Programmation par Contraintes

Module du Master “Systèmes Informatiques Intelligents” 2ème année

CHAPITRE III

Résolution d’un CSP binaire discret

Mr ISLI

Département d’Informatique

Faculté d’Electronique et d’Informatique

Université des Sciences et de la Technologie Houari Boumediène

BP 32, El-Alia, Bab Ezzouar

DZ-16111 ALGER

http://perso.usthb.dz/~aisli/TA_PpC.htm

aisli@usthb.dz

CHAPITRE III
Résolution d'un CSP binaire discret

Résolution d'un CSP binaire

- Le CSP admet-il des solutions ?
- Le CSP admet-il des solutions ? Si oui, en exhiber une
- Le CSP admet-il des solutions ? Si oui, les exhiber toutes

CHAPITRE III
Résolution d'un CSP binaire discret

Algorithmes de résolution

- Algorithmes de consistance locale
  - Propagation de contraintes : incomplets en général mais de complexité polynômiale
  - Consistance de nœud : consistance des sous-CSP de taille 1
  - consistance d'arc : consistance des sous-CSP de taille inférieure ou égale à 2
  - Complets pour certaines classes de CSP
- Algorithmes complets naïfs (recherche systématique d'une solution)
  - Générer et tester (GET)
  - Simple retour arrière (SRA)

CHAPITRE III
Résolution d'un CSP binaire discret

Algorithmes de résolution

- Algorithmes complets intelligents
  - Quand une nouvelle variable est instanciée, filtrage des domaines des variables non encore instanciées
    - Pour chaque variable non encore instanciée, supprimer de son domaine toutes les valeurs incompatibes avec la valeur venant d'être choisie pour la variable en cours d'instanciation (forward-checking) :
      - Anticipe les situations auxquelles fait face SRA : forward-checking = anticipation
    - Appliquer un algorithme de consistance d'arc au CSP résultant de la nouvelle instanciation de variable (look-ahead) :
      - Forward-checking = forme simplifiée de look-ahead
      - Filtrage au niveau des nœuds de l'arbre de recherche : meilleur que celui du forward-checking (nouveaux domaines=sous-ensembles de ceux que donnerait FC)
      - Mais plus gourmand en temps
    - Compromis entre filtrage et complexité
  - Instancier-et-propager (forme de branch and bound)

CHAPITRE III
Résolution d'un CSP binaire discret

Algorithmes de résolution d'un CSP binaire

- Heuristiques
  - Ordre d'instanciation des variables (statique, dynamique)
  - Ordre sur les valeurs du domaine de la variable en cours d'instanciation (statique, dynamique)

Algorithme « générer et tester » (GET)

1. Initialement, aucune instanciation n'est marquée
2. Considérer une instanciation $i=(e_1,\ldots,e_n)$ non marquée
3. Marquer i
4. Si i satisfait toutes les contraintes du CSP :
   - Retourner i
5. Si toutes les instanciations sont marquées
   - Retourner « échec : le CSP n'admet pas de solution »
6. Aller en 2.

CHAPITRE III
Résolution d'un CSP binaire discret

Algorithme GET :

fonction GET(A) : booléen
début
si toutes les variables de X sont instanciées alors
    si A est solution alors retourner VRAI sinon retourner FAUX finsi
sinon
    choisir une variable X_i de X qui n'est pas encore instanciée
    pour toute valeur V_i du domaine $D(X_i)$ faire
        si $GET(A \cup \{(X_i, V_i)\})$ alors retourner VRAI finsi fait
    retourner FAUX
finsi
fin

Algorithme GET : inconvénient majeur

- Si inexistence de solution ou unique solution consistant en la toute dernière instanciation
  - Parcours exhaustif de toutes les instanciations possibles : exploration de tout l'espace de recherche D(X1)x…xD(Xn).

Algorithme SRA : amélioration de GET

- Si la valeur choisie pour la variable en cours d'instanciation est incompatible avec les valeurs choisies pour les variables déjà instanciées ➔ retour arrière
  - Instancier avec une autre valeur du domaine si possible
  - Si toutes les valeurs déjà essayées
    - Retourner échec si toute première variable
    - Retourner à la variable précédente sinon (retour arrière chronologique)
- Sinon :
  - Si toute dernière variable retourner "succès"
  - sinon passer à la variable suivante et réitérer le processus

Algorithme « Simple Retour Arrière » (SRA)

fonction SRA(A) : booléen

début

choisir une variable X_i non encore instanciée

pour toute valeur V_i du domaine $D(X_i)$ faire

si $A \cup \{(X_i,V_i)\}$ satisfait les contraintes unaires sur X_i et toutes les contraintes binaires sur X_i et les variables déjà instanciées alors

si $A \cup \{(X_i,V_i)\}$ instanciation complète alors retourner VRAI

sinon si SRA($A \cup \{(X_i,V_i)\}$) alors retourner VRAI finsi finsi finsi

fait

retourner FAUX

fin

CHAPITRE III
Résolution d'un CSP binaire discret

Exemple : le problème des quatre reines

- Echiquier de 16 cases (4x4)
- 4 lignes L1, L2, L3 et L4
- 4 colonnes C1, C2, C3 et C4
- 4 reines R1, R2, R3 et R4 :
  - La reine R1 se déplace sur la colonne C1
  - La reine R2 se déplace sur la colonne C2
  - La reine R3 se déplace sur la colonne C3
  - La reine R4 se déplace sur la colonne C4

CHAPITRE III
Résolution d'un CSP binaire discret

Exemple : le problème des quatre reines

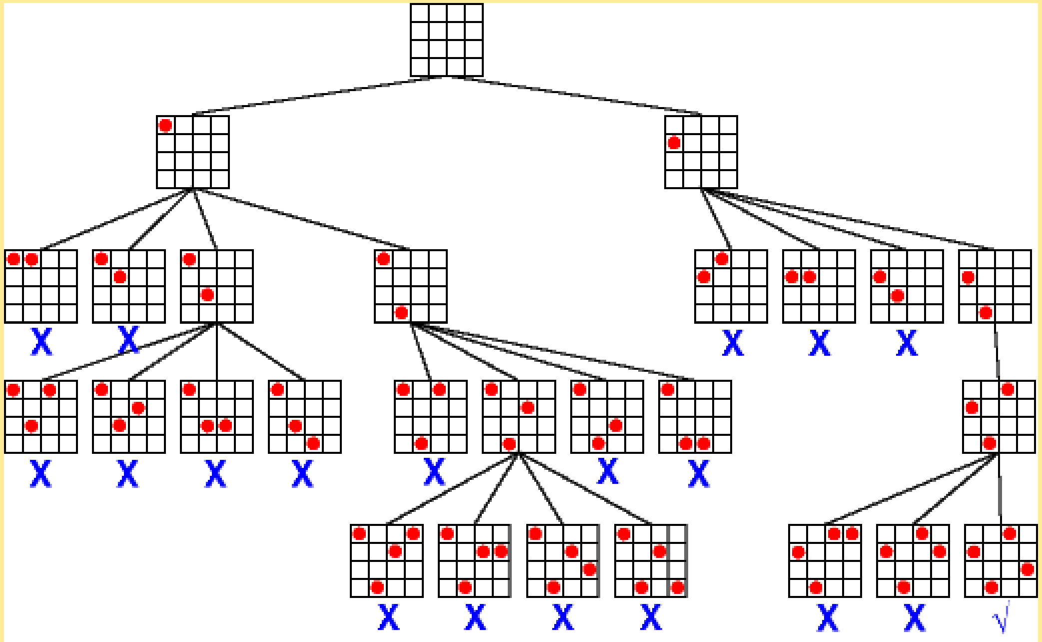
- Modélisation à l'aide d'un CSP P=(X,D,C)
  - $X = \{X_1, X_2, X_3, X_4\}$
  - $D = \{D(X_1),D(X_2),D(X_3),D(X_4)\}$ avec
    - $D(X_1)=D(X_2)=D(X_3)=D(X_4)\}=\{1,2,3,4\}$
  - $C=C_{LI}\cup C_{DD}\cup C_{DA}$
    - $C_{LI}=\{X_i\neq X_j, i\neq j, i\in\{1,2,3,4\}, j\in\{1,2,3,4\}\}$
    - $C_{DD}=\{X_i-i\neq X_j-j, i\neq j, i\in\{1,2,3,4\}, j\in\{1,2,3,4\}\}$:

      (i,X_i) et (j,X_j) ne doivent pas être sur la même parallèle à la 1ère bissectrice des axes (diagonale ascendante)
    - $C_{DA}=\{X_i+i\neq X_j+j, i\neq j, i\in\{1,2,3,4\}, j\in\{1,2,3,4\}\}$:

      (i,X_i) et (j,X_j) ne doivent pas être sur la même parallèle à la 2ème bissectrice des axes (diagonale descendante)

Résolution du problème des quatre reines

- algorithme SRA



Algorithme SRA : inconvénient majeur

- Améliore l’algorithme GET
  - Réduction de l’espace de recherche : les valeurs affectées à la variable en cours d’instanciation sont réduites à celles satisfaisant les contraintes avec les variables déjà instanciées
- Mais la détection des conflits reste tardive
- Solution ➔ FC : dès qu’une nouvelle variable est instanciée, supprimer des domaines des variables non encore instanciées les valeurs qui ne sont pas compatibles avec la valeur choisie

➔ anticipation

CHAPITRE III
Résolution d'un CSP binaire discret

Algorithme « Forward Checking » (FC)

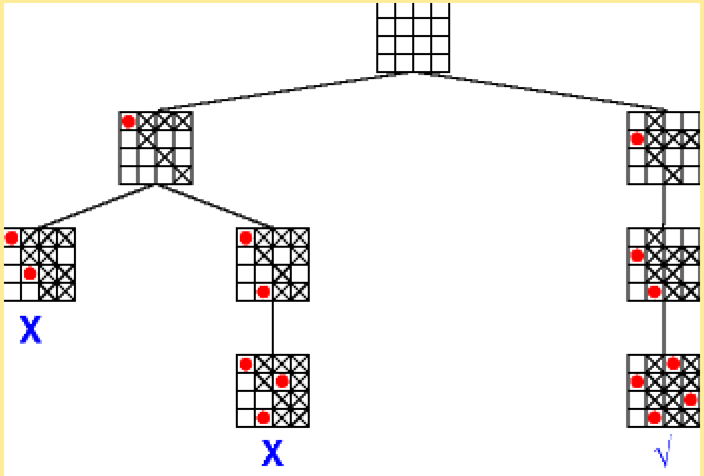
Sous forme d'une fonction récursive FC(A,Dom)

- P=(X,D,C) CSP dont il faut tester la consistance
- A et Dom : instanciation (partielle) + évolution des domaines
- A ensemble dont les éléments sont de la forme (X_i,V_i)
  - contient les valeurs V_i affectées aux variables X_i déjà instanciées
- A l'appel de la fonction FC :
  - A vide (aucune variable n'est instanciée)
  - Dom=D
- FC fonction booléenne retournant VRAI ssi P est consistant

Algorithme « Forward Checking » (FC)

fonction FC(A,Dom) : booléen
début
si toutes les variables de X sont instanciées alors retourner VRAI
sinon
  choisir une variable X_i de X qui n'est pas encore instanciée
  pour toute valeur V_i de $Dom(X_i)$ satisfaisant les contraintes unaires sur X_i faire
    D'=Dom //$D'(X_j)=Dom(X_j)$, pour toute variable X_j
    $D'(X_i)=\{V_i\}$;non_instanciees=$\{X_j \in X$: X_j non encore instanciée$\} \setminus \{X_i\}$;domaine_vide=faux
    tant que (non_instanciees$\neq\emptyset$) et (domaine_vide=faux) faire
        considérer une variable X_j de l'ensemble non_instanciees
        non_instanciees=non_instanciees$\setminus\{X_j\}$
        $D'(X_j)=\{V_j \in Dom(X_j) | \{(X_i,V_i),(X_j,V_j)\}$ satisfait les contraintes sur X_i et $X_j\}$
        si $D'(X_j)$ vide alors domaine_vide=vrai finsi fait
    si (domaine_vide=faux) alors si FC($A\cup\{(X_i,V_i)\}$,D') alors retourner VRAI finsi finsi fait
  retourner FAUX //la solution partielle A ne peut pas être étendue à la variable X_i
finsi fin

Exemple :



Exemple : appliquer l'algorithme FC au CSP P=(X,D,C)

- $X=\{X_1,X_2,X_3,X_4\}$
- $D(X_1)=D(X_2)=D(X_3)=D(X_4)=\{1,2,3,4,5,6,7,8\}$
- $C=\{c_1 : X_1$ impair ; $c_2 : X_2$ pair ; $c_3 : X_3$ impair ; $c_4 : X_4$ pair ;

  $c_5 : X_1<X_2$;

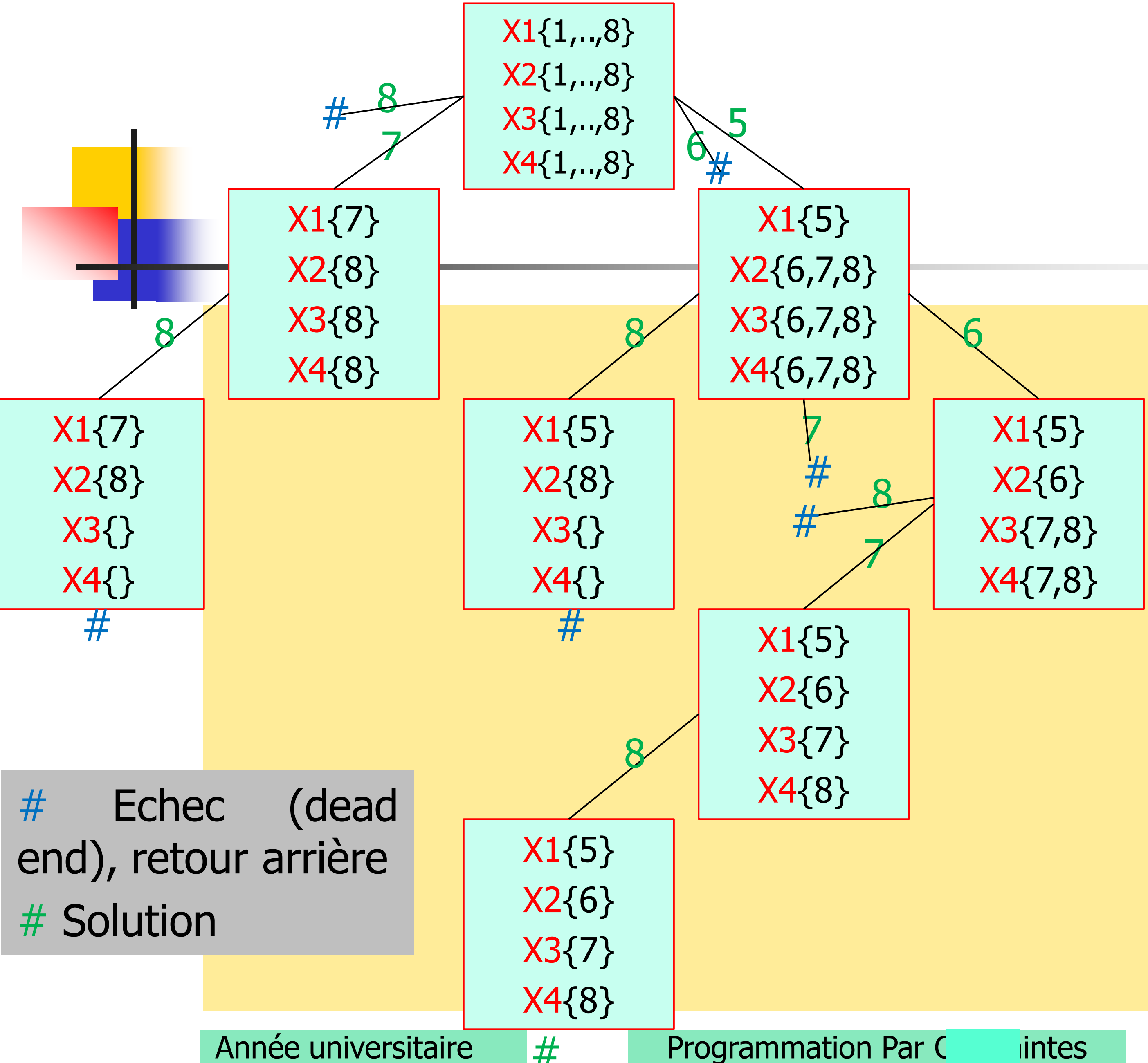
  $c_6 : X_2<X_4$;

  $c_7 : X_1<X_4$;

  $c_8 : X_1<X_3$;

  $c_9 : X_3<X_4$;

  $c_{10} : X_2<X_3\}$



Echec (dead end), retour arrière

Solution

Instanciation de X1 (contraintes unaires)

FC (var non encore inst)

Instanciation de X2 (contraintes unaires)

FC (var non encore inst)

Instanciation de X3 (contraintes unaires)

FC (var non encore inst)

Instanciation de X4 (contraintes unaires)

Variables toutes inst
Aucun domaine vidé

Algorithme FC : inconvénient majeur

- Anticipe des situations d'échec que SRA ne détecte que plus tardivement
- Mais ne supprime que les valeurs incompatibles avec la valeur choisie pour la variable en cours d'instanciation :
  - Conséquence : pour X_i et X_j non encore instanciées, $D(X_i)$ peut avoir des valeurs n'ayant plus de support dans $D(X_j)$
- Solution → look-ahead : filtrage avec un algorithme de consistance d'arc

CHAPITRE III
Résolution d'un CSP binaire discret

Filtrage durant la recherche récursive d'une solution :

- Filtrage à priori
  - Prétraitement
  - Avant le début effectif de la recherche
  - Aucune variable n'est encore instanciée
  - Réduction de l'espace de recherche
  - CSP surcontraint et inconsistant : inconsistance généralement détectée par le prétraitement
- Filtrage après chaque instanciation d'une nouvelle variable
  - Instancier puis propager

Consistance de nœud (node-consistency)

- Un CSP P=(X,D,C) est consistant de noeud si pour toute variable X_i de X, et pour toute valeur v de $D(X_i)$, l’instanciation partielle $\{(X_i,v)\}$ satisfait toutes les contraintes unaires de C portant sur X_i

Principe d’un algorithme de consistance de noeud

- filtrage des domaines
  - pour chaque variable X_i
    - supprimer de $D(X_i)$ toute valeur v telle que l’instanciation partielle $\{(X_i,v)\}$ viole les contraintes unaires portant exclusivement sur X_i

fonction NC((X,D,C))

début

pour i=1 à n faire

$D(X_i)=\{v \in D(X_i)$: $\{(X_i,v)\}$ satisfait toutes les contraintes unaires portant sur $X_i\}$

fait

retourner (X,D,C)

fin

CHAPITRE III
Résolution d'un CSP binaire discret

consistance d'arc (arc-consistency)

- Un CSP P=(X,D,C) est consistant d'arc si :
  - Il est consistant de nœud
  - pour tout couple (X_i,X_j) de variables, et pour toute valeur v_i de $D(X_i)$, il existe une valeur v_j de $D(X_j)$ telle que l'instanciation partielle $\{(X_i,v_i),(X_j,v_j)\}$ satisfait toutes les contraintes binaires de C portant exclusivement sur X_i et X_j

consistance d'arc (arc-consistency)

- Exemple de CSP consistant d'arc mais inconsistant :
  - CSP modélisant l'instance suivante du problème de coloriage d'un graphe : coloriage d'un triangle avec deux couleurs

consistance d’arc (arc-consistency)

- Exemple de CSP non consistant d’arc :
  - CSP modélisant l’instance suivante du problème de coloriage d’un graphe : coloriage d’un triangle ABC
    - couleurs permises pour A et B : V et R
    - couleur permise pour C : V

Principe d'un algorithme de consistance d'arc

- Rendre le CSP consistant de noeud
- Répéter tant que possible
  - Soit X_i une variable telle qu'il existe des valeurs v_i de $D(X_i)$ pour lesquelles:
    - il existe une variable X_j pour laquelle, pour toute valeur v_j de $D(X_j)$
      - l'instanciation partielle $\{(X_i,v_i),(X_j,v_j)\}$ viole les contraintes binaires portant exclusivement sur X_i et X_j
  - supprimer de $D(X_i)$ de telles valeurs v_i

Algorithmes de consistance d’arc

- REVISE :
  - réduit la taille des domaines
  - supprime les valeurs qui ne satisfont pas les contraintes binaires
- AC1 :
  1. appliquer REVISE à tous les arcs sur lesquels il y a une contrainte
  2. Si aucun domaine n’a été modifié, la procédure prend fin
  3. Sinon aller à 1
  - **Inconvénient :** réapplique REVISE à tous les arcs sur lesquels il y a une contrainte :
    - Même aux arcs non modifiés par la passe précédente
    - jusqu’à fermeture
- AC3 :
  - ne réapplique REVISE qu’aux arcs dont le domaine de la variable extrémité a été modifié : (X_k,X_i) tel que $D(X_i)$ modifié

CHAPITRE III
Résolution d'un CSP binaire discret

fonction REVISE((Xi,Xj),(X,D,C)) : booléen

début

DELETE ← FAUX

pour tous les V_i de $D(X_i)$ faire

si il n'y a pas de V_j dans $D(X_j)$ telle que $\{(X_i,v_i),(X_j,v_j)\}$ satisfait les contraintes binaires entre Xi et Xj **alors**

supprimer V_i de $D(X_i)$

DELETE ← VRAI

finsi

fin pour

retourner DELETE

fin

```
fonction AC1((X,D,C))
    début
    (X,D,C)=NC((X,D,C))
    Q ← {(Xi,Xj) : (i≠j) et (il existe une contrainte entre Xi et Xj)}
    répéter
        R ← FALSE
        pour tous les (Xi,Xj) de Q faire
            si REVISE((Xi,Xj),(X,D,C)) alors R ← TRUE finsi
        fin pour
    jusqu'à non R
    retourner (X,D,C)
    fin
```

```
fonction AC3((X,D,C))
    début
    (X,D,C)=NC((X,D,C))
     Q ← {(Xi,Xj) : (i≠j) et (il existe une contrainte entre Xi et Xj)}
    tant que Q≠∅ faire
            Prendre une paire (Xi,Xj) de variables de Q
            supprimer la paire de Q : Q ← Q\{(Xi,Xj)}
            si REVISE((Xi,Xj),(X,D,C)) alors
             Q←Q∪{(Xk,Xi) : il existe une contrainte entre Xk et Xi et Xk≠Xi et Xk≠Xj}
            finsi
    fait
    retourner (X,D,C)
fin
```

CHAPITRE III
Résolution d'un CSP binaire discret

Exemple 1 : appliquer les algorithmes de consistance d'arc AC1 et AC3 au CSP P=(X,D,C)

- $X=\{X_1,X_2,X_3,X_4\}$
- $D(X_1)=D(X_2)=D(X_3)=D(X_4)=\{1,2,3,4,5,6,7,8\}$
- $C=\{c_1$: X_1 impair ; c_2 : X_2 pair ; c_3 : X_3 impair ; c_4 : X_4 pair ;

  $c_5 : X_1<X_2$;

  $c_6 : X_2<X_4$;

  $c_7 : X_1<X_4$;

  $c_8 : X_1<X_3$;

  $c_9 : X_3<X_4$;

  $c_{10} : X_2<X_3\}$

➔ Voir Annexe 1

```
fonction Look_Ahead(A,(X,D,C)) : booléen
Début
(X,D,C)=AC3((X,D,C))
si inconsistance (vacuité d’un domaine) alors retourner FAUX finsi
si tous les domaines sont des singletons alors retourner VRAI
sinon
    choisir une variable disjonctive Xi de X qui n’est pas encore instanciée
    pour j=1 à n faire D’(Xj)=D(Xj) fait
    pour toute valeur Vi de D(Xi) faire
        D’(Xi)={Vi}
        si Look_Ahead(A∪{(Xi,Vi)},(X,D’,C)) alors retourner VRAI finsi
    fait
    retourner FAUX  //l’instanciation partielle A ne peut pas être étendue à la variable Xi
finsi
fin
```

CHAPITRE III
Résolution d'un CSP binaire discret

Prétraitement (AC3)

X1{1,3,5}
X2{2,4,6}
X3{3,5,7}
X4{4,6,8}

3

Instanciation de X1

Filtrage (AC3) :
Q={(X2,X1),(X3,X1),(X4,X1)}

X1{3}
X2{4,6}
X3{5,7}
X4{6,8}

6

Instanciation de X2

Filtrage (AC3) :
Q={(X1,X2),(X3,X2),(X4,X2)}

X1{3}
X2{6}
X3{7}
X4{8}

#

Raffinement singleton consistant d'arc donc consistant (solution)

Consistance de chemin (path-consistency)

Un CSP P=(X,D,C) est consistant de chemin si, pour tout chemin (Xi0, … ,Xij) de longueur j :

Xi0	Xi1	…	Xik	Xi(k+1)	…	Xij
V0	v1	…	vk	v(k+1)	…	vj

Pour tous (vo,vj) tels que unaires+binaires sur Xi0 et Xij satisfaites,

il existe v1, …, v(j-1) tels que pour tout k=0…(j-1):

(vk,v(k+1)) satisfait unaires+binaires sur Xik et Xi(k+1)

consistance de chemin (path-consistency)

On montre facilement qu'on peut se restreindre aux chemins de longueur 2 (triplets de variables).

consistance de chemin (path-consistency)

Un CSP P=(X,D,C) est consistant de chemin si :

Xi0	Xi1	Xi2
V0	v1	v2

Pour tous (v0,v2) tels que unaires+binaires sur Xi0 et Xi2 satisfaites, il existe v1 tel que :

- (v0,v1) satisfait unaires+binaires sur Xi0 et Xi1
- (v1,v2) satisfait unaires+binaires sur Xi1 et Xi2

v1 support de la paire (v0,v2)

Principe d’un algorithme de consistance de chemin

Filtrage des paires de valeurs permises :

- pour chaque paire (X_i,X_j) de variables
  - supprimer les paires permises (vi,vj) n’ayant pas de support dans le domaine d’une troisième variable Xk

Xi	Xk	Xj
V0	v1	v2

Supprimer la paire permise (v0,v2) si elle n’a aucune valeur v1 de D(Xk) comme support.

fonction REVISE_pc((X_i,X_k,X_j),(X,D,C)) : booléen

début

temp=$M_P[i,j] \cap M_P[i,k] \circ M_P[k,k] \circ M_P[k,j]$

si temp$\neq M_P[i,j]$ **alors**

$M_P[i,j]$=temp

retourner VRAI

sinon retourner FAUX

finsi

fin

fonction PC1((X,D,C))

début

Q ← $\{(X_i,X_k,X_j)\in X^3 : \neg(X_i=X_k=X_j)\}$

répéter

R ← FALSE

pour tous les (X_i,X_k,X_j) de Q faire

si REVISE_pc((X_i,X_k,X_j),(X,D,C)) alors R ← TRUE finsi

fin pour

jusqu'à non R

retourner (X,D,C)

fin

Résolution d'un CSP binaire discret

```
procedure PC2(M_P){
    début
    Q ← {(X_i,X_j)∈X^2 : (i≤j) et il existe une contrainte entre X_i et X_j}; entrée_nulle=faux
    tant que Q≠∅ et ¬entrée_nulle faire
      Prendre une paire (Xi,Xj) de variables de Q, et l'en supprimer (Q ← Q\{(X_i,X_j)}); k=1
      tant que k≤n et ¬(k=i=j) et ¬entrée_nulle faire
          temp=M_P[i,k]∩M_P[i,j]°M_P[j,j]°M_P[j,k]; si temp=[0] alors entrée_nulle=true
          si temp≠M_P[i,k] alors
                  M_P[i,k]=temp; M_P[k,i]=(temp)^t
                  si i≤k alors Q=Q∪{(X_i,X_k)} sinon Q=Q∪{(X_k,X_i)} finsi finsi
          temp=M_P[k,j]∩M_P[k,i]°M_P[i,i]°M_P[i,j]; si temp=[0] alors entrée_nulle=true
          si temp≠M_P[k,j] alors
                  M_P[k,j]=temp; M_P[j,k]=(temp)^t
                  si k≤j alors Q=Q∪{(X_k,X_j)} sinon Q=Q∪{(X_j,X_k)} finsi finsi fait fait fin
```

Exemple 2 : appliquer les algorithmes de consistance de chemin PC1 et PC2 au CSP P=(X,D,C)

- $X=\{X_1,X_2,X_3\}$
- $D(X_1)=D(X_2)=D(X_3)=\{1,2,3,4,5\}$
- $C=\{c_1 : X_1<X_2 ;$
  $c_2 : X_1<X_3 ;$
  $c_3 : X_2<X_3\}$

→ Voir Annexe 2

- K-consistance et k-consistance forte :

Soit P=(X,D,C) un CSP de taille n, et k un entier entre 1 et n

- K-consistance

P est k-consistant si toute solution de tout sous-réseau de taille k-1 s'étend à toute k^{ème} variable

- K-consistance forte

P est fortement k-consistant s'il est m-consistant pour tout entier m entre 1 et k

- K-consistance et k-consistance forte :

Soit P=(X,D,C) un CSP de taille n :

  - Si P est fortement n-consistant alors toute solution partielle s'étend à une solution globale
  - Recherche d'une solution sans retour arrière

- Consistance de noeud = 1-consistance forte
- Consistance d'arc = 2-consistance forte
- Consistance de chemin = 3-consistance forte

- Minimalité :

Soit P=(X,D,C) un CSP de taille n :

  - P est dit minimal si (il est consistant de noeud) et toute solution de tout sous-CSP de taille 2 peut s'étendre à une solution globale

CHAPITRE III
Résolution d’un CSP binaire discret

Filtrage durant la recherche : un algorithme de consistance locale tel que AC1, AC3, PC1 ou PC2

```
fonction Look_Ahead(X,D,C) : booléen
Début
AC3(X,D,C)
si inconsistance alors retourner FAUX finsi
si tous les domaines sont singletons alors retourner VRAI
sinon
    choisir une variable disjonctive Xi de X
    pour toute valeur Vi de D(Xi) faire
        D’(Xi)={Vi}
        D’(Xj)=D(Xj),pour tout j≠i
        si Look_Ahead(X,D’,C) alors retourner VRAI finsi
    fait
    retourner FAUX
finsi
fin
```

Filtrage durant la recherche : un algorithme de consistance locale tel que AC1, AC3, PC1 ou PC2

```
fonction Look_Ahead(M) : booléen
Début
PC2(M)
si inconsistance alors retourner FAUX finsi
si tous les domaines sont singletons alors retourner VRAI
Sinon   choisir une variable disjonctive X_i de X
        pour toute valeur V_i de D(X_i) faire
                M'=M
                M'[i,i][a,b]= 1 si a=b=V_i,
                              O sinon
                si Look_Ahead(M') alors retourner VRAI finsi
        fait
        retourner FAUX
finsi
fin
```

CHAPITRE III
Résolution d'un CSP binaire discret

consistance de chemin (path-consistency)

- Pour les CSP binaires discrets, la consistance de chemin est généralement jugée trop coûteuse
  - On se restreint aux consistances de noeud et d'arc
  - Un CSP binaire discret dont les domaines sont des singletons
    - Si arc-consistant alors consistant
- CSP temporels et CSP spatiaux
  - Vérifient les propriétés de consistance de noeud et de consistance d'arc
  - La consistance de chemin prend toute son importance