Projet : INFO-F-302 Informatique Fondamentale.

George Rusu et Maximilien Romain

21 mai 2017

Table des matières

1 Introduction

Le premier objectif de ce projet est de modeliser divers problemes en problemes de satisfaction de contraintes (CSP). Le second objectif est d'implementer un programme resolvant ces problemes en utilisant ChocoSolver.

link : http ://www-master.ufr-info-p6.jussieu.fr/2005/IMG/pdf/rp3.pdf

2 Question 1

L'ensemble des cases du jeux V où $\#V = n^2$
Variables de décision $X = \{x_{i,j} | \forall i, j (1 \leq i \leq n).(1 \leq j \leq n)\}$, n^2 variables de décision
Domaines : $D = \{vide, fous, cavalier, tour\}$
Contraintes : Pour chacune des pièces de l'echequier, si une case est occupé par un pions, alors dans la porté de ce pion, donc les cases attaquables par ce pion, doivent être obligatoirement vide. On parcours donc chaque case, et lorsque qu'une case est occupé, en fonction du pion qui occupe cette case, les contraintes changent.

2.1 Contraintes tour

Pour chaque colonne de l'echequier, si une case de cette colonne est occupée par une tour, alors le reste des cases de cette colonne doivent être vide :

$$c_{T_{col},j} = ((x_{1,j}, x_{2,j}, \ldots, x_{n,j}), \{(b_1, b_2, \ldots, b_n) | b_i = T, b_j = V, \forall j \neq i\})$$

Pour chaque ligne de l'echequier, si une case de cette ligne est occupée par une tour, alors le reste des cases de cette ligne doivent être vide :

$$c_{T_{i,ligne}} = ((x_{i,1}, x_{i,2}, \ldots, x_{i,n}), \{(b_1, b_2, \ldots, b_n) | b_i = T, b_j = V, \forall j \neq i\})$$

2.2 Contraintes fou

Les contraintes du fou sont composées dans le même état d'esprit que les contraintes des tours, cepandant la portée d'un fou change, et en fonction de la digonale sur laquelle est placé le fou, la distance maximal que ce fou peut atteindre change. En effet les plus grand diagonales se trouve en $(0, 0)$ et $(0, n-1)$. Par contrainte, on considère toujours la diagonale opposée, et on réunie les deux diagonales par un *and*.

$$c_{F,2*n-2} = ((x_{1,n-1}, x_{2,n}), \{(b_1, b_2) | b_1 = F, b_2 = V \vee b_1 = V, b_2 = F\}) \wedge$$
$$((x_{1,2}, x_{2,1}), \{(b_1, b_2) | b_1 = F, b_2 = V \vee b_1 = V, b_2 = F\})$$

$$\vdots$$

$$c_{F,n+1} = ((x_{1,2}, x_{2,3}, \ldots, x_{n-1,n}), \{(b_1, b_2, \ldots, b_{n-1}) | b_i = F, b_j = V, \forall j \neq i\}) \wedge$$
$$((x_{1,n-1}, x_{2,n-2}, \ldots, x_{n-1,1}), \{(b_1, b_2, \ldots, b_{n-1}) | b_i = F, b_j = V, \forall j \neq i\})$$
$$c_{F,n} = ((x_{1,1}, x_{2,2}, \ldots, x_{n,n}), \{(b_1, b_2, \ldots, b_n) | b_i = F, b_j = V, \forall j \neq i\}) \wedge$$
$$((x_{1,n}, x_{2,n-1}, \ldots, x_{n,1}), \{(b_1, b_2, \ldots, b_n) | b_i = F, b_j = V, \forall j \neq i\})$$
$$c_{F,n-1} = ((x_{2,1}, x_{3,2}, \ldots, x_{n,n-1}), \{(b_1, b_2, \ldots, b_{n-1}) | b_i = F, b_j = V, \forall j \neq i\}) \wedge$$
$$((x_{2,n}, x_{3,n-1}, \ldots, x_{n,2}), \{(b_1, b_2, \ldots, b_{n-1}) | b_i = F, b_j = V, \forall j \neq i\})$$

$$\vdots$$

$$c_{F,2} = ((x_{n-1,1}, x_{n,2}), \{(b_1, b_2) | b_1 = F, b_2 = V \vee b_1 = V, b_2 = F\}) \wedge$$
$$((x_{n-1,n}, x_{n,n-1}), \{(b_1, b_2) | b_1 = F, b_2 = V \vee b_1 = V, b_2 = F\})$$

Les diagonales de gauche à droite sont représenté par la matrice ci-dessous.

$$\begin{pmatrix} c_{F,n} & c_{F,n+1} & \cdots & c_{F,2*n-2} & \\ c_{F,n-1} & c_{F,n} & c_{F,n+1} & \vdots & c_{F,2*n-2} \\ \vdots & c_{F,n-1} & c_{F,n} & c_{F,n+1} & \vdots \\ c_{F,2} & \vdots & c_{F,n-1} & c_{F,n} & c_{F,n+1} \\ & c_{F,2} & \cdots & c_{F,n-1} & c_{F,n} \end{pmatrix} \quad (1)$$

2.3 Contraintes cavalier :

Les contraintes des cavaliers sont plus simple. En effet si on rencontre une case occupée par un cavalier, alors il n'y a que 8 contraintes pour les 8 cases que le cavalier peut atteindre. Si le cavalier est en (i, j), alors il ne peut qu'attaquer les cases où $i + e, j + e, e \in [+1, -1, +2, -2]$

$$c_{C,(i,j)} = ((x_{i,j}, x_{i+1,j+2}, x_{i+1,j-2}, x_{i-1,j+2}, x_{i-1,j-2}, x_{i+2,j+1}, x_{i+2,j-1}, x_{i-2,j+1}, x_{i-2,j-1}),$$
$$\{(b_1, b_2, \ldots, b_9)|b_1 = C, b_2, \ldots b_9 = V\})$$

3 Question 2

L'ensemble des cases du jeux V où $\#V = n^2$
Variables de décision $X = \{x_{i,j}|\forall i, j(1 \leq i \leq n).(1 \leq j \leq n)\}$, n^2 variables de décision
Domaines : $D = \{Vide, tour, fous, cavalier\}$
Contraintes : Pour chacune des pièces de l'echequier, si une case est vide, alors il existe au moins un pion qui menace cette case vide

3.1 Contraintes tours

Pour chaque case de l'echequier qui n'est pas occupé par un pion, alors il existe au moins une tour dans la ligne ou la colonne de la case non occupée.

$$c_T = ((x_{i,j}, \forall i, j(1 \leq i \leq n).(1 \leq j \leq n)),$$
$$\{b_{i,j}, \forall i, j(1 \leq i \leq n).(1 \leq j \leq n)|b_{i,j} = V, \forall l, (1 \leq l \leq n)\exists b_{l,j} = T \vee b_{i,j} = V, \forall k(1 \leq k \leq n), \exists b_{i,k} = T\})$$

Nous devons aussi indiqué qu'il n'y pas d'autre pion entre la case vide et la tour. On indique donc que les cases entre la case vide et la case occupé par une tour, doivent non occupées. Nous avons donc une telle contrainte pour la ligne et la colonne de la case vide menacé par une tour.

$$c_{couple,T_{colonnes}} = ((x_{i,j}, x_{l,j}), \{(b_{i,j}, b_{l,j})|b_{i,j} = T, b_{l,j} = V, \rightarrow b_{m,j}\forall m(i < m < l), b_{m,j} = V\})$$
$$c_{couple,T_{lignes}} = ((x_{i,j}, x_{i,k}), \{(b_{i,j}, b_{i,k})|b_{i,j} = T, b_{i,k} = V, \rightarrow b_{i,m}\forall m(j < m < k), b_{i,m} = V\})$$

3.2 Contraintes fous

Pour chaque case de l'echequier qui n'est pas occupé par un pion, alors il existe au moins un fou dans les deux diagonales de la case vide.

$$c_F = ((x_{i,j}, \forall i, j(1 \leq i \leq n).(1 \leq j \leq n)),$$
$$\{b_{i,j}, \forall i, j(1 \leq i \leq n).(1 \leq j \leq n)|b_{i,j} = V, \forall l, k(1 \leq l \leq n).(1 \leq k \leq n)\exists b_{l,k} = F \vee$$
$$b_{i,j} = V, \forall l, k(1 \leq l \leq n).(n \leq k \leq 1), \exists b_{l,k} = F\})$$

De nouveau nous devons nous assurer qu'il n'y a pas d'autre pions qui gène le fou entre la case vide et ce fou.

$$c_{couple,F_{diag1}} = ((x_{i,j}, x_{l,k}), \{(b_{i,j}, b_{l,k})|b_{i,j} = T, b_{l,k} = V, \rightarrow b_{x,y}\forall x, y(i < x < l).(j < y < k), b_{x,y} = V\})$$
$$c_{couple,F_{diag2}} = ((x_{i,j}, x_{l,k}), \{(b_{i,j}, b_{l,k})|b_{i,j} = T, b_{l,k} = V, \rightarrow b_{x,y}\forall x, y(i < x < l).(k < y < j), b_{x,y} = V\})$$

3.3 Contraintes cavaliers :

Pour chaque case de l'echequier qui n'est pas occupé par un pion, alors il existe au moins un cavalier qui menace la case vide. Pour cela il suffit de verifier qu'il existe une case dans les 8 cases correspondant à l'ensemble des mouvements du cavalier ($\{+1, -1, +2, -2\}$), soit occupé par un cavalier.

$$c_{C,(i,j)} = ((x_{i,j}, x_{i+1,j+2}, x_{i+1,j-2}, x_{i-1,j+2}, x_{i-1,j-2}, x_{i+2,j+1}, x_{i+2,j-1}, x_{i-2,j+1}, x_{i-2,j-1}),$$
$$\{(b_1, b_2, \ldots, b_9) | b_1 = V, \forall i, j \in [+1, +2, -1, -2], \exists b_{i,j} = C\})$$

3.4 Contrainte finale :

Finalement il n'y a qu'une seule grande contrainte à appliquer à notre problème. Etant donné qu'il suffit qu'un seule pion menace une case vide, nous pouvons relier nos contrainte par des conditions *or*.

$$C = (C_T \wedge c_{couple,T_{colonnes}} \wedge c_{couple,T_{lignes}}) \vee (c_F \wedge c_{couple,F_{diag1}} \wedge c_{couple,F_{diag2}}) \vee c_{C,(i,j)}$$

4 Question 2 avec ensemble

L'ensemble des cases du jeux V où $\#V = n^2$
Variables de décision $X = \{x_{i,j} | \forall i, j (1 \leq i \leq n).(1 \leq j \leq n)\}$, n^2 variables de décision
Domaines : $D = \{Vide, tour, fous, cavalier\}$
Contraintes : Pour chacune des pièces de l'echequier, si une case est vide, alors il existe au moins un pion qui menace cette case vide. Le problème ici, est que nous devons trouver un moyen d'appliquer un *OR* de contraintes, ce qui n'est pas possible. Nous allons donc définir divers ensemble, et ensuite écrire une grande contrainte appliquant des *Unions* de ces ensembles.

4.1 Ensemble Tours

Pour chaque case de l'echequier qui n'est pas occupé par un pion, alors il existe au moins une tour dans la ligne ou la colonne de la case non occupée.

$$e_T = \{(x_{i,j}, \forall i, j (1 \leq i \leq n).(1 \leq j \leq n)),$$
$$\{b_{i,j}, \forall i, j (1 \leq i \leq n).(1 \leq j \leq n) | b_{i,j} = V, \forall l, (1 \leq l \leq n) \exists b_{l,j} = T \vee b_{i,j} = V, \forall k (1 \leq k \leq n), \exists b_{i,k} = T\}\}$$

Nous devons aussi indiqué qu'il n'y pas d'autre pion entre la case vide et la tour. On indique donc que les cases entre la case vide et la case occupé par une tour, doivent non occupées. Nous avons donc une telle contrainte pour la ligne et la colonne de la case vide menacé par une tour.

$$e_{couple,T_{colonnes}} = \{(x_{i,j}, x_{l,j}), \{(b_{i,j}, b_{l,j}) | (b_{i,j} = T, b_{l,j} = V) \wedge (\forall m (i < m < l), b_{m,j} = V)\}\}$$
$$e_{couple,T_{lignes}} = \{(x_{i,j}, x_{i,k}), \{(b_{i,j}, b_{i,k}) | (b_{i,j} = T, b_{i,k} = V) \wedge (\forall m (j < m < k), b_{i,m} = V)\}\}$$

4.2 Ensemble fous

Pour chaque case de l'echequier qui n'est pas occupé par un pion, alors il existe au moins un fou dans les deux diagonales de la case vide.

$$e_F = \{(x_{i,j}, \forall i, j (1 \leq i \leq n).(1 \leq j \leq n)),$$
$$\{b_{i,j}, \forall i, j (1 \leq i \leq n).(1 \leq j \leq n) | b_{i,j} = V, \forall l, k (1 \leq l \leq n).(1 \leq k \leq n) \exists b_{l,k} = F \vee$$
$$b_{i,j} = V, \forall l, k (1 \leq l \leq n).(n \leq k \leq 1), \exists b_{l,k} = F\}\}$$

De nouveau nous devons nous assurer qu'il n'y a pas d'autre pions qui gène le fou entre la case vide et ce fou.

$$e_{couple,F_{diag1}} = \{(x_{i,j}, x_{l,k}), \{(b_{i,j}, b_{l,k}) | (b_{i,j} = T, b_{l,k} = V) \wedge (\forall x, y (i < x < l).(j < y < k), b_{x,y} = V)\}\}$$
$$e_{couple,F_{diag2}} = \{(x_{i,j}, x_{l,k}), \{(b_{i,j}, b_{l,k}) | (b_{i,j} = T, b_{l,k} = V) \wedge (\forall x, y (i < x < l).(k < y < j), b_{x,y} = V)\}\}$$

4.3 Ensemble cavaliers :

Pour chaque case de l'echequier qui n'est pas occupé par un pion, alors il existe au moins un cavalier qui menace la case vide. Pour cela il suffit de verifier qu'il existe une case dans les 8 cases correspondant à l'ensemble des mouvements du cavalier ($\{+1, -1, +2, -2\}$), soit occupé par un cavalier.

$$e_{C,(i,j)} = \{(x_{i,j}, x_{i+1,j+2}, x_{i+1,j-2}, x_{i-1,j+2}, x_{i-1,j-2}, x_{i+2,j+1}, x_{i+2,j-1}, x_{i-2,j+1}, x_{i-2,j-1}),$$
$$\{(b_1, b_2, \ldots, b_9) | b_1 = V, \forall i, j \in \{+1, +2, -1, -2\}, \exists b_{i,j} = C\}\}$$

4.4 Contrainte finale :

Finalement il n'y a qu'une seule grande contrainte à appliquer à notre problème. Etant donné qu'il suffit qu'un seule pion menace une case vide, nous pouvons relier nos ensemble par des *unions* $\cup$ et des *intersections* de contraintes $\cap$.

$$C = (e_T \cap e_{couple,T_{colonnes}} \cap e_{couple,T_{lignes}}) \cup (e_F \cap e_{couple,F_{diag1}} \cap e_{couple,F_{diag2}}) \cup e_{C,(i,j)}$$

5 Question 3

5.1 Fichier main.java

6 Question Bonus

7 Question 4

8 Question 5

Nous avons un musée qui est reprséenté par une grille de $n * m$
Variables de décision $X = \{x_{i,j} | \forall i, j(1 \leq i \leq n).(1 \leq j \leq m)\}$
Domaines : $D = \{0, 1, 2, 3, 4, 5\}$ correspondant respectivement à une case vide, une caméra Nord, une caméra Sud, une caméra Est, une caméra Ouest et un mur.
Contraintes : La contrainte du musée est définie comme ceci : chaque case vide doit être surveillé par au moins une caméra. La solution doit être trouvée avec un nombre de caméra le plus possible.

8.1 Contraintes CSP

Pour appliquer la contrainte, il faut donc prendre en compte chaque case du musée. Pour cela nous prenons chaque $x_{i,j}$ compris dans la taille du musée. Ensuite il faut vérifier que pour chacune de ces cases, si elle est non occupée, il faut qu'elle soit surveillé par au moins une caméra. Ainsi nous avons une suite d'ensemble de valeurs liées par des relations *OR*, qui nous permet de dire : existe au moins une caméra observant la case vide.

Afin d'être sûr qu'une caméra pointe vers le bon sens, lorsque nous parcourons les lignes et colognes d'une case vide, nous précisons bien qu'il y ai au moins une caméra regardant vers le Nord dans les cases se trouvant en dessous de notre case $x_{i,j}$, et ainsi de suite pours les autres directions.

$$C = ((x_{i,j}, \forall i, j(1 \leq i \leq n).(1 \leq j \leq m)), \{b_{i,j}, \forall i, j(1 \leq i \leq n).(1 \leq j \leq m) | [(b_{i,j} = 0) \wedge (\forall k(i < k \leq n), \exists b_{k,j} = 1)] \vee$$
$$[(b_{i,j} = 0) \wedge (\forall k(0 \leq k < i), \exists b_{k,j} = 2)] \vee$$
$$[(b_{i,j} = 0) \wedge (\forall k(j < k \leq m), \exists b_{i,k} = 3)] \vee$$
$$[(b_{i,j} = 0) \wedge (\forall k(0 \leq k < j), \exists b_{i,k} = 4)]\})$$

Afin de réduire le nombre de caméra, nous avons reformulé de la demande d'optimisation. C'est à dire que nous n'allons plus chercher à minimiser le nombre de caméra, mais à maximiser le nombre de case surveillé, ce qui équivaut à la même solution. En effet en réduisant le nombre de caméra, on maximise le nombre de case surveillé avec un petit nombre de caméra.

Il n'y a donc pas de contrainte *CSP* à écrire. Il suffit de donner dire au programme de trouver la solution avec le plus de case vide possible.

8.2 Partie implémentation

8.2.1 Parser

La première chose à faire pour résoudre le problème de minimisation du nombre de caméra dans un musée, et de donner au programme l'architecture du musée. C'est à dire donner au programme une carte représentant les murs et les trous à surveiller / combler par des caméras.

Pour ce faire le programme prend en paramètre d'éxécution un fichier texte sous le format suivant :

Une fois que le programme récupère ce fichier, il va récupérer tout d'abord les dimensions du musée sans compter les quatres mur de la pièce. Une fois terminé, ces valeurs sont donnée aux attributs du programme. Une matrice *grid* est ensuite crée. Cette matrice représente la situation de la pièce du musée avec les murs, et permettra ainsi au programme de placer les caméras aux bons endroits.

8.2.2 Placements des caméras

Maintenant que la matrice *grid* à été initialiser, nous pouvons appliquer nos contraintes en fonction de chaque case de cette matrice. Nous faisons alors appel à *minCamera()*, qui va parcourirs chaque case de notre matrice. Si la case est différent de "*", qui correspond à un mur, alors nous allons appliquer des contraintes. Pour ce faire nous initialisons un tableau qui va contenir toutes nos contraintes, ce tableau est *contrainteTotal_OR*. C'est ce tableau qui va permettre d'indiquer au solver, si une case est vide, alors elle est surveillé par au moins une caméra. Ensuite pour la case vide, il va falloir dire : il existe au moins une caméra Sud dans les cases au dessus de notre case vide, ou alors une caméra Nord dans les cases en dessous, ou alors une caméra Est dans les cases à gauche, ou une caméra Ouest dans les cases à droite. Pour ce faire on crée à nouveau un tableau (*existsCam_OR*) qui va contenir ces contraintes. On parcours alors en croix les cases adjacentes à la case vide, en prenant soin de nous arrêter lorsque nous rencontrons un mur. Lors de ce parcours, nous ajoutons à notre tableau la contrainte arithmétique : case parcouru = caméra visant notre case vide. Une fois le parcours en croix terminé, il suffit d'ajouter au tableau *contrainteTotal_OR* quatre contraintes indiquant que la case que nous examinons peut aussi être remplit par une caméra. Il ne reste plus qu'à poster les contraintes du tableau grâce à un *post()* de notre tableau.

8.2.3 Affichage de la solution

Pour afficher la solution, il ne reste plus qu'à parcourirs le tableau contenant les différentes positions des caméras utilisé par le solver (*this.salle*). Ce tableau contient la structure de la salle, les cases vides, et le entiers correspondant au caméra et leur sens.

8.2.4 Exemple