On Modeling A Dynamic Hybrid System with Constraints: Computing Aircraft Landing Trajectories

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

We study the problem of computing trajectories of a set of aircraft in their final descent, right before landing. Trajectories must be compatible with aircraft dynamics while keeping distance between aircraft large enough. Our objective is to determine the order in which aircraft land as well as their exact trajectories in order to minimize the maximal landing time. We study a highly simplified version of this hybrid problem where time and space are discretized. A constraint based model relying on several specific global constraints is introduced. Computational experiments are reported.

Key words: Air-Traffic Control, Scheduling, Constraint Programming

---

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, 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 exact trajectories in order to minimize the maximal landing time.

Similar problems have been studied in the literature. In the "static" version studien in Artiouchine, Baptiste, and Dürr (2004), Beasley et al. (2000) all possible trajectories are precomputed 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, described in Bayen and Tomlin (2003) does not ensure that trajectories do not conflict 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 of taking into account all constraints simultaneously has not been studied yet. In this paper 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 $(a_1, b_1), ..., (a_K, b_K)$ of K kings $\mathcal{R}_1, \ldots, \mathcal{R}_K$ on the chessboard, we look for a sequence of moves. The objective is to "empty" the $N * N$ chessboard while meeting the following constraints:

1. The distance between any pair of kings is at least 1.

2. A king is removed of 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). Deciding whether the move exists or not is the problem that we study in Section 5.

An instance of the problem and one of its optimal solution is depicted in Figure 3. In the following, we assume that lines and columns are numbered from 1 to N and to simplify all notations, a "fictive" 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 fictive square can be seen as

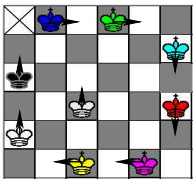

Figure 1: Non-Blocking Instance

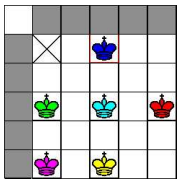

Figure 2: Blocking Instance

the airport where aircraft can stay as long as they wish. The squares that do not exist are coloured in gray on Figure 2.

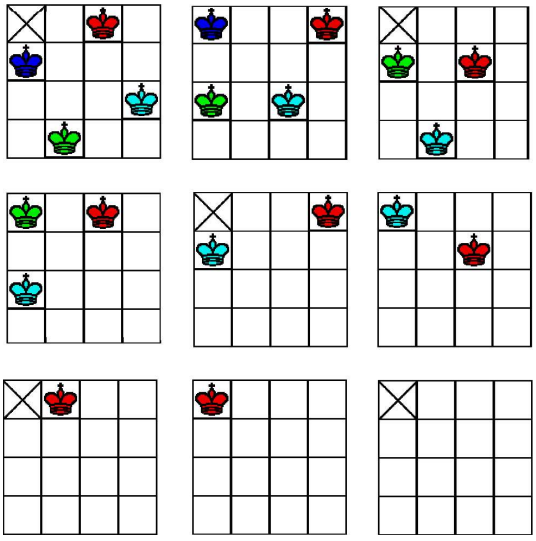

Figure 3: 4 kings reach the target square within 9 moves

The paper is organized as follows. We formally describe the $K-King$ problem and we provide an initial CSP model in Section 2. In Section 3, we describe additional propagation rules that drastically reduce the search space. Search strategies are defined in Section 4 and experimental results are reported in Section 5.

2. CSP Model

Three sets of variables are used to build a finite domain constraint model for the problem: Integer Position Variables, 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 extremely inefficient and leads to poor performance.

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* $T_1, ..., T_k$, that correspond to the time at which kings leave the board.

$$T_k = \min \left\{ \tilde{t} \;\middle|\; \forall t \geq \tilde{t},\ \binom{X_{k,t}}{Y_{k,t}} = \binom{0}{0} \right\}$$

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

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

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

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" if and only if $\mathcal{R}_k$ is located at (i, j) at time t.

Constraint propagation rules that make use of these variables are presented in Section 3.

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 can take the value *true* if king k can be 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:

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

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

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

Where $dom(X)$ denotes the domain of the variable X with $inf(X) = \min_{x \in dom(X)} x$ and $sup(X) = \max_{x \in dom(X)} x$.

2.1. The Fictive Square $(0,0)$

As stated previously, kings either stay in the $N \times N$ board or are located on the "fictive" 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 $(1,0),...,(N,0)$, $(0,1),...(0,N)$, we introduce the following set of constraints over position variables:

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

For the Boolean model this restriction is easy to formulate:

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

2.2. 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 $(0, 0)$.

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

Such constraints allow to instantiate immediately the Integer Position Variables and do not have a direct counterpart on the Boolean model.

2.3. Inter-Distance Constraints

Kings cannot stand in adjacent squares. Hence, relying on Position Variables, we have

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

Relying on boolean variables, we have a set of disjunctive constraints $\forall k' \neq k, \neg u_{k,i,j,t} \bigvee \neg u_{k',i',j',t}$ 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 3.

Note that there is no inter-distance constraints involving the fictive square. In sake of clarity, we omit the corresponding conditions in the following sections.

2.4. 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 Γ. Each square of the board corresponds to a vertex of Γ 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.

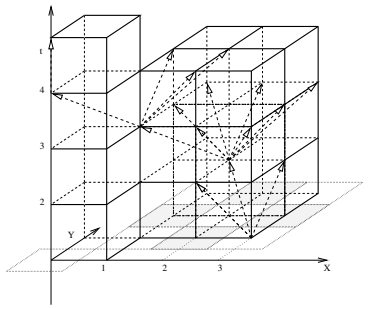

Figure 4: A part of transition graph

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

$$\Gamma\binom{x}{y} = \left\{ \binom{x'}{y'} \;\middle|\; \begin{array}{l} x' \in [\max(1, x-1), \min(x+1, N)] \\ y' \in [\max(1, y-1), \min(y+1, N)] \end{array} \right\} \setminus \left\{ \binom{x}{y} \right\}$$

Note again that more complex dynamics can be modeled with Γ.

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.

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

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 by the following relation:

$$\forall k \in [1, K], \forall t \in [1, sup(\bar{T}) - 1] \quad \binom{X_{k,t+1}}{Y_{k,t+1}} \in \Gamma\binom{X_{k,t}}{Y_{k,t}}$$

Using the Boolean Variables, we have a straightforward formulation:

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

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

3. Reducing Search Space with Global Constraints

3.1. 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.

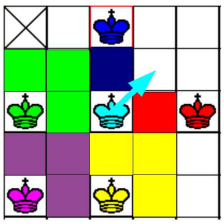

Figure 5: Mandatory Move Detection

Consider for instance a king $\mathcal{R}_k$ in a corner of the board at time t. Wherever it moves, 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 $t+1$.

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 ($|dom(X_k)| > 3$) or if ($|dom(Y_k)| > 3$). 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 smaller than 3×3 square, 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)| ≤ 3) ∧ (|dom(Y_k)| ≤ 3) then
  for i in sup(X_k) − 1 .. inf(X_k) + 1 do
    for j in sup(Y_k) − 1 .. inf(Y_k) + 1 do
      for all k' ≠ k do
        u_{k',i,j,t} := 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 these constraints corresponds to the propagation of our global constraint when variables $X_{k,t}$ and $Y_{k,t}$ are instantiated.

3.2. Rectangle Capacity

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

Proposition 3.1 *The maximal number of kings in a rectangle $l * h$ is at most $\left\lceil \frac{l}{2} \right\rceil \left\lceil \frac{h}{2} \right\rceil$*

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

The above limit can be reached as shown in Figure 7.

This property allows us to immediately deduce the fact that there is no solution for the instance presented in Figure 6. Indeed, we first deduce that all kings in the bottom line have to move upward (Section 3.1) and then,

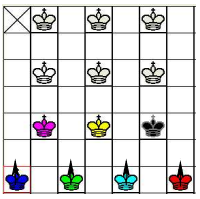

Figure 6: Blocking instance

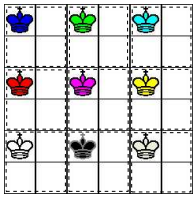

Figure 7: Max. rectangle capacity

based on the capacity of first 6 lines, Proposition 3.1 allows us to deduce that there is no solution.

More formally, we introduce the following notation in order 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:

$$\begin{array}{l} \forall t \in [1, sup(\bar{T})], \forall i, j \in [1, N], \forall l > 0, \forall h > 0 \\ \quad \#\left(\{k \mid \mathcal{R}_k(t) \in ([i, i+l-1] \times [j, j+h-1])\}\right) \leq \left\lceil \frac{l}{2} \right\rceil \left\lceil \frac{h}{2} \right\rceil \end{array}$$

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 $\left\lceil \frac{l}{2} \right\rceil \left\lceil \frac{h}{2} \right\rceil$ then, all kings besides 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 constraint 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 $O(K)$. Note that for one time point, we

have $O(N^4)$ rectangles to test. We consider in Section 5 two variants of the CSP model adding all such constraints or only N^2 constraints only for 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 $dom(X_k) \times dom(Y_k) \subset (x_{\min}, x_{\max}) \times (y_{\min}, y_{\max})$ **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**
      for all i **do**
        for all j **do**
          if $dom(X_k) \times dom(Y_k) \notin (x_{\min}, x_{\max}) \times (y_{\min}, y_{\max})$ **then**
            $u_{k,i,j,t} := false$

3.3. Trajectories

As stated in Section 2.4, dynamics are captured by

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

The symmetric equation also holds

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

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 3.2 *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 $sup(dom(T))$, all variables u are instantiated to false except $u_{k,0,0,sup(T)}$, the last variable in this list corresponds to the target square. Hence we have build 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). □

The above proposition shows that this simple mechanism is extremely efficient to propagate the existence of trajectories. Also note that this trajectory propagation scheme relies on the transition graph Γ an can thus be used for more complex dynamics.

3.4. Scheduling

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

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

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 Pape, 1996; Carlier and Pinson, 1990; Torres and Lopez, 1999; Vilím, 2004).

As all processing times are equal, we have also tested another 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 for each bound of each variable can participate in a solution for this constraint.

- Arc-consistency algorithm from Régin (1995) which eliminates from domains of variables all values which can not participate in a solution for this constraint.

For our scheduling problem, we divide and round Equation (3.4) 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.

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

3.5. 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 any solution of the problem.

4. 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.

Algorithm 3 Singleton Consistency Propagation Algorithm

```
Propagate AC on all constraints
repeat
  change ← false
  for all i ∈ V do
    for all a ∈ D_i do
      if P|D_i = a is not consistent then
        D_i ← D_i \ {a}
        Propagate AC on all constraints
        change = true
until change = false
```

Step 1 If not all exit times are known, then among variables T_k that are not bound, chose one such that $inf(T_k)$ 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 variable $X_{k,t}$ 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 build in lexicographical order. In this case all T_k values are not bound so among unbound position variable $X_{k,t}$, $Y_{k,t}$, we chose one with minimal value of $inf(T_k)$.

5. Experimental Results

An instance is characterized by the board size $N \times N$, the number of kings K and by the initial coordinates of kings (a_k, b_k). Several instance sets with up to 7×7 boards have been generated (several thousands of instances for a given board size and several hundreds of them are non-blocking).

Two sets of experiments have been led on these instances.

- As for some instances (see, Figure 2), no move is possible, we first test if one move can be achieved.
- 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 the constraint propagation schemes that have been proposed in the paper. All experiments were performed on Intel Pentium-M 1.5 GHz (Centrino) platform.

5.1. 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. Each line of Table 1 corresponds to a combination of algorithms tested in our study. Each row refers to a specific constraint propagation algorithm and a "+" means that the algorithm has been used in the the combination associated to the corresponding column.

- Row *Bool* stands for the Boolean inter-distance constraints, as described in Section 2.3.
- Row *Indisp* stands for the square unavailability constraint propagation (Algorithm 1).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Bool		+		+		+	+							
Indisp			+		+			+	+	+			+	
N^4check				+	+		+		+	+			+	+
N^2prop						+		+				+		
N^4prop							+		+				+	+
SAC										+	+	+	+	+

Table 1: Parameters

- Rows N^4check, N^2prop and N^4prop reefer to the rectangle capacity constraints described in Section 3.2. 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.

- Row SAC indicates whether Singleton Arc Consistency was applied or not (Algorithm 3).

Among instances for which no move is possible, many of them are identified immediately by constraint propagation. We report in Figure 8 the ratio between those instances and the total number of instances for which no move is possible. The 14 constraint propagation combinations have been tested and for each of them, we report the results on four kind of instances $6 * 5 \times 5$ (6 kings on a $5 * 5$ board, dark gray), $7 * 5 \times 5$ (7 kings on a $5 * 5$ board, black), $13 * 7 \times 7$ (13 kings on a $7 * 7$ board, white), $14 * 7 \times 7$ (14 kings on a $7 * 7$ board, light gray). Note that SAC plays a crucial role for detection of move absence. Almost 100% of all instances are detected by constraint propagation even when SAC is used alone. The very few remaining $13 * 7 \times 7$ 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

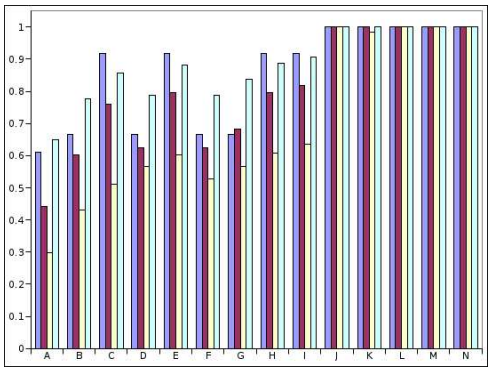

Figure 8: Results

SAC, the most crucial ingredient of the propagation scheme are the square unavailability propagation rules.

5.2. How To Empty the Chessboard ?

Several combination 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. All tested variants are listed in Table 2. Column "Dyn" stands for the propagation of dynamics as described in Section 3 and column *Sched* indicates whether the search strategy follows the two steps of Section 4 (first decide the time at which kings reach the exit, second build a feasible sequence of transitions) or attempts to build the trajectories (Step 2 only). Finally, the results of the initial variant of the problem based only on the variables T_k, $X_{k,t}$ and $Y_{k,t}$ with the search strategy based on the trajectory construction are reported in the column XY.

	Dyn	N^4prop	Sched
K			+
L			
M		+	+
N		+	
O	+		+
P	+		
Q	+	+	+
R	+	+	

Table 2: Parameters

Figures 9, 10, 11, 12, 13 and 14 report the results obtained on all instances of corresponding sizes for which a solution was 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 the computing the trajectories (K, M, O, Q), the first solution found is always optimal and the overall number of backtracks is kept low. The two other ingredients (dynamics constraint propagation, rectangle capacity constraint) are very useful to reduce the search space.

However, without the rectangle constraints propagation, there is about 10 percent of of the largest instances (13 kings on 7×7 board) which were not solved by this strategy within 5 minutes (variants K and O).

The initial variant makes much more backtracks than more complex ones but goes faster on the small instances. This difference is not as strong as for the largest instances which were accessible for our implementation of the boolean model.

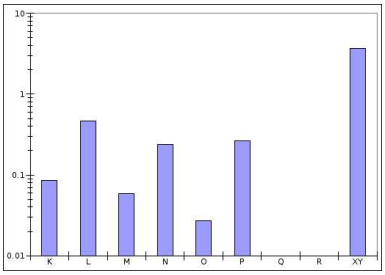

Figure 9: Backtracks on $6 * 5 \times 5$

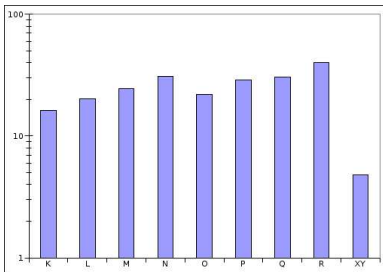

Figure 10: Time on $6 * 5 \times 5$ instances

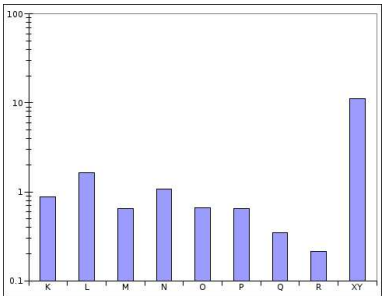

Figure 11: Backtracks on $9 * 6 \times 6$

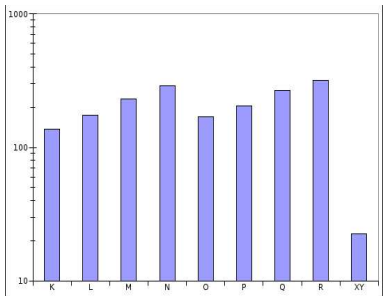

Figure 12: Time on $9 * 6 \times 6$ instances

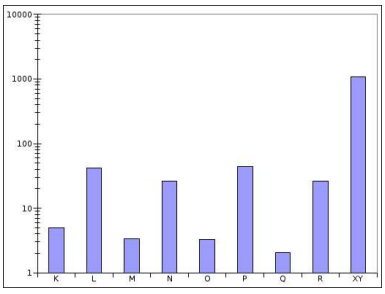

Figure 13: Backtracks on $13 * 7 \times 7$

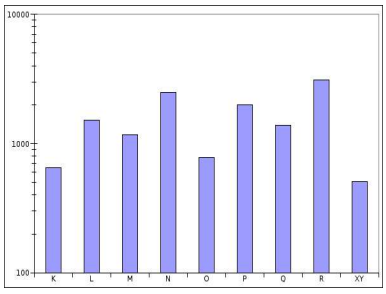

Figure 14: Time on $13 * 7 \times 7$ instances

5.3. 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 slightly more complex dynamics of aircraft together with the fact that the landings have to be spaced of some arbitrary value for safety reasons.

- In the second variant, in addition to the previous constraints, a small "wall", from (2, 1) to (2, 3) in our experiments, forbids a region of the airspace. This models the fact that a sector of the airspace is closed (this typically happens for military operations).

The combinations of Table 2 have been used gain 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 $6 * 5 \times 5$ and 50 randomly generated $8 * 6 \times 6$ instances of the first variant. Propagation of dynamics do improve the overall efficiency of the algorithm (L vs. P, N vs. R) while 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 20 and 19 report the same results for the small wall variant for 5×5 board with 6 kings. Compared to our previous experiments appears that the search strategy based on the scheduling problem does not improve the behavior of the search procedure but, however, the first solution found is optimal. Finally, the initial variant of the model (shown in XY column) was not able to solve about one third of the instances in one minute and the average was computed over the instances which were solved.

6. 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

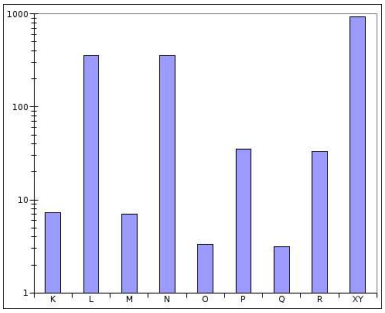

Figure 15: Backtracks on $6 * 5 \times 5$

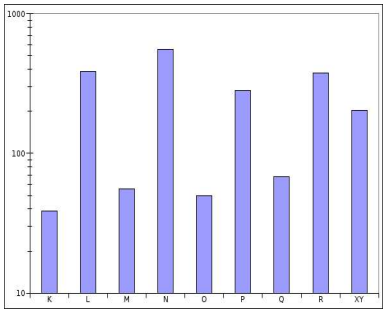

Figure 16: Time on $6 * 5 \times 5$ instances

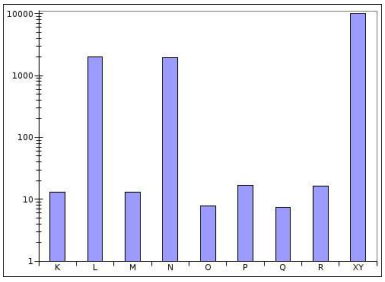

Figure 17: Backtracks on $8 * 6 \times 6$

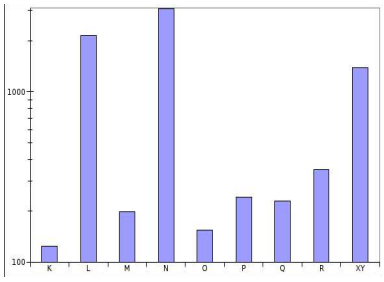

Figure 18: Time on $8 * 6 \times 6$ instances

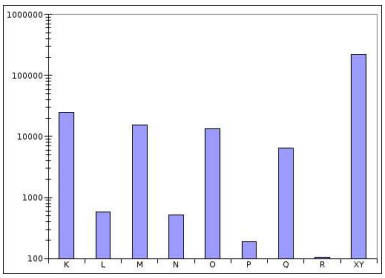

Figure 19: Backtracks on $6 * 5 \times 5$ with a wall

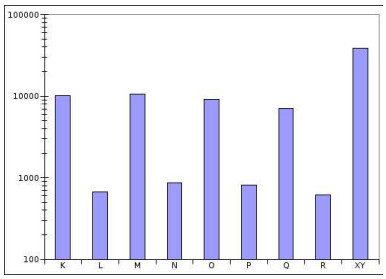

Figure 20: Time on $6 * 5 \times 5$ instances with a wall

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 limitations to our approach but we believe that the most important is that we are not able to deal with large problems and thus, we have to roughly discretize time and space. There are several directions in which we could improve our search procedure: The search strategy could be improved with "No-Good recoding" to avoid solving exactly the same subproblem several times. We could also try to improve the propagation process.

Finally, we would like to mention that we have not been able to prove that the problem of deciding whether a move is possible or not is NP-Complete. So it could be that there is polynomial time algorithm for this problem.

Acknowledgment

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

References

Artiouchine, K., P. Baptiste, C. Dürr. 2004. Runway sequencing with holding patterns. Tech. rep., Laboratoire d'Informatique de l'Ecole Polytechnique.

Baptiste, P., C. Le Pape, W. Nuijten. 2001. *Constraint-based Scheduling*. Kluwer.

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*.

Bartàk, R. 2004. A new algorithm for singleton arc consistency. *Proceedings of FLAIRS-04*.

Bayen, A.M., C.J. Tomlin. 2003. Real-time discrete control law synthesis for hybrid systems using milp: Application to congested airspace. *American control conference*.

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 AAAI-98*. 359–366.

Régin, J.-C. 1995. Développement d'outils algorithmiques pour l'Intelligence Artificielle. application à la chimie organique. Ph.D. thesis, Université Montpellier II.

Torres, P., P. Lopez. 1999. On not-first/not-last conditions in disjunctive scheduling. *European Journal of Operational Research* .

Vilím, P. 2004. $o(n \log n)$ filtering algorithms for unary resource constraint. Jean-Charles Régin, Michel Rueher, eds., *Proceedings of CP-AI-OR*, *LNCS*, vol. 3011. Springer, 335–347.