La programmation par contraintes
Séance 2

Philippe Morignot
pmorignot@yahoo.fr

Plan du cours

1. Variables, domaines, contraintes.
2. ***Structure et algorithmes***.
3. Travaux Pratiques en MiniZinc (1 / 2).
4. Travaux Pratiques en MiniZinc (2 / 2).
5. Travaux Pratiques notés.

Structure du problème
Sous-problèmes indépendants

- Si un CSP est décomposable en composantes connexes CSP_i : si l'assignation A_i est une solution de CSP_i, $U\ A_i$ est une solution de $U\ CSP_i$

- Supposons que chaque CSP_i ait ***c*** variables parmi ***n***, chacune ayant un domaine avec ***d*** valeurs :
  - ***n/c*** sous-problèmes, chacun prenant au plus d^c de travail.
  - Complexité en temps : $O(n/c * d^c)$ soit linéaire en ***n***.
  - A comparer avec la complexité en temps d'un CSP : $O(d^n)$

- <u>Exemple :</u> un CSP booléen (***d*** = 2) avec 80 variables (***n*** = 80) et 4 sous-problèmes (***c*** = 20).

  Pire des cas pour ce CSP indépendant = $10^{6,62}$ (environ 1s)

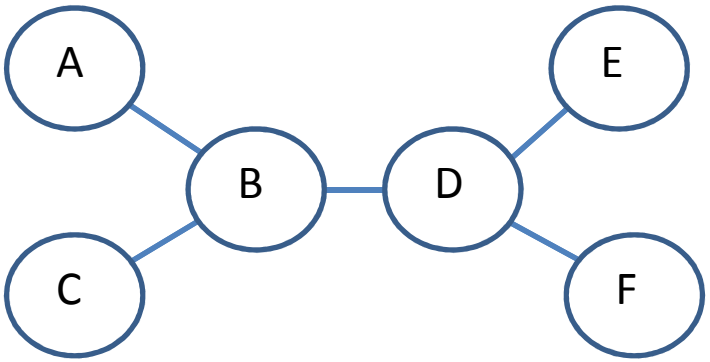
  Pire des cas pour ce CSP = 10^{24} (920 milliards d'années)

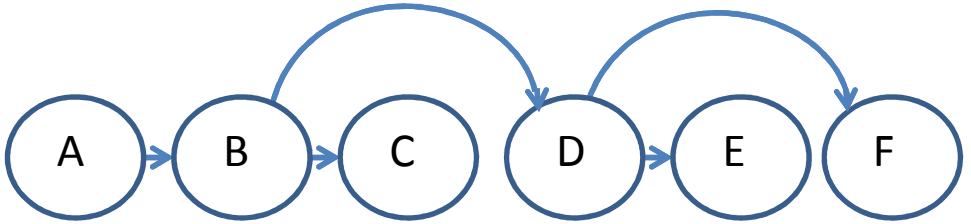
Structure du problème
Un arbre

- L'ensemble des contraintes est un arbre :

  Pour toute paire de variables (V_i, V_j)

  V_i et V_j sont reliées par au plus un chemin.



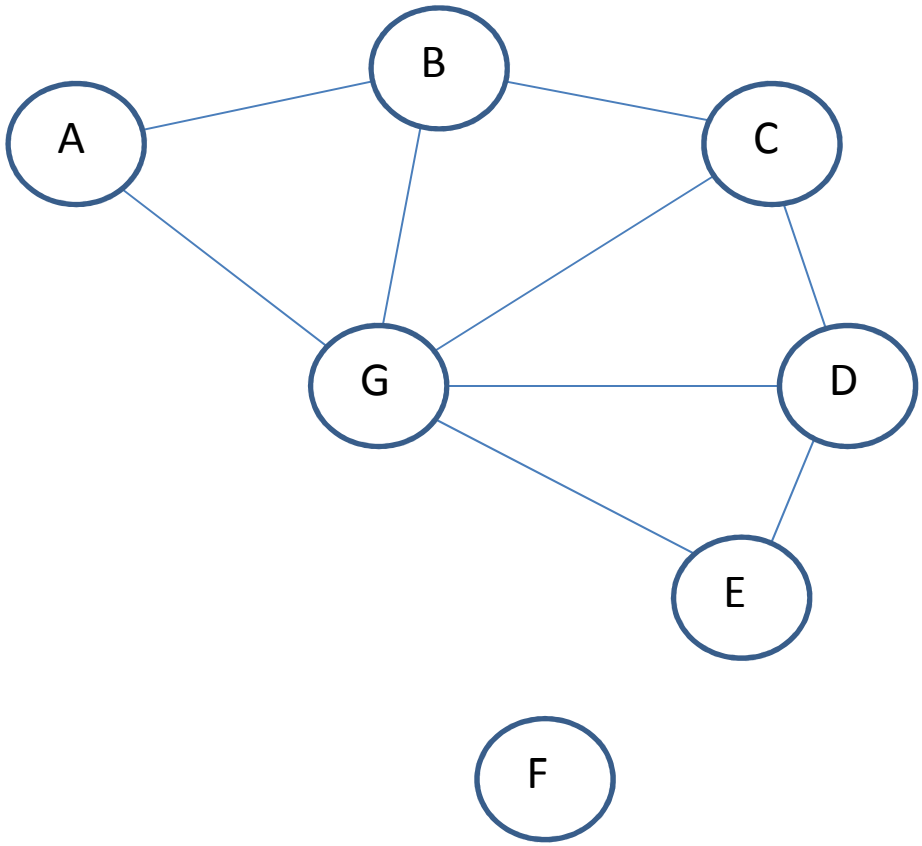


Structure du problème
Un arbre

- Algorithme ARBRE(csp) // canevas
  1. Choisir une variable racine, et trier les variables en un ordre compatible avec l'arbre (soit $V_1, V_2, \ldots, V_n$), de façon que le parent de la variable V_i soit avant elle.
  2. POUR *j* de *n* à 2, appliquer l'arc-consistance à chaque arc (V_i, V_j) avec V_i le parent de V_j (en enlevant des valeurs du domaine de V_i).
  3. POUR *j* de 1 à *n*, assigner n'importe quelle valeur à V_j consistante avec celle de son parent V_i.
- Complexité en temps : $O(nd^2)$

  Si un ensemble de contraintes est un arbre, alors il peut être résolu par un algorithme de complexité linéaire en temps.

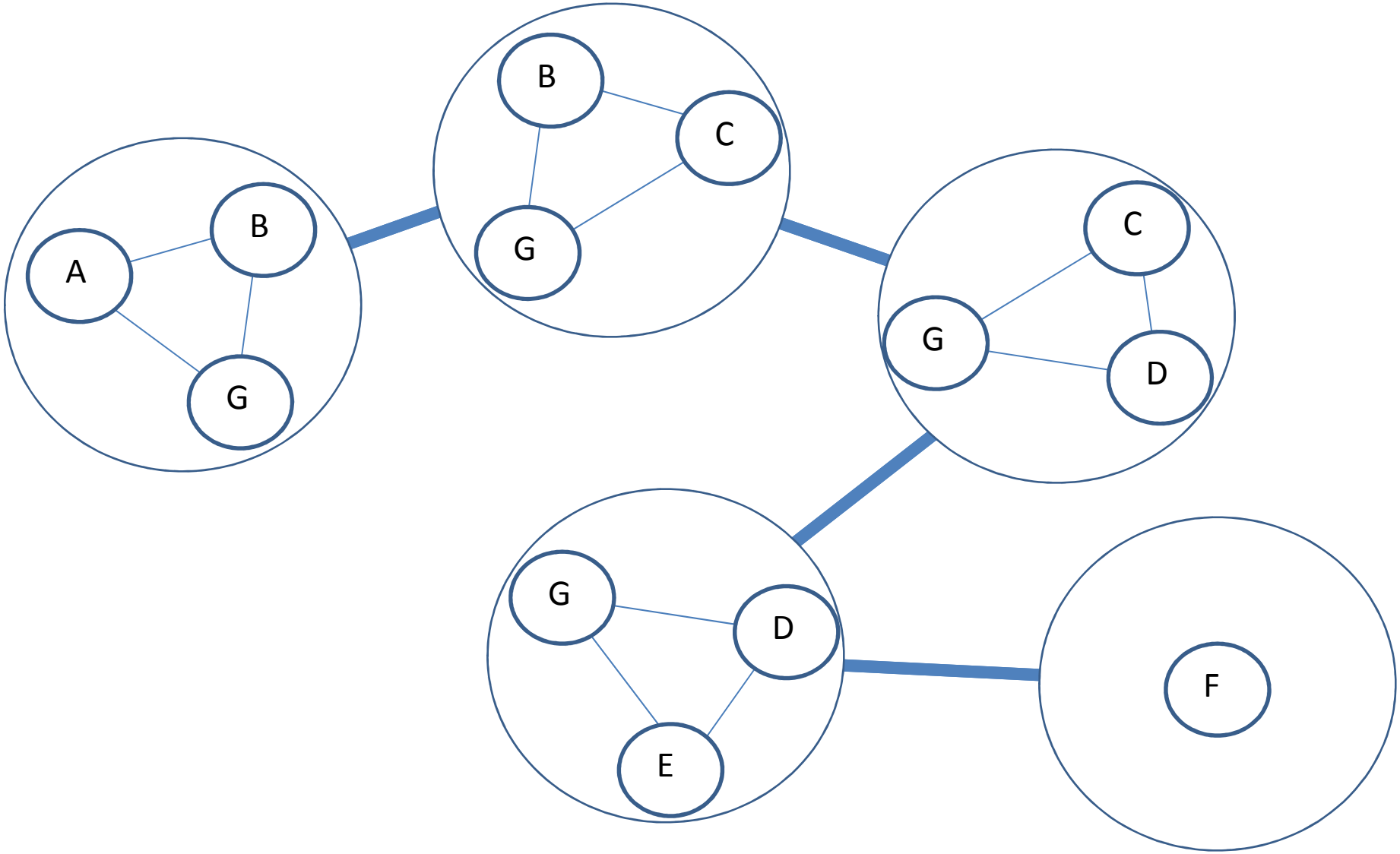
Structure du problème
Se ramener à un arbre : enlever des variables



Structure du problème
Se ramener à un arbre : enlever des variables

- Etapes de l'algorithme ENSEMBLE-COUPES(**csp**)
  1. CHOISIR un sous ensemble ***S*** de *Variables(**csp**)*, tel que la structure des contraintes devient un arbre après enlèvement de ***S***.
  2. POUR TOUTE assignation ***A*** de variables dans ***S*** qui satisfait les contraintes dans ***S***
     a) Enlever du domaine des variables restantes toutes les valeurs inconsistantes avec ***A***
     b) SI le CSP restant à une solution, ALORS la retourner avec ***A***.
- Complexité en temps :
  - En posant ***c*** = Card(***S***), $O(d^c (n - c)d^2)$
  - Trouver le plus petit ensemble ***S*** est difficile …

Structure du problème
Se ramener à un arbre : décomposition arborescente



Structure du problème
Se ramener à un arbre : décomposition arborescente

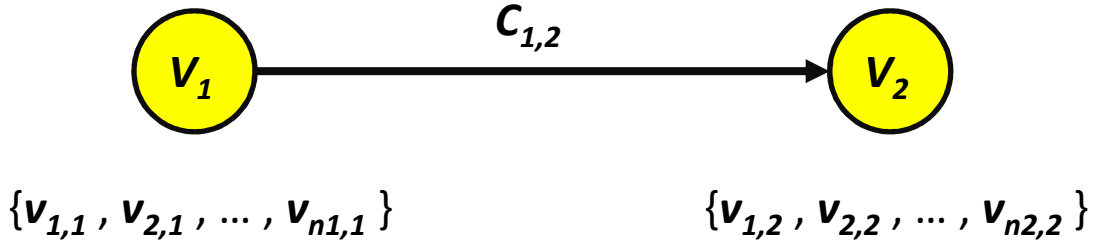
- Comment faire une décomposition arborescente :
  - Toute variable du problème apparait dans au moins un sous problème.
  - Si les variables V_i et V_j sont reliées par une contrainte, alors V_i et V_j (et la contrainte) apparaissent ensemble dans au moins un sous problème.
  - Si la variable V_i apparait dans deux sous problèmes de l'arbre, alors elle apparait dans le chemin entre ces deux sous problèmes.
- Etapes de l'algorithme **MACRO-VARIABLES**(***csp***)
  1. Résoudre chaque sous problème du ***csp***

     SI pas de solution, ALORS pas de solution pour le problème global ***csp***.
  2. Appliquer **ARBRE**() sur les macro-variables (sous-problèmes, assignations) du ***csp***.
- Complexité en temps :
  - Soit la largeur de l'arbre ***w*** :

    $\boldsymbol{w} = \min_{\boldsymbol{décomposition}} (\max_i$ (taille d'un sous problème $\boldsymbol{csp_i}$ selon ***décomposition***) – 1)
  - $O(\boldsymbol{n\ d^{w+1}})$
  - Mais trouver une décomposition de largeur minimale est difficile …

Rappel : consistance d'arc

- Arc-consistance ou consistance d'arc : pour des contraintes binaires orientées, l'arc de la variable V_1 vers la variable V_2 est arc-consistant ssi pour chaque valeur $v_{i,1}$ de V_1, il existe au moins une valeur $v_{i,2}$ de V_2 qui est consistante avec $v_{i,1}$ selon cet arc.



- Complexité en temps : $O(d^2)$
- Que faire si une valeur $v_{i,1}$ servait à une variable V_0 ?

Algorithmes
Consistance de nœud (1-consistance)

Algorithme CONSISTANCE-NOEUD(*csp*)

POUR TOUT ***var*** dans Variable(*csp*)

POUR TOUT ***val*** dans Domaine(***var***)

SI une contrainte unaire sur ***var*** est inconsistante avec ***val***

ALORS enlever ***val*** de Domaine(***var***)

- Complexité : *O(nd)*

Algorithmes
Consistance d'arc : AC-1

- **Algorithme AC-1**(***csp***)

  Q <- { (***V***$_i$, ***V***$_j$) dans arcs(***csp***) avec ***i*** ≠ ***j*** }

  REPETER

  | ***change*** <- FALSE

  | POUR TOUT (***V***$_i$, ***V***$_j$) dans ***Q*** FAIRE

  | | ***change*** = REVISE(***V***$_i$, ***V***$_j$) ou ***change***

  JUSQU'A CE QUE not(***change***)

- Si une seule révision sur un arc se produit, tous les arcs seront reconsidérés à la prochaine itération. Alors que les seuls arcs impactés par la révision de ***V***$_k$ sont les arcs (***V***$_l$, ***V***$_k$) parce qu'une valeur support dans le domaine de ***V***$_l$ peut avoir été enlevée.

Algorithmes
Consistance d'arc : AC-1

- **Algorithme REVISE** (V_i, V_j) // ENLEVE-VALEURS-DOMAINE

  est-enleve <- FALSE

  POUR TOUT v_i dans Domaine(V_i) FAIRE

  | SI il n'existe pas de v_j dans Domaine(V_j) tel que (v_i, v_j) est consistant
  | ALORS
  | | enlever v_i de Domaine(V_i)
  | | **est-enleve** <- TRUE

  return **est-enleve**

- Enlève du domaine de V_i les valeurs inconsistantes avec l'arc ($V_{i,}$) et renvoie TRUE si au moins une valeur a été enlevée (l'arc a été révisé).
- Complexité en temps : O(d^2).

Algorithmes
Consistance d'arc : AC-3

- **Algorithme AC-3**(csp)

  $\boldsymbol{Q}$ <- { ($\boldsymbol{V_i}$, $\boldsymbol{V_j}$) dans arcs(*csp*) avec $i \neq j$ }
  TANT QUE $\boldsymbol{Q} \neq \emptyset$ FAIRE
  | ($\boldsymbol{V_i}$, $\boldsymbol{V_j}$) <- POP_FRONT($\boldsymbol{Q}$)
  | SI REVISE($\boldsymbol{V_i}$, $\boldsymbol{V_j}$)
  | ALORS POUR TOUT $\boldsymbol{V_k}$ dans Voisinage($\boldsymbol{V_i}$) \ { $\boldsymbol{V_j}$ } FAIRE
  | | ajouter ($\boldsymbol{V_k}$, $\boldsymbol{V_i}$) à $\boldsymbol{Q}$

- Complexité en temps : $O(n^2d^3)$
  - Un CSP binaire a au plus $O(n^2)$ arcs ; l'arc (V_k, V_i) peut être ajouté au pire d fois parce que V_i a au plus d valeurs à enlever ; vérifier la consistance d'un arc par REVISE() peut être fait en $O(d^2)$.
- Lorsque AC-3 révise un arc pour la 2e fois, il re-teste des valeurs dont on sait déjà si elle sont consistantes ou pas.
  - Algorithme AC-4 qui considère des paires de valeurs
  - Et AC-5, AC-6, AC-7, …

Algorithmes
Consistance d'arc : AC-4

- **Algorithme INITIALISE**(*csp*)

  Q <- {}

  POUR TOUT ***x***, ***y*** FAIRE ***support***$_{x,y}$ <- {}

  POUR TOUT (***V***$_i$, ***V***$_j$) dans arc(*csp*) FAIRE

  | POUR TOUT ***v***$_i$ dans Domaine(***V***$_i$) FAIRE

  | | ***total*** <- 0

  | | POUR TOUT ***v***$_j$ dans Domaine(***V***$_j$) FAIRE

  | | | SI (***v***$_i$, ***v***$_j$) est consistant avec l'arc (***V***$_i$, ***V***$_j$) ALORS

  | | | | ***total*** <- ***total*** + 1

  | | | | ***support***$_{j,v_j}$ = ***support***$_{j,\,v_j}$ U (***i***, ***v***$_i$) // v_j de V_j est une valeur support de v_i de V_i

  | | *compteur*[(***i***,***j***), ***v***$_i$] = ***total*** // Nombre de valeurs supports de la valeur v_i

  | | SI *compteur*[(***i***,***j***), ***v***$_i$] == 0 ALORS

  | | | enlever ***v***$_i$ de Domaine(***V***$_i$)

  | | | ***Q*** <- ***Q*** U (***i***, ***v***$_i$) // La variable V_i pour la valeur v_i est à réviser

  return ***Q***

Algorithmes
Consistance d'arc : AC-4

- **Algorithme AC-4**(*csp*)

  Q <- INITIALISE(*csp*)

  TANT QUE ***Q*** non vide FAIRE

  | (***j***, $\boldsymbol{v}_j$) <- POP_FRONT(***Q***)

  | POUR TOUT (***i***, $\boldsymbol{v}_i$) dans $\boldsymbol{support}_{j, vj}$ FAIRE

  | | *compteur*[(***i***, ***j***), $\boldsymbol{v}_i$] <- *compteur*[(***i***, ***j***), $\boldsymbol{v}_i$] – 1

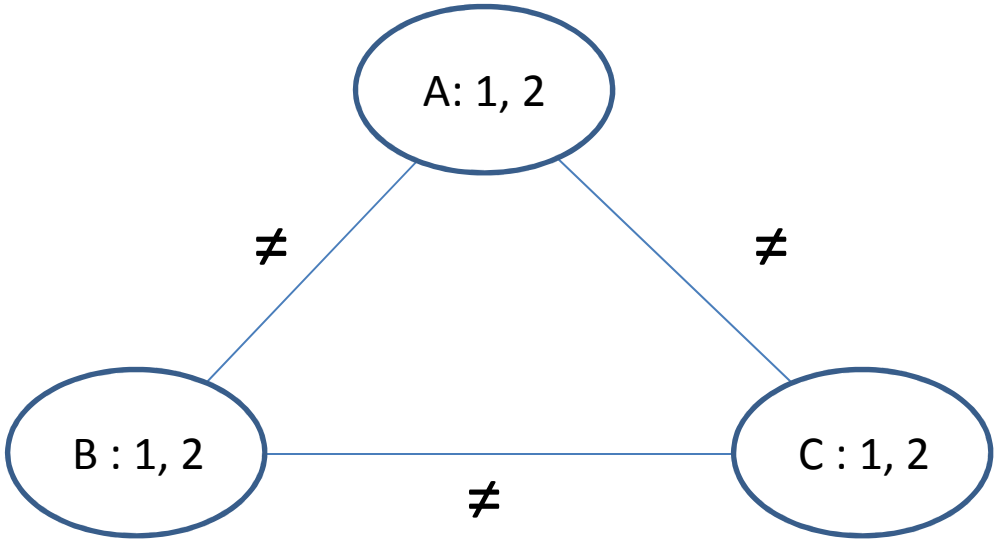
  | | SI *compteur*[(***i***, ***j***), $\boldsymbol{v}_i$] == 0 ET $\boldsymbol{v}_i$ est dans Domaine($\boldsymbol{V}_i$) ALORS

  | | | enlever $\boldsymbol{v}_i$ de Domaine($\boldsymbol{V}_i$)

  | | | ***Q*** <- ***Q*** U (***i***, $\boldsymbol{v}_i$)

- Après initialisation, AC-4 n'effectue une re-révision que pour les paires de valeurs affectées par des révisions précédentes.
- Complexité en temps :
  - INITIALISE() est en O(n^2d^2), AC-4 sans INITIALISE() est en O(n^2d)
  - Donc la complexité en temps d'AC-4 est $O(n^2d^2)$ // Gain d'un facteur *d*

Algorithmes
Consistance de chemin

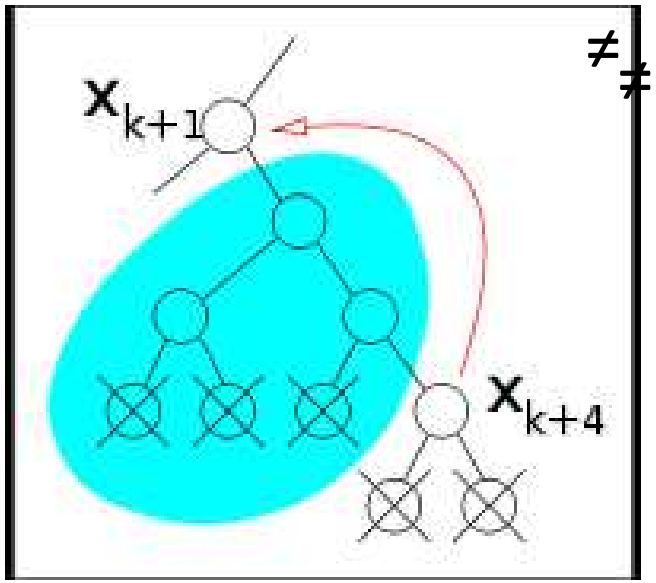


- Le graphe de contraintes ci-dessus est arc-consistant.

  Pourtant il n'y a pas d'affectation possible des variables …

- Un CSP inclut 3SAT, aussi on ne peut pas s'attendre à trouver un algorithme de complexité polynomiale pour prouver la consistance.
  - Notion de consistance de chemin (ou 3-consistance)
  - Plus généralement, notions de k-consistance et k-consistance forte

Algorithmes
Backtracking et Backjumping



Algorithmes
Le retour-arrière (backtracking)

- Algorithme **RECHERCHE-RETOUR-ARRIERE**(*csp*)
  return RETOUR-ARRIERE-RECURSIF({}, *csp*)

- Algorithme **RETOUR-ARRIERE-RECURSIF**(*assignation*, *csp*)
  SI *assignation* est totale ALORS return *assignation*
  var <- CHOISIR-VARIABLE-NONAFFECTEE(Variables(*csp*), *assignation*, *csp*)
  POUR TOUT *val* dans ORDONNE-VALEURS-DOMAINE(*var*, *assignation*, *csp*)
  > SI *va*l est consistante avec *assignation* suivant Contraintes(*csp*) ALORS
  > > ajoute (*var* = *val*) à *assignation*
  > > *resultat* <- **RETOUR-ARRIERE-RECURSIF**(*assignation*, *csp*)
  > > SI *resultat* ≠ ECHEC ALORS return *resultat*
  > > enleve (*var* = *val*) de *assignation*

  return ECHEC

Algorithmes
Le saut arrière (backjumping)

- **Algorithme SAUT-ARRIERE-RECURSIF(*assignation*, *csp*)**

  SI ***assignation*** est totale ALORS return ***assignation***

  var <- CHOISIR-VARIABLE-NONAFFECTEE(Variables(***csp***), ***assignation***, ***csp***)

  POUR TOUT ***val*** dans ORDONNE-VALEURS-DOMAINE(***var***, ***assignation***, ***csp***)

  FAIRE

  SI ***val*** est consistante avec ***assignation*** suivant Contraintes(***csp***) ALORS

  ajoute ***var*** dans le conflict-set des variables reliées par une contrainte

  ajoute (***var*** = ***val***) à ***assignation***

  resultat <- SAUT-ARRIERE-RECURSIF(***assignation***, ***csp***)

  SI ***resultat*** ≠ ECHEC ALORS return ***resultat***

  enleve ***var*** du conflict set des variables précédentes

  SI ***var*** ∈ conflict-set(***resultat***) ALORS

  enleve (***var*** = ***val***) de ***assignation***

  SINON return **resultat**

  return ECHEC U conflict-set(**var**)

Conclusion

- La structure de l'ensemble des contraintes compte :
  - Problèmes indépendants.
  - Problèmes en forme d'arbre.
  - Se ramener à un arbre :
    - par l'enlèvement de variables (coupes);
    - par regroupement de variables (macro-variables).
- La consistance de nœud vérifie les contraintes unaires.
- La consistance d'arc, sur les contraintes binaires orientées, s'améliore en complexité de AC-1 à AC-3 puis à AC-4 jusqu'à AC-8.
- Le saut arrière améliore le retour arrière chronologique, tout en ne perdant pas de solution.

Exercices de modélisation

- Donner une formulation précise en terme de CSP des problèmes suivants :
  1. Placer des petits rectangles dans un grand rectangle, sans chevauchement.
  2. On a *n* professeurs et *m* salles de classe, une liste de cours à donner, et une liste de créneaux horaires. Chaque professeur peut donner certains cours seulement. Constituer l‘affectation.