

Partie 2 – Problèmes de satisfaction de contraintes

2.2 LA RESOLUTION

2.2.1 CONSISTANCE ET PROPAGATION

Détection des incohérences

- L'enjeu est de détecter au plus tôt des incohérences dans les valeurs des variables pour éviter des assignations inutiles
- L'idée est de supprimer des valeurs impossibles des domaines des variables afin de diminuer l'espace de recherche
- Cette suppression peut-être envisagée :
 - en **amont du BT** pour réduire les domaines à considérer
 - après chaque assignation pour réduire les domaines des variables restant à assigner : on parle de **propagation**
 - » on parle alors de **domaines dynamiques** : *les domaines évoluent au cours du BT*

Méthodes de détection d'incohérences

- Définition de propriétés de consistance que les domaines des variables doivent respecter
 - » Ex : une valeur du domaine d'une variable doit avoir au moins une occurrence dans chacune des contraintes où elle apparaît
 - A partir de telles propriétés, on cherche à calculer une réduction des domaines qui vérifie cette propriété et garantit que les solutions du CSP sont conservées
 - » Ex : soit $C(x)$ l'ensemble des contraintes contenant x :
$$\text{Dréduit}(x) = D(x) \cap \bigcap_{c \in C(x)} (c|_x)$$
- Définition opérationnelle de règles de réduction de domaine attachées aux contraintes
 - » Ex : on peut associer à la contrainte $x \neq y$ la règle :
si v est assigné à x *alors* $D(y) \leftarrow D(y) - \{v\}$

Exemple : la simple consistance une autre façon de voir BT

- On appelle **tuple valide** d'une contrainte c de portée $\langle x_1, x_2, \dots, x_k \rangle$ un tuple $t \in c$ tel que $t \in \text{Dom}(x_1) \times \text{Dom}(x_2) \times \dots \times \text{Dom}(x_k)$
- Une contrainte est **simple-consistante** (ou 1-consistant) si elle possède un tuple valide
- Un CSP est **simple-consistant** (ou 1-consistant) si chaque contrainte est simple-consistante : $\forall c \in C, \forall x \in \text{Portée}(c) \exists t \in \text{Tuples}(c) \text{ t.q. } t \text{ valide}$
- Propriété : un CSP non simple-consistante n'a pas de solution.
- Si on voit le Backtrack comme un méthode qui modifie les domaines à chaque assignation d'un v à x en les réduisant à un singleton : $\text{Dom}(x) := \{v\}$ alors la **simple consistance** est la propriété de consistante du backtrack

Backtrack : vision domaines dynamiques

Fonction BT(List V, Network R) : Booléen

Début

```
si |V| = |R.X| alors
    afficher R.D;
    retourner true;
finsi;
x ← ChoixVariableNonAssignée(R.X, V);
pour tout v ∈ Tri(R.D(x)) faire
    sauvegarder R.D;
    R.D(x) := {v}
    si simple-consistant(R.C) alors
        si BT(V ∪ {x}, R) alors retourner true; finsi;
    finsi;
    restaurer R.D;
finpour;
retourner false;
```

Fin

Partie 2 – Problèmes de satisfaction de contraintes

2.2 LA RESOLUTION

2.2.1 L'ARC-CONSISTANCE

L'arc-consistance

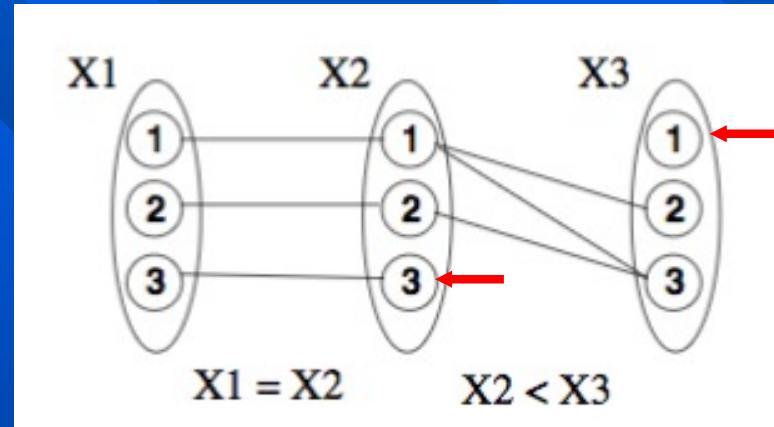
- On appelle **tuple support** d'une valeur v d'une variable x dans une contrainte c un tuple $t \in c$ tel que $t|_x = v$ (la proj. de t sur x est v)
 - Ex : le tuple $(2,3)$ de c avec $\text{portée}(c) = \langle x, y \rangle$ est support de la valeur 2 de x dans c et de la valeur 3 de y dans c
- Une contrainte c est **arc-consistante** si pour toute variable x de sa portée, chaque valeur v du domaine de x possède un **tuple support valide** : $\forall x \in \text{Portée}(c), \forall v \in D(x), \exists t \in \text{Tuples}(c) \text{ t.q. } t|_x = v \text{ et } t \text{ valide}$
- Un CSP est **arc-consistant** (ou 2-consistant) si chaque contrainte est arc-consistante :
$$\forall c \in C, \forall x \in \text{Portée}(c), \forall v \in D(x), \exists t \in \text{Tuples}(c) \text{ t.q. } t|_x = v \text{ et } t \text{ valide}$$

(ce qui revient à) si chaque valeur du domaine de chaque variable possède un **tuple support valide** dans chaque contrainte où la variable apparaît :

$$\forall x \in X, \forall v \in \text{Dom}(x), \forall c \in C \text{ t.q. } x \in \text{Portée}(c), \exists t \in \text{Tuples}(c) \text{ t.q. } t|_x = v \text{ et } t \text{ valide}$$

Arc-Consistance

- Arc-consistante : chaque valeur du domaine de chaque variable doit posséder un **tuple support valide** dans chaque contrainte où la variable apparaît
- Exemple :
 - **Non Arc-Consistant car**
 - » valeur 1 de x_3 pour $x_2 < x_3$
 - » valeur 3 de x_2 pour $x_2 < x_3$



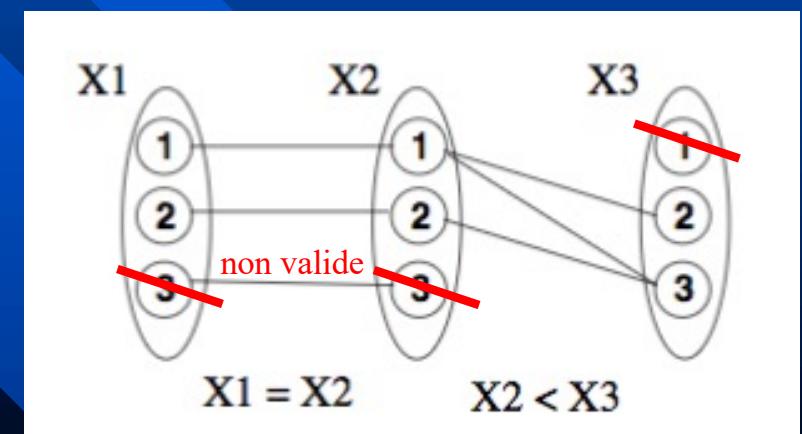
Arc-Consistance

- On peut restaurer l'arc-consistance en supprimant :
 - des domaines les valeurs sans support
 - attention cela change la validité des tuples
- La fermeture arc-consistance d'un réseau (X, D, C) consiste à calculer le plus grand domaine de chaque variable tel que $D_{AC} \subseteq D$ et (X, D_{AC}, C) est arc-consistant
 - Cette fermeture est unique

- Exemple :

Arc-Consistant

$$D_{AC}(X_1) = \{1, 2\}, D_{AC}(X_2) = \{1, 2\}, D_{AC}(X_3) = \{2, 3\}$$



AC3 : un algo pour tester/restaurer l'arc-consistance

[Mackworth, 1977]

- Une fonction *Révise* qui filtre le domaine d'une variable x relativement à une contrainte c en vérifiant que chaque valeur v du domaine de cette variable possède un tuple t support dans la contrainte
 - t appartient au produit cartésien des domaines des variables de c
 - t contient pour x la valeur v
- La fonction *AC* maintient un ensemble Q de couples var/cont (x, c) tels que $x \in P(c)$ qu'elle considère un à un :
 - *Révise* est appelée
 - si le $D(x)$ est modifié alors l'ensemble des couples (x', c') tels que c' porte sur x et x' (avec $c' \neq c$ et $x' \neq x$) est ajouté à Q pour être (re)considérer.

AC3 : un algo pour tester/restaurer l'arc-consistance

[Mackworth, 1977]

Fonction Revise (Variable x, Constraint c, Network R) : Booléen

Début

```
modif  $\leftarrow$  false ;  
Pour tout v  $\in$  R.D(x) Faire  
    Si il n'existe pas t  $\in$  c tel que  $t|_x = v$  et t valide* pour R.D alors  
        supprimer v de R.D(x) ;  
        modif  $\leftarrow$  true ;  
    finsi;  
    finpour;  
    retourner modif ;  
Fin;
```

* valide : chaque valeur du tuple est dans le domaine courant de la variable correspondante

AC3 : un algo pour tester/restaurer l'arc-consistance

[Mackworth, 1977]

Fonction AC3 (Network R) : Booléen

Début

```
Q ← {(x,c) | c ∈ R.C et x ∈ P(c)};  
tant que Q ≠ {} faire  
    retirer (x,c) de Q;  
    si Revise(x,c,R) alors  
        si R.D(x)={} alors retourner false ;  
        sinon  
            Q ← Q ∪ {(x',c') | c' ∈ R.C, x ∈ P(c'), x' ∈ P(c'), c'≠c, x'≠x};  
        finsi;  
    finsi;  
    fintantque;  
    retourner true;  
Fin;
```

K-consistance

- Filtrage (1-consistance)
- Arc-consistance (2-consistance)
- Chemin-consistance (3-consistance)
- **k-consistance** = toute instanciation consistante de $k-1$ variables peut-être étendue à une instanciation consistante avec une variable supplémentaire (k)
- **Propriété** : un CSP à n variables possède une solution s'il est k -consistant pour tout $1 \leq k \leq n$
 - Remarque : on ne peut pas « rendre » un CSP k -consistant pour $k > 2$

