UQAC
Université du Québec
à Chicoutimi

Résumé de l'article : Programmation par contraintes sur les séquences infinies

Session : Automne 2014

Intelligence Artificielle
(8INF846)

Département d'informatique et de mathématiques

Présenté à : Professeur: Popescu, Ilie

Travail de (Groupe 5) : *Valentin Corre*
Triki Zied
Houssem Amamou

Date de remise : 31/10/2014

1. Introduction

La programmation par contraintes permet de résoudre des problèmes combinatoires de grandes tailles d'une manière générique et de faire des affectations de valeurs aux variables qui satisfont toutes les contraintes. Cette approche, et en raison de son succès, couvre actuellement d'autres domaines, en outre l'allocation de ressources ou de raisonnement spatial et temporel, comme la fouille de données ou les logiques temporelles qui sont le sujet de cet article.

Malgré la diversité des domaines d'application, il reste difficile de modéliser des problèmes dont la valeur des variables varie au cours du temps et pour lesquels l'objectif est simplement de satisfaire des contraintes temporelles, et ce de façon indéfinie sur la durée. La programmation par contraintes utilisent en général une résolution itérative, qui consiste à allonger le temps imparti jusqu'à ce qu'une solution soit trouvée.

Nous présentons dans cet article un formalisme pour traiter des problèmes temporels en utilisant cette approche à travers des applications qui couvrent différents domaines.

2. Problème de satisfaction de contraintes

Un problème de satisfaction de contraintes est composé d'un tuple à trois ensembles : un ensemble fini de variables, un ensemble de domaines associés à ces variables et un ensemble fini de contraintes. Une solution d'une CSP doit satisfaire toutes les contraintes. Comme les domaines des variables sont de cardinalité finies, alors le nombre de solution est fini. Grâce aux propagateurs, on peut réduire l'espace de recherche en identifiant les régions infructueuses cependant dans notre problème l'espace de recherche est infini. On peut alors utiliser des structures de données qui générer l'ensemble des solutions.

3. StCSPs

Nous allons traiter des problèmes de satisfactions de contraintes sur flux. Un flux est une séquence finie ou infinie de datons. L'operateur « Next » et « (.) » permettent de symboliser des parties d'un flux. Ce papier introduit aussi la notion de variable flux ainsi que son domaine de définition.

Les termes sont construits à partir des variable flux et l'opérateur « Next ». L'ensemble des termes peuvent être construit à partir d'une variable flux ou un autre terme.

Il est cependant difficile de représenter l'ensemble des affectations qui satisfont l'ensemble des contraintes autrement que par la définition de la contrainte elle-même. Il s'agit d'une contrainte globale au sens des flux, c'est-à-dire qui porte sur l'ensemble des datons de chaque terme. Les contraintes de voisinage sont des contraintes appliquées localement sur les flux. Comme l'ensemble des datons peut être infini, il est impossible de le contraindre par des applications locales, d'où l'utilisation d'un mécanisme de réplication.

Une solution à une StCSP est une affectation qui satisfait toutes les contraintes. On définit la notion de temporalité qui est le plus grand nombre d'itérations de l'opérateur « Next » parmi les termes figurant dans le problème. Un k-StCSP est un StCSP de temporalité k

4. Représentation de l'ensemble des solutions

Nous allons présenter une méthode permettant de retrouver les solutions des StCSP de temporalité 1 puis nous allons mettre une avant une méthode qui transforme des StCSP de grande taille en 1-StCSP.

Les contraintes d'un 1-StCSP peuvent être de trois formes

- Contraintes point-à-point : c'est une contraintes dans laquelle tous les termes ont une temporalité 0.
- Contraintes mixtes : c'est une contrainte de transition. Elle contient des termes de temporalité 1 et de temporalité 0.
- Contraintes singulières : c'est une contrainte qui ne contient que des termes de temporalité 1.

Un 1-StCSP qui ne contient que des contraintes de voisinage peut être transformé en un système de transitions.

Une solution de ce système est un ensemble de tuples α. Nous notons

$$\tau[\alpha] = (\alpha_1(0), \alpha_2(0), \ldots, \alpha_n(0))\tau[\mathrm{Next}(\alpha)]$$

Une solution S correspond alors à l'exécution d'un système de transitions. L'ensemble des états est l'ensemble des combinaisons qui satisfont toutes les contraintes point-à-point :

$$Q = \{\tau \in d(X_1) \times \cdots \times d(X_n) \mid \forall C \in \mathcal{C}^0, \tau \vDash C\}$$

La relation de transition est construite avec les contraintes mixtes :

$$\delta = \{(\tau_0, \tau_1) \in Q \times Q \mid \forall C \in \mathcal{C}^1, (\tau_0, \tau_1) \vDash C\}$$

Il s'ensuit que toutes les traces d'exécution du système de transitions T (S) associé à un StCSP S de temporalité 1 correspondent = des solutions de S, car, par construction, les contraintes point-à-point et les contraintes mixtes de S sont satisfaites.

L'ensemble de solution d'un 1-StCSP ne comporte que des contraintes de voisinage qui est simple à calculer. Ceci est prouvé car les solutions de P(S) correspondent à l'ensemble des transitions de T (S) parce que les contraintes point-à-point sont satisfaites à chaque instant, et les contraintes mixtes sont satisfaites pour chaque transition. D'où la possibilité de calculer le

système de transition associé à un 1-StCSP en résolvant le problème de satisfaction de contraintes P(S) correspondant.

5. Réduction de temporalité

Cette partie de l'article explique comment transformer un StCSP de temporalité supérieur à 1 en un 1-StCSP équivalent. Le principe est d'ajouter de la mémoire au StCSP en utilisant des variables qui contiennent les valeurs des variables aux instants futurs. Ces variables sont reliées entre elles par des contraintes d'égalité. Ici, le système peut contenir de l'information sur les valeurs futures.

Soit Nextk(X) de temporalité k. Soient Xk-1 une nouvelle variable du domaine D(Xk-1) = D(X), et Xk-1 = Nextk-1(X) une nouvelle contrainte d'égalité. Ainsi on peut remplacer partout Nextk(X) par

Next(Xk-1) et donc la temporalité est donc maintenant k-1. Puisque la temporalité est finie, on va forcément arrivé à 1 en répétant cette opération k-1 fois.

Système de transitions étiqueté associé à un k-StCSP

Un système de transitions étiqueté est un tuple TE = (Q, D, E, L) tel que :

- Q est un ensemble d'état
- Q x Q contient d, d est la relation de transitions
- E est un ensemble d'étiquettes
- L : Q -> E est une fonction d'étiquetage qui associe à chaque état une étiquette

Les étiquettes correspondent à la projection de Q sur l'ensemble des variables de S, et la fonction d'étiquetage effectue cette opération.

6. Exemple

Le carrefour

Le but de l'exemple est de synchroniser des feux de circulations à un carrefour. Dans l'exemple on ne traite que de deux feux.

On définit :

- $S_E = \{X_E, D_E, C_E\}$
- $X_E = \{X, Y\}$, X et Y représente deux feux.
- $D_E = D(X) = D(Y) = \{V, O, R\}$, V représente le couleur verte du feu, O la couleur orange et R la couleur rouge.
- C_E représente les contraintes qui permettent de modéliser les états autorisés pour le système et son évolution au cours du temps.
- R_C représente les états autorisés du système :

R_C	
V	O
V	R
O	V
R	V
O	O
R	R

Si on utilise plusieurs feux (plus de 2), il faut prendre chaque paire de feux et leur appliquer la relation RC.

- C_C : RC(X, Y) représente l'application de la relation RC aux feux X et Y.
- R_{Ev} représente la relation qui caractérise l'évolution des feux :

R_{Ev}	
V	O
O	R
R	R
R	V

- E_X : Rev(X,Next(X)) et EY : Rev(Y,Next(Y)) représentent les contraintes mixtes.

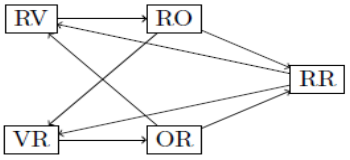
Puisque on utilise seulement $Next^k()$ avec k = 1, il est inutile d'appliquer la procédure de réduction de temporalité (5)).

Finalement, Les variables du CSP sont X = {X0, Y0, X1, Y1}

Les contraintes de compatibilité sont C0 : RC(X0, Y0)
C1 : RC(X1, Y1)

Les contraintes mixtes sont EX : REV(X0, X1)
EY : REV(Y0, Y1)

Solutions : Exemple d'enchaînement possible pour les feux :



X	V	O	R	R	V	O	R	...
Y	R	R	V	O	R	R	R	...

Production automobile

Dans cet exemple, une production automobile doit produire une infinité de voitures sans jamais surcharger les machines qui installent les options. Les voitures sont triées en différentes classes en fonction des options qu'elles utilisent. Les options sont indépendantes les unes des autres et sont modélisées par des variables binaires.

Certaines options sont plus difficiles à installer que d'autres. Cette notion de difficulté est encodée par une capacité k/n, qui exprime que pour n voitures consécutives, au plus k voitures pourront recevoir l'option.

On considère 6 classes de voiture {c1, c2 … c6} qui sont des combinaisons de 5 options {o1, o2, … o6).

c_1	o_1	$\neg o_2$	o_3	o_4	$\neg o_5$
c_2	$\neg o_1$	$\neg o_2$	$\neg o_3$	o_4	$\neg o_5$
c_3	$\neg o_1$	o_2	$\neg o_3$	$\neg o_4$	o_5
c_4	$\neg o_1$	o_2	$\neg o_3$	o_4	$\neg o_5$
c_5	o_1	$\neg o_2$	o_3	$\neg o_4$	$\neg o_5$
c_6	o_1	o_2	$\neg o_3$	$\neg o_4$	$\neg o_5$

La capacité pour chaque option est :

o_1	o_2	o_3	o_4	o_5
1/2	2/3	1/3	2/5	1/5

On définit :

- SV = (X, D, C)
- O1, O2, O3, O4, O5 qui représente les options au moment présent.

La combinaison (O_1, O_2, O_3, O_4, O_5) doit correspondre à une des classes définies.

- (O_1;O_2;O_3;O_4;O_5) appartient {
  (1; 0; 1; 1; 0);
  (0; 0; 0; 1; 0);
  (0; 1; 0; 0; 1);
  (0; 1; 0; 1; 0);
  (1; 0; 1; 0; 0);
  (1; 1; 0; 0; 0)}
- C^1: (O^1 + Next (O^1) <= 1)
- C^2: (O^2 + Next (O^2) + Next² (O_2)) <= 2)
- C^3: (O^3 + Next (O^3) + Next² (O_3) <= 1)
- C^4: (O^4 + Next (O^4) + Next² (O_4) + Next^3 (O_4) + Next^4 (O_4) <= 2)
- C^5: (O^5 + Next (O^5) + Next² (O_5) + Next^3 (O_5) + Next^4 (O_5) <= 1)
- CC : {C1, C2, C3, C4, C5} représente les contraintes de capacité

Ce problème est de temporalité 4 (Next^4()). Il faut donc utiliser le principe de réduction de temporalité. Après réduction, le problème contient 8 variables supplémentaires et donc le CSP a 26 variables. Le système de transition correspondant est suffisamment grand pour ne pas être inclus dans la page.

7. Recherche incrémentale

Il n'est pas nécessaire de calculer le système de transition en entier. Grâce à un algorithme, il est possible grâce à quelques transitions d'en déduire le système en entier. Voici cet algorithme :

Algorithme : Recherche incrémentale

$H \leftarrow \emptyset$
Pour chaque solution (τ_1, τ_2) de $\mathcal{P}$
  ajouter τ_1 et τ_2 à H
  $\tau \leftarrow$résoudre($\mathcal{P}$,τ_2)
  Si $\tau \neq \perp$
    afficher τ_2
    afficher τ_1
    afficher τ
    sortir
  Sinon
    enlever τ_1 et τ_2 de H
  Fin Si
Fin Pour

Procédure résoudre($\mathcal{P}, \tau_1$)
  Pour chaque solution τ_2 de $\mathcal{P}[\tau_1]$
    Si $\tau_2 \in H$
      renvoyer τ_2
    Sinon
      ajouter τ_2 à H
      $\tau \leftarrow$résoudre($\mathcal{P}$,τ_2)
      Si $\tau \neq \perp$
        afficher τ_2
        renvoyer τ
      Fin Si
      supprimer τ_2 de H
    Fin Si
  Fin Pour
  renvoyer $\perp$
Fin Procédure

FIGURE 8 – Résolution incrémentale d'un StCSP

8. Conclusion

Nous avons présenté dans cet article de nombreux formalismes concernant le raisonnement temporel en s'appuyant sur les méthodes habituellement utilisées pour la satisfaction de formules de logiques temporelles, qui ne sont pas basées sur la programmation par contraintes et celles basées sur cette approche pour la modélisation de problèmes temporels sans horizon.

Enfin, beaucoup de problèmes restent ouverts. Les possibilités concernant la modélisation restent à explorer, et les possibilités d'applications sont encore à l'étude, mais semblent nombreuses.