Les K-rois : une métaphore du problème d'ordonnancement spatio-temporel des avions autour d'un aéroport

Konstantin Artiouchine, Philippe Baptiste, Juliette Mattioli

Les $K-$rois : une métaphore du problème d'ordonnancement spatio-temporel des avions autour d'un aéroport

Konstantin Artiouchine [1,2] **Philippe Baptiste** [1] **Juliette Mattioli** [2]

[1] LIX, École Polytechnique, 91128 Palaiseau, France,
[2] Thales TRT, Domaine de Corbeville, 91404 Orsay CEDEX, France
{Prenom.Nom}@polytechnique.fr Juliette.Mattioli@thalesgroup.com

Résumé

Nous étudions le problème du calcul des trajectoires d'un ensemble d'avions pendant leur descente finale juste avant l'atterrissage. Ces trajectoires sont soumises d'une part aux contraintes liées à la dynamique des avions et d'autre part au fait qu'une distance de sécurité suffisamment grande doit être maintenue. Nous cherchons à déterminer l'ordre dans lequel les avions se posent ainsi que leurs trajectoires respectives avec comme objectif la minimisation du temps d'atterrissage maximal. Nous présentons dans ce papier une version simplifiée du problème hybride où le temps et l'espace sont discrétisés. Un modèle en PPC introduisant quelques contraintes globales est presenté.

Abstract

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.

Mots clefs : Contrôle du trafic aerien, Ordonnancement, Programmation par Contraintes

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

1 Introduction

Les modèles pour la dynamique d'un système reposent souvent sur des équations différentielles dérivées des lois physiques. Ces systèmes sont étudiés dans le domaine de la théorie du contrôle. Les systèmes hybrides sont caractérisés par une forte interaction des modèles continus et discrèts (e.g., une utilisation simultanée des variables continues et discrètes).

Nous étudions ici un problème hybride de calcul des trajectoires des avions lors de leur descente finale. Ces trajectoires doivent être compatibles avec la dynamique des avions et satisfaire les contraintes induites par les distances de sécurité entre avions. Tant que les avions sont en l'air, la distance entre chaque paire d'avions doit être supérieure à une valeur fixée. En outre, un intervalle de sécurité entre deux atterrissages consécutifs sur la même piste doit être maintenue. Notre objectif est de déterminer l'ordre d'atterrissage et les trajectoires de tous les avions qui minimisent la date d'atterrissage du dernier avion.

Des problèmes similaires ont déjà été étudiés. La version "statique" est étudiée dans [1], [6] où les auteurs supposent que toutes les trajectoires possibles sont connues. Cela permet d'associer à chaque avion un ensemble de fenêtres temporelles dans lesquelles son atterrissage est autorisé. Le problème de séquencement des vols ainsi obtenu est par la suite abordé comme un problème d'ordonnancement classique. Une fois toutes les dates d'atterrissage connues, les plans de vol correspondant aux trajectoires peuvent être calculés. L'inconvénient majeur de cette approche, détaillée dans [5], est que les trajectoires obtenues peuvent introduire des conflits potentiels puisque les contraintes de

distance peuvent être violées. Ceci est lié au fait que le calcul de l'ordonnancement et le calcul des trajectoires sont effectués indépendamment l'un de l'autre. En pratique, cette approche est réaliste si les avions volent à des altitudes différentes.

A notre connaissance, il n'existe pas d'approches permettant de prendre en compte simultanément toutes les contraintes du problème. Dans ce papier, on étudie une métaphore très simplifiée du problème original : "le problème des $K-$rois". Le temps est discrétisé et l'espace aérien est représenté par un échiquier avec une case fictive représentant la piste d'atterrissage. Les avions se déplacent comme les rois de jeu d'échecs et l'objectif est de vider l'échiquier le plus rapidement possible. Un problème encore plus simple est de vérifier qu'à partir d'une disposition donnée il existe au moins un mouvement.

Plus formellement, en partant d'une disposition initiale donnée par les coordonnées $(a_1, b_1), ..., (a_K, b_K)$ de K rois $\mathcal{R}_1, \ldots, \mathcal{R}_K$ sur un échiquier de taille $N \times N$, on cherche une séquence de transitions permettant de "vider" l'échiquier en respectant les contraintes suivantes :

1. Les rois ne peuvent pas être en prise (selon les régles classiques du jeu d'échecs).
2. Un roi peut être enlevé de l'échiquier tout de suite après son passage sur la case cible (marquée par une croix sur les Figures 1 et 2).
3. A chaque itération, chaque roi se déplace obligatoirement de sa position courante à une case adjacente tant qu'il n'est pas sorti du jeu.

Un exemple d'une instance du problème est représenté sur la Figure 1. Les flèches sur la figure indiquent les éventuels déplacements des rois d'un instant à l'autre. Il est important de savoir qu'il existe des situations à partir desquelles il n'existe pas de mouvement possible. Un exemple d'une telle disposition est donné sur la Figure 2. Une autre instance du problème avec 4 rois sur une grille 4×4 et sa solution optimale sont représentées sur la figure 3.

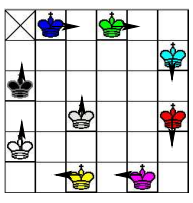

FIG. 1 – Un mouvement

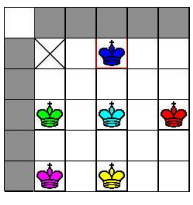

FIG. 2 – Un blocage

Dans la suite du papier on suppose que les lignes et les colonnes de l'échiquier sont numérotées de 1 à N. Afin de simplifier les notations, on introduit une case "fictive" avec les coordonnées $(0, 0)$. Par convention, tous les rois déjà sortis de l'échiquier se trouvent sur cette case. On peut voir cette case comme l'aéroport où les avions peuvent rester autant de temps qu'ils veulent. Les cases non-existantes sont dessinées en gris sur la Figure 2.

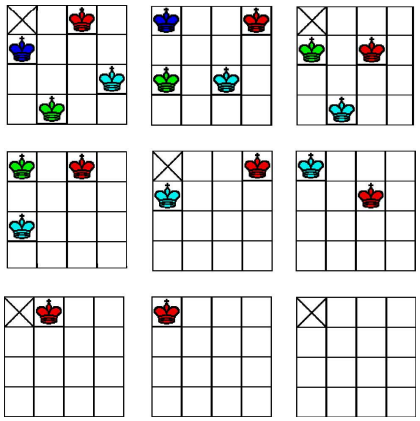

FIG. 3 – Une solution optimale en 9 transitions pour 4 rois

Cet article est composé de 5 sections. La description formelle du problème des $K-$rois et un modèle initial en PPC sont décrits dans la Section 2. Nous introduisons dans la Section 3 les régles de propagation supplémentaires permettant de réduire l'espace de recherche. Les heuristiques de branchement sont définies dans la Section 4 et les résultats expérimentaux sont donnés dans la Section 5.

2 Modèle PPC

Nous utilisons trois ensembles de variables à domaines finis pour la construction d'un modèle en Programmation par Contraintes : variables de position entières et booléennes, et les variables de dates de sortie. Le modèle extrêmement simple utilisant seulement les variables entières de position peut être construit mais comme on le verra par la suite la propagation est très inefficace et mène à de faibles performances.

La position $\mathcal{R}_k(t)$ du roi $\mathcal{R}_k$ à l'instant t est donnée par une paire $(X_{\mathrm{k,t}}, Y_{\mathrm{k,t}})$ de *Variables Entières de Position* où $X_{\mathrm{k,t}}$ (resp. $Y_{\mathrm{k,t}}$) désigne la colonne (resp. la ligne) de l'échiquier. On rappelle que le coin en haut et à gauche de l'échiquier a pour coordonnées $(1, 1)$.

Nous introduisons un ensemble supplémentaire de variables appellées *Dates de Sortie* $T_1, ..., T_k$ associées aux instants auxquels les rois sont enlevés de l'échiquier :

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

Nous introduisons dans le modèle l'ensemble de contraintes suivant afin d'imposer cette relation :

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

En plus, on introduit la variable $\bar{T}$ égale au maximum de ces variables (c.à.d le premier instant où l'échiquier est vide).

Dans une solution optimale on ne peut pas avoir exactement la même disposition des rois plus d'une fois (sinon cette solution ne serait pas optimale). Le nombre total de pas de temps peut donc être borné par une constante.

Ces variables nous permettent de définir le modèle PPC "de base" pour notre problème. Nous introduisons néanmoins les *variables booléennes de position* $u_{\mathrm{k,i,j,t}}$ pour permettre les propagations irrealisables avec les variables de position entières. La variable $u_{\mathrm{k,i,j,t}}$ est instanciée à "vrai" si et seulement si $\mathcal{R}_k$ se trouve sur la case (i,j) à l'instant t. Les règles de propagation utilisant ces variables sont présentées dans la Section 3.

Les variables de position entières $X_{\mathrm{k,t}}$ et $Y_{\mathrm{k,t}}$ sont reliées aux variables de position booléennes en deux étapes. La première étape consiste en l'introduction de deux ensembles de variables supplémentaires $x_{\mathrm{k,i,t}}$ et $y_{\mathrm{k,j,t}}$ qui peuvent avoir la valeur "vrai" si le roi $\mathcal{R}_k$ peut se trouver sur la colonne i (resp. ligne j) à l'instant t. Ces variables sont ensuite reliées au modèle booléen comme suit :

$$\forall k \in [1,K], \forall t \in \left[1, sup\left(\bar{T}\right)\right], \forall i \in [0,N] \quad x_{\mathrm{k,i,t}} = \bigvee_{j \in [0,N]} u_{\mathrm{k,i,j,t}}$$

$$\forall k \in [1,K], \forall t \in \left[1, sup\left(\bar{T}\right)\right], \forall j \in [0,N] \quad y_{\mathrm{k,j,t}} = \bigvee_{i \in [0,N]} u_{\mathrm{k,i,j,t}}$$

La deuxième étape consiste à établir une connexion entre ces variables et les variables du modèle principal.

$$\forall k \in [1,K], \forall t \in \left[1, sup\left(\bar{T}\right)\right], \forall i \in [0,N] \quad i \notin dom\left(X_{\mathrm{k,t}}\right) \iff \neg\, x_{\mathrm{k,i,t}}$$

$$\forall k \in [1,K], \forall t \in \left[1, sup\left(\bar{T}\right)\right], \forall j \in [0,N] \quad j \notin dom\left(Y_{\mathrm{k,t}}\right) \iff \neg\, y_{\mathrm{k,j,t}}$$

Où $dom\left(X\right)$ dénote le domaine de la variable X avec $inf\left(X\right) = \min_{x \in dom(X)}$ et $sup\left(X\right) = \max_{x \in dom(X)}$.

2.1 La case fictive $(0,0)$

Comme précédemment annoncé, chaque roi peut soit rester sur l'échiquier de taille $N \times N$, soit se trouver sur une case "fictive" supplémentaire avec les coordonnées $(0,0)$.

Les variables de position $X_{\mathrm{k,t}}$ et $Y_{\mathrm{k,t}}$ peuvent donc prendre des valeurs dans $[0,N]$. Nous introduisons les contraintes suivantes afin d'interdire les cases non-existantes $(1,0), ..., (N,0)$, $(0,1), ... (0,N)$:

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

Pour le modèle booléen cette restriction est simple :

$$\forall k \in [1,K], \forall t \in \left[1, sup\left(\bar{T}\right)\right], \forall i \in [1,N], \quad u_{\mathrm{k,i,0,t}} = false$$

$$\forall k \in [1,K], \forall t \in \left[1, sup\left(\bar{T}\right)\right], \forall j \in [1,N], \quad u_{\mathrm{k,0,j,t}} = false$$

2.2 Dispositions initiale et finale

La position initiale du roi $\mathcal{R}_k$ est donnée par (a_k, b_k). À l'instant $\bar{T}$ tous les rois ont pour coordonnées $(0,0)$:

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

Ces conditions permettent d'instancier immédiatement une partie des variables de position et ne sont pas explicitement appliquées au modèle booléen.

2.3 Contraintes de distance

Les rois ne peuvent pas se trouver sur les cases adjacentes. En consequence, en utilisant les variables de position on obtient :

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

Un ensemble de contraintes disjonctives sur les variables booléennes peut être formulé sous la forme suivante :$\forall k' \neq k, \neg u_{\mathrm{k,i,j,t}} \bigvee \neg u_{\mathrm{k',i',j',t}}$ où i,j et i',j' satisfont $|i - i'| \leq 1$ et $|j - j'| \leq 1$. Néanmoins, cela augmente significativement le nombre de contraintes du modèle et amène à une forte réduction de la performance. Cet ensemble de *contraintes booléennes de distance* peut être remplacé par des contraintes globales plus compactes et plus génériques décrites dans la section 3.

Une remarque importante est que la case fictive $(0,0)$ n'apparait jamais dans les contraintes d'inter-distance. Afin de simplifier les notations, les conditions correspondantes seront omises dans les sections suivantes.

2.4 Contraintes sur la dynamique

Comme la dynamique des rois est extrêmement simple, on peut facilement utiliser la modélisation suivante :

- La longueur d'un déplacement est au plus 1 : $\forall k \in [1,K], \forall t \in [1, sup(\bar{T}) - 1], |X_{k,t+1} - X_{k,t}| \leq 1$ et $|Y_{k,t+1} - Y_{k,t}| \leq 1$
- Le déplacement est obligatoire : $\forall k \in [1,K], \forall t \in [1, sup(\bar{T}) - 1], (X_{k,t} \neq X_{k,t+1})$ ou $(Y_{k,t} \neq Y_{k,t+1})$

Une dynamique plus complexe peut être prise en compte par un graphe des transitions Γ. Chaque case de l'échiquier à un instant donné est associée à un sommet de ce graphe. Il existe une arête entre deux sommets (x,y) et (x',y') si et seulement si un roi peut se déplacer de (x,y) à (x',y') en un seul pas.

La Figure 4 montre une partie de ce graphe des transitions adjacentes à la case cible. Les sommets correspondent aux points de la grille et les arêtes sont représentées par les flèches en pointillés. Pour notre

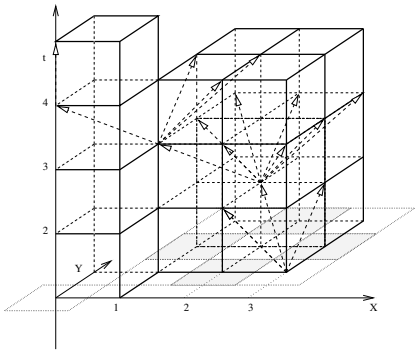

FIG. 4 – Graphe de transitions

dynamique, à chaque case (x,y) avec $1 \leq x \leq N$ et $1 \leq y \leq N$ et $(x,y) \neq (1,1)$ on obtient :

$$\Gamma\begin{pmatrix}x\\y\end{pmatrix} = \left\{ \begin{pmatrix}x'\\y'\end{pmatrix} \middle| \begin{array}{l} x' \in [x-1, x+1] \cap [1,N] \\ y' \in [y-1, y+1] \cap [1,N] \end{array} \right\} \setminus \left\{ \begin{pmatrix}x\\y\end{pmatrix} \right\}$$

Comme précisé précédemment, Γ peut être utilisé pour la modélisation d'une dynamique plus complexe.

Le roi se trouvant sur la case $(1,1)$ peut éventuellement être enlevé de l'échiquier (déplacement vers $(0,0)$) où rester sur l'échiquier en se déplaçant sur une autre case adjacente :

$$\Gamma\left(\begin{pmatrix}1\\1\end{pmatrix}\right) = \left\{ \begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}1\\2\end{pmatrix}, \begin{pmatrix}2\\1\end{pmatrix}, \begin{pmatrix}2\\2\end{pmatrix} \right\}$$

Une fois enlevé, le roi ne quitte plus la case cible :

$$\Gamma\left(\begin{pmatrix}0\\0\end{pmatrix}\right) = \left\{ \begin{pmatrix}0\\0\end{pmatrix} \right\}.$$

Dans la suite, on utilisera une formulation plus générique de la dynamique. En particulier, les contraintes globales pour la dynamique sont appliquées au graphe des transitions arbitraires plutôt que sur une dynamique plus précise.

La dynamique d'un roi est alors définie par la relation de récurrence suivante :

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

En utilisant la formulation booléenne on obtient directement :

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

En plus, on a aussi $u_{k,i,j,t} \Rightarrow \neg\, u_{k,i,j,t+1}$.

3 Propagation des contraintes globales

3.1 Indisponibilité des cases

La première contrainte globale redondante introduite dans cette section est une généralisation des contraintes de distance entre les rois. Au cours de la recherche, il est possible de faire des déductions supplémentaires à partir de l'information partielle sur la position des rois.

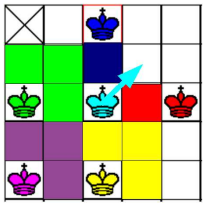

FIG. 5 – Déductions de l'indisponibilité des cases

Considérons par exemple l'instance sur la Figure 5. Quelque soit le mouvement du roi $\mathcal{R}_k$ qui se trouve au coin de l'échiquier à l'instant t donné, les trois cases adjacentes à sa position ne pourront pas être occupées par autres rois à l'instant $t+1$. Nous pouvons donc utiliser cette information pour réduire les domaines des variables de position de tous les rois $\mathcal{R}_{k'}$, $k' \neq k$ à l'instant $t+1$.

Ce mécanisme simple peut être étendu comme suit : on considère à un instant t l'intersection I des voisinages de toutes les positions admissibles c.à.d. pas encore interdites pour un roi $\mathcal{R}_k$. Toutes les cases dans I sont alors adjacentes à la position du roi $\mathcal{R}_k$ à l'instant

$t+1$. Les autres rois $\mathcal{R}_{k'}$, $k' \neq k$ ne peuvent donc pas se trouver sur l'une des cases de I à l'instant $t+1$.

On peut remarquer que l'ensemble I est vide si $(|dom\,(X_k)| > 3)$ ou $(|dom\,(Y_k)| > 3)$. En conséquence, il est facile de vérifier que l'algorithme 1 propage toutes les déductions nécessaires sur les variables $u_{\mathrm{k,i,j,t}}$.

Lorsque la position d'un roi est contenue dans un carré 3×3, l'algorithme 1 réduit les domaines de variables $u_{\mathrm{k,i,j,t}}$ et sa complexité est linéaire en nombre de rois ($O(K)$). Cette contrainte globale fait

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
```

plus de déductions que les contraintes d'inter-distance booléennes décrites dans la section précédente. En fait, les mêmes déductions sont réalisées lorsque les variables $X_{\mathrm{k,t}}$ et $Y_{\mathrm{k,t}}$ sont instanciées.

3.2 Capacité des rectangles

Le deuxième ensemble de contraintes globales est dérivé de la proposition suivante :

Proposition 3.1 *Le nombre maximal de rois se trouvant dans un rectangle $l \times h$ ne peut pas excéder* $\left\lceil \frac{l}{2} \right\rceil \left\lceil \frac{h}{2} \right\rceil$

Preuve: On ne peut pas avoir plus d'un roi dans un rectangle 2×2. Le nombre de rois dans un rectangle de taille $l \times h$ est donc borné par le nombre de rectangles de taille 2×2 qu'on peut placer dans ce rectangle. □

Comme le montre la figure 7, cette limite peut être atteinte.

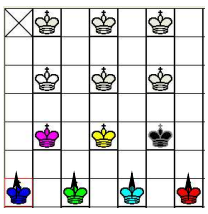

FIG. 6 – Un blocage

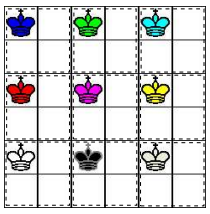

FIG. 7 – Capacité

Cette propriété permet de déduire immédiatement que l'instance sur la Figure 6 est bloquante. En effet, les contraintes décrites dans la Section 3.1 nous permettent de déduire que tous les rois de la dernière ligne doivent se déplacer verticalement. À partir de cette information on peut voir que la capacité de 6 premières lignes est forcément dépassée. Il n'existe donc pas de solution pour cette instance.

De manière plus formelle, on introduit la notation suivante qui indique que le roi $\mathcal{R}_k$ se trouve dans le rectangle $[i, i+l-1] \times [j, j+h-1]$:

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

Une solution partielle ne pourra donc pas être complétée si la condition suivante est violée pour un rectangle donné :

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

où $\#(S)$ désigne la cardinalité de l'ensemble S.

En outre, si le nombre de rois dans un rectangle donné est égal à la capacité de ce rectangle alors les cases à l'intérieur deviennent "interdites" pour les autres rois. Plus formellement, si pour le rectangle $[i, i+l-1] \times [j, j+h-1]$ la cardinalité de l'ensemble $S = \{\mathcal{R}_k \mid \mathcal{R}_k(t) \in ([i, i+l-1] \times [j, j+h-1])\}$ des rois se trouvant à l'intérieur à l'instant t est exactement $\left\lceil \frac{l}{2} \right\rceil \left\lceil \frac{h}{2} \right\rceil$ alors tous les rois sauf ceux dans S sont forcement à l'extérieur de ce rectangle à l'instant t c.à.d. $\forall \mathcal{R}_k,\ \mathcal{R}_k(t) \notin S \Rightarrow \mathcal{R}_k(t) \notin ([i, i+l-1] \times [j, j+h-1])$.

Algorithm 2 Propagation de la contrainte de capacité d'un rectangle

```
Require: X_k, Y_k, x_min, x_max, y_min, y_max, t
  nbS := 0
  for all k do
    if dom(X_k) × dom(Y_k) ⊆ [x_min, x_max] × [y_min, y_max] then
      nbS := nbS + 1
  if nbS > ⌈(x_max − x_min)/2⌉ ⌈(y_max − y_min)/2⌉ then
    There is no possible solution
  if nbS = ⌈(x_max − x_min)/2⌉ ⌈(y_max − y_min)/2⌉ then
    for all k do
      for all i do
        for all j do
          if dom(X_k) × dom(Y_k) ∉ [x_min, x_max] × [y_min, y_max] then
            u_{k,i,j,t} := false
```

L'algorithme 2 décrit la propagation de cette contrainte pour un rectangle $(x_{\min}, x_{\max}) \times (y_{\min}, y_{\max})$ à l'instant t. La complexité d'une

exécution de cet algorithme est $O(K)$. On peut noter que pour un instant donné on peut avoir $O(N^4)$ rectangles à vérifier. Nous considérons dans la Section 5 deux variations relatives du modèle en rajoutant les contraintes de ce type pour tous les rectangles ou seulement pour N^2 rectangles avec un coin fixé à $(1,1)$.

3.3 Trajectoires

Comme introduit dans la Section 2.4, la dynamique peut être décrite par

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

ainsi que la relation inversée :

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

Nous imposons la cohérence d'arc généralisée (GAC-scheme de [8]) sur cet ensemble de contraintes afin d'éliminer les valeurs incohérentes à partir des domaines de variables $u_{\mathrm{k,i,j,t}}$. Une contrainte est arc-cohérente si toutes les valeurs appartenant aux domaines des variables de cette contrainte peuvent participer à une solution. GAC élimine itérativement toutes les valeurs incohérentes et l'information correspondante est propagée sur toutes les autres contraintes potentiellement concernées.

La proposition suivante montre que cette propagation impose que la case cible soit accessible.

Proposition 3.2 *Après l'application de GAC pour les ensembles de contraintes (1) et (2), si la valeur "vrai" appartient au domaine de la variable* $u_{\mathrm{k,i,j,t}}$*, alors il existe une trajectoire pour le roi* $\mathcal{R}_k$ *à partir de sa position initiale jusqu'à la case cible passante par* (i,j) *à l'instant* t*.*

Preuve: Tout d'abord on peut montrer que le roi peut atteindre la case cible si il peut se trouver à la case (i,j) à l'instant t.

La valeur "vrai" est enlevée par GAC à partir du domaine de $u_{\mathrm{k,i,j,t}}$ si toutes les variables de la partie droite de (1) sont instanciées à "faux". Donc, si GAC n'a pas déduit une incohérence alors pour chaque valeur "vrai" dans les domaines il existe une variable non-instanciée à "faux" dans la partie droite de la contrainte (1) correspondante.

En conséquence, on construit une liste de variables $u_{\mathrm{k,i,j,t}}, u_{k,i',j',t+1}, u_{k,i'',j'',t+2}, \ldots$ contenant la valeur "vrai" dans son domaine afin de construire un chemin de la position (i,j) à l'instant t. Comme à l'instant $sup(dom(T))$ toutes les variables u sont instanciées à "faux" sauf $u_{k,0,0,sup(T)}$ la dernière variable de la liste ainsi construite est associée à la case cible.

Pour prouver qu'il existe une trajectoire de la position initiale du roi $\mathcal{R}_k$ jusqu'à la case (i,j) à l'instant t on peut appliquer le raisonnement similaire aux inégalités (2). □

La proposition précédente montre que ce mécanisme extrêmement simple est efficace pour propager les déductions liées à l'existence des trajectoires. Nous pouvons aussi remarquer que ce schéma de propagation est défini pour un graphe de transitions Γ similaire et peut donc être utilisé pour une dynamique plus complexe.

3.4 Ordonnancement

À cause des contraintes d'inter-distance, il y a au moins deux pas de temps entre deux sorties de rois consécutives :

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

Ceci nous permet de renforcer le modèle en contraintes en utilisant une contrainte redondante de ressource disjonctive. Une tâche avec la durée 2 est associe à chaque roi $\mathcal{R}_k$. La date de début de cette tâche est T_k. Les contraintes associées modélisent le fait que toutes ces tâches doivent être exécutées sur une machine de capacité unitaire (c.à.d. leurs exécutions ne peuvent pas s'entrelacer). Il existe de nombreuses techniques de propagation pour ce problème d'ordonnancement (voir [2] pour une revue). Nous utilisons les règles de "edge-finding" [9] et "not-first/not-last" [3, 9, 14, 15].

Comme toutes les durées des tâches sont égales nous avons également étudié une autre technique de propagation basée sur la contrainte "All-Different". Cette contrainte impose le fait que toutes les valeurs prises par les différentes variables d'un ensemble soient deux à deux distinctes. Deux variantes peuvent être utilisées :

- L'algorithme de la cohérence aux bornes décrit dans [12]. Cette propagation garantit que chaque borne du domaine de chaque variable peut participer à une solution pour cette contrainte.
- L'algorithme de la cohérence d'arc décrit dans [13] qui élimine des domaines des variables toutes les valeurs qui ne peuvent pas participer à aucune solution de la contrainte.

Pour notre problème d'ordonnancement on transforme l'équation (3.4) comme suit : on introduit deux ensembles de variables $\tau_k^- = \lfloor T_k/2 \rfloor$ et $\tau_k^+ = \lceil T_k/2 \rceil$.

Toutes ces variables doivent être deux à deux distinctes. Elles peuvent donc être réliées par des les contraintes de type "All-Different" :

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

Ces contraintes permettent de propager plus que les contraintes de ressource disjonctive. Cependant, les expérimentations ont montré que ces propagations supplémentaires sont rarement utiles. Dans la suite on utilise seulement le "edge-finding" et "not-first/not-last".

3.5 Propagation de SAC

La propagation de la cohérence de singletons (Singleton Arc Consistency) décrite dans [7], [4] est appliquée sur les variables $X_{k,2}$ et $Y_{k,2}$ comme décrit dans l'algorithme 3. Cet algorithme vérifie qu'après l'instantiation d'une variable à une valeur il est possible de rendre le problème arc-cohérent. Cette propagation renforcée est fortement corrélée avec la technique de "shaving" [10, 11].

Étant donné un réseau de contraintes P, cet algorithme garantit que le réseau de contraintes $P|D_i = a$, obtenu à partir de P par la réduction du domaine D_i de la variable V_i à un singleton a, est arc-consistant. Dans le cas contraire, cette instantiation ne peut pas participer à une solution du problème.

Algorithm 3 Propagation de SAC

```
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
```

4 Heuristiques de recherche

Les expérimentations initiales ont permis de choisir l'heuristique de recherche suivante : une fois que le problème d'ordonnancement est résolu on construit les trajectoires des rois associées.

Étape 1 S'il existe encore des dates de sortie non-déterminées alors on en choisit une variable T_k avec $inf\,(T_k)$ minimal parmi les dates non-fixées. Une fois la variable choisie, on essaye de l'instancier à sa valeur minimale. Lors d'un retour-arrière cette valeur est éliminée du domaine de T_k.

Étape 2 Si toutes les dates de sortie sont connues alors on considère le premier instant t tel qu'il existe au moins une position d'un roi encore inconnue (s'il n'existe pas de t correspondant alors le problème est résolu). On choisit parmi les variables $X_{\mathrm{k,t}}$ et $Y_{\mathrm{k,t}}$ celle qui correspond à la valeur minimale de T_k (grâce à l'étape 1, ces valeurs sont connues) et on énumère les valeurs dans son domaine en ordre croissant.

Une autre variante de l'heuristique de recherche consiste à n'utiliser que l'étape 2 (c.à.d. construire les trajectoires en ordre lexicographique). Dans ce cas, les variables T_k n'étant pas encore instanciées donc parmi les variables de position $X_{\mathrm{k,t}}$ et $Y_{\mathrm{k,t}}$ on choisit celle correspondante à la valeur minimale de $inf\,(T_k)$.

5 Résultats expérimentaux

Une instance du problème est caractérisée par la taille de l'échiquier $N \times N$, le nombre de rois K et par les positions initiales des rois (a_k, b_k). Plusieurs classes d'instances ont été générées, la taille maximale de l'échiquier allant jusqu'à 7×7. Le nombre d'instances par classe est de l'ordre de quelques milliers et quelques centaines de ces instances sont non-bloquantes.

Deux sortes d'expérimentations ont été effectuées :

- Tout d'abord, on identifie les instances pour lesquelles il n'existe pas de mouvement (Figure 2).
- Si un mouvement existe alors on cherche à déterminer la suite de transitions permettant de vider l'échiquier au plus vite.

Dans les deux cas on évalue la performance relative des différentes variantes du modèle (en utilisant ou non les règles de propagation proposées). Toutes les expérimentations ont été effectuées sur Intel Pentium-M 1.5 GHz (Centrino).

5.1 L'existence d'un mouvement

La première question étudiée dans cette section est de déterminer si à partir d'une disposition initiale donnée il existe un mouvement admissible ou pas. Nous avons analysé le comportement de plusieurs combinaisons d'algorithmes de propagation. Chaque colonne du tableau 1 correspond à l'une de ces combinaisons. Chaque ligne correspond à une règle de propagation et un signe "+" dans la colonne correspondante indique que la propagation associée a participé dans la combinaison associée.

- La ligne *Bool* indique si les contraintes d'inter-distance booléennes décrites dans la Section 2.3 ont été utilisées.

	A	B	C	D	E	F	G
Bool		+		+		+	+
Indisp			+		+		
N^4*check*				+	+		+
N^2*prop*						+	
N^4*prop*							+
SAC							

	H	I	J	K	L	M	N
Bool							
Indisp	+	+	+			+	
N^4*check*		+	+			+	+
N^2*prop*	+				+		
N^4*prop*		+				+	+
SAC			+	+	+	+	+

Tab. 1 – Paramètres

- La ligne *Indisp* correspond à l'application de l'algorithme de l'indisponibilité des cases (Algorithme 1).
- Les lignes N^4*check*, N^2*prop* et N^4*prop* font référence aux propagations des contraintes de capacité des rectangles décrites dans la Section 3.2. N^4*check* signifie que seulement la satisfaisabilité à été vérifiée pour tous $O(N^4)$ rectangles. La propagation a été appliquée soit pour tous les rectangles soit pour $O(N^2)$ rectangles avec un coin fixé à $(1, 1)$.
- La ligne *SAC* indique si la propagation de consistance de singletons à été utilisée (Algorithme 3).

La propagation seule permet d'identifier immédiatement une partie des instances pour lesquelles il n'existe pas de mouvement. La Figure 8 montre le rapport entre le nombre de ces instances et le nombre total des instances sans solution. Ces résultats sont donnés pour 4 classes d'instances et pour les 14 variantes du modèle. Les colonnes du graphique correspondent aux différentes classes : $6 * 5 \times 5$ (6 rois sur l'échiquier 5×5, gris foncé), $7 * 5 \times 5$ (7 rois sur 5×5, noir), $13 * 7 \times 7$ (13 rois sur 7×7, blanc), $14 * 7 \times 7$ (14 rois sur 7×7, gris clair).

On peut voir que la consistance des singletons joue un rôle crucial dans la détection de l'absence d'un mouvement. Pratiquement 100% de toutes ces instances ont été détectées par Algorithme 3. Les instances restantes de taille $13 * 7 \times 7$ sont détectées si une propagation supplémentaire est appliquée. Selon ces résultats expérimentaux, il semble que, après SAC l'algorithme le plus performant est celui de l'indisponibilité des cases.

5.2 Le problème complet

Comme pour le problème précédent, plusieurs combinaisons d'algorithmes de propagation ont été

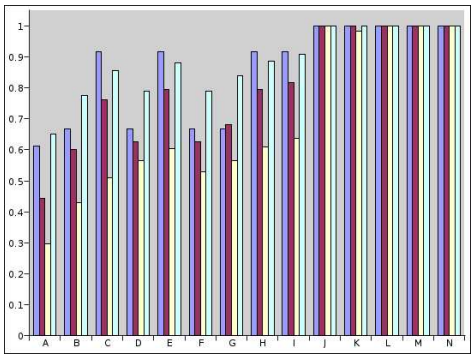

Fig. 8 – Pourcentage des blocages détectés à la phase de la propagation

étudiées pour le calcul des trajectoires optimales, c.à.d. les ensembles de trajectoires tels que la date à laquelle l'échiquier est vide est minimale. Toutes les combinaisons utilisées sont présentées dans le tableau 2. La ligne "Dyn" correspond à la propagation des contraintes sur la dynamique décrite dans la Section 3 et la ligne *Sched* correspond à l'heuristique de recherche choisie (un "+" montre que les deux étapes de la section 4 ont été utilisées). Les contraintes d'indisponibilité des cases ont été activées dans toutes les variantes et un "+" dans la ligne N^4*prop* signifie que les $O(N^4)$ contraintes de capacité des rectangles ont été propagées (vs. vérification de cohérence seule).

Finalement, les résultats du modèle initial basé seulement sur les variables T_k, $X_{\mathrm{k,t}}$ et $Y_{\mathrm{k,t}}$ avec l'heuristique de recherche basée sur la construction des trajectoires sont montrés sur les graphiques dans la colonne *XY*.

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

Tab. 2 – Paramètres

Les Figures 9, 10, 11, 12, 13 et 14 donnent les résultats obtenus sur toutes les instances non-bloquantes (des classes correspondantes) pour lesquelles une solution a été trouvée par toutes les combinaisons de contraintes. Les graphiques donnent sur une échelle logarithmique, le temps processeur moyen en millisecondes et le nombre de retours-arrière pour chaque classe d'instances et pour chaque combinaison. Un résultat inattendu est que si le problème d'ordonnancement est résolu avant la construction des trajectoires (K, M, O, Q) alors la première solution trouvée est toujours optimale et le nombre total des retours-arrière reste petit. Les deux autres éléments

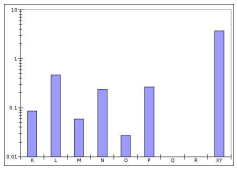

FIG. 9 – BT $6 * 5 \times 5$

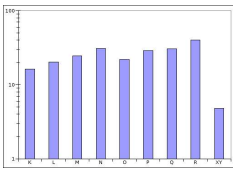

FIG. 10 – Temps $6 * 5 \times 5$

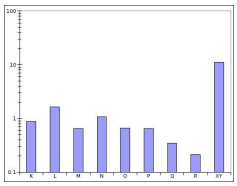

FIG. 11 – BT $9 * 6 \times 6$

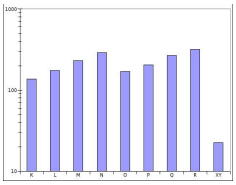

FIG. 12 – Temps $9 * 6 \times 6$

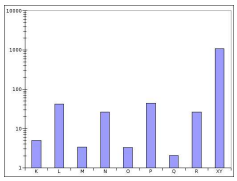

FIG. 13 – BT $13 * 7 \times 7$

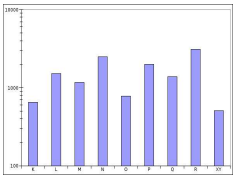

FIG. 14 – Temps $13 * 7 \times 7$

(les contraintes dynamiques et les contraintes de capacité des rectangles) sont utiles pour la réduction de l'espace de recherche.

Cependant, sans la propagation des contraintes de capacité des rectangles, à peu près de 10% d'instances de la plus grande taille (13 rois sur 7×7) n'étaient pas résolus à l'aide de cette heuristique après 5 minutes (variantes K et O).

La variante initiale fait beaucoup plus de retours-arrière que les variantes plus compliquées mais cependant elle est plus rapide sur les instances de petite taille. Cette différence n'est pas aussi importante pour les instances de plus grande taille, qui étaient accessibles pour notre implémentation du modèle booléen.

5.3 Problème modifié

Deux autres variations du problème initial sont facilement abordables selon cette approche.

- La première modification consiste à interdire à un roi de revenir sur la case qu'il vient de quitter. En outre, le laps de temps entre deux sorties consécutives est au moins de trois pas de temps. Ces deux contraintes supplémentaires représentent une dynamique des avions un peu plus réaliste ainsi que le fait que la distance de sécurité peut dépendre d'autres facteurs que la distance physique.
- En plus des contraintes précédentes, une zone interdite en forme de petit "mur" est ajoutée sur les cases de $(2, 1)$ à $(2, 3)$. Ceci modélise le fait que une partie de l'espace aérien peut être rendue indisponible aux avions (ceci est fréquent lors d'opérations militaires).

Les combinaisons présentées dans la section précédente (tableau 2) ont été utilisées pour cette analyse.

Les Figures 15, 16, 17, 18 donnent le nombre des retours-arrière et le temps nécessaire pour trouver la solution optimale de la première modification et prouver sa optimalité pour toutes les instances $6 * 5 \times 5$ et 50 instances aléatoirement choisies de la taille $8 * 6 \times 6$. La propagation des contraintes des chemins améliorent les performances (L vs. P, N vs. R) tandis que l'impact de la propagation des contraintes de rectangles n'est pas aussi important. Comme dans la version non-modifiée, il est plus avantageux de résoudre d'abord le problème d'ordonnancement que de commencer directement par la construction des trajectoires. Finalement, la variante initiale n'est plus compétitive.

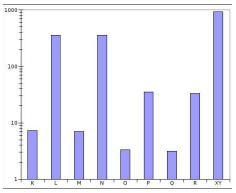

FIG. 15 – BT on $6 * 5 \times 5$

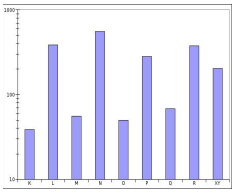

FIG. 16 – Temps $6 * 5 \times 5$

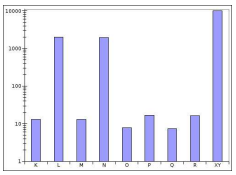

FIG. 17 – BT on $8 * 6 \times 6$

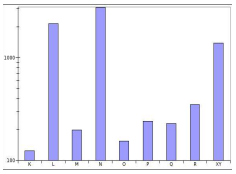

FIG. 18 – Temps $8 * 6 \times 6$

Les Figures 20 et 19 donnent les mêmes résultats pour 6 rois sur l'échiquier de la taille 5×5. En comparaison avec les résultats précédents on peut voir que l'heuristique basée sur le problème d'ordonnancement n'améliore plus les performances. Néanmoins, la première solution trouvée est toujours optimale. Finalement, la variation initiale du modèle (colonne XY) n'a pas réussi à résoudre environ un tiers d'instances après une minute et on ne présente que les résultats pour les instances résolues par cette variante.

6 Conclusion

Nous avons introduit un modèle en Programmation par Contraintes pour un problème d'optimisation diffi-

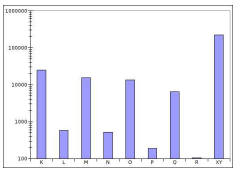

FIG. 19 – BT 6∗5×5 (mur)

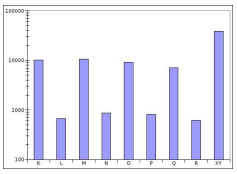

FIG. 20 – Temps $6 * 5 \times 5$ (mur)

cile dans lequel on cherche à déterminer les dates d'atterrissage et les trajectoires des avions à l'approche d'un aéroport (Terminal Radar Approach CONtrol area). Plusieurs modèles et algorithmes ont été proposés et évalués. Nous avons pu identifier les éléments de base pour la construction d'une approche en PPC pour ce type de problèmes. Nous avons aussi démontré que notre modèle peut être étendu pour la résolution de situations plus complexes.

Notre approche a plusieurs limitations, mais à notre avis la plus importante est qu'on ne peut pas utiliser cette approche pour des instances de grande taille. Cela nous oblige à discrétiser grossièrement le temps et l'espace. Il y a plusieurs améliorations possibles à la procédure de recherche. Par exemple, on peut utiliser une méthode qui permet d'éviter de résoudre plusieurs fois le même sous-problème. Il est également possible que la propagation puisse encore être améliorée.

Finalement, nous voudrions mentionner le fait que nous n'avons pas prouvé que le problème de vérification de l'existence d'un mouvement est NP-complet. Il existe donc peut-être un algorithme polynomial pour ce problème.

Remerciements

Les auteurs souhaitent remercier Jean-Marc Steyaert pour plusieurs discussions instructives sur les systèmes hybrides.

Références

[1] K. Artiouchine, P. Baptiste, and C. Dürr. Runway sequencing with holding patterns. Technical report, Laboratoire d'Informatique de l'Ecole Polytechnique, 2004.

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

[3] P. Baptiste and C. Le Pape. Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling. In *Proceedings of the fifteenth workshop of the U.K. planning special interest group*, 1996.

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

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

[6] J.E. Beasley, M. Krishnamoorthy, Y.M.Sharaiha, and D.Abramson. Scheduling aircraft landings - the static case. *Transportation Science*, 34(2) :180–197, 2000.

[7] C. Bessière and R. Debruyne. Theoretical analysis of singleton arc consistency. In *Proceedings of ECAI-04 workshop on Modeling and Solving Problems with Constraints*, 2004.

[8] C. Bessière and J.-C. Régin. Arc consistency for general constraint networks : preliminary results. In *Proceedings of IJCAI, Nagoya, Japan*, pages 398–404, 1997.

[9] J. Carlier and E. Pinson. A practical use of jackson's preemptive schedule for solving the job-shop problem. *Annals of Operations Research*, 26 :269–287, 1990.

[10] J. Carlier and E. Pinson. Adjustments of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78 :146–161, 1994.

[11] P. D. Martin and D. B. Shmoys. Approach to computing optimal schedules for the job-shop scheduling problem. In *Proceedings of 5th Conference on Integer Programming and Combinatorial Optimization*, 1996.

[12] J.-F. Puget. A fast algorithm for the bound consistency of alldiff constraints. In *Proceedings of AAAI-98*, pages 359–366, 1998.

[13] J.-C. Régin. *Développement d'outils algorithmiques pour l'Intelligence Artificielle. Application à la chimie organique*. PhD thesis, Université Montpellier II, 1995.

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

[15] P. Vilím. $o(n \log n)$ filtering algorithms for unary resource constraint. In Jean-Charles Régin and Michel Rueher, editors, *Proceedings of CP-AI-OR*, volume 3011 of *LNCS*, pages 335–347. Springer, 2004.