions.

France is overflown by all European air-carriers and even more. Its airspace is a patchwork of about 90 sectors. Each of them is under the responsibility of a pair of controllers. A flight crosses several sectors along its route. The radar controller works on a radar position and gives instructions to pilots via a radio link. He (she) maintains separation between planes and keep them away from specific dangers such as military areas, storms. The planning controller takes it

on to find convenient entry and exit flight levels and the right coordination with neighbouring sectors. When traffic allows it, sectors can be grouped (there are about 120 possible groups of sectors).

The European ATFM activity is structured in three levels:

1. **strategic level**: at this level, long term measures are defined such as the traffic orientation scheme that dictates the routes operators have to use to go from specific departure areas to specific arrival areas. National measures are also defined at this level: modulation of controllers working hours, agreements between military and civil air traffic services, or use of main platforms scheduling;
2. **pre-tactical level**: an important feature of the sector is its capacity, i.e. the maximum number of flights that can enter the sector per hour. This capacity is variable along the day and along the year. Generally, it is greater than the demand when one team of controllers manages one sector. However, some sectors are regularly overloaded due to a limited number of controllers, to structural reasons, or to peak traffic: in that case, the demand can be greater than the capacity during certain periods of the day. The pre-tactical ATFM consists in preparing, two days before the tactical day, a regulation plan which is a set of flow rate restrictions intended to avoid overloads within critical sectors.
3. **tactical level**: is sub-divided into two processes:
   - **slot allocation**: airline operators affected by the regulation plan have to ask for departure slots two hours before scheduled take-off, so that each aircraft enters critical sectors at the right time. In the French flow management unit, slots are allocated according to a *first-demander-first-served* principle.
   - **real time supervision**: during the pre-tactical phase, relying on traffic periodicity, flow managers forecast the traffic to come using recorded data. Because of last time changes (weather conditions, technical failures, ...), it is necessary to monitor the effects of the regulation plan and to adapt some restrictions in real time to cope with excess demand and under-used capacities.

2.2 The Slot Allocation Problem

First of all, we will focus on solving the slot allocation problem under capacity constraints; a small example is presented in Sect. 4. We will then extend the model in order to integrate another type of constraints, called "flow rate constraints", to organize the delays undergone by the flights in the first application.

Capacity Constraint Definition. A capacity constraint is a relation between an airspace volume A (a sector or a group of sectors), a temporal period T and an hourly rate $N/\delta t$ (N is the maximum number of aircraft that can enter the

sector each δt minutes). The constraint is satisfied if during T, at most N flights per contiguous slices of δt minutes width enter A[1]. N is called **capacity** of A. The problem consists in avoiding overloads all along the tactical day by delaying certain flights. In our model, we have made the choice that capacity constraints affect all flights without any discrimination: no flight is privileged with regard to `CLP` slot allocation.

Description Of The Slot Allocation Problem Model. The slot allocation problem under capacity constraint can be defined by its input and output data. The input data are:

- the demand: constituted of a set of filled flight plans:
  $\{F_i : (O_i, D_i, SR_i : (S_{i,1}, EET_{i,1}, \ldots, S_{i,n}, EET_{i,n}))\}$,
  where F_i is a flight identifier, O_i and D_i are its origin and destination, $S_{i,1}, \ldots, S_{i,n}$ are the sectors crossed by the flight, $EET_{i,1}, \ldots, EET_{i,n}$ are the expected (by the flight carrier) entry times in those sectors ($EET_{i,1}$ is the expected *departure* time of the flight). There are 6000 flight plans a day on average.
- the resources: defined by a set of airspace volume capacity constraints:
  $\{CC_j : (\{S_{j,1}, \ldots, S_{j,m}\}, Capa_j, H1_j, H2_j)\}$
  where CC_j is a constraint identifier, $S_{j,1}, \ldots, S_{j,m}$ are the constrained airspace volumes by the capacity constraint, $Capa_j$ is the capacity (half-hourly maximum number of flights entering in the constrained airspace volume), $H1_j$ and $H2_j$ are the bounds of the application period of the constraint. An example of such a capacity constraint is

  $$CC_1 : (\prime UT\prime, \prime TU\prime,\ 26,\ 600,\ 660)$$

  that expresses that at most 26 aircraft can enter the group of sectors $\{\prime UT\prime, \prime TU\prime\}$ from 10am to 11am (in minutes from 0am).

The output data is a set of satisfactory departure times $\{SET_{k,1}\}$ of the flights F_k such that all capacity constraints are satisfied and the average delay undergone by a flight is minimized.

3 `clp(FD)` in a Nutshell

As introduced in Logic Programming by the CHIP language, `clp(FD)` [6] is a constraint logic language based on finite domains, where constraint solving is done by propagation and consistency techniques originating from Constraint Satisfaction Problems [15,17,10]. The novelty of `clp(FD)` is the use of a unique primitive constraint which allows users to define their own high-level constraints. The black-box approach gives way to glass-box approach.

[1] We have developed a new constraint, the `atmost_interval` constraint that enables the implementation of a certain number of aircraft per contiguous slices of δt-minute width; the cumulative constraint (of CHIP) allows reasoning on sliding windows of δt-minute width and is so too stringent for our needs.

3.1 The Constraint *X in r*

The main idea is to use a single primitive constraint X *in* r, where X is a *finite domain (FD) variable* and r denotes a *range*, which can be not only a *constant range*, e.g. 1..10 but also an *indexical range* using:

- $min(Y)$ which represents the minimal value of Y (in the current store),
- $max(Y)$ which represents the maximal value of Y,
- $val(Y)$ which represents the value of Y as soon as Y is ground.

A fragment of the syntax of this (simple) constraint system is given in table 1.

```
c ::= X in r (constraint)

r ::= t..t   (interval range)
      {t}    (singleton range)
      ...

t ::= C      (parameter)
      n      (integer)
      min(X) (indexical min)
      max(X) (indexical max)
      val(X) (delayed value)
      t + t  (addition)
      t - t  (subtraction)
      t * t  (multiplication)
      ...
```

Table 1. fragment of the constraint system syntax

The intuitive meaning of such a constraint is: "X must belong to r in any store".

The initial domain of an FD variable is $0..\infty$ and is gradually reduced by X *in* r constraints which replace the current domain of X (D_X) by $D'_X = D_X \cap r$ at each modification of r. An inconsistency is detected when D'_X is empty. Obviously, such a detection is correct if the range denoted by r can only decrease. So, there are some monotone restrictions about the constraints [11]. To deal with the special case of anti-monotone constraints we use the general *forward checking* propagation mechanism [8] which consists in awaking a constraint only when its arguments are *ground* (i.e. with singleton domains). In `clp(FD)` this is achieved using a new indexical term $val(X)$ which delays the activation of a constraint in which it occurs until X is ground.

As shown in the previous table, it is possible to define a constraint w.r.t. the *min* or the *max* of some other variables, i.e. reasoning about the bounds of the intervals (*partial lookahead* [9]). `clp(FD)` also allows operations about

the whole domain in order to also propagate the "holes" (*full lookahead* [9]). Obviously, these possibilities are useless when we deal with boolean variables since the domains are restricted to 0..1.

3.2 High-Level Constraints and Propagation Mechanism

From X *in* r constraints, it is possible to define high-level constraints (called *user constraints*) as Prolog predicates. Each constraint specifies how the *constrained variable* must be updated when the domains of other variables change. In the following examples X, Y are FD variables and C is a *parameter* (runtime constant value).

```
'x+y=c'(X,Y,C):- X in C-max(Y)..C-min(Y),  (C1)
                 Y in C-max(X)..C-min(X).  (C2)

'x-y=c'(X,Y,C):- X in min(Y)+C..max(Y)+C,  (C3)
                 Y in min(X)-C..max(X)-C.  (C4)
```

The constraint `x+y=c` is a classical FD constraint reasoning about intervals. The domain of X is defined w.r.t. the bounds of the domain of Y.

In order to show how the propagation mechanism works, let us trace the resolution of the system $\{X+Y=4, X-Y=2\}$ (translated via `'x+y=c'(X,Y,4)` and `'x-y=c'(X,Y,2)`): after executing `'x+y=c'(X,Y,4)`, the domain of X and Y are reduced to 0..4 (C_1 is in the current store: X *in* $-\infty..4$, $C_2 : Y$ *in* $-\infty..4$). And, after executing `'x-y=c'(X,Y,2)`, the domain of X is reduced to 2..4 ($C_3 : X$ *in* 2..6), which then reduces the domain of Y to 0..2 ($C_4 : Y$ *in* 0..2).

Note that the unique solution $\{X=3, Y=1\}$ has not yet been found. So, in order to efficiently achieve consistency, the traditional method (arc-consistency) only checks that, for any constraint C involving X and Y, for each value in the domain of X there exists a value in the domain of Y satisfying C and vice-versa. So, once arc-consistency has been achieved and the domains have been reduced, an enumeration (called labeling) has to be done on the domains of the variables to yield the exact solutions. Namely, X is assigned to one value in D_X, its consequences are propagated to other variables, and so on. If an inconsistency arises, other values for X are tried by backtracking. Note that the order used to enumerate the variables and to generate the values for a variable can improve the efficiency in a very significant manner (see heuristics in [9]).

In our example, when the value 2 is tried for X, C_2 and C_4 are woken (because they depend on X). C_2 sets Y to 2 and C_4 detects the inconsistency when it tries to set Y to 0. The backtracking reconsiders X and tries value 3 and, as previously, C_2 and C_4 are reexecuted to set (and check) Y to 1. The solution $\{X=3, Y=1\}$ is then obtained.

3.3 Optimizations

The uniform treatment of a single primitive for all complex user constraints leads to a better understanding of the overall constraint solving process and allows

for (a few) global optimizations, as opposed to the many local and particular optimizations hidden inside the black-box. When a constraint X *in* r has been reexecuted, if $D'_X = D_X$ it was useless to reexecute it (i.e. it has neither failed nor reduced the domain of X). Hence, we have designed three simple but powerful optimizations for the X *in* r constraint [6,2] which encompass many previous particular optimizations for FD constraints:

- some constraints are *equivalent* so only the execution of one of them is needed. In the previous example, when C_2 is called in the store $\{X \text{ in } 0..4, Y \text{ in } 0..\infty\}$ Y is set to 0..4. Since the domain of Y has been updated, all constraints depending on Y are reexecuted and C_1 (X *in* 0..4) is woken unnecessarily (C_1 and C_2 are equivalent).
- it is useless to reexecute a constraint as soon as it is entailed. In `clp(FD)`, only one approximation is used to detect the entailment of a constraint X *in* r which is "*X is ground* ". So, it is useless to reexecute a constraint X *in* r as soon as X is ground.
- when a constraint is woken more than once from several distinct variables, only one reexecution is necessary. This optimization is obvious since the order of constraints, during the execution, is irrelevant for correctness.

These optimizations make it possible to avoid on average 50% of the total number of constraint executions on a traditional set of FD benchmarks (see [6,2] for full details) and up to 57% on the set of boolean benchmarks presented below.

3.4 Performances

Full implementation results about the performances of `clp(FD)` can be found in [6,2], and show that this "glass-box" approach is sound and can be competitive in terms of efficiency with the more traditional "black-box" approach of languages such as CHIP. On a traditional set of benchmark programs, mostly taken from [9], the `clp(FD)` engine is on average about four times faster than the CHIP system, with peak speedup reaching eight.

3.5 `atmost_interval` Constraint

To model capacity constraints, we needed to define a new constraint, the `atmost_interval` constraint.

The symbolic constraint `atmost_interval`$(N, [X_1, ..., X_m], L, U)$ is a user-defined constraint that holds if and only if at most N variables X_i are included within the interval $[L, U]$. This constraint can be defined via the relation:

$$Cardinal\{X_i / L \leq X_i \leq U\} \leq N$$

A boolean B_i is associated with each variable X_i and set to 1 if $L \leq X_i \leq U$ and to 0 otherwise. The sum of all B_i must be less than or equal to N. It is worth noticing that such a constraint should be "wired" in CHIP by the designers of the system whereas it is defined in `clp(FD)` as a user constraint.

4 Slot Allocation Satisfying Capacity Constraints

4.1 A Small Example

In the graphical representation (Fig. 2), 4 flights are represented as connected segments. Each segment corresponds to the crossing of a sector by a flight and is characterized by its length proportional to the crossing duration. The capacity constraints are represented on time axis: only 2 aircraft are allowed to enter S1 between times 2 and 3, and only 3 aircraft are allowed to enter S2 between times 3 and 5. Variables V_i represent the expected departure times of flights; variables V'_j are S1 and S2 the expected entry times (S1 and S2 are the only constrained sectors so only those variables are necessary).

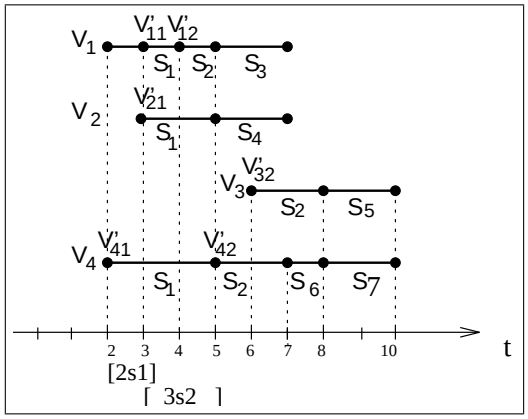

Fig. 2. Graphical representation of a small problem

4.2 `clp(FD)` Model

The slot allocation problem under capacity constraints can be modelled using 3 types of constraints:

1. *Domain constraints on departure time variables*: we saw that a CLP variable corresponding to the departure time is associated with each flight; in order to satisfy capacity constraints, the departure of a flight can be delayed, up to 3 hours (= 180 min, during our experiments).
   Each departure time variable will have to satisfy the following constraint:

   $$V_i \; in \; EET_{i,1}..EET_{i,1} + max\_delay$$

   where V_i is the departure time variable of flight F_i and $EET_{i,1}$ is the constant corresponding to the requested time of the flight (see 2.2 and 4.2).

2. *relations between sector entry time variables and departure time variables*: for each capacity constraint, a variable is created for a flight if the entering time in the first sector S_i of its route SR_j that belongs to A (where A is the constraint group of sectors - possibly a singleton) is within T (the constraint period) - see 4.1 and 4.2. Therefore, we set a new constraint on each of these variables V', as follows:

$$V'\# = V + \Delta$$

where V is the departure time and Δ is the time the flight needs to reach the sector S_i (translation constant).
3. `atmost_interval` *constraints*: finally, each capacity constraint is obviously modelled using an `atmost_interval` constraint, defined in 3.5. Its arguments are the capacity, the list of the variables identified in step 2, and the bounds of the constraint period interval (see 3.5 and 4.1).

This model is interesting because of its simplicity and transparency: since a flight can cross many sectors, it can be affected by several `atmost_interval` constraints. Regulators speak about "combining" restrictions but it is difficult for them to evaluate the effects of such restrictions. Such an overlapping problem is modelled in a transparent way. Another interest of our model is its extensibility: for instance, it would be obvious to affect a distinct delay to flights if we considered that some special flights could not be delayed.

4.3 `clp(FD)` Implementation of our Small Example

The `clp(FD)` implementation of the small problem presented in 4.3 is provided in table 2. The solution found by `clp(FD)` to this problem is $S1 = \{V1 = 2, V2 = 3, V3 = 6, V4 = 4\}$. Flights 1, 2 and 3 can take-off at their requested time, while flight 4 undergoes a 2 unit-of-time delay.

4.4 Optimization Trials - Heuristics

To solve real cases, we needed to implement some heuristics that we describe in the three points here below :

1. labeling strategy: `clp(FD)` labeling works on a list of variables L and backtracks first on the last element of L, then on the last but one and so on. This has a shortcoming: a solution of average delay d_1 can be labeled before a solution of average delay d_2 with $d_2 \leq d_1$. In our small example, the solution $S2 = \{V1 = 2, V2 = 4, V3 = 6, V4 = 2\}$ is not found whereas it is better in term of average delay than $S1$. For that reason, we have implemented a new labeling strategy that enumerates solutions in the order of increasing average delays. The solution $S2$ is encountered by such a labeling before $S1$. But we could not use this labeling in practical examples because it is too slow to find a solution. To solve practical problems, we have used an heuristic that leads `clp(FD)` labeling to find a good solution first. It consists

```
Solution([V1, V2, V3, V4]) :-
      V1 in 2..12,
      V2 in 3..13,
      V3 in 6..16,
      V4 in 2..12,
      V'11 #= V1+1,
      V'21 #= V2,
      V'41 #= V4,
      V'12 #= V1+2,
      V'22 #= V2+2,
      V'32 #= V3,
      V'42 #= V4+3,
      atmost_interval(2, [V'11, V'21, V'41], 2, 3),
      atmost_interval(3, [V'12, V'22, V'32, V'42], 3, 5),
      labeling([V1, V2, V3, V4]).
```

Table 2. implementation of our small problem with `clp(FD)` constraints

first in ordering take-off variables in L: the lower bound of the domain of an element i of L is always less than or equal to the lower bound of the domain of its successor in L. The second part of the heuristic consists in setting constraints according to an increasing order among the beginning of their application period. Thanks to this heuristic, `clp(FD)` finds a solution that minimizes the average delay;

2. time granularity: the variable domains have bounds from 0 to 1440 (number of minutes of a day); if we allow flights to be delayed up to 3 hours, domains can contain 180 values. Those size considerations can be redhibitory in practical examples (see the size of such examples in next section). So, to reduce memory size model, we have chosen to divide all variables and domain bounds by a "time granularity" that can be 5 or 10 minutes (or else);
3. discrete approach: because of the number of variables and constraints involved, it is difficult to treat a day taken as a whole. So, we have cut it in slices of 4 hours: when a flight is delayed by the constraints of a slice, its maximum delay is reduced accordingly.

4.5 Results

Figures 3 and 4 show traffic histograms of UM sector before and after the `clp(FD)` process: Fig. 3 depicts an overload between 10a.m. and 11a.m. while Fig. 4 has absorbed it.

Table 3 provides some runtime characteristics: the total number of variables is equal to the sum of the number of "indomain" constraints and of the number of "equality" constraints.

`clp(FD)` was processed on a pattern containing about 100 days of a year. This proves a certain stability with regard to the density of the trafic. When no

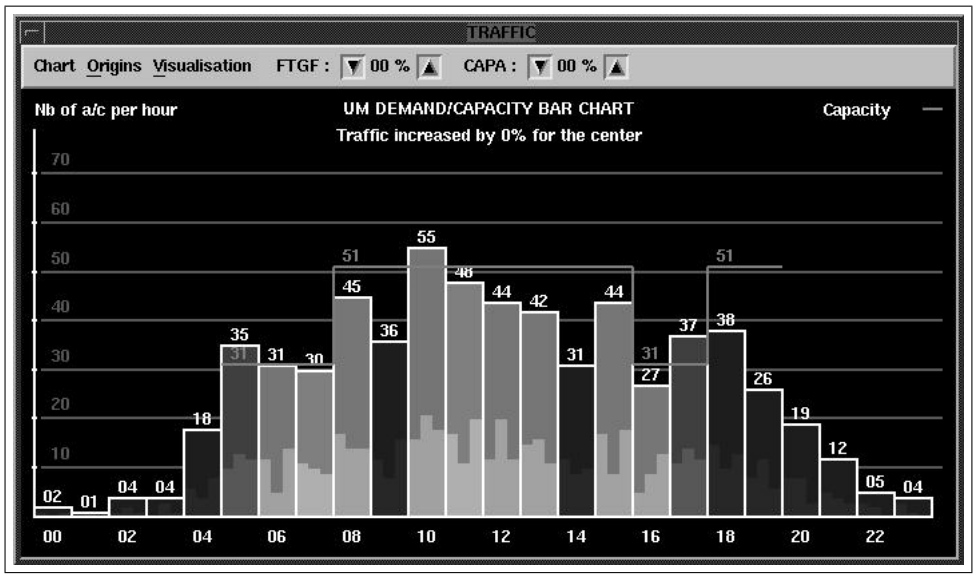

Fig. 3. UM traffic before CLP process

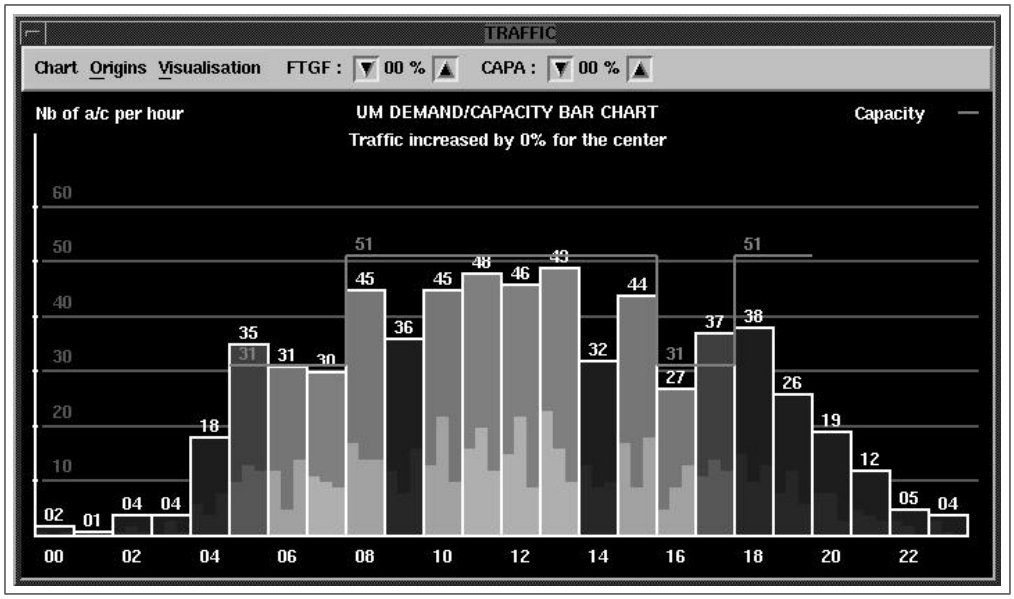

Fig. 4. UM traffic after CLP process

period length (h)	runtime	average delay (among delayed flights)	average delay (among all flights)	max delay	delayed flights (p.c.)	number of indom. constr.	number of atmost constr.	number of equality constr.	number of flights
3	$0'39''$	$19.16'$	$5.38'$	$50'$	28.10	957	193	13031	4714
4	$2'03''$	$18.83'$	$4.45'$	$55'$	23.64	1184	256	21700	4714
5	$4'08''$	$17.50'$	$1.90'$	$50'$	10.88	1397	320	29006	4714
5	$0'44''$	$25.78'$	$10.20'$	$75'$	39.58	815	218	12069	4714
6	$1'42''$	$5.73'$	$2.51'$	$16'$	43.80	1071	282	19269	4714
3	*error*					814	199	9967	4714
10	*error*					1761	393	40187	4714

Table 3. Some runtime examples

solution is found, we decrease the period length and/or increase the maximum delay that can be undergone by a flight.

To conclude this subsection, we can underline the fact that dealing only with capacity constraints to make slot allocation has some weaknesses: delays are distributed among all flights without any discrimination. So, if a regulation plan were created in such a way, it would have been impossible to adapt the restrictions to last minute changes during the real time supervision phase. Hereafter, we describe an extension of the model seen so far which permits to organize delays in a fairly manner. It corresponds to the way French flow managers work.

4.6 Extension of the Model to Integrate Flow Rate Constraints

Flow Constraint Definition. A flow constraint (called in Europe a "regulation") is a relation between a traffic flow F, a temporal period T and a rate $N/\delta t$ (N is the maximum number of aircraft that can feed the flow each δt minutes). F can be defined by a set of origins and/or a set of destinations and/or a set of beacons and/or a set of sectors and/or a flight level layer. Those properties of a flow are the characteristics that a flight must fulfill to be submitted to the flow constraint.

Generic example of flow constraint:
from *SetOfOrigins* **entering** *GroupOfSectors*
h1 **-** *h2* **:** $N/\delta t$

Instantiated example of flow constraint:
from *UK* **to** *Balearic*
10am **-** *11am* **:** $1/8$

The constraint is satisfied if during T, there are at most N flights belonging to F per slice of δt-minute width.

`clp(FD)` **Model.** We can detail the 3 types of constraints shown in 4.4 that are necessary to model the extension to flow rate constraints:

1. *domain constraints on departure time variables*: this step is identical to the first step defined in 4.4;
2. *relations between sector entry time variables and departure time variables*: capacity constraints are modelled in the same way than in 4.4; for each flow constraint, a variable V' that corresponds to the adequate instant is created for a flight if the flight belongs to the constrained flow and "reaches" the constraint (at instant V') within T. "Reaches" means that we will be interested in different instants of the flight according to the constraint type. This instant can be a sector entry time, a beacon over-flying time, a departure or arrival time. Except this, step 2 is identical to the second step above-mentioned in 4.4;
3. `atmost_interval` *constraints*: this step is identical to the third step above-mentioned in 4.4;

As we can see, the integration of flow rate constraints is very natural. This illustrates the declarativity and extensibility of our model.

4.7 A Simulation Aid Tool for Regulators - Cost estimation of Regulation Plans

Whatever the point of view may be, either local or global, regulators cannot have a precise idea of the effects of restrictions on traffic flows they impose because of the very large volume of data, the great interdependency between sectors, and the complexity of the air route network.

The interest of a simulation aid tool is to let the prominent rôle and the final choice to the regulators when they have at their disposal the cost estimation of a regulation plan provided by `clp(FD)`. Cost estimation can integrate criteria such as the average or maximum delay, the number of delayed flights, the number and duration of planned restrictions, the average number of restrictions affecting a flight, the difference between demand and capacity (it allows to save a security margin for imponderables). Such a tool can help them to avoid imposing unnecessary restrictions on flows. Our work has been integrated into the simulation aid tool SPORT (from which we have provided hardcopies in this paper). This integration has been easy to do because `clp(FD)` allows to obtain a C runtime program.

5 Conclusion and Further Works

This paper has shown how CLP is well adapted to solve ATFM problems such as departure slot allocation satisfying different types of constraints. Describing a possible extension of this practical application (flow rate regulation), we have highlighted expressiveness and flexibility of the CLP approach. It seems that

numerous ATFM applications can benefit from CLP advantages. Among them, we will now investigate other applications like rerouting, automatic search of flow rate regulations, evaluation of a capacity change cost, evaluation of flow constraints cost. The efficiency of `clp(FD)` language gives us a good hope to realize interesting further works.

References

1. M. Bonnard, S. Manchon, and P. Planchon. Bilan des études de la division AOC sur la régulation du trafic aérien, 1992.
2. P. Codognet and D. Diaz. Compiling constraints in `clp(FD)`, draft, 1993.
3. D. Colin de Verdière. Utilisation des techniques de recherche opérationnelle pour les études Air Traffic Management, 1992.
4. D. Diaz. `clp(FD)` *User's Manual*. INRIA, Le Chesnay, France, 1994.
5. D. Diaz. `wamcc` *Prolog Compiler User's Manual*. INRIA, Le Chesnay, France, 1994.
6. D. Diaz and P. Codognet. A minimal extension of the WAM for `clp(FD)`. In *10th International Conference on Logic Programming*, Budapest, Hungary, 1993. MIT Press.
7. J.M. Garot. Airspace Management in Europe: issues and solutions. In *IFORS 1993: 13th International Conference of Operational Research*, Lisbon, Portugal, 1993.
8. R.M. Haralick and G.L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, no 14:pp 263–313, 1980.
9. P. Van Hentenryck. Constraint satisfaction in logic programming. In *Logic Programming Series*, Cambridge, 1989. MIT Press.
10. P. Van Hentenryck, Y. Deville, and C.M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, no 57:pp 291–321, 1992.
11. P. Van Hentenryck, V. Saraswat, and Y. Deville. Constraint processing in cc(FD), draft, 1991.
12. P. Van Hentenryck, H. Simonis, and M. Dincbas. Constraint satisfaction using constraint logic programming. *Artificial Intelligence*, no 58:pp 113–159, 1992.
13. J. Jaffar and J.L. Lassez. Constraint logic programming. In *Principles Of Programming Languages*, Munich, Germany, January 1987.
14. J. Jourdan. Modelisation of terminal zone aircraft sequencing in constraint logic programming, 1992.
15. A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, no 8:pp 99–118, 1977.
16. S. Manchon, D. Chemla, C. Gobier, and P. Kerlirzin. Dossier de spécifications du Système Prétactique pour Optimiser la Régulation du Trafic aérien: SPORT V4.3, 1992.
17. B.A. Nadel. Constraint satisfaction algorithms. *Computationnal Intelligence*, no 5:pp 188–224, 1989.