Programmation par Contraintes

Introduction

Thi-Bich-Hanh Dao

LIFO - Université d'Orléans

Le module Contraintes

Organisation

- CM : 15h
- TD : 15h

Les intervenants

- Thi-Bich-Hanh Dao (thi-bich-hanh.dao@univ-orleans.fr)
- Denys Duchier (denys.duchier@univ-orleans.fr)

Evaluation

Contrôle continu

- Projet à réaliser
- L'absence : 0

Contrôle terminal

- Contrôle à la fin du semestre
- L'absence au contrôle terminal : ABI

Note finale

- Note finale = moyenne pondérée des CC et du CT

Objectifs du cours

- Comprendre les techniques de la programmation par contraintes
- Identifier des classes de problèmes où la programmation par contraintes est bénéfique
- Modéliser des simples problèmes sous forme de contraintes
- Exprimer des problèmes dans un langage de programmation par contraintes

Références

- K. Apt, Principles of Constraint Programming
- R. Dechter, Constraint Programming
- F. Rossi, P. van Beek, T. Walsh, Handbook of Constraint Programming

Sudoku

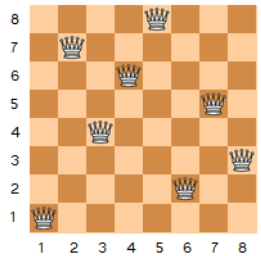
			2		7	8		
							4	
					5		6	2
		2		4			1	
				6				4
9		1			8		3	
		3				7		
	5	6		9				
	2					5		

- Remplir les cases avec les entiers de 1 à 9
- *Contraintes* : chaque entier apparaît une seule fois dans
  - chaque ligne
  - chaque colonne
  - chaque région

Comment le résoudre ?

Les reines

- Placer 8 reines sur un échiquier
- *Contraintes* : pas d'attaque entre les reines, c-à-d. une seule reine sur
  - chaque ligne
  - chaque colonne
  - chaque ligne diagonale
- Une solution :



Comment le résoudre ?

Problème de satisfaction de contraintes

- Ces problèmes sont modélisés d'une façon très simple comme un problème de satisfaction de contraintes (Constraint Satisfaction Problem - CSP)
  - Un ensemble de variables $X = \{X_1, \ldots, X_n\}$,
  - Un ensemble de domaines $D = \{D_1, \ldots, D_n\}$, chaque domaine D_i définit l'ensemble de valeurs que peut prendre la variable X_i ($X_i \in D_i$)
  - Un ensemble de contraintes $C = \{C_1, \ldots, C_n\}$
- Une *solution* est une affectation d'une valeur à chaque variable, telle que toutes les contraintes sont satisfaites

Les reines comme un CSP

Les variables

- puisqu'on ne peut avoir plus qu'une reine sur une même ligne, on déduit que sur chaque ligne il y a une seule reine
- on cherche donc la colonne pour la reine de la ligne 1, pour la reine de la ligne 2, etc.
- on utilise donc une variable X_i pour la reine de la ligne i, avec l'échiquier on utilise 8 variables $X_1, \ldots, X_8$
- ces variables ont le même domaine – l'ensemble des numéros de colonnes $= \{1, \ldots, 8\}$

Les reines comme un CSP

Les contraintes

- pas de deux reines sur la même colonne :

$$\begin{array}{llll} X_1 \neq X_2, & X_1 \neq X_3, & \ldots & X_1 \neq X_8 \\ & X_2 \neq X_3, & \ldots & X_2 \neq X_8 \\ & & \ldots & \\ & & & X_7 \neq X_8 \end{array}$$

- pas de deux reines sur la même ligne diagonale :

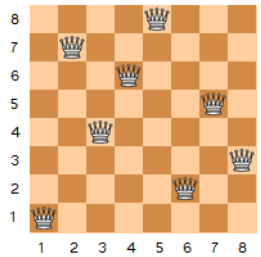
$$\begin{array}{llll} |X_1 - X_2| \neq |1-2|, & |X_1 - X_3| \neq |1-3|, & \ldots & |X_1 - X_8| \neq |1-8| \\ & |X_2 - X_3| \neq |2-3|, & \ldots & |X_2 - X_8| \neq |2-8| \\ & & \ldots & \\ & & & |X_7 - X_8| \neq |7-8| \end{array}$$

Les reines comme un CSP

Les solutions

- Une solution = une affectation de valeur à chaque variable X_i, telle que les contraintes sont satisfaites
- Exemple :

$$X_1 = 1,\ X_2 = 6,\ X_3 = 8,\ X_4 = 3,\ X_5 = 7,\ X_6 = 4,\ X_7 = 2,\ X_8 = 5$$



Problème d'optimisation sous contraintes

- Pour plusieurs problèmes, en même temps que les contraintes, on souhaite optimiser une valeur, ex.
  - affectation de tâches aux machines en minimisant le temps d'exécution,
  - choix d'investissements en maximisant le gain,
  - ...
- Problèmes d'optimisation sous contraintes (Constrained Optimization Problem – COP) :
  - un problème CSP
  - et une fonction objective à optimiser (minimiser/maximiser)

Résoudre un CSP/COP

- CSP/COP : complexité NP-Difficile ou pire ...
  - ex. : problème de coloration de graphe formulé par un CSP
  - coloration de graphe est NP-Complet
  - CSP est NP-Difficile
- ... et en général pas d'approximation raisonnable
- Méthodes :
  - algorithmes spécifiques pour des problèmes spécifiques
  - programmation linéaire
  - recherche locale
  - **programmation par contraintes**

Principes de la programmation par contraintes

- Le programmeur spécifie le problème sous forme d'un CSP
- Le solveur cherche une (des) solution(s) du problème
  - à priori le solveur peut trouver des (les) solutions tout seul
  - mais pour le propulser le programmeur peut : indiquer une stratégie de recherche, ajouter des contraintes redondantes mais utiles, ...

Programmation par contraintes

- Modélisation : formaliser le problème sous forme d'un CSP/COP
  - identifier les variables et leur domaine
  - exprimer les contraintes
- Résolution : utiliser des solveurs ou librairies
  - implanter les contraintes en utilisant des librairies
  - spécifier au solveur la méthode de recherche utilisée
  - indiquer des heuristiques

Principe de résolution

Combinaison/itération de 2 étapes

- Propagation de contraintes
  - réduire le domaine des variables
  - détecter des combinaisons impossibles
- Enumération
  - instancier une variable ou découper le domaine d'une variable
  - créer des sous-cas à explorer

Procédure de résolution

```
def Solve():
    while not Happy() :
        Pre_Process()
        Constraint_Propagation()
        if not Happy():
            Split()
            Process_By_Case()
```


Les procédures internes

- `Pre_Process()` : mettre les contraintes en une forme attendue
- `Happy()` : condition d'arrêt
  - lorsqu'une solution est trouvée
  - lorsque toutes les solutions sont trouvées
  - lorsqu'il n'y a pas/plus de solution
  - lorsqu'il n'y a pas de meilleure solution
- `Constraint_Propagation()` : propagation de contraintes
- `Split()` : découper le domaine d'une variable ou découper une contrainte, créer les sous-cas
- `Process_By_Case()` : ordonner les cas créé par `Split()` et appeller `Solve()` pour explorer chacun des cas

Applications

- Gestion du temps : emploi du temps, planification d'équipes, de rotation d'équipes
- Gestion et affectation de ressources : gestion de personnels, de moyens de transports, gestion de production
- Planification et ordonnancement : planification de production, de livraison, de maintenance, d'itinéraires, ordonnancement d'ateliers
- Configuration : logiciels, ordinateurs, voitures, ...
- Finance : gestion des options, optimisation de placements financiers
- Traitement de langage naturel : construction d'analyseur efficace
- Biologie moléculaire : séquences ADN, construction de models 3D de protein
- Systèmes graphiques interactifs : cohérence géométrique en analyse de scène
- Ingénierie électronique : diagnostic de pannes, test et vérification de circuits

Applications industrielles (1)

- ELF : planification de production de raffinerie
- EDF : planification et ordonnancement de transport de déchet nucléair
- CDC : gestion de portefeuille
- Rhone-Poulenc : aide à la conception de produits équilibrés
- SNCF : planification de maintenance de locomotives
- Renault : planification de livraison de pièces détachées, configuration de véhicules industriels
- Dassault Aviation : ordonnancement de ligne d'assemblage, de production
- Militaire : optimisation de plan de fréquence, planification de carrière, affectation de régions opérationnelles, planification d'escadrons mobiles

Applications industrielles (2)

- Luftansa, Air France : planification de roulements d'équipages
- Chrysler, Ford : problèmes de configuration
- Whirlpool : planification de production
- Port de Hong-Kong : associer porte-conteneurs aux terminaux
- National Westminster Bank : planification et allocation de formation des personnels
- Banque Bruxelles Lambert : gestion des employés sur l'année entière
- Siemens : placement et routage de VLSI
- Télécommunication : allocation de fréquences

Solveurs de PPC

- *Gecode* (librairie C++)
- Choco (librairie Java)
- JChoco, Koalog (librairie Java)
- Ilog Solver (librairie C++)
- Oz (langage)
- Sictus Prolog, Gnu Prolog, Eclipse (langage)
- Numérica (langage de modélisation, domaine réel)
- ...

Langages de modélisation

- Zinc
- *MiniZinc*
- Essence
- ...

Les reines en MiniZinc

```
include "globals.mzn";

int: n;
array [1..n] of var 1..n: queens;

% queens are in different columns
constraint all_different(queens);

% queens are in different diagonals
constraint forall(i,j in 1..n where i<j) (
    queens[i] + i != queens[j] + j /\
    queens[i] - i != queens[j] - j
);

solve satisfy;

output [show(queens)];
```