Programmation Par Contraintes

Aide à la Décision et Intelligence Artificielle
Licence 3

Dr. **Abdelkader OUALI**

abdelkader.ouali@unicaen.fr

Département Informatique
UFR des sciences
Université de Caen Normandie

2021

Algorithme de **BackTrack**

Algorithme 1 : Pseudo-code de **BackTrack**

Entrées :

- Un CSP $P = (X, D, C)$ `// membre de la classe BacktrackSolver`
- Une instantiation partielle $\mathcal{I}$ `// argument donné à la fonction BT`
- Les variables non instanciées $\mathcal{V}$ `// argument donné à la fonction BT (classe LinkedList dans le TP)`

Output : Solution

Fonction `BT`($\mathcal{I}, \mathcal{V}$)

```
    // condition d'arrêt de la récurrsivité
    si V = ∅ alors
        retourner I
    // choisir une variable non encore instanciée
    x_i ← Retirer(V)
    // choisir une valeur dans le domaine de x_i
    pour v_i ∈ D(x_i) faire
        N ← I ∪ (x_i, v_i)
        si IsConsistent(N) alors
            R ← BT(N, V)
            si R ≠ Nul alors
                retourner R
    Mettre(V, x_i)
    retourner Nul
```

Fonction `IsConsistent`($\mathcal{N}$)

```
    pour c ∈ C faire
        si Portée(c) ⊆ Variables(N) alors
            si ¬ c.Satisfait(N) alors
                retourner Faux
    retourner Vrai
```

Algorithme de **NC**

Algorithme 2 : Pseudo-code de **NC**

Entrées :

- Un CSP (X, D, C) `// membre de la classe`
- Un argument pour suivre l'évolution des domaines ED

Output :

- Les domaines dans ED vérifient la cohérence de noeud
- La fonction retourne Vrai si aucun domaine n'est vide, sinon elle retourne Faux

Fonction `enforceNodeConsistency`(ED)
- **pour** *toute variable* $x \in X$ **faire**
  - **pour** *toute valeur* $v \in ED(x)$ **faire**
    - **pour** *toute* **contrainte unaire** $c \in C$ **faire**
      - **si** $\neg$ c.`Satisfait`$((x, v))$ **alors**
        - $ED(x) \leftarrow ED(x) \setminus \{v\}$ `// Supprimer v du domaine de x`
- **pour** *toute variable* $x \in X$ **faire**
  - **si** $ED(x) = \emptyset$ **alors retourner** Faux
- **retourner** Vrai

Algorithme de **Revise**

Algorithme 3 : Pseudo-code de **Revise**

Entrées :

- Un CSP (X, D, C) `// membre de la classe`
- Un argument pour la 1er variable x_i
- Un argument pour la 2ème variable x_j
- Un argument pour suivre l'évolution des domaines *ED*

Output :

- Les domaines dans *ED* vérifient l'arc-cohérence
- La fonction retourne *Vrai* si au moins un domaine des variables est réduit, sinon elle retourne *Faux*

Fonction `REVISE`(x_i, x_j, ED)
    $Del \leftarrow Faux$
    pour *toute* $v_i \in ED(x_i)$ **faire**
        $viable \leftarrow Faux$
        pour *toute* $v_j \in ED(x_j)$ **faire**
            $toutSatisfait \leftarrow Vrai$
            pour *toute contrainte binaire* $c \in C$ *portant sur* x_i *et* x_j **faire**
                $U \leftarrow \{(x_i, v_i), (x_j, v_j)\}$
                si $\neg$ c.`Satisfait`$(\mathcal{N})$ **alors**
                    $toutSatisfait \leftarrow Faux$
                    break
            si *toutSatisfait* **alors**
                $viable \leftarrow Vrai$
                break
        si $\neg$ *viable* **alors**
            $ED(x_i) \leftarrow ED(x_i) \setminus \{v_i\}$ `// Supprimer` v_i `du domaine de` x_i
            $Del \leftarrow Vrai$
    retourner *Del*

Algorithme de **AC1**

Algorithme 4 : Pseudo-code de **AC1**

Entrées :

- Un CSP (X, D, C) `// membre de la classe`
- Un argument pour suivre l'évolution des domaines *ED*

Output :

- Les domaines *ED* sont arc-cohérent si aucun domaine n'est vide
- La fonction retourne *Faux* si au moins un domaine est vide, sinon elle retourne *Vrai*

```
Fonction AC1(ED)
    si ¬NC(ED) alors
        retourner Faux

    faire
        change ← Faux
        pour tout couple (x_i, x_j) dans X faire
            si REVISE(x_i, x_j, ED) alors
                change ← Vrai

    tant que change = Vrai

    pour toute variable x ∈ X faire
        si ED(x) = ∅ alors retourner Faux
    retourner Vrai
```

Algorithme de **MAC** solver

Algorithme 5 : Pseudo-code de **MAC**

Entrées :

- Un CSP $P = (X, D, C)$ `// membre de la classe BacktrackSolver`
- Une instantiation partielle $\mathcal{I}$ `// argument donné à la fonction MAC`
- Les variables non instanciées $\mathcal{V}$ `// argument donné à la fonction MAC (classe LinkedList dans le TP)`
- Suivre l'évolution des domaines ED `// arugment de la fonction MAC`

Output : Solution

```
Fonction MAC(I, V, ED)
    // conditions d'arrêt de la récurrsivité
    si V = ∅ alors
        retourner I
    sinon
        // Réduction des domaines des variables par l'arc-cohérence
        si ¬ AC1(X, ED, C) alors
            retourner Nul
        // choisir une variable non encore instanciée
        x_i ← Retirer(V)
        // choisir une valeur dans le domaine de x_i
        pour v_i ∈ ED(x_i) faire
            N ← I ∪ (x_i, v_i)
            si IsConsistent(N) alors
                R ← MAC(N, V, ED)
                si R ≠ Nul alors
                    retourner R
        Mettre(V, x_i)
        retourner Nul
```

Algorithme de **AC3** (n'est pas forcément requis pour le TP)

Algorithme 6 : Pseudo-code de **AC3**

Entrées :

- Un CSP (X, D, C) `// membre de la classe`
- Un argument pour suivre l'évolution des domaines *ED*

Output :

- Les domaines *ED* sont arc-cohérent si aucun domaine n'est vide
- La fonction retourne *Faux* si au moins un domaine est vide, sinon elle retourne *Vrai*

Fonction `AC3(`*ED*`)`
 si $\neg$`NC(`*ED*`)` **alors**
  retourner Faux
 $Q \leftarrow \{(x_i, x_j) : (i \neq j) \wedge (\text{ il existe une contrainte entre } x_i \text{ et } x_j)\}$
 tant que $Q \neq \emptyset$ **faire**
  $Q \leftarrow Q \setminus \{(x_i, x_j)\}$ `// Prendre une paire de variables` (x_i, x_j) `de` Q
  si `REVISE(`x_i, x_j, ED`)` **alors**
   `// il y a eu réduction du domaine de` x_i
   $Q \leftarrow Q \cup \{(x_k, x_i) : \text{il existe une contrainte entre } x_k \text{ et } x_i \text{ et } x_k \neq x_i \text{ et } x_k \neq x_j\}$
 pour *toute variable* $x \in X$ **faire**
  si $ED(x) = \emptyset$ **alors retourner** Faux
 retourner Vrai