56-63

137232

N93-18665

Scheduling of an Aircraft Fleet

Massimo Paltrinieri (*) Alberto Momigliano (**) Franco Torquati

Bull HN Italia
Direzione Sistemi Esperti
Pregnana Milanese, Milano
Italia

Abstract

Scheduling is the task of assigning resources to operations. When the resources are mobile vehicles, they describe routes through the served stations. To emphasize such aspect, this problem is usually referred to as the routing problem. In particular, if vehicles are aircraft and stations are airports, the problem is known as aircraft routing. This paper describes the solution to such a problem developed in OMAR (Operative Management of Aircraft Routing), a system implemented by Bull HN for Alitalia. In our approach, aircraft routing is viewed as a Constraint Satisfaction Problem. The solving strategy combines network consistency and tree search techniques.

1. Introduction

Two of the main concerns for a major airline are flight planning and aircraft routing.

Flight planning involves both technical and market issues, such as the choice of the cities to be served and the weekly frequency of flights. It produces an aircraft rotation, valid for a whole season, which we shall refer to as the *virtual plan* (see fig. 1); it consists of a periodical time table where flights are organized in lines, one for each *virtual* aircraft, an hypothetical resource that could perform them in absence of technical and maintenance constraints.

Aircraft routing assignes tail numbers - the identifiers of the aircraft - to flights, usually for a time window of 24 hours. This process, called *predictive routing*, is trial and error: routes are drawn on the virtual plan, performing switches, i.e. connections between flights on different lines of the plan, to satisfy the constraints that prevent an aircraft to cover the next flight on the same line. When there are no more tasks available for the given aircraft, an assignment to an already scheduled task is possibly invalidated. If the scheduler is not able to cover all the activities with the available resources, maintenance are delayed or, in some extreme cases, flights are dalayed or even cancelled. The schedule produced by predictive routing is coded in the *routing plan*, which differs from the virtual plan in replacing virtual with actual aircraft and arranging programmed maintenance. The routing plan is often modified in real time to avoid or contain, propagation of delays. Such an activity is said *reactive routing*.

This paper describes the Prolog kernel of OMAR (Operative Management of Aircraft Routing), an interactive system designed to provide predictive and reactive routing of the Alitalia fleet. Routing is formulated as a Constraint Satisfaction Problem (CSP): each variable (task) has a domain of possible values (aircraft) while constraints (relations between variables) are used to restrict such domains. Since the refined domains are not in general single-valued, solutions must be found by search, iteratively selecting an aircraft and assigning it to a set of consecutive flights. Aircraft selection is driven by the first fail principle: the most constrained aircraft is scheduled first. A controlled form of backtracking is implemented to partially recover from heuristics flaws while maintaining predictable response time.

Present addresses:

(*) Stanford University - Department of Computer Science - Stanford, CA 94305 - palmas@cs.stanford.edu

(**) Carnegie Mellon University - Department of Philosophy - Pittsburgh, PA 15213 - am4e@andrew.cmu.edu

2. Problem Definition

In this section we give a formal definition of both predictive and reactive aircraft routing.

The constraints of the problem are captured by the function *label*, that associates to each task the set of aircraft that can perform it. The function $start_{qs}$ returns the airport from which an aircraft has to depart after time q_S, the start time of the scheduling window.

Predictive Routing

Input

set T of tasks
set AP of airports
set AC of aircraft
set Q of times
schedule start time q_S and schedule end time q_e
total order $\leq$ on $Q \cup \{q_S\} \cup \{q_e\}$ s.t. $\forall q \in Q, q_S \leq q \leq q_e$

total function departing time,	dt: T -> Q
total function arrival time,	at: T -> Q
total function departing airport,	da: T -> AP
total function arrival airport,	aa: T -> AP
total function label,	label: T -> 2^{AC}
total function $start_{qs}$,	$start_{qs}$: AC -> AP

Output

an *aircraft routing*, i.e a total function s: T -> AC, s.t.

(i) $\forall t \in T$, $s(t) \in label(t)$
(ii) if $s^{-1}(ac)$ is not empty, then its elements can be ordered in a sequence (the routing path of ac)

$r_{ac} = \langle t_{ac,0}, t_{ac,1}, \ldots, t_{ac,n} \rangle$ such that

$da(t_{ac,0}) = start_{qs}(ac)$
$aa(t_{ac,i-1}) = da(t_{ac,i})$ $\quad i=1,\ldots,n$
$at(t_{ac,i-1}) < dt(t_{ac,i})$ $\quad i=1,\ldots,n$

Reactive Routing

Input

aircraft routing as defined above
an unexpected event

Output

an aircraft routing that copes with the unexpected event and most closely conforms to the given routing.

3. Aircraft Routing as a Constraint Satisfaction Problem

A task is said *programmed* if its departure and arrival airports and times are fixed. Flights, as well as main maintenance, are programmed, whereas secondary maintenance not necessary. The duration of each task is a given constant. Let us assume that we have a set $T = \{T_h, h=1,\ldots,m\}$ of programmed tasks to be scheduled in a time window of 24 hours.

Two tasks T_h and T_k are said to be *connectible* (denoted $T_h \rightarrow T_k$), if the following Prolog clause holds:

```
connectible(Th,Tk):-
        task_arrival_airport(Th,Airp),
        task_departure_airport(Tk,Airp),
        task_arrival_time(Th,MinArrT),
        task_departure_time(Tk,MaxDepT),
        ground_time(Airp,GrT),
        ArrT0 is MinArrT + GrT,
        ArrT0 < MaxDepT.
```

In other words, task T_h is connectible to task T_k iff the arrival airport of the former is equal to the departure airport of the latter and the arrival time of the former plus the ground time precedes the departure time of the latter. The graph of the connectibility relation is said the *connection graph*. It is directed and acyclic. Fig. 2 shows the connection graph for the portion of virtual plan in fig. 1.

We say that T_h *precedes* T_k and write $T_h < T_k$ iff (T_h, T_k) is in the transitive closure of ->. If neither $T_h < T_k$ nor $T_k < T_h$, then T_h and T_k are said *incompatible*, denoted $T_h >/< T_k$: incompatible tasks cannot be assigned to the same aircraft. A *routing path* P is a finite sequence of elements from T

$$P = \langle T_1, T_2, \ldots, T_n \rangle$$

such that $T_h \rightarrow T_{h+1}$ for each h, $1 \leq h < n$. A path S is *operable* by aircraft Ac if each task in the path is operable by Ac, i.e. there are no technical reasons that forbid the assignment to Ac.

An initial state for the fleet is a one-to-one map from Acs, the set of aircraft in the fleet, to a subset of T, the set of programmed tasks. The image of Acs under such map is the set of *initial* tasks of T, which correspond to those nodes in the connection graph with no entering arcs. The set of *final* tasks is the set nodes in the connection graph with no exiting arcs. In the following, paths will have an initial task as first element of the sequence; the idea is that paths are the formalization of the routes that an individual aircraft may cover, starting from its initial state.

We look at the elements of T as variables which take their values from the domain Acs. As already mentioned, a label of a task is the set of aircraft that can perform it. This concept can be extended to the set of all tasks: the *labeling* of the set T is a map l : T -> *P*(Acs), where *P*(Acs) is the powerset of Acs.

Constraints are relations in Acs x *P*(T) that are used to refine the labels of tasks. They come in two types: a *commitment* constraint between aircraft Ac and tasks $T_1,\ldots,T_n$ requires that Ac executes at least one of those tasks; an *exclusion* constraint between an aircraft Ac and and tasks $T_1,\ldots,T_n$ requires for Ac to be excluded from those tasks.

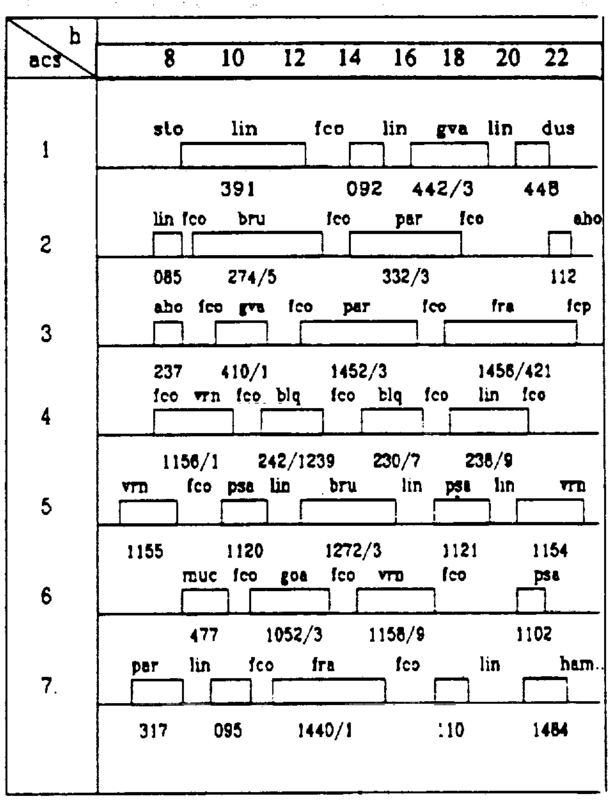

Fig. 1. A portion of about one-fourth of the virtual plan for the DC-9 fleet.

Each singleton labeling that satisfies all the constraints is an *aircraft routing*, i.e. a solution to the routing problem formalized in sect. 2. Such a singleton labeling generates a partition of the set T of tasks such that each element of the partition is a *routing path* for a distinct aircraft.

4. Routing Process

The routing process implemented in OMAR starts loading the state of the fleet and the relevant information on the tasks to be scheduled from the Alitalia database. A necessary, but not sufficient, condition for the existance of a fleet routing is checked, namely whether the number of resources available to be assigned to each task is always greater than or equal to zero. We briefly describe the algorithm, linear in the number of tasks, that tests such condition.

Each airport airport served by the fleet identifies a sequence of chronologically ordered events belonging to one of two classes: departures or arrivals. Each task entails two events, its arrival and departure, unless it is initial, in which case we consider only the arrival. A resource counter representing, at each time, the balance between arrivals and departures, is associated at every airport. The resource counter is initially set to 0 and is incremented or decremented, at each flight arrival or flight departure, respectively. If, scanning the whole plan, the counter of some airport becomes negative, the necessary condition is not satisfied and no routing exists. On the other hand, if the counters are always grater than or equal to zero, then the condition is satisfied and the system enters its next stage.

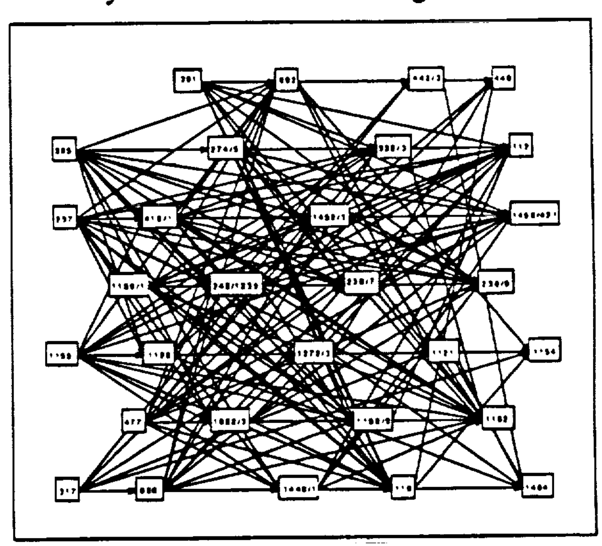

Fig. 2. The connection graph for the virtual plan in fig. 1.

A sample list of events at Linate airport is shown below.

Time	Event	Flight	Resource Level
17:50+0	d	448	0
17:25+35	a	267	1
17:45+35	a	074	2
18:30+0	d	316	1

Observe that the arrival of flight 267 at 17:25, given the ground time of 35 minutes, follows the departure of the flight 448 at 17:50.

The constraint satisfaction algorithm refines the labels so that most dead-ends are avoided and expiry maintenance requirements are implicitly satisfied: this means that aircraft planned for the latter tasks are excluded by those routes that do not lead to the set of airports where maintenance jobs are possible.

If the network is not found consistent, no complete routing exists and the control goes to the human scheduler who relaxes the constraints. It is our opinion that this kind of expertise cannot be adequately simulated by a computer, since the knowledge required to recognize the causes of an inconsistent situation and suggest a solution is too extended and fuzzy. If, on the other hand, everything is succesfull, the system is ready to schedule.

The aircraft are sorted in decreasing order according to the number of occurrences inside the labeling; the idea is that the aircraft coming first in this order are the most constrained ones, since they have a smaller number of tasks on which they can be enrouted. Routes are then created according to such an order by the Prolog procedures sketched below.

```
route_gen([Ac/Acs],Lab,NewLab):-
        pathgen(Ac,Lab,TmpLab),
        !,
        route_gen(Acs,TmpLab,NewLab).
route_gen([],Lab,Lab).

path_gen(Ac,Lab,NewLab):-
        last_started(Ac,Task),
        path_gen(Ac,Task,Lab,NewLab).

path_gen(Ac,Task,Lab,NewLab):-
        select(Ac,Task,Lab,NextTask,TmpLab),
        path_gen(Ac,NextTask,TmpLab,NewLab).
path_gen(_Ac,_Task,Lab,Lab).
```

The recursive procedure *route_gen/3* terminates when the list of aircraft to be scheduled is empty. It searches for a solution in depth-first mode, generating a descendant of the most recently expanded node and backtracking if some dead end is reached. If we relied exclusively on backtracking, the process duration would be unpredictable. Fortunately, we have developed some criteria that help us to discard paths likely to fail. On each aircraft *Ac*, *route_gen/3* calls *pathgen*/3, passing as parameters the aircraft *Ac* and the labeling *Lab* and returning a new labeling *TmpLab* in which the tasks assigned to *Ac* are the generated path. The procedure *pathgen*/4 builds a path recursively, task after task, starting from the first one returned by *last_started/2*.

A limited amount of backtracking is allowed: different choices are considered only during the coupling of a task with one of its direct offsprings. Yet paths cannot be invalidated after its completion (note the use of the cut sign '!' after *pathgen/3*). In case of failure, the interaction with the user is more effective. In our experience, after the relevant modifications have been performed, another run of the scheduler is usually sufficient to achieve a complete solution.

Let us analize the path generation process in more detail. The problem is not trivial, since there are both local and global optimizations which influence the choice at various extents, often in opposite directions. For instance, we could always choose the first task departing after the given one (local optimization), but this could generate a new line switch hard to manage in the overall routing (global optimization).

```
select(Ac,Task,Lab,NextTask,NewLab) :-
        propose(Ac,Task,Lab,NextTask),
        check_rc(Task,NextTask),
        update_lab(Ac,NextTask,Lab,NewLab).

propose(Ac,Task,Lab,NextTask):-
        get_methods(Ac,Task,Methods),
        member(Method,Methods),
        offsprings(Task,Offs),
        choose(Method,Ac,Offs,Task,NextTask).

get_method(Ac,Task,Methods):-
        rule(Condition,Methods),
        apply(Condition,Ac,Task).

rule(open_switch, [close_switch,straight,closest,stop]).

rule(default, [straight,open_switch,closest,stop]).
```

The basic step of the path generation process is performed by the Prolog procedure *select/5* shown above. Given an aircraft *Ac*, just assigned to a flight or maintenance (*Task*), *select/5* extends the path of *Ac* to a new flight or maintenance (*NextTask*). The procedure *propose/4* returns *Nextask*, then *check_rc/2* checks whether the resource counter becomes negative: in such a case it fails, otherwise it succeedes and the labeling is updated, aircraft *Ac* being assigned to *NextTask*. The path of *Ac* is extended with *NexTask* by *propose/4* as follows: first, a list *Methods* of methods compatible with *Ac* and *Task* is selected by *get_methods/3*; then, one *Method* is chosen nondeterministically from such a list; after, the offsprings of *Task* in the connection graph are returned by *offsprings/2* and finally, one of them, *NextTask*, is returned by *choose/5*, which basically applies *Method* to the given *Ac* and *Task*.

A method is a technique to choose the next task that extends a given path. Methods are gathered in lists and are associated to conditions. The relation between conditions and lists of methods is defined by *rule/2*. Two sample rules are shown above for the *open_switch* (remember that an aircraft opens a switch when its path is extended on a different row) and the *default* conditions. Given *Ac* and *Task*, if a condition is applicable to *Ac* and *Task*, which is checked by *apply/3*, a list of methods is returned by *get_methods/3*. Such methods are tried in the same order as they appear in the *Methods* list, the first one being the most desirable. For any possible *Ac* and *Task* there is at least one rule whose condition is satisfied, thus a list of methods is always selected, eventually by the *default* rule. In such a case, the list of methods tries to extend the path on the same line of the virtual plan with the *straight* method, which is considered optimal, otherwise a switch is opened by *open_switch*; if it is not possible to open a switch, the closest flight is selected by *closest* to minimize the consumption of the resources; if even this method is not applicable, the path is terminated by *stop*.

5. Conclusions

Aircraft routing is a problem for which no exact solution is known. Consequently, all models are heuristic and research is now concentrating on the systematic interaction between human and computer.

OMAR is an interactive system for the routing of the Alitalia fleet. Its kernel is presently composed of 20,000 lines of Quintus Prolog source code, and the system's response time is satisfactory. Once the derived structures have been computed from the primary database, the fleet routing is returned nearly in constant time (approximatively 30 seconds for a fleet of 26 aircraft with 170 flights).

Moreover, if the constraints are compatible with complete schedules, there is a very high probability that the system succeeds finding one of them. Of course, we cannot expect that the solution perfectly matches the user's expectations. According to our experience, however, an intervention by the user modifying, on average, five assignments, is suffucent to reach such an accomplishment.

In the tests supplied by Alitalia so far, OMAR's solutions can be compared with those of a senior scheduler.

References

[Da] Davis E., Constraint Propagation with Interval Labels, *Artificial Intelligence*, 32, 1987, 281-331.

[De&Pe] Dechter R. & Pearl J., Network-Based Heuristics for Constraint Satisfaction Problems, *Artificial Intelligence*, 34, 1988, 1-38.

[Et&Ma] Etschmeier M.M. & Mathaisel D.F.X., Aircraft Scheduling: the State of the Art, *XXIV AGIFORS Symposium*, Strassbourg, 1984,181-225.

[Ha&El] Haralick R.M. & Elliot G.L., Increasing Tree Search Efficiency for Constraint Satisfaction Problems, *Artificial Intelligence*, 14, 1980, 263-313.

[Na] Nadel B.A., Tree Search and Arc Consistency in Constraint Satisfaction Problems, in Kanal & Kumar (eds), *Search in Artificial Intelligence*, Springer-Verlag, 1988.

[Ri] Richter H., Optimal Aircraft Rotations based on Optimal Flight Timing, *VIII AGIFORS Symposium*,1968,34-69.

[Ste&Sha] Sterling L. & Shapiro E., *The Art of Prolog Programming*, MIT Press, Cambridge, Massachussets, 1986.

[Wal] Waltz D., Understanding Line Drawings of Scenes with Shadows, in *The Psychology of Computer Vision*, edited by P. H. Winston, McGraw-Hill Company, 1975.