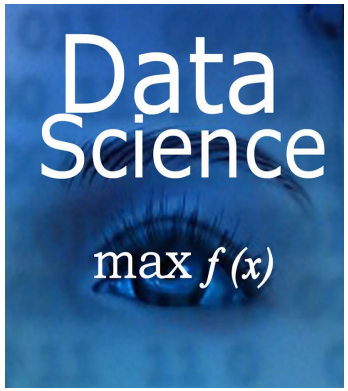
Programmation par Contraintes

Notions, algorithmes et consistance

Maria Malek

2 janvier 2017



Satisfaction de Contraintes

- Problèmes d'emploi de temps.
- Gestion d'agenda.
- Gestion de trafic.
- Problèmes de planification et d'optimisation (comme le problème de routage de réseaux de telecommunication).

Modélisation du problème

- Un problème de satisfaction de contraintes (PSC) est un triplet (X,D,C) :
  - $X = \{X, ..X_n\}$ est un ensemble fini de variables à résoudre,
  - D est la fonction qui définit le domaine de chaque variable $(D(X_i))$
  - $C = \{C_1, ..C_m\}$ est un ensemble fini de contraintes.
  - Une contrainte est une relation entre un sous ensemble de variables noté W et un sous ensemble de valeurs noté T : C=(W,T) avec $W \subseteq X$ et $T \subseteq D^{|W|}$.

Affectations & Solution

- Une affectation est un ensemble de couples (variables-valeurs) : $A = \{(X_j \leftarrow v_j)\}$ avec $X_j \in X$ et $v_j \in D(X_j)$.
- Une affectation est *partielle* si elle ne concerne qu'une partie de variables et *totale* sinon.
- D est la fonction qui définit le domaine de chaque variable ($D(X_i)$).
- Une affectation A est valide par rapport à C si la relation définie dans chaque contrainte C_i est vérifiée pour les valeurs des variables affectées dans A.
- Une solution à un problème de resolution de contrainte est une affectation totale valide A.

L'algorithme Simple Backtrack

simple-backtrack(X,D,C) : affectation

- VAR A :affectation, i : *entier*
- $i \leftarrow 1$
- *TANTQUE* $i \leq n$
  - (*) choisir $v_i \in D(X_i)$
  - *SI* $A \cup \{X_i \leftarrow v_i\}$ est valide
    - $A \leftarrow A \cup \{X_i \leftarrow v_i\}$
  - *SINON*
    - retour au dernier point de choix (*)
  - $i \leftarrow i + 1$
- *RETOURNER* A

L'algorithme Anticipation

Anticipation (X,D,C) : affectation

- VAR A :affectation, i,j : *entier*
- $i \leftarrow 1$
- *TANTQUE* $i \leq n$
  - (*) choisir $v_i \in D(X_i)$
  - $A \leftarrow A \cup \{X_i \leftarrow v_i\}$
  - $j \leftarrow i+1$
  - *TANTQUE* $j \leq n$
    - $D(X_j) = \{v_j \in D(X_j) \text{tq} A \cup \{X_j \leftarrow v_j\} \text{est valide}\}$
    - *SI* $D(X_j) = \{\}$
      1. retour au dernier point de choix (*)
    - $j \leftarrow j+1$
  - $i \leftarrow i+1$
- *RETOURNER* A

L'algorithme Choix du plus petit domaine

min-domaine(X,D,C) : affectation

- VAR A :affectation, i,j,k : ENTIER
- $i \leftarrow 1$
- *TANTQUE*v$i \leq n$
  - choisir la variable X_k non affectée ayant le plus petit domaine
  - (*) choisir $v_k \in D(X_k)$
  - $A \leftarrow A \cup \{X_k \leftarrow v_k\}$
  - *TANTQUE* il existe des variables X_j non affectée
    - $D(X_j) = \{v_j \in D(X_j)$tq $A \cup \{X_j \leftarrow v_j\}$est valide$\}$
    - *SI* $D(X_j) = \{\}$
      1. retour au dernier point de choix (*)
  - $i \leftarrow i + 1$
- *RETOURNER* A

La notion de Consistance

- Une *consistance* est une propriété qui doit être assurée à chaque étape de la recherche de la solution dans l'objectif de couper l'arbre de recherche.
- NON coûteuse à calculer.
- Retire des domaines des variables, les valeurs qui ne participent pas à aucune solution.
- Types de consistance :
  - **La consistance de nœud** consiste à éliminer des domaines de variables toutes les valeurs qui n'appartiennent pas aux solutions des contrainte unaires.
  - **La consistance d'arc** contraintes binaires : le système sera représenté par un graphe.

La propriété arc consistance

- Soit s_X le domaine courant de la variable X, soit c une contrainte sur X et Y :
- la valeur $v_x \in s_x$ est *arc-consistante pour c* si : $\exists v_y \in s_y$ tel que $(v_x, v_y) \in sol(c)$;
- De même on dit que la contrainte c est *arc-consistante* si : $\forall v_x \in s_x$, v_x est arc-consistante et vice-versa en échangeant le rôle de X et de Y.
- Finalement, un système de résolution de contraintes est *Arc-consistant* ssi chaque contrainte est arc-consistante.

Les algorithmes qui rendent un système AC

- L'algorithme AC1 propage la mise à jours des domaines dans le graphe jusqu'à la stabilisation des domaines s.
- L'algorithme AC3 Utilise une file d'attente. Dès qu'un domaine est modifié, on ne place dans la file que les arcs pouvant être affectés par la modification.
- L'algorithme AC4 affine m la performance en ajoutant une stratégie permettant de choisir d'une façon optimale les valeurs à verifier.
- Ces algorithmes sont basés sur la fonction *réviser* qui prend en paramètre une contrainte binaire $c(X, Y)$ et qui met à jours les domaines des deux variables.

L'algorithme Réviser

Reviser(c : contrainte ; X,dx : variable) : Bool

- VAR supprime :Bool , v,w : Valeurs
- *supprime* $\leftarrow$ *faux*
- *TANTQUE* $v \in dx$
  - *SI* il n'y a pas de $w \in d_y$ tq $(v, w) \in sol(c)$
    - supprimer v de dx
    - *supprime* $\leftarrow$ *vrai*
- *RETOURNER* supprime

L'algorithme AC1

AC1 (C :PSC)

- VAR change :bool , c : Contrainte, X : Variable
- *change* $\leftarrow$ *vrai*
- *TANTQUE change*
  - *change* $\leftarrow$ *faux*
  - *TANTQUE* il y a des couples (c,X)
    - *change* $\leftarrow$ *change* $\vee$ *reviser*(c, X)

L'algorithme AC3

AC3 (C :PSC)

- VAR Q :File
- *change* $\leftarrow$ *vrai*
- *TANTQUE* Q est non vide
  - récupérer la tête (c,X)
  - *SI* reviser(c,X)
    - $Q \leftarrow (Q \cup \{(c', Z) \mid (var(c') = \{X, Z\}) \wedge Z \neq Y\})$

L'algorithme AC4

- Permet d'éliminer les verifications inutiles en c
- Nous construisons la table de support, pour une contrainte $c(X, Y)$.
  - (Y, w) est un support pour (X, v) si $(v, w) \in sol(C)$
  - Quand (Y, w) est supprimée, il suffit de réétudier les valeurs dont (Y, w) était les supports.
  - Implémentation de cet algorithme en utilisant un systèmes de règles.
  - Notations : la valeur v du domaine de X sera noté $x(v)$.

AC4 : Construction des règles

- Pour une contrainte $c(X, Y)$ on construit la règle dont la tête est $x(a)$. on sélectionne dans $sol(c)$ les couples (a, w)
  Appelons B ce sous ensemble de $sol(c)$
  - $x(a) < -\{y(w)|w \in B\}$
  - On associe à chaque atome a la liste $Supporte_a$, qui contient les atomes dont a est le support.
  - On associe à chaque règle $(c, x(a))$ un compteur $Compt_{(c,x(a))}$, initialisé à la taille du corps de la règle.

AC4 : Exemple de construction de règles

- Soit la contrainte c définie par $X = \{X, Y\}$, avec les domaines $D(X) = D(Y) = [0, 3]$,
  $sol(c) = \{(0,1), (1,1), (1,2), (2,1), (2,3)\}$
  - $x(0) < -y(1).$
    $x(1) < -y(1), y(2).$
    $x(2) < -y(1), y(3).$
    $x(3) < -$
  - $y(0) < -$
    $y(1) < -x(0), x(1), x(2).$
    $y(2) < -x(1).$
    $y(3) < -x(2).$

L'algorithme AC4

AC4 (C :PSC)

- VAR $Q = \{x(a)|x(a) < - \in R\}$
- *TANTQUE*Q est non vide
  - récupérer la tête $x(a)$
  - *PourTous* $y(b) \in Supporte_{x(a)}$
    - decrémenter $Compt_{c,y(b)}$
    - *SI* $Compt_{c,y(b)} = 0$
      $Q = Q \cup \{y(b)\}$