Intelligence Artificielle

Introduction à la Programmation par Contraintes

Maria Malek

Département Informatique

Satisfaction de Contraintes

- Problèmes d'emploi de temps.

Satisfaction de Contraintes

- Problèmes d'emploi de temps.
- Gestion d'agenda.

Satisfaction de Contraintes

- Problèmes d'emploi de temps.
- Gestion d'agenda.
- Gestion de trafic.

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 $v_j \in D(X_j)$.

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.

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 ($D(X_i)$).

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.

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* $j \leq n$
      - $D(X_j) = \{v_j \in D(X_j)tqA \cup \{X_j \leftarrow v_j\}$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* $i \leq n$
    - 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.

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.

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.

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 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 $(v_x, v_y) \in sol(c)$;

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.

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.

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.

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.

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)
      - $Q \leftarrow (Q \cup \{(c', Z) \mid (var(c') = \{X, Z\}) \wedge Z \neq Y\})$