Contraintes pour l'aide à la décision

Propagation de contraintes

Thi-Bich-Hanh Dao

LIFO - Université d'Orléans

Ressources

- Handbook of Constraint Programming, Chapitre 3 : Constraint Propagation, Christian Bessière
- Constraint Propagation and Backtracking-Based Search, Roman Barták
- Constraint Programming, notes de cours, Yves Deville

Contenu

- Propagation de contraintes et consistances
- Consistance de nœud
- Consistance d'arc
  - Algorithme AC-1
  - Algorithme AC-3
  - Algorithme AC-4
  - Consistance d'arc pour contraintes non binaires
- Consistance de borne

Problèmes NP-Difficile

- Exemple : Port maritime, associer navires aux quais pour le chargement/déchargement
  restrictions en temps/coût
- Pour 5 quais et 10 navires, calculer toutes les permutations possibles pour connaître leur temps/coût
  $\Longrightarrow 5^{10}$ alternatives
  sur un ordinateur qui teste 1000 alternatives par seconde, temps de résolution environ 3 heures
- Le port se développe, 10 quais et 20 navires, le même ordinateur résout en 3 milliards années ! ! !
- Plusieurs critères pour le choix de quai pour un navire : capacité de quai/pas de deux navires pour un quai en même temps/etc.
  $\Longrightarrow$ les contraintes permettent de réduire l'espace de recherche, problème plus traitable

CSP

- Un ensemble de variables $X_1, \ldots, X_n$ de domaines $D_1, \ldots, D_n$
- Un ensemble de contraintes $c_1, \ldots, c_m$
  - chaque contrainte c_i porte sur un sous-ensemble de variables $X_{i_1}, \ldots, X_{i_{k_i}}$
  - chaque contrainte c_i est un sous-ensemble de $D_{i_1} \times \cdots \times D_{i_{k_i}}$
- Une solution est un tuple $(d_1, \ldots, d_n) \in D_1 \times \cdots \times D_n$ tel que pour chaque contrainte c_i sur $X_{i_1}, \ldots, X_{i_{k_i}}$

$$(d_{i_1}, \ldots, d_{i_{k_i}}) \in c_i$$

Contraintes

- Contraintes unaires et contraintes binaires (pour le moment)
- Au plus une contrainte binaire sur chaque couple X_i, X_j
- Contrainte unaire sur X_i est représentée par c_i
- Contrainte binaire sur X_i et X_j $(i < j)$ est représentée par c_{ij} et c_{ji}
- Pour chaque contrainte c_{ij} et $u \in D_i$, $v \in D_j$, on a une valeur booléenne $c_{ij}(u, v)$

$$c_{ij}(u, v) = \begin{cases} 1 & \text{si } (u, v) \in c_{ij} \\ 0 & \text{sinon} \end{cases}$$

Complexités

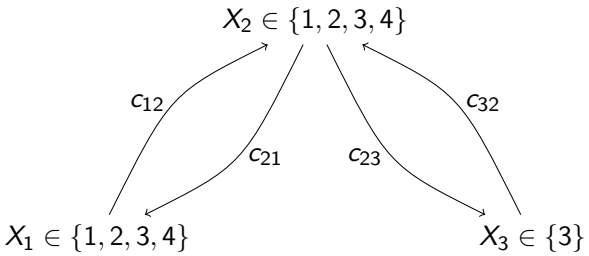
- Nombre de variables : n
- Taille du plus grand domaine : d
- Nombre de contraintes binaires : e
- Supposition : $c_{ij}(u, v)$ est vérifié en temps constant $O(1)$
- Nombre maximum de contraintes binaires : $O(n^2)$
- Complexité en espace des domaines : $O(nd)$

Graphe de contraintes

- Un CSP (contraintes unaires et binaires) peut se représenté par un graphe
  - Nœuds : $X_1, \ldots, X_n$
  - Les arcs :
    - arc (i, i) pour chaque contrainte c_i
    - arcs (i, j) et (j, i) pour chaque contrainte c_{ij} et c_{ji}

Exemple

- CSP :
  - X_1, X_2, X_3, avec $D_1 = D_2 = \{1, 2, 3, 4\}$, $D_3 = \{3\}$
  - contraintes $X_1 \leq X_2$, $X_2 \neq X_3$
- Graphe de contraintes



Propagation de contraintes

- Programmation par contraintes : combinaison de
  - Propagation de contraintes
  - Recherche de solution
- Propagation :
  - Réduire le domaine des variables, sans perdre de solution
  - Utiliser les techniques de consistance
  - Chaque contrainte est considérée individuellement

Consistance de nœud

- Un nœud X_i est nœud-consistant si toute contrainte unaire c_i est satisfaite sur toute valeur $v \in D_i$.
- Consistance de nœud assurée en $O(nd)$

Agorithme NC

pour chaque *variable* X_i **faire**
  pour chaque *valeur* $v \in D_i$ **faire**
    si *une contrainte unaire* c_i *n'est pas satisfaite sur* v **alors**
      supprimer v de D_i

Consistance d'arc

- Une contrainte c_{ij} sur deux variables X_i, X_j est arc-consistante ssi
  - pour chaque valeur $u \in D_i$, il existe une valeur $v \in D_j$ telle que (u, v) satisfait c_{ij}
- Arc-consistance est directionnelle
  - arc (X_i, X_j) consistance ne garantie pas la consistance de l'arc (X_j, X_i)

$X_1, X_2 \in 1..4, X_1 < X_2$, c_{12} arc-consistant mais pas c_{21} (gauche), c_{12} et c_{21} sont arc-consistants (droite)

$$X_1 \in 1..3 \overset{c_{12}}{\underset{c_{21}}{\rightleftarrows}} X_2 \in 1..4 \qquad X_1 \in 1..3 \overset{c_{12}}{\underset{c_{21}}{\rightleftarrows}} X_2 \in 2..4$$

- Un CSP est arc-consistant ssi tous les arcs (dans les deux directions) sont arc-consistants

Révision d'arc

- Assurer l'arc (X_i, X_j) consistant
- REVISE(X_i, X_j) supprime les valeurs inconsistantes du domaine D_i
- Complexité $O(d^2)$

Agorithme REVISE(X_i, X_j)

Deleted $\leftarrow$ false
pour chaque $u \in D_i$ **faire**
  si *il n'existe pas de valeur* $v \in D_j$ *telle que* $(u, v) \in c_{ij}$ **alors**
    supprimer v de D_i
    Deleted $\leftarrow$ true
retourner *Deleted*

Algorithme AC-1

Agorithme AC-1

répéter
  Changed $\leftarrow$ false
  pour chaque *contrainte* c_{ij} **faire**
    Changed $\leftarrow$ REVISE(X_i, X_j) ou Changed
jusqu'à *Changed == false*;

- Complexité AC-1 est $O(end^3)$ ou $O(n^3d^3)$
- pas efficace car même un petit changement d'un domaine entraîne la révision de toutes les contraintes, même pour celles qui ne sont pas concernées

Algorithme AC-3

- Amélioration : utiliser une queue pour mémoriser les arcs dont la consistance peut être mise en cause par une révision
- Complexité en temps $O(ed^3)$ ou $O(n^2d^3)$
- Complexité en espace $O(e)$ ou $O(n^2)$

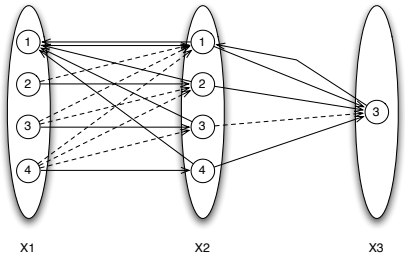
Agorithme AC-3

$Q \leftarrow \emptyset$
pour chaque *arc* c_{ij} **faire** $Q \leftarrow Q \cup \{(i, j)\}$;
tant que $Q \neq \emptyset$ **faire**
  choisir et supprime un arc (k, m) de Q
  si *REVISE*(X_k, X_m) **alors**
    $Q \leftarrow Q \cup \{(i, k) \mid c_{ik}$ est une contrainte, $i \neq m\}$

Exemple (1/3)

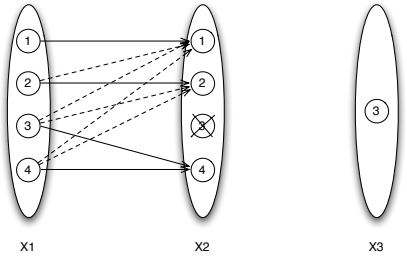
- CSP :
  - X_1, X_2, X_3, avec $D_1 = D_2 = \{1, 2, 3, 4\}$, $D_3 = \{3\}$
  - contraintes $X_1 \leq X_2$, $X_2 \neq X_3$
- Initialisation $Q = \{(1, 2), (2, 1), (2, 3), (3, 2)\}$
- REVISE(1,2) : 1 test pour trouver le support de 1, 2 tests pour trouver le support de 2, etc.
  nécessite au total 1+2+3+4=10 tests pour décider que toutes les valeurs de D_1 sont consistantes
- REVISE(2,1) : nécessite 4 tests pour décider que toutes les valeurs de D_2 sont consistantes
- REVISE(2,3) : nécessite 4 tests et supprime la valeur 3 de D_2
  $\Longrightarrow$ (1,2) est mis dans Q
- REVISE(3,2) : nécessite 1 test pour décider que la seule valeur de D_3 est consistante

Exemple (2/3)



- Avec REVISE(1,2), REVISE(2,1), REVISE(2,3), REVISE(3,2)
- flèche pleine (test succès), flèche pointillée (test échec)
- après ces 4 révisions, $Q = \{(1, 2)\}$

Exemple (3/3)



- REVISE(1,2) : refait 9 tests (dont un seul nouveau) pour décider que toutes les valeurs de D_1 sont consistantes

AC-3 et AC-4

- AC-3
  - n'est pas optimal car REVISE ne se souvient pas de support pour chaque valeur
  - refait les tests même pour les valeurs à priori non concernées
- AC-4 :
  - mémoriser les informations concernant les supports pour chaque valeurs
  - Algorithme optimal en temps
  - Nécessite de maintenir des structures de données pour réduire le nombre de tests à l'intérieur de REVISE

Structures de données de AC-4

Compter le nombre de supports $counter[X_i, u, X_j]$

- $counter[X_i, u, X_j]$: nombre de supports pour la valeur $u \in D_i$ dans le domaine D_j, pour une contrainte c_{ij}

$$counter[X_i, u, X_j] \ = \ \#\{v \in D_j \mid c_{ij}(u, v)\}$$

- Dès que $counter[X_i, u, X_j] = 0$, la valeur u est supprimée du domaine D_i
- Structure de données initialisée dans INITIALIZE et mise à jour dans AC-4
- Taille de structure $O(ed)$ ou $O(n^2 d)$

Structures de données de AC-4

L'ensemble des valeurs de supports $S[X_j, v]$

- $S[X_j, v]$: l'ensemble des valeurs de D_i supportées par la valeur $v \in D_j$

$$S[X_j, v] \ = \ \{u \in D_i \mid c_{ij}(u, v)\}$$

- Cet ensemble permet d'accéder à tous les valeurs supportées lorsque u est supprimée de D_i
- Structure de données *statique* initialisée dans INITIALIZE et *pas* de mise à jour
- Taille de structure $O(ed^2)$ ou $O(n^2d^2)$

Algorithme INITIALIZE

Agorithme INITIALIZE

$Q \leftarrow \emptyset$
pour chaque X_j **faire**
  pour chaque $v \in D_j$ **faire**
    $S[X_j, v] \leftarrow \emptyset$
pour chaque *contrainte* c_{ij} **faire**
  pour chaque $u \in D_i$ **faire**
    $total \leftarrow 0$
    pour chaque $v \in D_j$ **faire**
      si $c_{ij}(u, v)$ **alors**
        $total \leftarrow total + 1$
        $S[X_j, v] \leftarrow S[X_j, v] \cup \{(X_i, u)\}$
    $counter[X_i, u, X_j] \leftarrow total$
    si $counter[X_i, u, X_j] = 0$ **alors**
      supprimer u du domaine D_i
      $Q \leftarrow Q \cup \{(X_i, u)\}$
retourner Q

Algorithme AC-4

Agorithme AC-4

$Q \leftarrow INITIALIZE()$
tant que $Q \neq \emptyset$ **faire**
  choisir et supprime un couple (X_j, v) de Q
  pour chaque $(X_i, u) \in S[X_j, v]$ **faire**
    si $u \in D_i$ **alors**
      $counter[X_i, u, X_j] \leftarrow counter[X_i, u, X_j] - 1$
      si $counter[X_i, u, X_j] = 0$ **alors**
        supprimer u de D_i
        $Q \leftarrow Q \cup \{(X_i, u)\}$

Exemple (1/2)

- CSP :
  - X_1, X_2, X_3, avec $D_1 = D_2 = \{1, 2, 3, 4\}$, $D_3 = \{3\}$
  - contraintes $X_1 \leq X_2$, $X_2 \neq X_3$
- INITIALIZE :

$counter[X_1, 1, X_2] = 4$ $counter[X_1, 2, X_2] = 3$ $counter[X_1, 3, X_2] = 2$ $counter[X_1, 4, X_2] = 1$

$counter[X_2, 1, X_1] = 1$ $counter[X_2, 2, X_1] = 2$ $counter[X_2, 3, X_1] = 3$ $counter[X_2, 4, X_1] = 4$

$counter[X_2, 1, X_3] = 1$ $counter[X_2, 2, X_3] = 1$ $counter[X_2, 3, X_3] = 0$ $counter[X_2, 4, X_3] = 1$

$counter[X_3, 3, X_2] = 3$

$S[X_1, 1] = \{(X_2, 1), (X2_2), (X_2, 3), (X_2, 4)\}$
$S[X_1, 2] = \{(X_2, 2), (X2_3), (X_2, 4)\}$
$S[X_1, 3] = \{(X_2, 3), (X2_4)\}$
$S[X_1, 4] = \{(X_2, 4)\}$

$S[X_2, 1] = \{(X_1, 1), (X_3, 3)\}$
$S[X_2, 2] = \{(X_1, 1), (X_1, 2), (X_3, 3)\}$
$S[X_2, 3] = \{(X_1, 1), (X_1, 2), (X_1, 3)\}$
$S[X_2, 4] = \{(X_1, 1), (X_1, 2), (X_1, 3), (X_1, 4), (X_3, 3)\}$

$S[X_3, 3] = \{(X_2, 1), (X2_2), (X_2, 4)\}$

$Q = \{(X_2, 3)\}$

Exemple (2/2)

- AC-4
  - $(X_2, 3)$ sorti de Q, réduire le nombre de supports de chaque couple dans $S[X_2, 3] = \{(X_1, 1), (X_1, 2), (X_1, 3)\}$

$$counter[X_1, 1, X_2] = 3$$
$$counter[X_1, 2, X_2] = 2$$
$$counter[X_1, 3, X_2] = 1$$

  - aucun counter devient 0, algorithme s'arrête

Complexité AC-4

- Complexité en temps $O(ed^2)$ ou $O(n^2d^2)$
- Complexité en espace $O(ed^2)$ ou $O(n^2d^2)$
- Complexité optimale en temps
- Complexité en espace importante, un autre algorithme AC-6 avec même complexité en temps et une meilleur complexité en espace $O(ed)$ ou $O(n^2d)$

L'arc-consistance est-elle suffisante ?

- Si le CSP est arc-consistant :
  - arrive-t-on à une solution ? Non
  - sait-on si une solution existe ? Non
- L'arc-consistance est utile car :
  - on peut arriver à un cas d'arrêt
    - ⋆ un domaine devient vide : pas de solution
    - ⋆ tous les domaines deviennent singleton : on arrive à une solution
  - et en général l'espace de recherche est réduit

Consistance d'arc pour contraintes non binaire

- Contrainte $c(X_1, \ldots, X_k)$
- c est hyper-arc consistante ssi

$$\forall i \in [1, k],\ \forall v_i \in D_i$$
$$\exists v_1 \in D_1 \ldots \exists v_{i-1} \in D_{i-1} \exists v_{i-1} \in D_{i-1} \ldots \exists v_k \in D_k$$
$$c(v_1, \ldots, v_k)$$

- Chaque valeur pour chaque variable participe à une solution de la contrainte

Algorithmes AC pour contraintes non binaire

- Algoirthmes AC existants peuvent être étendus pour gérer contraintes non binaire : coût important !
- Consistance d'arc généralisée (GAC) trop couteuse
- GAC-4 : complexité $O(ed^k)$, où k est l'arité maximale des contraintes
- Classes de contraintes spécifiques pour lesquelles il existe des algorithmes de consistance efficaces : contraintes globales (à suivre ...)

Consistance de borne

- Le domaine d'une variable est représenté par des bornes (valeurs minimale et maximale) : $X \in min(X)..max(X)$
- Une contrainte c est borne-consistante si pour chaque variable X de c
  - il existe $d_1, \ldots, d_k$ pour les autres variables $X_1, \ldots, X_k$ tels que
    - $min(X_i) \leq d_i \leq max(X_i)$ pour tout i
    - $\{X \leftarrow min(X), X_1 \leftarrow d_1, \ldots, X_k \leftarrow d_k\}$ est une solution de c
  - il existe $d'_1, \ldots, d'_k$ pour les autres variables $X_1, \ldots, X_k$ tels que
    - $min(X_i) \leq d_i \leq max(X_i)$ pour tout i
    - $\{X \leftarrow max(X), X_1 \leftarrow d'_1, \ldots, X_k \leftarrow d'_k\}$ est une solution de c
- Un CSP arithmétique est borne-consistant si toutes ses contraintes le sont.

Consistances

Règles de propagation

- Les règles de propagation sont générées des contraintes pour modifier les bornes
- Exemples : $X = Y + Z$ équivalent à $Y = X - Z$ et $Z = X - Y$, règles
  - $min(X) \geq min(Y) + min(Z)$, $max(X) \leq max(Y) + max(Z)$
  - $min(Y) \geq min(X) - max(Z)$, $max(Y) \leq max(X) - min(Z)$
  - $min(Z) \geq min(X) - max(Y)$, $max(Z) \leq max(X) - min(Y)$

  si $X \in 1..10$, $Y \in 2..6$, $Z \in 1..10$, la consistance de borne réduit les domaines à

$$X \in 3..10, \quad Y \in 2..6, \quad Z \in 1..8$$