

# The $K$ King Problem, an Abstract Model for Computing Aircraft Landing Trajectories: On Modeling a Dynamic Hybrid System with Constraints

Konstantin Artiouchine

Thales TRT, Domaine de Corbeville, 91404 Orsay CEDEX, France and CNRS LIX, Ecole Polytechnique, 91128 Palaiseau, France, Konstantin.Artiouchine@polytechnique.org

Philippe Baptiste

CNRS LIX, Ecole Polytechnique, 91128 Palaiseau, France, Philippe.Baptiste@polytechnique.fr

Juliette Mattioli

Thales TRT, Domaine de Corbeville, 91404 Orsay CEDEX, France, Juliette.Mattioli@thalesgroup.com

Motivated by the problem of computing trajectories of a set of aircraft in their final descent, we introduce the  $K$  king problem, a dramatic simplification of the initial problem in which time and space are discretized. A constraint-based model relying on several specific global constraints is introduced. Computational experiments are reported and show that small instances of this problem can be solved in reasonable time.

*Key words:* air-traffic control, scheduling, constraint programming

*History:* Accepted by John Hooker, Area Editor for Constraint Programming and Optimization; received April 2005; revised February 2006; accepted June 2007.

---

## 1. Introduction

Systems are often modeled by differential equations derived from physical laws that capture the dynamics of the system. Control theory is devoted to this field of research. Hybrid systems are characterized by the interaction of continuous and discrete models (e.g., the system is modeled by variables taking values from a continuous set and variables taking values from a discrete one).

Motivated by long-term industrial applications of THALES, (a major electronic systems company acting in areas such as defense, aerospace, airline security and safety, and transportation), we study a hybrid system where aircraft trajectories have to be computed when

aircraft reach the final descent in the Terminal Radar Approach CONtrol area (TRACON). Trajectories must be compatible with aircraft dynamics while keeping the distance between aircraft, at any time, larger than some predefined value. Moreover, there is minimum delay between any two consecutive landings on the same runway under which safety is not granted. Our objective is to determine the order in which aircraft land as well as their trajectories in order to minimize the maximal landing time.

Similar problems have been studied in the literature. In the “static” version studied in (Artiouchine et al. 2004) and (Beasley et al. 2000), all possible trajectories are pre-computed for each aircraft taken independently from others. From this computation, a set of possible landing times is associated to each aircraft. The problem of sequencing aircraft on the runway, while meeting one of the possible landing times, is then solved as a standard scheduling problem. Once the landing times are known, corresponding aircraft flight plans can be computed. Note that this process does not ensure that trajectories do not conflict with each other since inter-distance constraints can be violated. This drawback comes from the fact that the scheduling problem and the trajectory-computation problems are not solved simultaneously.

To our knowledge, the problem in which all constraints are simultaneously taken into account has not been studied. We focus on a *highly simplified* version of the problem: The “ $K$  king” problem. *Time is discretized and airspace is modeled as a two-dimensional chessboard with a special square that represents the runway. Aircraft move as kings do on a chessboard and the objective is to empty the chessboard as soon as possible.* An even simpler problem is to check if a move is possible or not.

More formally, starting from an initial layout described by the coordinates  $(a_1, b_1), \dots, (a_K, b_K)$  of  $K$  kings  $\mathcal{R}_1, \dots, \mathcal{R}_K$  on the chessboard, we look for a sequence of moves. The objective is to “empty” the  $N \times N$  chessboard while meeting the following constraints:

1. The distance between any pair of kings is greater than 1.
2. A king is removed off the board at the time point following its arrival on the runway square (marked with a cross on Figure 1).
3. At each time point, all kings move simultaneously to an adjacent square.

Arrows on Figure 1 show a possible simultaneous move of kings meeting all constraints. Note that for some initial configuration, it may happen that no move at all is possible (see Figure 2).

An instance of the problem and one of its optimal solutions is depicted in Figure 3. In

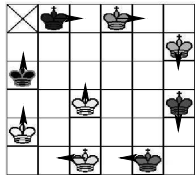


Figure 1: Non-Blocking Instance

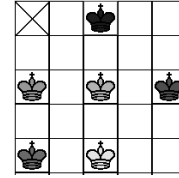


Figure 2: Blocking Instance

the following, we assume that lines and columns are numbered from 1 to  $N$  and to simplify notation, a “virtual” square with coordinates  $(0,0)$  is added to the board. By convention, all pieces already removed from the board are positioned on this square. This virtual square can be seen as the airport where aircraft can stay as long as they wish. Note that the location of the runway at the upper-left corner is arbitrary and the model could be changed to locate the runway in any square.

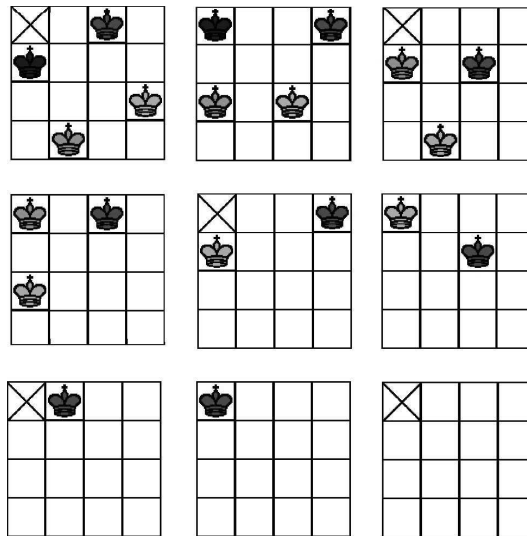


Figure 3: 4 Kings Reach the Target Square Within 9 Moves

## 2. Limitations of the $K$ king Model

The  $K$  king problem is a high-level abstraction of the initial problem. There are major features of the trajectories-computation problem that are not immediately captured by our model.

- In the  $K$  king model, everything is discretized and thus, a *trajectory* is a sequence of chessboard squares. This is a dramatic simplification of the standard physical interpretation of what a trajectory is. The initial “hybrid” problem is reduced to a discrete optimization problem.
- *2D vs. 3D.* Aircraft of comparable sizes usually fly at a similar “flight level” (a standard nominal altitude of an aircraft) and air traffic controllers impose strict “vertical separation limits” between flight levels (typically 1000 ft in the upper air space). So the vertical traffic is less intense than the horizontal traffic. For this reason we have chosen to study a simplified 2D chessboard rather than a 3D one.
- *Aircraft dynamics.* In the  $K$  king model we assume that each aircraft moves from one square to an adjacent square at each time point. This is again a dramatic simplification of the dynamics of aircrafts. A more realistic model would take into account the fact that the speed of an aircraft can vary within some range (hence from one square, a king could move to the squares whose “distance” to the current position is upper-and-lower bounded by some given limits). This could be integrated in the  $K$  king model (see the concluding remarks of Section 7 on how to represent aircraft dynamics within the constraint model).

We formally describe the  $K$  king problem and we provide an initial CSP (constraint satisfaction problem) model in Section 3. In Section 4, we describe additional propagation rules that drastically reduce the search space. Search strategies are defined in Section 5 and experimental results are reported in Section 6.

### 3. CSP Model

Three sets of variables are used to build a finite domain constraint model for the problem: *integer position variables* (Cartesian  $X, Y$  coordinates of kings), *Boolean position variables* and *exit-time variables*. A very basic model, relying on integer position variables only, can be designed but, as we will see later, the propagation is weak and leads to poor performance. For each feature/constraint of the problem we describe how it can be stated with the  $X, Y$  variables only, and how it can be stated with the Boolean position variables. Of course, when using Boolean position variables, extra propagation is performed and we expect better results.

The position  $\mathcal{R}_k(t)$  of the king  $\mathcal{R}_k$  at time  $t$  is given by a pair  $(X_{k,t}, Y_{k,t})$  of integer position variables where  $X_{k,t}$  (respectively  $Y_{k,t}$ ) stands for the column (resp. row) of the chessboard. As stated earlier, we assume that the coordinates of the upper left square of the chessboard is  $(1, 1)$ .

We introduce an additional set of exit-time variables  $T_1, \dots, T_k$ , that correspond to the time at which kings leave the board:

$$T_k = \min \left\{ \tilde{t} \mid \forall t \geq \tilde{t}, \begin{pmatrix} X_{k,t} \\ Y_{k,t} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right\}.$$

In the constraint model we introduce the following set of constraints to enforce this equation:

$$\forall k \in [1, K], \forall t \in [1, T] \quad \begin{pmatrix} X_{k,t} \\ Y_{k,t} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \iff T_k \leq t.$$

Moreover,  $\bar{T}$  denotes the variable corresponding to the maximum over these variables (i.e., the time at which the chessboard becomes empty). In an optimal solution, all kings do not have exactly the same position on the board twice (otherwise the solution would be suboptimal). Hence, the  $\bar{T}$  variable can be bounded by a constant value.

Such variables allow us to define a basic CP model for the problem. Still, we introduce additional Boolean position variables  $u_{k,i,j,t}$  for additional constraint-propagation that cannot take place directly on the initial variables.  $u_{k,i,j,t}$  is instantiated to “true” when  $\mathcal{R}_k$  is located at  $(i, j)$  at time  $t$ . Constraint-propagation rules that make use of these variables are presented in Section 4.

Integer position variables  $X_{k,t}$  and  $Y_{k,t}$  are linked to the Boolean position variables  $u_{k,i,j,t}$  by introducing two additional sets of variables  $x_{k,i,t}$  and  $y_{k,j,t}$ , which take the value *true* if and only if king  $\mathcal{R}_k$  is on line  $i$  (column  $j$ ) at time  $t$ . These variables enable us to establish a bi-directional link. At first, they are connected to the Boolean model as follows:

$$\begin{aligned} \forall k \in [1, K], \forall t \in [1, \text{sup}(\bar{T})], \forall i \in [0, N] \quad x_{k,i,t} &= \bigvee_{j \in [0, N]} u_{k,i,j,t} \\ \forall k \in [1, K], \forall t \in [1, \text{sup}(\bar{T})], \forall j \in [0, N] \quad y_{k,j,t} &= \bigvee_{i \in [0, N]} u_{k,i,j,t}. \end{aligned}$$

Second, we establish the connection between these variable sets and the main model:

$$\begin{aligned} \forall k \in [1, K], \quad \forall t \in [1, \text{sup}(\bar{T})], \quad \forall i \in [0, N] \quad i \notin \text{dom}(X_{k,t}) &\iff \neg x_{k,i,t} \\ \forall k \in [1, K], \quad \forall t \in [1, \text{sup}(\bar{T})], \quad \forall j \in [0, N] \quad j \notin \text{dom}(Y_{k,t}) &\iff \neg y_{k,j,t}. \end{aligned}$$

where  $\text{dom}(X)$  denotes the domain of the variable  $X$  and

$$\text{inf}(X) = \min_{x \in \text{dom}(X)} x \quad \text{sup}(X) = \max_{x \in \text{dom}(X)} x.$$

### 3.1. The Virtual Square (0, 0)

As stated previously, kings either stay in the  $N \times N$  board or are located on the “virtual” additional square with coordinates (0,0). So, domains of the position variables  $X_{k,t}, Y_{k,t}$  are  $[0, N]$ . In order to forbid the squares  $(1, 0), \dots, (N, 0), (0, 1), \dots, (0, N)$ , we introduce the following set of constraints over position variables:

$$\forall k \in [1, K], \forall t \in [1, \text{sup}(\bar{T})] \quad \left[ (X_{k,t} = 0) \wedge (Y_{k,t} = 0) \right] \vee \left[ (X_{k,t} \neq 0) \wedge (Y_{k,t} \neq 0) \right]$$

For the Boolean model this restriction is easy to formulate:

$$\begin{aligned} \forall k \in [1, K], \quad \forall t \in [1, \text{sup}(\bar{T})], \quad \forall i \in [1, N], \quad u_{k,i,0,t} &= \text{false} \\ \forall k \in [1, K], \quad \forall t \in [1, \text{sup}(\bar{T})], \quad \forall j \in [1, N], \quad u_{k,0,j,t} &= \text{false} \end{aligned}$$

### 3.2. Initial and Target Layouts

The initial coordinates of king  $\mathcal{R}_k$  is  $(a_k, b_k)$  and at time  $\bar{T}$ , all kings are located on (0,0).

$$\forall k \in [1, K], \quad \begin{pmatrix} X_{k,1} \\ Y_{k,1} \end{pmatrix} = \begin{pmatrix} a_k \\ b_k \end{pmatrix}, \quad \begin{pmatrix} X_{k,\text{sup}(\bar{T})} \\ Y_{k,\text{sup}(\bar{T})} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Such constraints allow the instantiation of the integer position variables.

### 3.3. Inter-Distance Constraints

Kings cannot stand in adjacent squares. Hence, relying on position variables, we have

$$\forall k, k' \in [1, K], \forall t \in [1, \text{sup}(\bar{T})], \quad k \neq k', \quad \left( |X_{k,t} - X_{k',t}| > 1 \right) \vee \left( |Y_{k,t} - Y_{k',t}| > 1 \right)$$

Note that there are no inter-distance constraints involving the virtual square.

Relying on Boolean variables, we have a set of disjunctive constraints

$$\forall k' \neq k, \neg u_{k,i,j,t} \vee \neg u_{k',i',j',t}$$

for all squares  $i, j$  and  $i', j'$  such that  $|i - i'| \leq 1$  and  $|j - j'| \leq 1$ . However, this leads to a large number of constraints that make the model slow. We refer to these as the *Boolean inter-distance constraints* and present a more compact and more generic global constraint in Section 4.

### 3.4. Constraints on Dynamics

As dynamics are extremely simple, they can be easily modeled as follows:

- The length of a move is at most 1, i.e.,  $\forall k \in [1, K], \forall t \in [1, \sup(\bar{T}) - 1], |X_{k,t+1} - X_{k,t}| \leq 1$  and  $|Y_{k,t+1} - Y_{k,t}| \leq 1$
- Kings actually move, i.e.,  $\forall k \in [1, K], \forall t \in [1, \sup(\bar{T}) - 1], (X_{k,t} \neq X_{k,t+1})$  or  $(Y_{k,t} \neq Y_{k,t+1})$

Dynamics of a more complex system can be captured by a transition graph  $\Gamma$ . Each square of the board corresponds to a vertex of  $\Gamma$  and there is an edge between two vertices  $(x, y)$  and  $(x', y')$  if and only if a king can move from  $(x, y)$  to  $(x', y')$  within the same single move.

Figure 4 shows a part of this transition graph involving the target square. Vertices correspond to grid points and transitions are drawn as dotted arrows.

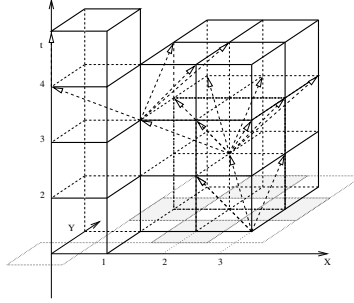


Figure 4: A Part of Transition Graph

With our dynamics, for any square  $(x, y)$  with  $1 \leq x \leq N, 1 \leq y \leq N$  and  $(x, y) \neq (1, 1)$ , we have

$$\Gamma\left(\begin{matrix} x \\ y \end{matrix}\right) = \left\{ \begin{pmatrix} x' \\ y' \end{pmatrix} \mid \begin{matrix} x' \in [\max(1, x-1), \min(x+1, N)] \\ y' \in [\max(1, y-1), \min(y+1, N)] \end{matrix} \right\} \setminus \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \right\}.$$

Note again that more complex dynamics can be modeled with  $\Gamma$ .

Once the king has reached the square  $(1, 1)$  it can leave the board (i.e., it moves to  $(0, 0)$ ) or it can continue its trajectory on the chessboard. Note that any optimal solution where a king moves from  $(1, 1)$  to a square that is not  $(0, 0)$  can be changed into a solution where the king immediately moves to  $(0, 0)$ . However, we study in Section 6.3 several extensions of the problem. Among them we consider the case where the time between consecutive exits

is strictly greater than 2. In this case, a king might have to wait and thus to continue its trajectory on the chessboard before reaching  $\binom{0}{0}$ :

$$\Gamma\left(\binom{1}{1}\right) = \left\{\binom{0}{0}, \binom{1}{2}, \binom{2}{1}, \binom{2}{2}\right\}.$$

A king, once it has left the board, does not move any longer, i.e.

$$\Gamma\left(\binom{0}{0}\right) = \left\{\binom{0}{0}\right\}.$$

In the following we most often rely on the generic graph formulation. In particular, the global constraints introduced later rely on the generic transition graph rather than on more specific dynamics. This allows us to describe the dynamics of a single king:

$$\forall k \in [1, K], \forall t \in [1, \text{sup}(\bar{T}) - 1] \quad \begin{pmatrix} X_{k,t+1} \\ Y_{k,t+1} \end{pmatrix} \in \Gamma\left(\begin{pmatrix} X_{k,t} \\ Y_{k,t} \end{pmatrix}\right)$$

Using the Boolean variables, we have a straightforward formulation:

$$\forall k \in [1, K], \forall i, j \in [1, N], \forall t \in [2, \text{sup}(\bar{T})] \quad u_{k,i,j,t} \Rightarrow \bigvee_{\binom{i'}{j'} \in \Gamma^{-1}\left(\binom{i}{j}\right)} u_{k,i',j',t-1}.$$

The symmetric equation also holds:

$$\forall k \in [1, K], \forall i, j \in [1, N], \forall t \in [1, \text{sup}(\bar{T}) - 1] \quad u_{k,i,j,t} \Rightarrow \bigvee_{\binom{i'}{j'} \in \Gamma\left(\binom{i}{j}\right)} u_{k,i',j',t+1}.$$

Moreover, we also have  $u_{k,i,j,t} \Rightarrow \neg u_{k,i,j,t+1}$ .

## 4. Reducing the Search Space with Global Constraints

### 4.1. Square-Unavailability

The first specific global constraint is related to the distance constraints between kings. During the construction of the solution we can make additional deductions based on partial information about kings' positions.

Consider, for instance, a king  $\mathcal{R}_k$  in a corner of the board at time  $t$ . Wherever it moves at  $t + 1$ , we are sure that none of the three immediate neighbors of the corner are occupied by another king at  $t + 1$ . We can of course use this information to propagate on the position variables of all kings  $\mathcal{R}_{k'}, k' \neq k$  at time  $t + 1$ .

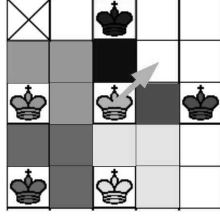


Figure 5: Mandatory Move Detection

This simple mechanism can be extended as follows: Consider, at some time  $t$ , the intersection  $I$  of the immediate neighbors of all admissible (i.e. not yet forbidden) squares for a king  $\mathcal{R}_k$ . All squares in  $I$  are immediate neighbors of  $\mathcal{R}_k$  at time  $t + 1$ , hence kings  $\mathcal{R}_{k'}$ ,  $k' \neq k$  cannot stand in the squares in  $I$  at time  $t + 1$ .

First, note that the intersection  $I$  is empty if  $(|dom(X_{k,t})| > 3)$  or if  $(|dom(Y_{k,t})| > 3)$ . Given this remark, it is easy to see that Algorithm 1 propagates the necessary deductions on the variables  $u_{k,i,j,t}$ .

When the king is known to be in to be in a rectangle smaller than a  $3 \times 3$  square, (i.e., when the domains of the variables  $X_{k,t}$  and  $Y_{k,t}$  are small enough), Algorithm 1 propagates the necessary deductions on all Boolean position variables and its overall complexity is  $O(K)$ .

---

**Algorithm 1** Propagation of Square-Unavailability for King  $\mathcal{R}_k$  at Time  $t$

---

```

if  $((|dom(X_{k,t})| \leq 3) \wedge (|dom(Y_{k,t})| \leq 3))$  then
  for  $i$  in  $sup(X_{k,t}) - 1 .. inf(X_{k,t}) + 1$  do
    for  $j$  in  $sup(Y_{k,t}) - 1 .. inf(Y_{k,t}) + 1$  do
      for all  $k' \neq k$  do
         $u_{k',i,j,t} := \text{false}$ 

```

---

Note that this global constraint subsumes the propagation of the Boolean inter-distance constraints as described in the previous section. Indeed, the propagation of the Boolean inter-distance constraints corresponds to the propagation of our global constraint when variables  $X_{k,t}$  and  $Y_{k,t}$  are instantiated.

## 4.2. Rectangle Capacity

The second set of specific global constraints is based on the following proposition.

**Proposition 4.1** *The maximal number of kings in a rectangle of size  $l \times h$  is at most  $\lceil \frac{l}{2} \rceil \lceil \frac{h}{2} \rceil$*

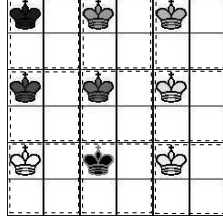


Figure 6: Maximum Rectangle Capacity

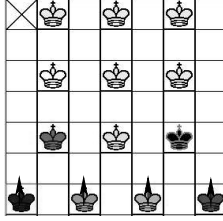


Figure 7: Blocking Instance

*Proof:* There is at most one king per  $2 \times 2$  square. So the number of kings in a rectangle of size  $l \times h$  is limited by the number of  $2 \times 2$  rectangles that can be contained inside.  $\square$

The above limit can be reached as shown in Figure 6.

This property allows us to deduce immediately the fact that there is no solution for the instance presented in Figure 7. Indeed, we first deduce that all kings in the bottom line have to move upward (Section 4.1) and then, based on the capacity of the first six lines, Proposition 4.1 allows us to deduce that there is no solution.

More formally, we introduce the following notation to describe the fact that rectangle  $[i, i + l - 1] \times [j, j + h - 1]$  contains the king  $\mathcal{R}_k$ :

$$\mathcal{R}_k(t) \in ([i, i + l - 1] \times [j, j + h - 1]) \Leftrightarrow (i \leq X_{k,t} < i + l) \wedge (j \leq Y_{k,t} < j + h)$$

So, a partial solution cannot be completed if the following condition is violated for any particular rectangle:

$$\forall t \in [1, \sup(\bar{T})], \forall i, j \in [1, N], \forall l > 0, \forall h > 0 \\ \#(\{k \mid \mathcal{R}_k(t) \in ([i, i + l - 1] \times [j, j + h - 1])\}) \leq \lfloor \frac{l}{2} \rfloor \lfloor \frac{h}{2} \rfloor$$

where  $\#(S)$  stands for the cardinality of the set  $S$ .

Moreover, if the number of kings inside a rectangle becomes equal to the capacity of this rectangle, then its inner squares are “forbidden” to other kings. More formally, if for some rectangle  $([i, i + l - 1] \times [j, j + h - 1])$  the cardinality of the set

$$S = \{\mathcal{R}_k \mid \mathcal{R}_k(t) \in ([i, i + l - 1] \times [j, j + h - 1])\}$$

of kings that are in this rectangle at  $t$ , is exactly  $\lceil \frac{l}{2} \rceil \lceil \frac{h}{2} \rceil$ , then all kings other than those in  $S$  have to be out of the rectangle at  $t$ , i.e.,  $\forall \mathcal{R}_k, \mathcal{R}_k(t) \notin S \Rightarrow \mathcal{R}_k(t) \notin ([i, i+l-1] \times [j, j+h-1])$ .

Algorithm 2 describes the propagation of the constraint for some rectangle  $(x_{\min}, x_{\max}) \times (y_{\min}, y_{\max})$  at time  $t$ . The complexity of one pass of this algorithm is  $O(K)$  if no propagation is achieved. If some deduction is made (i.e., when  $nbS$  is equal to the maximal number of kings that can be put in the rectangle) then the propagation takes  $O(KN^2)$  steps. Note that for one time point, we have  $O(N^4)$  rectangles to test. We consider in Section 6 two variants of the CSP model, adding all such constraints or only  $N^2$  constraints for only the rectangles with one corner fixed to  $(1, 1)$ .

---

**Algorithm 2** Propagation of the Rectangle-Capacity Constraint

---

**Require:**  $X_k, Y_k, x_{\min}, x_{\max}, y_{\min}, y_{\max}, t$   
 $nbS := 0$   
**for all**  $k$  **do**  
    **if**  $dom(X_{k,t}) \times dom(Y_{k,t}) \subseteq (x_{\min}, x_{\max}) \times (y_{\min}, y_{\max})$  **then**  
         $nbS := nbS + 1$   
**if**  $nbS > \left\lceil \frac{(x_{\max} - x_{\min})}{2} \right\rceil \left\lceil \frac{(y_{\max} - y_{\min})}{2} \right\rceil$  **then**  
    There is no possible solution  
**if**  $nbS = \left\lceil \frac{(x_{\max} - x_{\min})}{2} \right\rceil \left\lceil \frac{(y_{\max} - y_{\min})}{2} \right\rceil$  **then**  
    **for all**  $k$  **do**  
        **if**  $dom(X_{k,t}) \times dom(Y_{k,t}) \not\subseteq (x_{\min}, x_{\max}) \times (y_{\min}, y_{\max})$  **then**  
            **for all**  $i, j \in (x_{\min}, x_{\max}) \times (y_{\min}, y_{\max})$  **do**  
                 $u_{k,i,j,t} := false$

---

### 4.3. Trajectories

As stated in Section 3.4, dynamics are captured by

$$\forall k \in [1, K], \forall i, j \in [1, N], \forall t \in [1, sup(\bar{T}) - 1] \quad u_{k,i,j,t} \Rightarrow \bigvee_{\binom{i'}{j'} \in \Gamma\left(\binom{i}{j}\right)} u_{k,i',j',t+1}. \quad (1)$$

The symmetric equation also holds:

$$\forall k \in [1, K], \forall i, j \in [1, N], \forall t \in [2, sup(\bar{T})] \quad u_{k,i,j,t} \Rightarrow \bigvee_{\binom{i'}{j'} \in \Gamma^{-1}\left(\binom{i}{j}\right)} u_{k,i',j',t-1}. \quad (2)$$

We enforce generalized arc-consistency on this constraint network (GAC-scheme from (Bessière and Régin 1997)) to eliminate non-consistent values from the domains of the variables  $u_{k,i,j,t}$ . A constraint is arc-consistent if all the values remaining in the domains of

its variables can participate in a solution for this constraint. GAC filtering eliminates all inconsistent values and after each deletion the propagation is done for all constraints that can be concerned.

The following proposition shows that GAC ensures that the target square is reachable.

**Proposition 4.2** *GAC on constraints (1) and (2) ensures that if the value true belongs to the domain of some variable  $u_{k,i,j,t}$  then there is a trajectory for  $\mathcal{R}_k$  from its initial location to the target square that steps over  $(i, j)$  at time  $t$ .*

*Proof:* First, we show that the king can reach the target square if it can reach  $(i, j)$  at time  $t$ .

A value *true* is removed by GAC from the domain of  $u_{k,i,j,t}$  when all variables at the right side of (1) are instantiated to “false.” Thus, if GAC has not detected an inconsistency, then for each value “true” in the domain of a variable there is variable non-instantiated to “false” in the right part of (1).

So, to build a path from the position  $(i, j)$  at time  $t$  to the target we build a list of variables  $u_{k,i,j,t}, u_{k,i',j',t+1}, u_{k,i'',j'',t+2}$ , etc. that contain the value “true” in their domain. Since at time  $\text{sup}(\text{dom}(T))$ , all variables  $u$  are instantiated to false except  $u_{k,0,0,\text{sup}(T)}$ , the last variable in this list corresponds to the target square. Hence we have built a path from  $(i, j)$  at time  $t$  to the target square

To show that there is a trajectory from the initial position of the king  $\mathcal{R}_k$  to the position  $(i, j)$  at time  $t$  we can apply similar reasoning to the second set of inequalities (2).  $\square$

The above proposition shows that this simple mechanism is extremely efficient in propagating the existence of trajectories. Also note that this trajectory-propagation scheme relies on the transition graph  $\Gamma$ . It can thus be used for more complex dynamics.

#### 4.4. Scheduling

Because of inter-distance constraints, we know that there are at least two time points between two consecutive king exits:

$$\forall k, k' \in [1, K], \quad k \neq k', \quad (T_k \leq T_{k'} + 2) \vee (T_{k'} \leq T_k + 2). \quad (3)$$

This allows us to strengthen the constraint model using a redundant single-machine scheduling constraint: We associate to each king  $\mathcal{R}_k$  an activity with processing time 2 whose

starting time is  $T_k$  and we add a constraint stating that all these activities must be scheduled on a single machine (i.e., they cannot overlap in time). Many constraint-propagation techniques have been proposed for single machine scheduling (see (Baptiste et al. 2001) for a review). We rely on edge-finding (Carlier and Pinson 1990) and not-first/not-last rules (Baptiste and Le Pape 1996, Carlier and Pinson 1990, Torres and Lopez 2000, Vilím 2004).

As all processing times are equal, we have also tested another (new) propagation scheme based on the well known “all different” constraint. This global constraint ensures that all variables in a set take pairwise distinct values. Two variants have been proposed:

- Bounds-consistency algorithm described in (Puget 1998). The propagation ensures that both the upper and lower bound of domains’ variables participate in a solution.
- Arc-consistency algorithm from (Régin 1995) which eliminates from domains of variables all values that cannot participate in a solution.

For our scheduling problem, we divide and round (3) as follows. Two additional sets of variables  $\tau_k^- = \lfloor T_k/2 \rfloor$  and  $\tau_k^+ = \lceil T_k/2 \rceil$  are introduced. These variables must satisfy the following constraints that can be handled by an all-different constraint:

$$\forall i \in [1, K], \forall j \in [1, K], \quad i \neq j \quad \tau_i^- \neq \tau_j^- \quad \tau_i^+ \neq \tau_j^+.$$

This new formulation is very efficient and compared to edge-finding, it can possibly propagate more. However, initial experiments have shown that this almost never happens while the use of additional variables is costly. Recently, (Artiouchine and Baptiste 2005, Quimper et al. 2006) studied the “inter-distance constraint” that ensures that the distance between any pair of variables is at least equal to a given value. They describe a polynomial time algorithms to achieve arc-B-consistency (arc-consistency restricted to bounds of domains). This constraint-propagation mechanism has also been tested and initial experiments have shown that, for the  $K$  king problem, this complex and costly algorithm does not prune much more than does edge-finding. Hence, we only use edge-finding together with not-first/not-last.

## 4.5. SAC Propagation

To strengthen propagation we enforce singleton arc-consistency (Bessière and Debruyne 2004, Bartàk 2004) on the variables  $X_{k,2}$  and  $Y_{k,2}$  as outlined in Algorithm 3. This algorithm ensures that after assigning the value to the variable it is still possible to make the problem

consistent. This strengthened constraint-propagation is strongly related to “shaving” (Carlier and Pinson 1994, Martin and Shmoys 1996).

Given a constraint network  $P$  this algorithm ensures the network  $P|D_i = a$  with the domain  $D_i$  of variable  $V_i$  reduced to the singleton  $\{a\}$  can be made arc-consistent. If not, then this assignment will not participate in a solution to the problem.

---

**Algorithm 3** Singleton-Consistency Propagation Algorithm

---

```

Propagate AC on all constraints
repeat
  change  $\leftarrow$  false
  for all  $i \in V$  do
    for all  $a \in D_i$  do
      if  $P|D_i = a$  is not consistent then
         $D_i \leftarrow D_i \setminus \{a\}$ 
        Propagate AC on all constraints
        change = true
until change = false

```

---

## 5. Search Strategy

Following preliminary experiments, we decided to implement the following search strategy. First decide the time at which kings reach the exit. Second, build a feasible sequence of transitions.

**Step 1** If not all exit-times are known, then among variables  $T_k$  that are not bound, chose one such that  $\text{inf}(T_k)$  is minimal. Try then to bound  $T_k$  to the minimal value in its domain. Upon backtracking, remove the corresponding value from the domain of  $T_k$ .

**Step 2** If all exit-times are known, then compute the first time point  $t$  at which the position of at least one king is not known (if no such  $t$  exists then the problem is solved). Among unbound position variables  $X_{k,t}$  or  $Y_{k,t}$ , we then chose one with minimal  $T_k$  (thanks to Step 1, all  $T_k$  values are bound). We then try all possible values in the domain of the variable in increasing order.

We have also implemented a variant of this search strategy in which we rely on Step 2 only, i.e. trajectories are built in lexicographical order. In this case not all  $T_k$  values are bound so among unbound position variable  $X_{k,t}$ ,  $Y_{k,t}$ , we chose one with minimal value of  $\text{inf}(T_k)$ .

## 6. Experimental Results

An instance is characterized by the board size  $N \times N$ , the number of kings  $K$ , and the initial coordinates  $(a_1, b_1), \dots, (a_K, b_K)$  of kings. We have chosen to study extensively “small” instances of the problem (up to 14 kings on a  $7 \times 7$  board) rather than randomly picking some larger instances. As we will see in the following sections, these small  $7 \times 7$  instances are large enough to make the problem difficult to solve and they allow us to conclude on the efficiency of the various models and constraint-propagation algorithms.

Note that when the ratio between the number of aircraft and the size of the airspace is low, then the problem is easy to solve because most often the shortest paths from the initial positions of the aircraft to the airport are “valid” trajectories (i.e., the inter-distance constraints are not violated). To get a hard instance, we need to have a dense airspace.

Two sets of experiments have been performed:

- As for some instances (see Figure 2), no move is possible, we first test if one move can be achieved. We refer to these instances as “Blocking” instances.
- We then look for a sequence of transitions to empty the chessboard as soon as possible.

In both case, we try to evaluate the efficiency of our constraint-propagation schemes. All experiments were performed on an Intel Pentium-M 1.5 GHz (Centrino) platform.

### 6.1. Can Kings Move?

The question here is to determine if, given an initial layout, there is a feasible move or not. Several combinations of constraint-propagation algorithms have been tested. In the following experiments, “A+B” identifies the combination of constraint-propagation algorithms A and B. constraint-propagation and/or models are identified by one of the following acronyms:

- $\boxed{Basic}$  corresponds to the most basic model in which we use the Integer Position variables  $(X_{kt}, Y_{kt})$  (i.e., we do not use the Boolean variables  $u$ ) and no global constraint.
- $\boxed{Bool}$  stands for the Boolean inter-distance constraints, as described in Section 3.3 ( $\forall k' \neq k, \neg u_{k,i,j,t} \vee \neg u_{k',i',j',t}$  for all squares  $i, j$  and  $i', j'$  such that  $|i - i'| \leq 1$  and  $|j - j'| \leq 1$ ).
- $\boxed{Indisp}$  stands for the square-unavailability constraint propagation (Algorithm 1).

- $\boxed{N^4check}$ ,  $\boxed{N^2prop}$ , and  $\boxed{N^4prop}$  refer to the rectangle-capacity constraints described in Section 4.2. With  $N^4check$ , only the satisfiability of the constraints is tested on all  $O(N^4)$  rectangles. Propagation is achieved either on  $O(N^2)$  rectangles (one of the corners being always  $(1, 1)$ ) or on all rectangles.
- $\boxed{SAC}$  indicates whether singleton arc-consistency was applied or not (Algorithm 3).

We report in Figure 8 the ratio between blocking instances identified by constraint-propagation alone and the total number of blocking instances. 14 constraint-propagation combinations have been tested and for each of them we report the results on four sets of instances:  $6 * 5 * 5$  (6 kings on a  $5 * 5$  board, dark gray),  $7 * 5 * 5$  (7 kings on a  $5 * 5$  board, black),  $13 * 7 * 7$  (13 kings on a  $7 * 7$  board, white),  $14 * 7 * 7$  (14 kings on a  $7 * 7$  board, light gray). To compute the ratios, we first computed the set of all blocking instances (by enumeration) and, for each of these instances, the 14 constraint propagation algorithms were run.

Surprisingly, many of the blocking instances are identified immediately by constraint-propagation. Note that SAC plays a crucial role for detection of move absence. Almost 100% of instances are detected by constraint-propagation even when SAC is used alone. The very few remaining  $13 * 7 * 7$  instances that are not detected by SAC propagation are detected as soon as extra propagation relying on rectangle-capacity or on square-unavailability is performed. *According to these experiments, it seems that, beyond SAC, the most crucial ingredient of the propagation scheme are the square-unavailability propagation rules.*

An interesting question is whether constraint-propagation can be sufficient to detect that there is no possible move. Unfortunately, we have not been able to answer this question and we do not know if the corresponding problem is NP-hard or not.

## 6.2. How Should We Empty the Chessboard?

Several combinations of propagation rules have been tested to compute all minimal trajectories, i.e., all trajectories such that the date at which the board is empty is minimal.

- $\boxed{Dyn}$  stands for the propagation of dynamics as described in Section 4 (arc-consistency on the Boolean constraint network).
- $\boxed{Sched}$  indicates that the search strategy follows the two steps of Section 5 (first decide the time at which kings reach the exit and second build feasible trajectories).  $\boxed{Traj}$  is

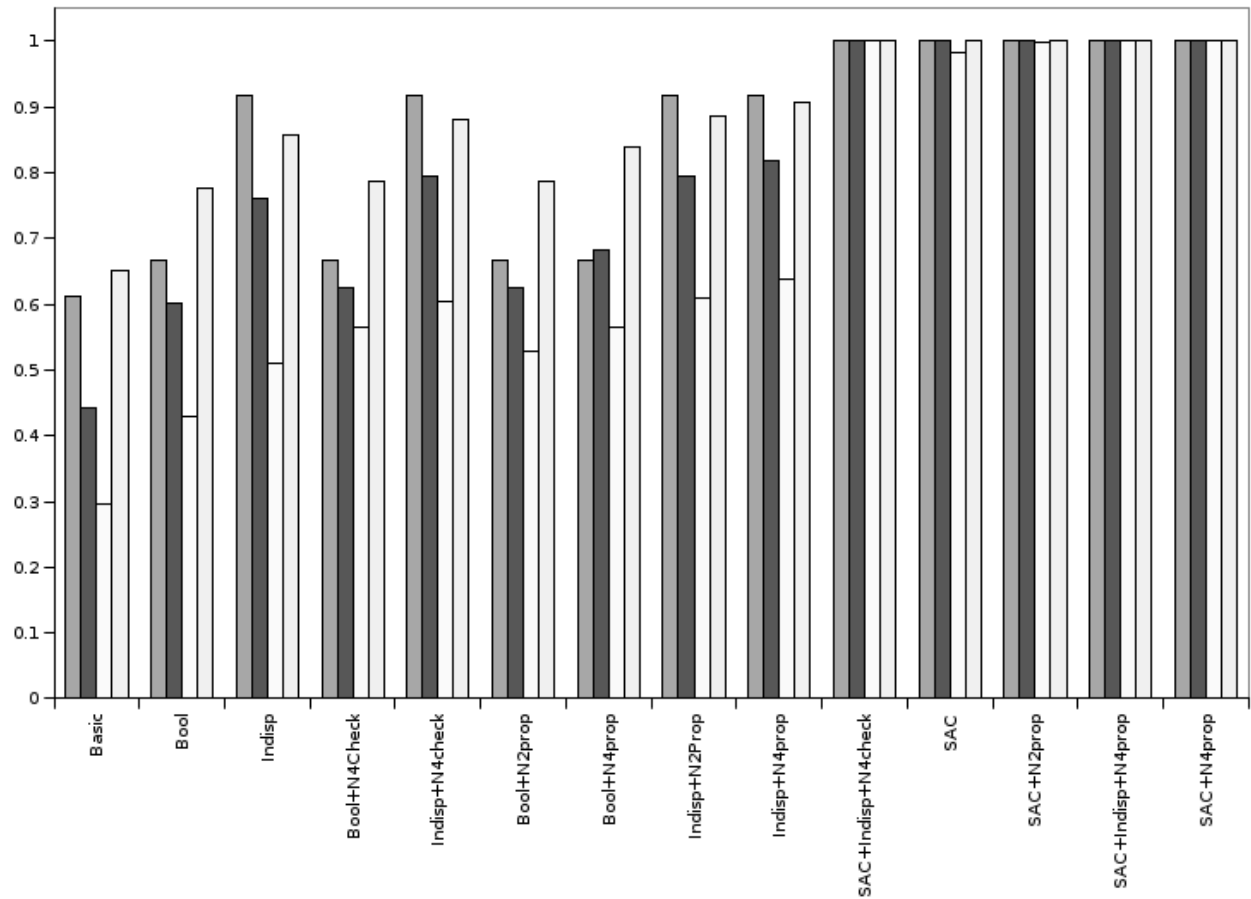


Figure 8: Ratio Between Blocking Instances Identified by Constraint-Propagation and the Total Number of Blocking Instances for  $6 * 5 * 5$  (Dark Gray),  $7 * 5 * 5$  (Black),  $13 * 7 * 7$  (White),  $14 * 7 * 7$  (Light Gray).

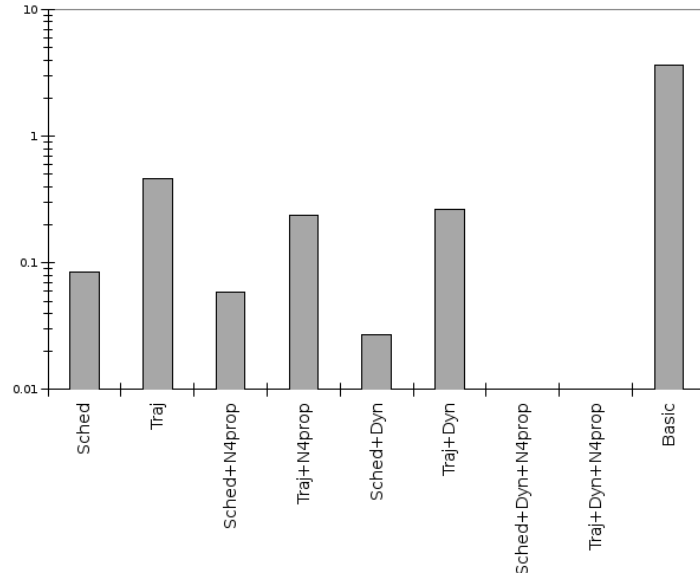


Figure 9: Minimizing the Date at Which the Board is Empty, Backtracks,  $6 * 5 \times 5$  Instances

used when we try to build the trajectories immediately (Step 2 only).

As before, the model in which we do not use the Boolean variables  $u$  (i.e., the model based only on the variables  $T_k$ ,  $X_{k,t}$ , and  $Y_{k,t}$ ) is referred to as the “`Basic`” model.” For this model, our search strategy is to build step-by-step the trajectories (step 2 of Section 5). As propagation is very weak for this model, the heuristic that consists in computing the variables  $T_k$  before building trajectories (step 1 followed by step 2) leads to very poor results and is not studied in the following experiments.

Figures 9—14 report the results obtained on all instances with size  $6 * 5 \times 5$ ,  $9 * 6 \times 6$  and  $13 * 7 \times 7$  for which an optimal solution is found by all the variants of the problem. For each set and each combination of propagation rule, the average number of backtracks, and average time in milliseconds is reported on a *logarithmic* scale.

Surprisingly, when the scheduling problem is solved before computing the trajectories (Sched, Sched +  $N^4prop$ , Sched + Dyn, Sched + Dyn +  $N^4prop$ ) the first solution found is always optimal (i.e., the search never backtracks to the scheduling portion of Step 1 once it has reached Step 2) and the overall number of backtracks is kept low. We have designed some special cases where this does not happen but in practice, the propagation seems to be (nearly always) strong enough to ensure the consistency of the CSP as soon as the scheduling problem is solved. The two other ingredients (propagation of dynamics, rectangle-capacity

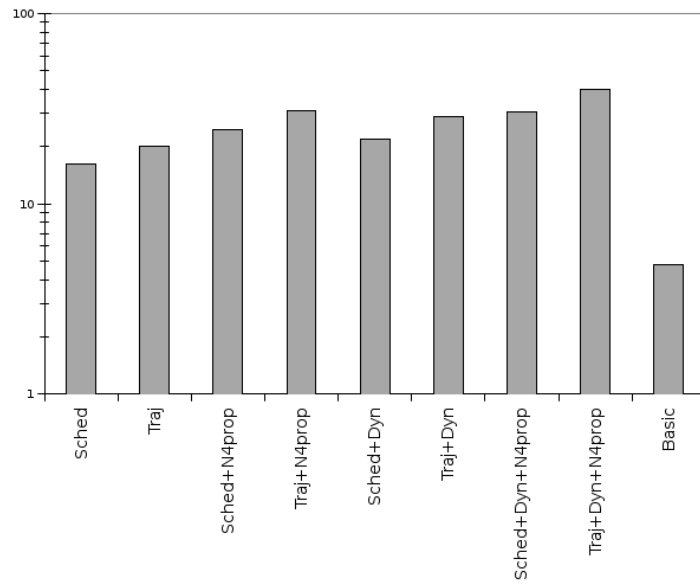


Figure 10: Minimizing the Date at Which the Board is Empty, CPU,  $6 * 5 \times 5$  Instances

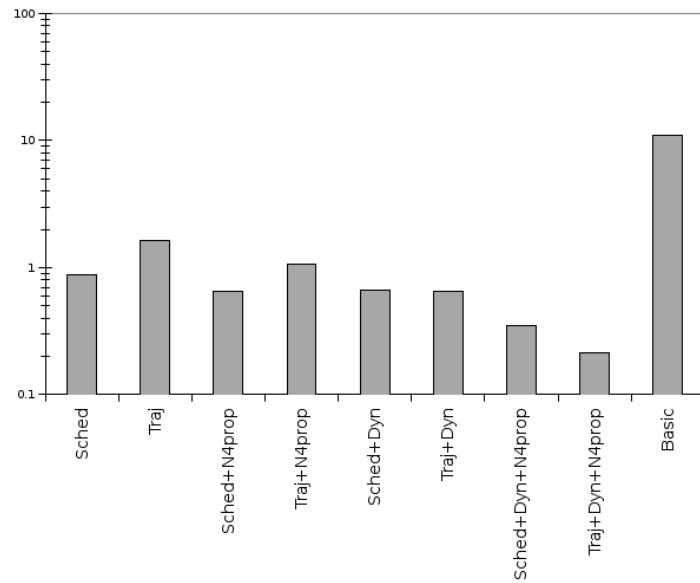


Figure 11: Minimizing the Date at Which the Board is Empty, Backtracks,  $9 * 6 \times 6$  Instances

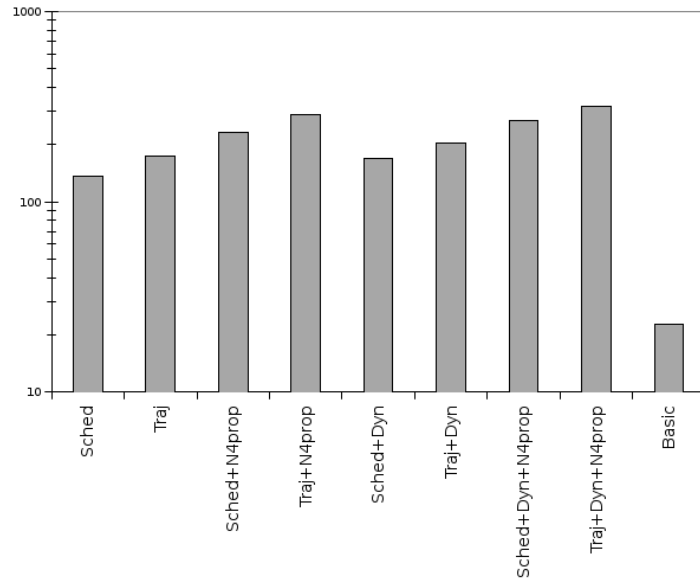


Figure 12: Minimizing the Date at Which the Board is Empty, CPU,  $9 * 6 \times 6$  Instances

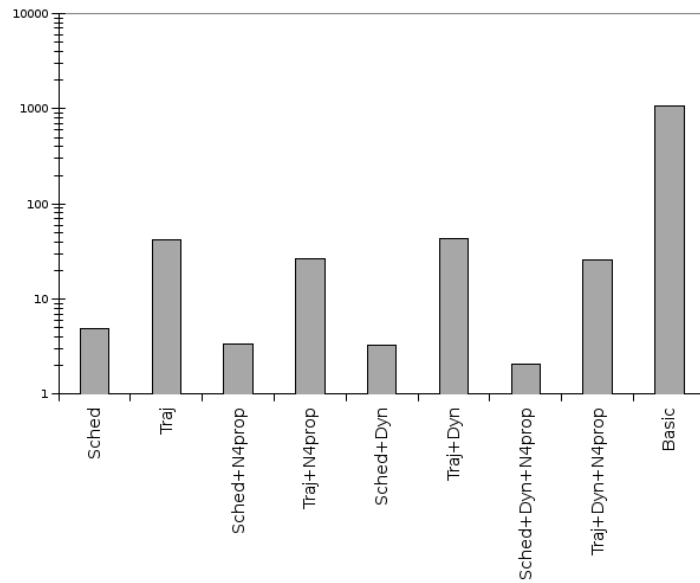


Figure 13: Minimizing the Date at Which the Board is Empty, Backtracks,  $13 * 7 \times 7$  Instances

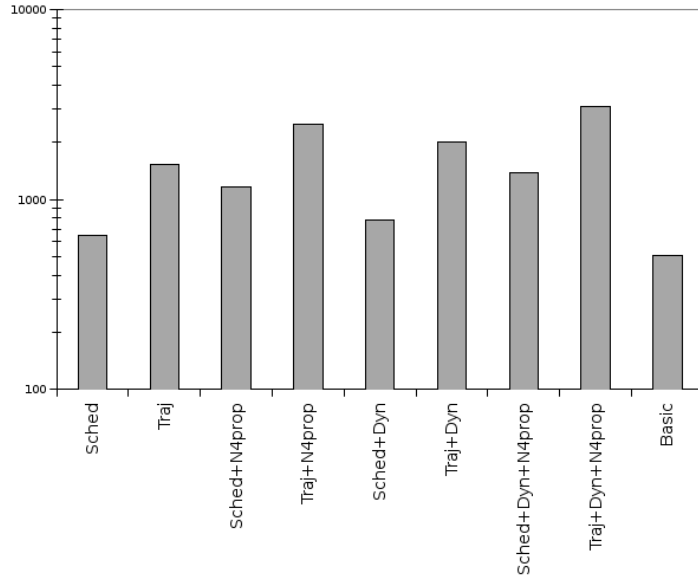


Figure 14: Minimizing the Date at Which the Board is Empty, CPU, 13 \* 7 x 7 Instances

constraint) are very useful to reduce the search space.

However, without the rectangle-constraints propagation, about 10 % of the largest instances (13 \* 7 x 7) were not solved by this strategy within five minutes.

The initial variant makes many more backtracks than do more complex variants but is faster on small instances. This difference is not as strong as for the largest instances that were accessible for our implementation of the Boolean model.

### 6.3. Modified Problem

We have tested two variants of the initial problem that can be easily implemented within our framework:

- In the first variant, a king cannot immediately return to the square it has left and the time between two consecutive exits is at least 3. These two additional constraints model respectively more complex aircraft dynamics and the fact that the landings have to be spaced for safety reasons.
- In the second variant, in addition to the previous constraints, a small “wall” forbids a region of the airspace. This models the fact that a sector of the airspace is closed (this typically happens for military operations). In the following, the wall is exactly  $\{(2, 1), (2, 2), (2, 3)\}$ .

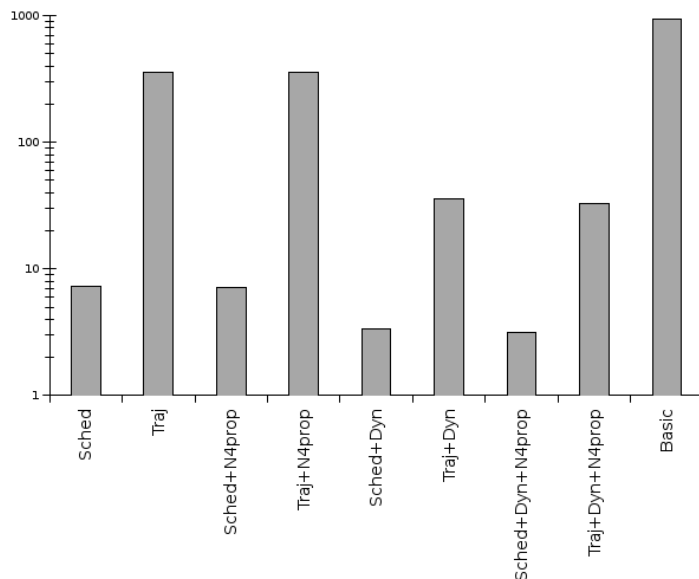


Figure 15: Modified Problem: At Least 3 Units of Time Between Consecutive Exits, Backtracks,  $6 * 5 * 5$  Instances

The same combinations of constraint-propagation algorithms as before have been used again in this experimental study.

Figures 15, 16, 17, 18 report the number of backtracks required to find the optimal solution (and prove its optimality) for all  $6 * 5 * 5$  and 50 randomly generated  $8 * 6 * 6$  instances of the first variant (at least 3 units of time between consecutive exits). Propagation of dynamics do improve the overall efficiency of the algorithm (L vs. P, N vs. R) while the effect of the propagation of rectangle constraints is rather small. Also note that the search strategy based on the resolution of the scheduling problem outperforms the strategy based on pure trajectory construction. Finally, the initial variant is not competitive.

Figures 19 and 20 report the same results for the small wall variant in a  $5 * 5$  board with 6 kings. The “Basic” variant was not able to solve about one third of the instances in one minute and the average values are computed over the instances which are solved. Compared to our previous experiments it appears that the search strategy based on scheduling does not improve the behavior of the search procedure. The combination “Traj + Dyn +  $N^4prop$ ” drastically prunes the search space.

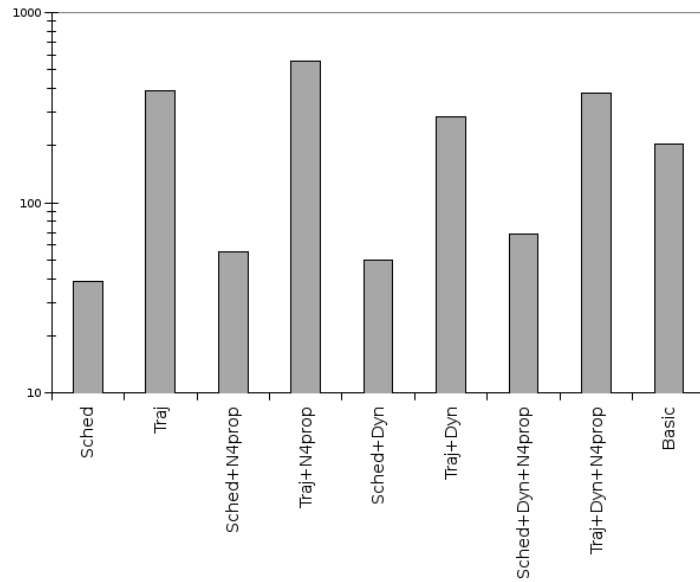


Figure 16: Modified Problem: At Least 3 Units of Time Between Consecutive Exits, CPU,  $6 * 5 * 5$  Instances

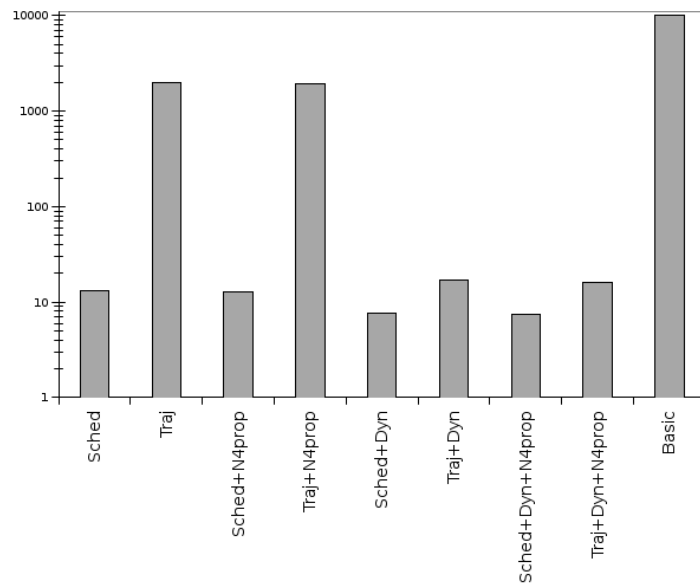


Figure 17: Modified Problem: At Least 3 Units of Time Between Consecutive Exits, Backtracks,  $8 * 6 * 6$  Instances

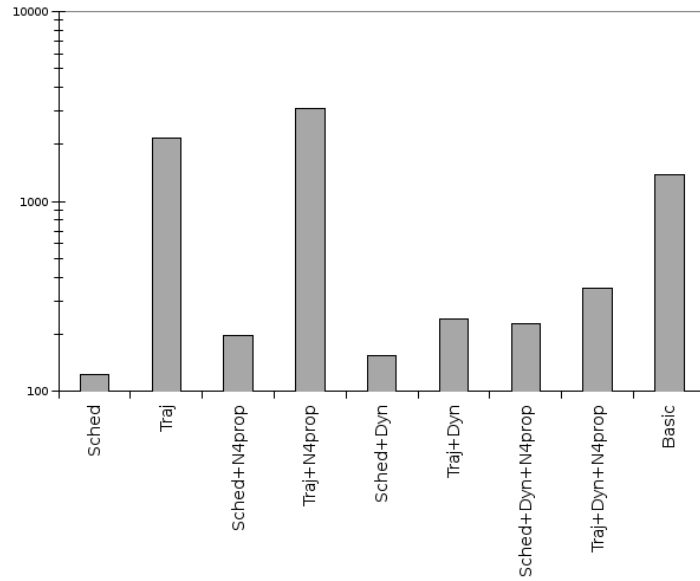


Figure 18: Modified Problem: At Least 3 Units of Time Between Consecutive Exits, CPU,  $8 * 6 * 6$  Instances

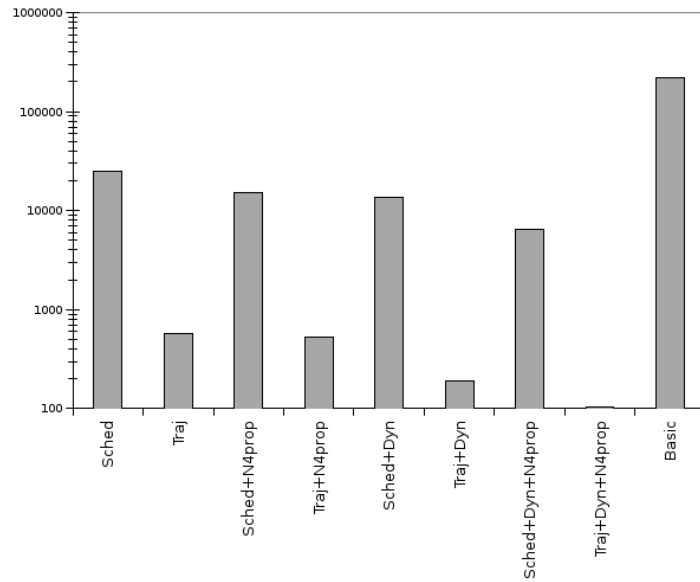


Figure 19: Modified Problem: Small Wall Variant, Backtracks,  $6 * 5 * 5$  Instances

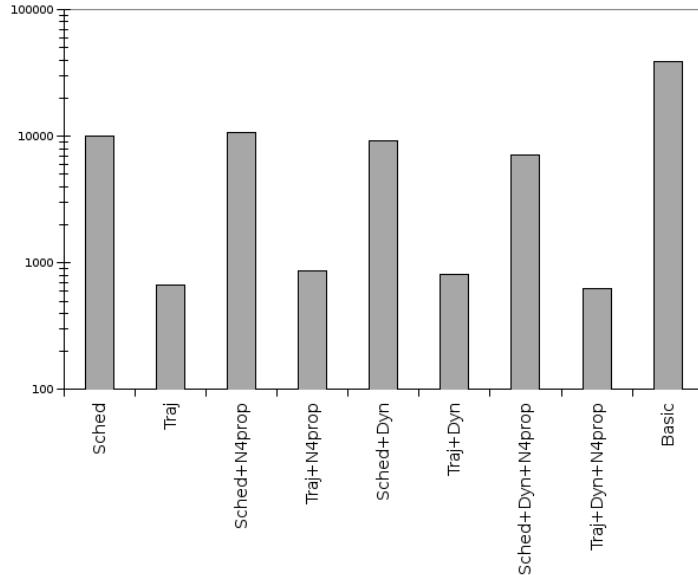


Figure 20: Modified Problem: Small Wall Variant, CPU,  $6 * 5 \times 5$  Instances

## 7. Conclusion

We have introduced a CSP model to solve a complex optimization problem in which aircraft trajectories and landing times are computed to minimize the maximal landing time of aircraft waiting in the Terminal Radar Approach CONTROL area. Several models and several constraint-propagation algorithms have been introduced and tested and we have highlighted the key ingredients of a successful CP approach for this problem. We have also shown that our model could be extended to deal with more complex situations.

There are several directions in which we could improve our search procedure. The search strategy could be improved with “no-good recoding” Schiex and Verfaillie (1994) to avoid solving the same subproblem several times. We could also try to improve the propagation process.

There are several limitations to our approach but we believe that the most important one is that we are not able to deal with large instances. Scalability is an important issue, but the *size of realistic instances is not as large as one could think at first*. There are two reasons for this:

- Let us first recall that the trajectory-computation problem is hard because of the inter-distance constraint (without the inter-distance constraint, we can compute, for each aircraft, its shortest path to the runway). Because of the inter-distance constraint, the

shortest paths are likely to be incompatible and thus we have a hard combinatorial problem to solve. Now, let us examine two typical situations: (1) When the airspace is very large compared to the number of aircraft, the inter-distance constraints do not play a central role in the problem as, “on the average”, aircraft are not too close together. So for such problem we believe that simple heuristics work well to solve the problem. (2) When there are many aircraft in the airspace, the inter-distance constraints play a central role. However, such dense traffic occurs at only very few places (a small part of the TRACON, very close to the runway) and a limited number of aircraft are involved. So these hard problems are small.

- Moreover, aircraft follow predetermined routes (with some kind of flexibility). This makes the problem much smaller as there are many positions in the chess board that are forbidden.

To summarize, we believe that large problems are easy and that hard problems are rather small. Still, we have to extend the size of the problems we can solve, but  $20 \times 20$  grids would already be large enough to be of interest for more practical applications. This is far from what we can achieve now, but not as far as what we could think at first.

Finally, we have not been able to prove that the problem of deciding whether a move is possible is NP-Complete.

## Acknowledgment

The authors are grateful to Jean-Marc Steyaert for several enlightening discussions on hybrid systems.

## References

- Artiouchine, K., P. Baptiste. 2005. Inter-distance constraint: an extension of the all-different constraint for scheduling equal length jobs. Springer, ed., *Proc. of the Eleventh International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science*, vol. 3709. Barcelona, Spain, 62–76.
- Artiouchine, K., P. Baptiste, C. Dürr. 2004. Runway sequencing with holding patterns. *To appear in European Journal of Operational Research* .

- Baptiste, P., C. Le Pape. 1996. Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling. *Proceedings of the fifteenth workshop of the U.K. planning special interest group*.
- Baptiste, P., C. Le Pape, W. Nuijten. 2001. *Constraint-based Scheduling*. Kluwer Academic Publishers, Norwell, MA, USA.
- Bartàk, R. 2004. A new algorithm for singleton arc consistency. *Proceedings of The FLorida Artificial Intelligence Research Society*. Miami, Florida, USA.
- Beasley, J.E., M. Krishnamoorthy, Y.M. Sharaiha, D. Abramson. 2000. Scheduling aircraft landings - the static case. *Transportation Science* **34** 180–197.
- Bessière, C., R. Debruyne. 2004. Theoretical analysis of singleton arc consistency. *Proceedings of ECAI-04 workshop on Modeling and Solving Problems with Constraints*.
- Bessière, C., J.-C. Régin. 1997. Arc consistency for general constraint networks: preliminary results. *Proceedings of IJCAI, Nagoya, Japan*. 398–404.
- Carlier, J., E. Pinson. 1990. A practical use of Jackson’s preemptive schedule for solving the job-shop problem. *Annals of Operations Research* **26** 269–287.
- Carlier, J., E. Pinson. 1994. Adjustments of heads and tails for the job-shop problem. *European Journal of Operational Research* **78** 146–161.
- Martin, P. D., D. B. Shmoys. 1996. Approach to computing optimal schedules for the job-shop scheduling problem. *Proceedings of 5th Conference on Integer Programming and Combinatorial Optimization*.
- Puget, J.-F. 1998. A fast algorithm for the bound consistency of alldiff constraints. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. Winsconsin, USA, 359–366.
- Quimper, C.-G., A. López-Ortiz, G. Pesant. 2006. A quadratic propagator for the inter-distance constraint. *Proceedings of AAAI-06*.
- Régin, J.-C. 1995. Développement d’outils algorithmiques pour l’Intelligence Artificielle. application à la chimie organique. Ph.D. thesis, Université Montpellier 2, France.

- Schiex, T., G. Verfaillie. 1994. Nogood recording for static and dynamic constraint satisfaction problem. *International Journal of Artificial Intelligence Tools* **3** 187–207.
- Torres, P., P. Lopez. 2000. On not-first/not-last conditions in disjunctive scheduling. *European Journal of Operational Research* **127** 332–343.
- Vilím, P. 2004.  $o(n \log n)$  filtering algorithms for unary resource constraint. J.-C. Régin, M. Rueher, eds., *Proceedings of CP-AI-OR, LNCS*, vol. 3011. Springer, 335–347.