Informatique Fondamentale, SAT Solvers

Nicolas Feron, Kishiro Nishio

Mai 2017

1 Introduction

Voici le rapport du projet d'informatique fondamentale sur les problèmes de satisfaction de contraintes. Il s'agit de problèmes seulement identifiés par des contraintes à satisfaire. Toutes les variables définissant ces contraintes utilisées lors de ce projet sont de type booléen pour avoir des comparaisons et sommes de de variables aisées et optimisées. Par contre, pour avoir des variables de type booléen, il est nécessaire dans certains cas de rajouter une dimension (ce qui augmente le nombre de boucle "for" dans le code source).

La librairie utilisée est Choco Solver.

2 Modèles

2.1 Variables

Nous ne traitons que des entiers donc i $\leq$ n $\longrightarrow$ i $\in$ N $\wedge$ i $\leq$ n
n = taille du tableau
k1 = nombre de tours
k2 = nombre de fous
k3 = nombre de cavaliers

Si ce n'est pas précisé:
les lignes sont itérées sur i, k $\leq$ n
les colonnes sur j, l $\leq$ n
la valeur de la case sur V = {T,F,C} et v $\in$ V
le nombre de pièce k $\in$ {k1,k2,k3} clarifie fortement les notations.

2.2 Question 1: Indépendance

Modèle composé de plusieurs petites contraintes, évidemment, elles peuvent être mise sous forme d'une grosse contrainte séparée par des 'et' mais cela rendrait la lecture désagréable.

1. Unicité:
$$\sum_i \sum_j \sum_v X_{i,j,v} \leq 1$$
Cette contrainte peut aussi être exprimée avec une forme:
$$T \longrightarrow \neg(F \vee C)$$

2. Chaque pièce est présente le nombre de fois requis:
Pour chaque valeur v $\in$ (T,F,C) on somme les variables en position (i,j)
rappel: on a des booléens
$$\sum_i \sum_j \wedge_{v/k} X_{i,j,v} \leq k$$

3. indépendance des tours:
Si il y a une tour, aucune case menacée n'est occupée
$$k \neq i \wedge l \neq j$$

$$\wedge_{i,j} \neg X i, j, T \vee_{k,l,v} (\neg X_{k,j,v} \wedge \neg X_{i,l,v})$$

4. indépendance des fous:
   Si il y a un fou, aucune case menacée n'est occupée

$$(i,j) \neq (k,l) \wedge |i-k| = |j-l|$$
$$\wedge_{i,j} \neg X_{i,j,F} \vee_{k,l,v} \neg X_{k,l,v}$$

5. indépendance des cavaliers:
   Si il y a un cavalier, aucune case menacée n'est occupée

$$k \in \{i-2, i-1, i+1, i+2\} \wedge 0 \leq k < n$$
$$l \in \{j-2, j-1, j+1, j+2\} \wedge 0 \leq l < n$$
$$|i-k| + |j-l| = 3$$
$$\wedge_{i,j} \neg X_{i,j,C} \vee_{k,l,v} \neg X_{k,l,v}$$

2.3 Question 2: Domination

En plus des contraintes d'unicité et pièces requises, nous avons une grosse contrainte avec ses termes séparés par des "ou".

1. Unicité:
$$\sum_i \sum_j \sum_v X_{i,j,v} \leq 1$$
   Cette contrainte peut aussi être exprimée avec une forme:
$$T \longrightarrow \neg(F \vee C)$$

2. Chaque pièce est présente le nombre de fois requis:
   Pour chaque valeur v ∈ (T,F,C) on somme les variables en position (i,j)
   rappel: on a des booléens
$$\sum_i \sum_j \wedge_{v/k} X_{i,j,v} \leq k$$

3. Domination:
   Les domaines de k et l sont les mêmes que pour les indépendances, respectivement tour, fou puis cavalier
$$\wedge_{i,j,v} X_{i,j,v} \vee_{k,l} (X_{k,j,T} \vee X_{i,l,T}) \vee_{k,l} X_{k,l,F} \vee_{k,l} X_{k,l,C}$$

Pour simplifier le problème, dans le code source, nous avons décidé de les transformer en plusieurs contraintes composées uniquement de négations et "et". Nous créons une arrayList de BoolVar et y mettons ces variables pour chaque case (donc tous les éléments séparés par des "or"). Ensuite on cast cette liste dynamique en un vecteur de même taille, en passant, on les inverse (négation). Et enfin, on cast ce vecteur en une contrainte dont on prend l'opposé et la post dans le problème.
En effet,

$$A \vee B \vee C$$

peut s'écrire

$$\neg(\neg A \wedge \neg B \wedge \neg C)$$

On aurait aussi pu utiliser la méthode Constraint.merge() (qui permet un code plus lisible) mais nous ne l'avions pas vu dans la javadoc à ce stade.

4. update Domination:
Nous avons eu vent vers la fin du projet, que les cases occupées ne sont pas "transparentes". Pour être plus clair, une tour ne menace plus la fin d'une ligne si il y a déjà une pièce sur son chemin (nous pensions qu' ajouter cette difficulté des obstacles était l'intérêt de la question 5).

$$v0 \in V$$

Domaines pour les tours

$$k \neq i \wedge l \neq j$$

$$k = i \wedge l < j \longrightarrow m = k \wedge 0 \leq n < l$$

$$k = i \wedge l > j \longrightarrow m = k \wedge l < n < largeur$$

$$k < i \wedge l = j \longrightarrow n = l \wedge 0 \leq m < k$$

$$k > i \wedge l = j \longrightarrow n = l \wedge k < m < longueur$$

Domaines pour les fous

$$(i,j) \neq (k,l) \neq (k,l) \wedge |i-k| = |j-l| = |m-n|$$

$$i < m \wedge j < n \longrightarrow 0 \leq m < i \wedge 0 \leq n < j$$

$$i < m \wedge j > n \longrightarrow 0 \leq m < i \wedge j < n \leq largeur$$

$$i > m \wedge j < n \longrightarrow i < m \leq longueur \wedge 0 \leq n < j$$

$$i > m \wedge j > n \longrightarrow i < m \leq longueur \wedge j < n \leq largeur$$

Rien ne change au niveau des cavaliers car ils sautent au dessus des pièces

Contrainte

$$\wedge_{i,j,v} X_{i,j,v} \vee_{k,l} ((X_{k,j,T} \vee X_{i,l,T}) \wedge_{m,n,v0} \neg X_{m,n,v0})$$

$$\vee_{k,l}(X_{k,l,F} \wedge_{m0,n0} \neg X_{m0,n0,v0}) \vee_{k,l} X_{k,l,C}$$

2.4 Question 4: Domination des cavaliers en minimisant leur nombre

Cette fois-ci, nous n'avons plus besoin des contraintes d'unicité et nombre de pièces. Cependant, en plus du tableau de BoolVar contenant nos variables, nous avons aussi une variable entière IntVar qui est la somme de nos BoolVar (et donc le nombre de cavaliers).

1. Domination par cavalier:

$$k \in \{i-2, i-1, i+1, i+2\} \wedge 0 \leq k < n$$

$$l \in \{j-2, j-1, j+1, j+2\} \wedge 0 \leq l < n$$

$$|i-k| + |j-l| = 3$$

$$\wedge_{i,j} X_{i,j} \vee_{k,l} X_{k,l}$$

2. nombre de cavaliers doit être minimisé:

$$min \sum_{i,j} X_{i,j}$$

   Ceci est facilement mis en code source par la ligne:

```
model.sum(variables, "=", tot_knights).post();
```

   Ensuite pour minimiser ce nombre, il faut demander au solver de fournir la solution optimale en précisant si on veut minimiser ou maximiser la variable donnée:

```
Solution sol = model.getSolver()\
       .findOptimalSolution(tot_knights, Model.MINIMIZE);
```

2.5 Question 5: Musée

La question peut-être ramenée à une domination de tour (à menace uni-directionnelle), ce seront les caméras. En ajoutant la difficulté des obstacles, murs et autres caméras qui obstruent la vision (menace).

2.5.1 Variables

- i est toujours un entier $\wedge$ 0 $\leq$ i $<$ longueur du musée
- j est toujours un entier $\wedge$ 0 $\leq$ j $<$ largeur du musée
- V = {O,E,S,W,N} respectivement objet, est, sud, ouest, nord
- v $\in$ V

2.5.2 Contraintes

1. Ajout des murs:
   Il faut forcer la valeur des murs à 1 (après avoir parsé un fichier indiquant leurs positions).
   $$\wedge_{i,j} mur(i,j) \longrightarrow X_{i,j}$$
2. Unicité:
   $$\sum_i \sum_j \sum_v X_{i,j,v} \leq 1$$
3. nombre de caméras à minimiser:
   On regarde tout sauf les v = O,
   $$v \in V/\{O\}$$
   $$min \sum_{i,j,v} X_{i,j,v}$$
4. Chaque case est occupée par une caméra ou "menacée" par une caméra:
   Lorsqu'on regarde dans une direction, on ne regarde que les caméras qui nous intéressent ainsi que si un obstacle obstrue la vision.
   $$k = i \wedge l < j \longrightarrow v0 = E \wedge m = k \wedge 0 \leq n < l$$
   $$k = i \wedge l > j \longrightarrow v0 = W \wedge m = k \wedge l < n < largeur$$
   $$k < i \wedge l = j \longrightarrow v0 = N \wedge n = l \wedge 0 \leq m < k$$

$$k > i \wedge l = j \longrightarrow v0 = S \wedge n = l \wedge k < m < longeur$$

On ingore les murs pour le premier terme de la contrainte

$$v \in V/\{O\}$$

$$v1 \in V$$

$$\wedge_{i,j,v} X_{i,j,v} \vee_{k,l} (X_{k,l,v0} \wedge_{m,n} \neg X_{m,n,v1})$$

2.5.3 Implémentation

Au niveau de l'implémentation, nous avons une grande méthode ajoutant la dernière contrainte présentée. Cette dernière pourrait être divisée en quatre si on préfère avoir des méthodes plus petites (mais dans ce cas-ci, elles ont besoin de beaucoup d'arguments).
Cette méthode jongle avec des tableaux de BoolVar qu'on ajoute par la suite en une contrainte comme présenté dans la domination. On utilise ces tableaux pour trouver les obstacles entre une caméra et la case actuelle.
Dans le reste des cas, on utilise la fonction Constraint.merge() pour les mettre en une et transformer des "or" en négations et "et" comme présenté précédemment. Enfin après cette fusion, la constrainte est posté, cette opération est effectuée une fois par case.

3 Implémentation

3.1 Résumé des classes

3.1.1 Echiquier

abstrait

- ChessPiece: représente une pièce quelconque
  Deux méthodes doivent être surchargées: void addToBoard(Board) et String toString()

concret

- Rook: représente une tour (extends ChessPiece)
- Bishop: représente un fou (extends ChessPiece)
- Knight: représente un cavalier (extends ChessPiece)
- Board: représente le tableau de jeu
- CSP: classe statique contenant toutes les méthodes de résolution CSP
- KnightsToDominate: contient le main du programme de minimisation de cavaliers
- Main: contient le main du programme indépendance/domination

3.1.2 Musée

abstrait

- MuseumObject: représente un objet quelconque placé dans le musée
  Trois méthodes doivent être surchargées: void addToMuseum(Museum), char getValue() et String toString()

concret

- Camera: représente une caméra (extends MuseumObject)
- Obstacle: représente un obstacle (extends MuseumObject)
- Museum: représente un musée
- Parser: contient le parser de fichier pour importer un musée (les murs)
- CSP: classe statique contenant toutes les méthodes de résolution CSP
- WatchMuseum: contient le main du programme

3.2 Choix sur les variables

Au début, il y a eu une hésitation pour ce qui était d'ajouter la valeur "menacé" ("surveillé" pour le musée), dans la dimension de v. Cela aurait permis de mettre des contraintes plus courtes. Par manque de temps, il nous a été impossible de tester les deux pour voir lequel est le plus optimisé et dans quel cas il l'est.