Intelligence Artificielle

Satisfaction et Propagation de Contraintes

Stéphane Marchand-Maillet

Contenu

- Types de problèmes
- Représentation de contraintes
- Problèmes de Satisfaction de Contraintes
- Algorithmes de résolution

Problème des *k*-reines

Problème:

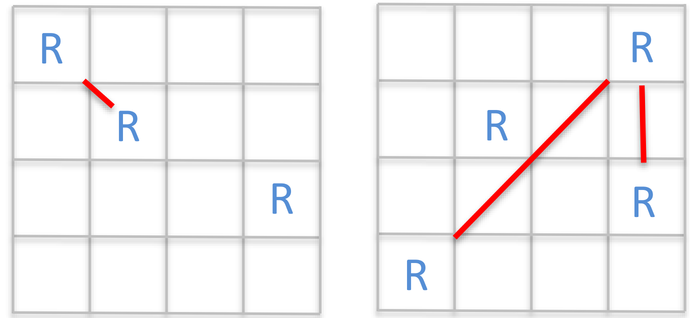
Placer un maximum de reines (d'échec) sur un échiquier de taille *kxk* sans qu'elles s'attaquent entre elles

→On peut en placer au maximum une seule par colonne/ligne/diagonale → *k*-reines

	R	X	X	X
R	X	X		
R	X		X	
R	X			X

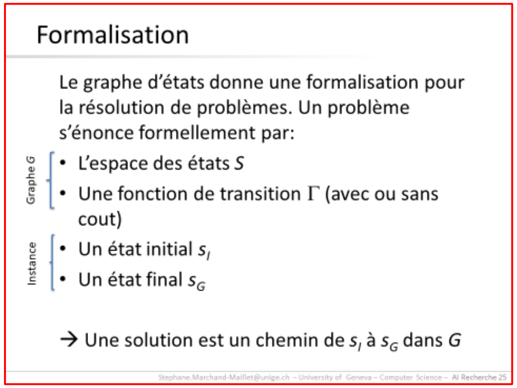
$k = 4$

Problème de recherche (1)



- Etats S:
  - Configurations de 0…k reines sur l'échiquier
- Transitions Γ:
  - $s' \in \Gamma(s)$ si s' affecte une reine de plus que s
- Etat initial s_I:
  - Echiquier vide
- Etat final s_G:
  - k reines sans attaque mutuelle

→ $k^2!/(k^2-k)!$ états


Formalisation

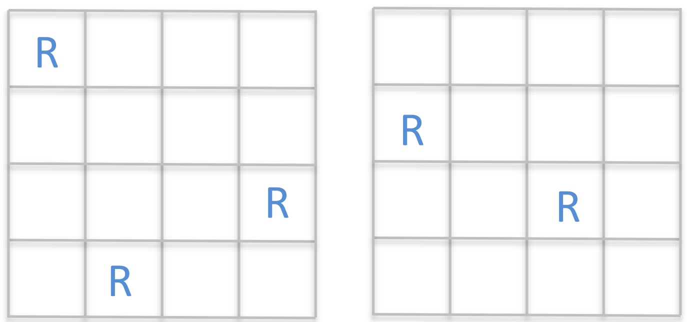
Le graphe d'états donne une formalisation pour la résolution de problèmes. Un problème s'énonce formellement par:

Graphe G
- L'espace des états S
- Une fonction de transition Γ (avec ou sans cout)

Instance
- Un état initial s_I
- Un état final s_G

→ Une solution est un chemin de s_I à s_G dans G

Problème de recherche (2)



- Etats *S*:
  - Configurations valides de 0…*k* reines sur l'échiquier
- Transitions Γ:
  - $s' \in \Gamma(s)$ si *s'* affecte une reine de plus que *s*
- Etat initial s_I:
  - Echiquier vide
- Etat final s_G:
  - *k* reines sans attaque mutuelle

→Beaucoup <u>moins</u> d'états
mais comment atteindre les états *valides*?

Discussion

- On a des contraintes mutuelles:
  - La position d’une reine contraint la position des autres
  - On veut satisfaire toutes les contraintes
- On va utiliser la propagation de contraintes:
  - Pour réduire l’espace de recherche
  - Déduire les états non-valides
- On peut utiliser des heuristiques:
  - Pour favoriser les états valides
  - Pour accélérer la recherche

Représentation de contraintes

Un problème de satisfaction de contraintes (PSC) est représenté par:

- Un ensemble de variables

$$X=(x_1,\ldots, x_N)$$

chaque variable x_i ayant un domaine de valeurs admissibles D_i (en général discret et fini)

- Un ensemble de contraintes sur les variables:

$$C = (c_1,\ldots, c_M)$$

chaque contrainte c_j est une proposition logique (égalité, inégalité, …) qui s'applique sur les variables X

Résolution de PSC

Solution: on cherche:

$$X^* \text{ tel que } x_i^* \in D_i \text{ et True}(c_j) \text{ pour tout } i, j$$

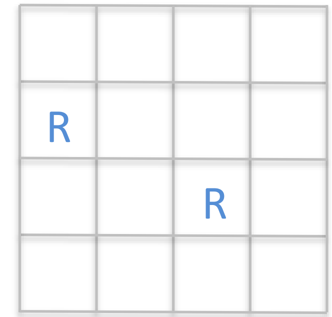
Un affectation valide est une affectation de certaines variables dans leurs domaines et satisfaisant les contraintes

Une solution est une affectation valide affectant toutes les variables

On va explorer (par recherche) l'espace des affectations pour construire une solution au PSC

On doit définir une stratégie de recherche

Exemple de PSC



k-reines:

- $X=(x_{ij})_{i,j=1..k}$ valeurs binaires des k^2 cases

- Domaines : $D_{ij} = \{0,1\}$ pour chaque $i,j = 1\ldots k$

- Contraintes:

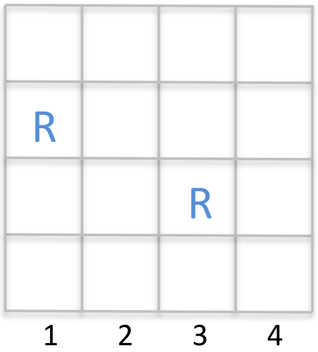
  $x_{ij} = 1 \Rightarrow x_{im} = 0$ for all $m \neq j$ (contraintes binaires)

  $x_{ij} = 1 \Rightarrow x_{mj} = 0$ for all $m \neq i$ (contraintes binaires)

  $x_{ij} = 1 \Rightarrow$ diagonales = 0 (contraintes binaires)

  $\Sigma\, x_{ij} = k$ (contrainte multiple)

Exemple de PSC



k-reines (autre formulation) :

- $X=(x_i)_{i=1..k}$ valeurs de ligne pour les *k* colonnes

- Domaines : $D_i = \{1,\ldots,k\}$ pour chaque $i = 1\ldots k$

- Contraintes:

  $x_i = m \Rightarrow x_j \neq m$ for all $j \neq i$ (contrainte binaire)

  De même pour les diagonales (contrainte binaire)

Exemple (à construire)

Modéliser (Enigme d'Einstein):

On a 14 indices. Il y a cinq maisons, cinq hommes de nationalité et de professions différentes habitent avec leur animal préféré cinq maisons de couleurs différentes où ils boivent leur boisson préférée.

1. L'Anglais habite la maison rouge.
2. L'Espagnol adore son chien.
3. L'Islandais est ingénieur.
4. On boit du café dans la maison verte.
5. La maison verte est située immédiatement à gauche de la maison blanche.
6. Le sculpteur possède un âne.
7. Le diplomate habite la maison jaune.
8. Le Norvégien habite la première maison à gauche.
9. Le médecin habite la maison voisine de celle où demeure le propriétaire du renard.
10. La maison du diplomate est voisine de celle où il y a un cheval.
11. On boit du lait dans la maison du milieu.
12. Le Slovène boit du thé.
13. Le violoniste boit du jus d'orange.
14. Le Norvégien demeure à côté de la maison bleue.

Questions:

- Qui boit de l'eau ?
- Qui élève le zèbre ?

Source: http://enigmesetdevinettes.com/enigme/enigme-einstein/

Modélisation

Exemple (à construire)

Modéliser (Enigme d'Einstein):

On a 14 indices. Il y a cinq maisons, cinq hommes de nationalité et de professions différentes habitent avec leur animal préféré cinq maisons de couleurs différentes où ils boivent leur boisson préférée.

1. L'Anglais habite la maison rouge.
2. L'Espagnol adore son chien.
3. L'Islandais est ingénieur.
4. On boit du café dans la maison verte.
5. La maison verte est située immédiatement à gauche de la maison blanche.
6. Le sculpteur possède un âne.
7. Le diplomate habite la maison jaune.
8. Le Norvégien habite la première maison à gauche.
9. Le médecin habite la maison voisine de celle où demeure le propriétaire du renard.
10. La maison du diplomate est voisine de celle où il y a un cheval.
11. On boit du lait dans la maison du milieu.
12. Le Slovène boit du thé.
13. Le violoniste boit du jus d'orange.
14. Le Norvégien demeure à côté de la maison bleue.

Questions:

- Qui boit de l'eau ?
- Qui élève le zèbre ?

Source: [illegible]

Stephane.Marchand-Maillet – Université de Genève | Intelligence Artificielle | AI Contraintes - 11

- Variables?
- Domaines?
- Contraintes?

Classification des PSC

- Le domaine de chaque variable est discret et fini:
  - Le PSC est fini
  - La solution peut se trouver par recherche
    - pire des cas: énumération

- Le domaine de certaines variables est infini:
  - On ne peut pas énumérer les affectations
  - Résolution similaire à la programmation linéaire

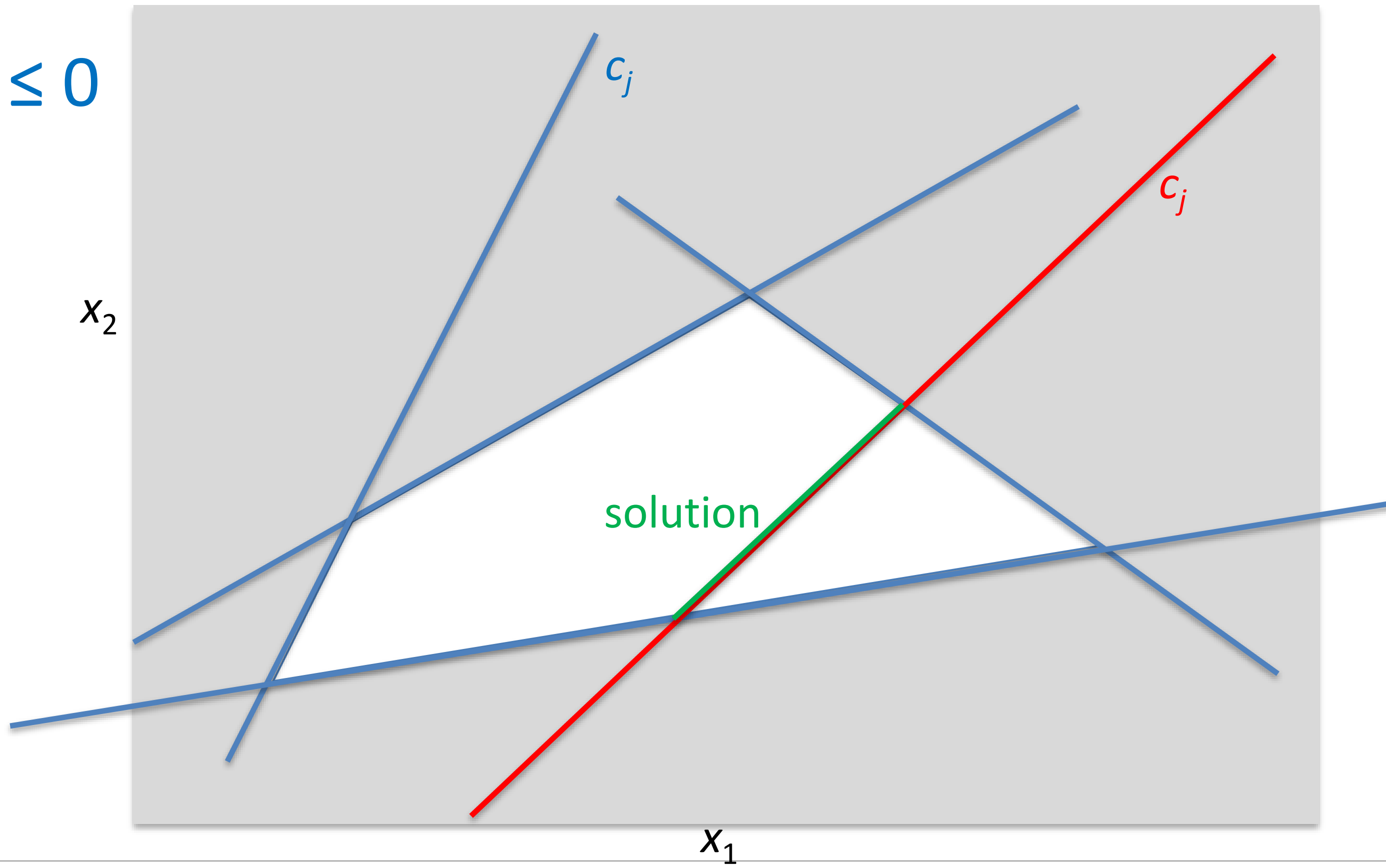
Dans ce cours, on ne traite que des PSC finis

Rappel: Programmation linéaire

$X=(x_i)_{i=1..k}$, $D_i = \mathbb{R}$ pour chaque $i = 1...k$

$c_j : \Sigma\, a_{ij}x_i + b_j = 0$

$c_j : \Sigma\, a_{ij}x_i + b_j \leq 0$

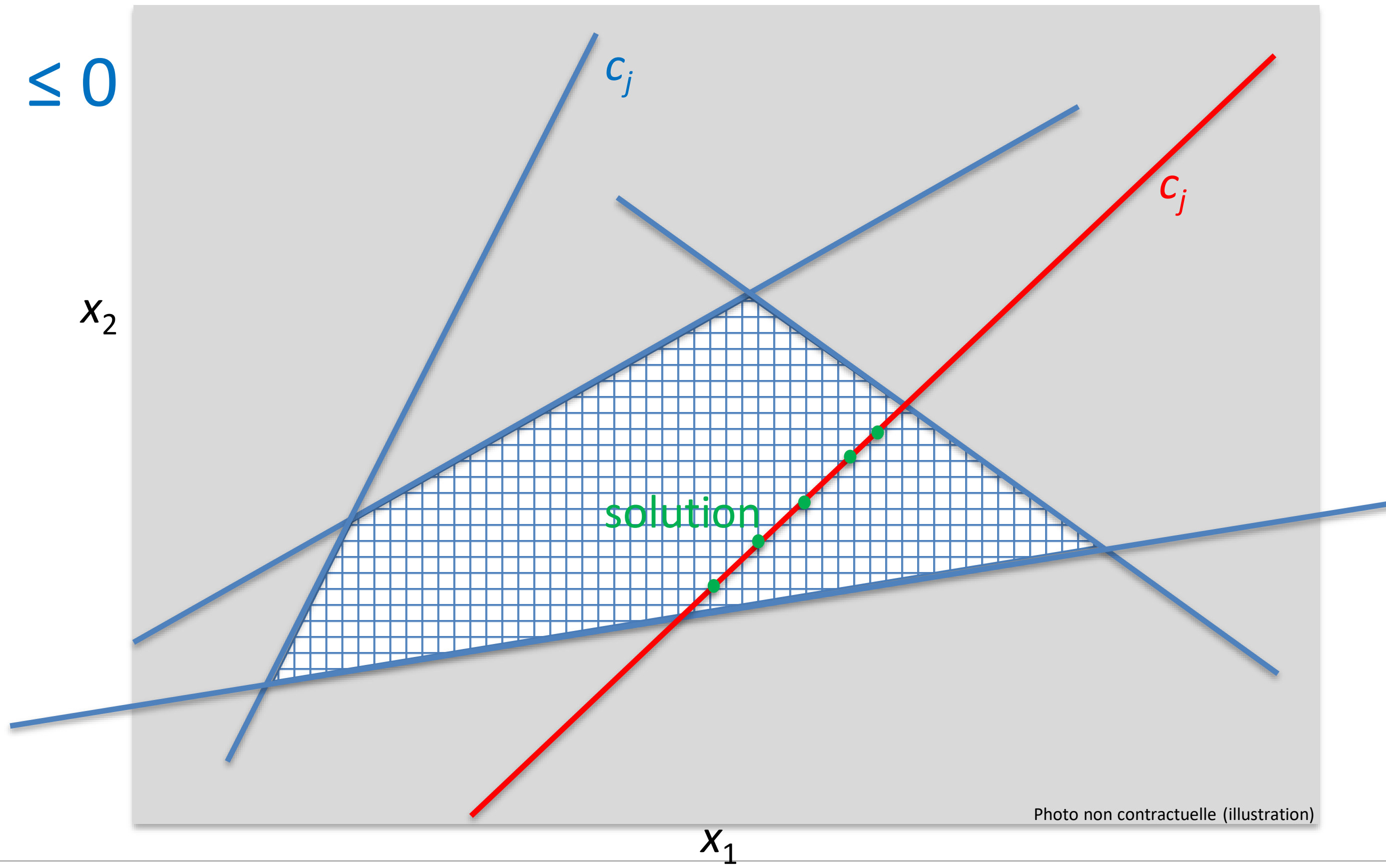


Analogie

$X=(x_i)_{i=1..k}$, D_i = {(discret fini)} pour chaque i = 1…k

$c_j : \Sigma_i\, a_{ij}x_i + b_j = 0$

$c_j : \Sigma_i\, a_{ij}x_i + b_j \leq 0$



Graphe de contraintes

Si les contraintes sont binaires, les variables sont liées 2 à 2:

→On crée le graphe

$$G=(V,E)=(\text{“variables”}, \text{“contraintes”})$$

- Les composantes connexes de ce graphe représentent des groupes de variables liées

→ On peut les traiter indépendamment

- On peut utiliser ce graphe pour suivre la propagation de contraintes

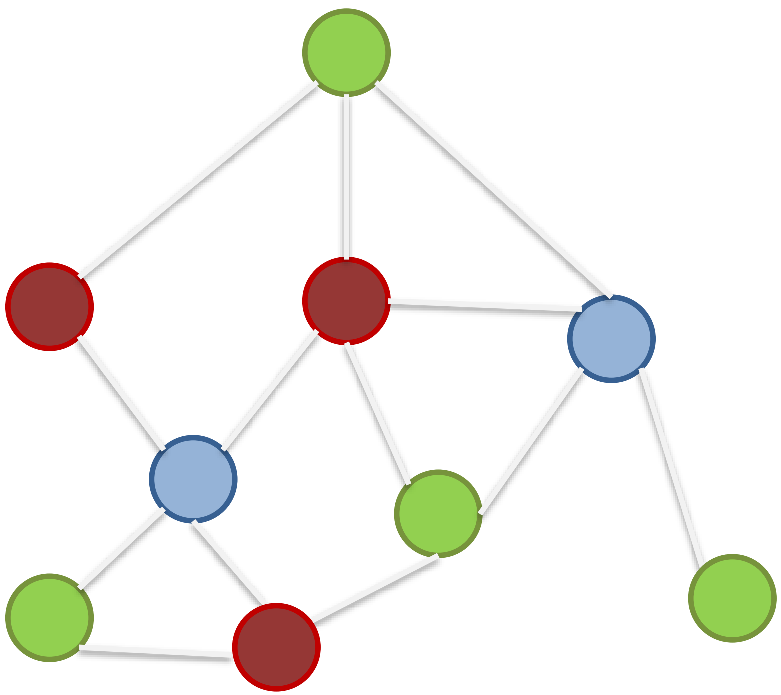
Exemple trivial

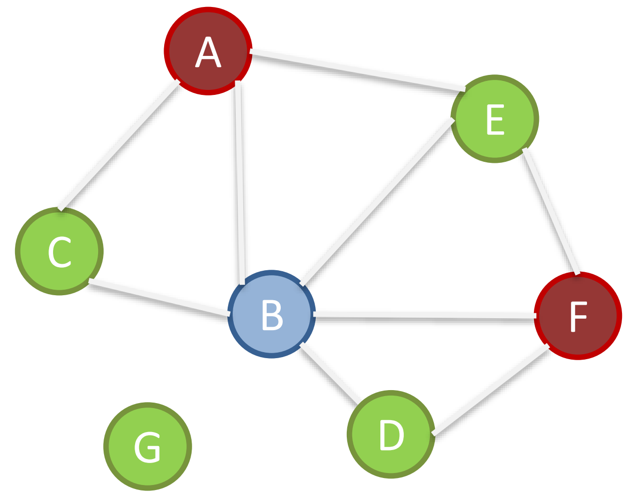
Coloration de graphe:

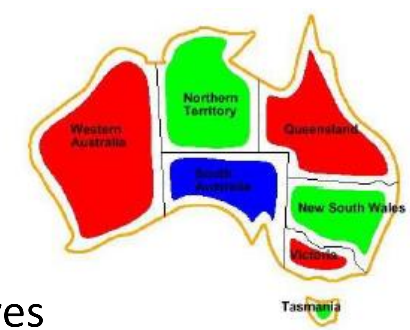
Etant donné un graphe, colorier les nœuds sans que 2 nœuds voisins soient de la même couleur

- PSC: variables? domaines?
- Par définition un PSC binaire (couleurs de nœuds différentes 2 à 2)
- Par définition soutenu par un graphe
- Si le graphe a des composantes connexes, on peut les traiter indépendamment

Coloration de graphe

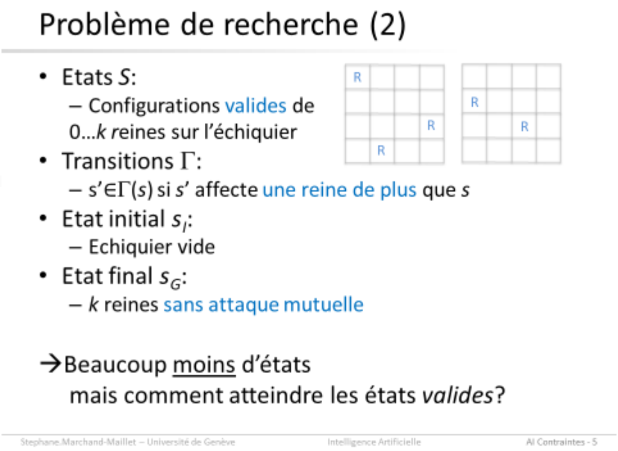






Applications: coloration de carte, cooccurences, …, contraintes exclusives

Recherche de solution



Espace d'états:

- Espace des affectations valides

Transition:

- Affecter une nouvelle variable en satisfaisant les contraintes

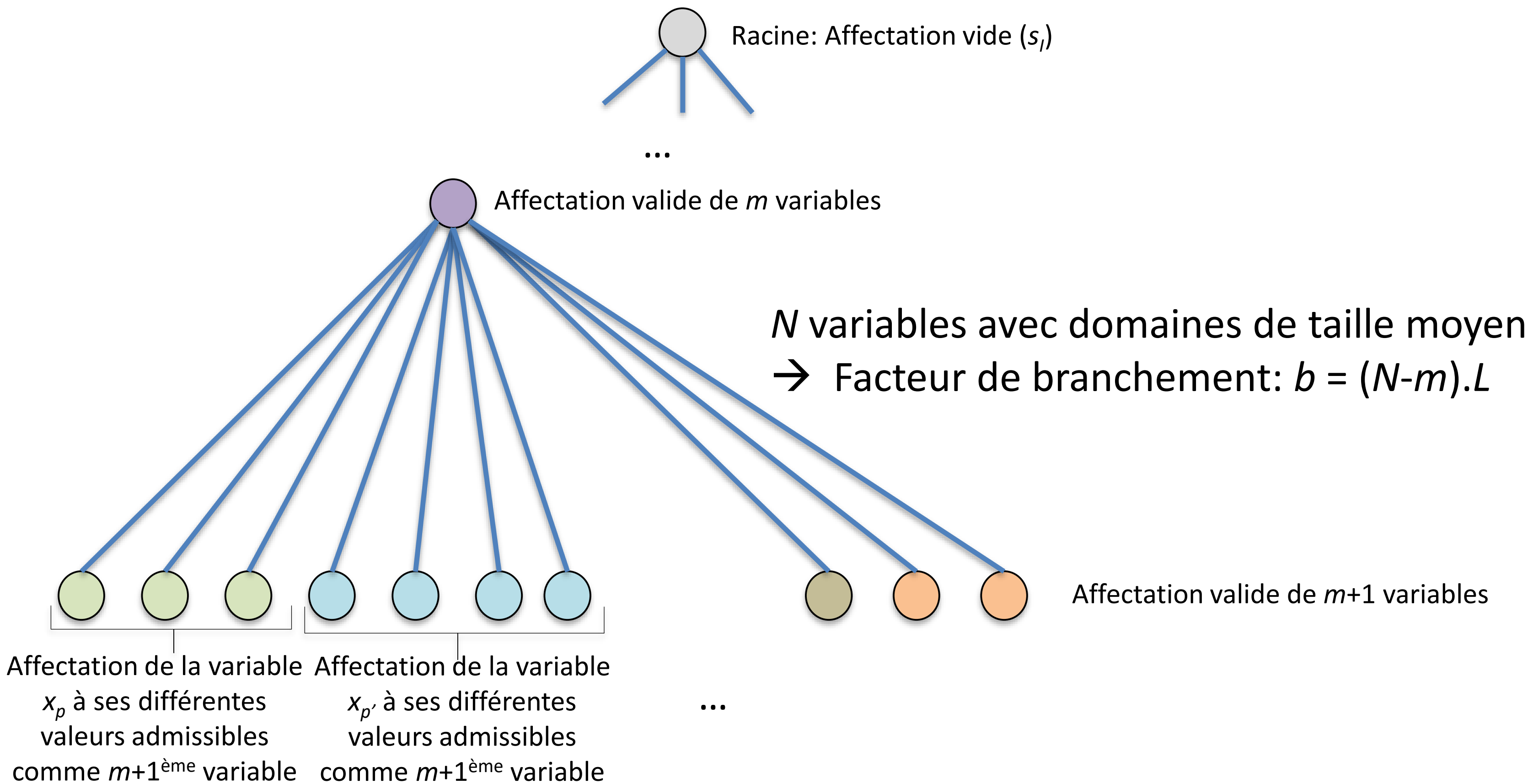
Etat initial s_I:

- Affectation vide (aucune variable affectée)

Etat final s_G:

- Une solution

Arbre de recherche (PSC)



N variables avec domaines de taille moyenne *L*:

→ Facteur de branchement: $b = (N\text{-}m).L$

On obtient un facteur de branchement ingérable

Commutativité des affectations

Remarque:

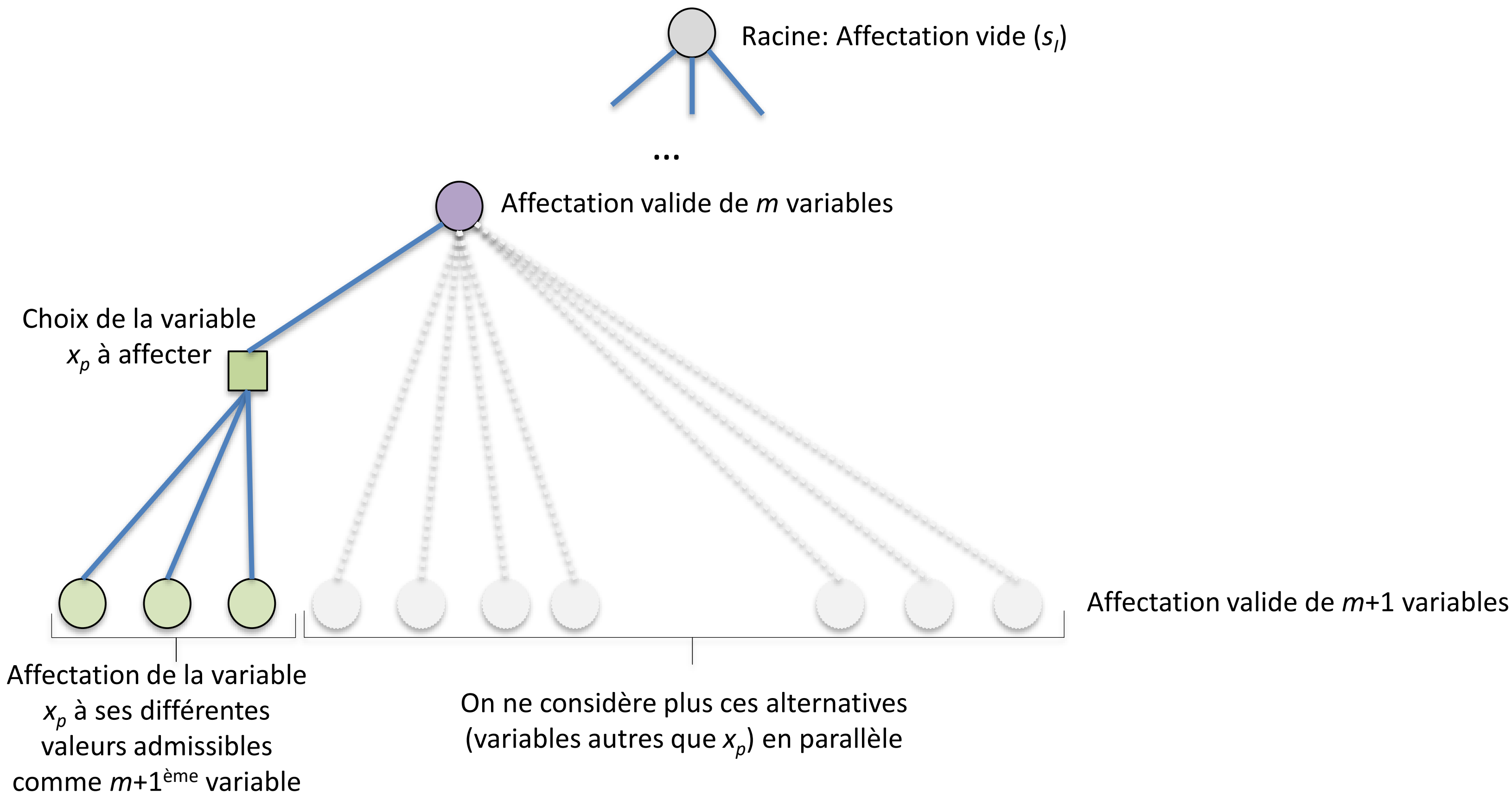
Une affectation valide ne dépend pas de l'ordre dans lequel elle a été construite

Transition:

- Choix de une variable x_p à affecter
- Affectation de x_p à ses valeurs admissibles (D_p)

→ Réduction du facteur de branchement

Arbre de recherche (PSC)



N variables avec domaines de taille moyenne L:

→ Facteur de branchement: $b = L$

Algorithme de *Backtracking*

PSC_BACKTRACKING(A: affectation)

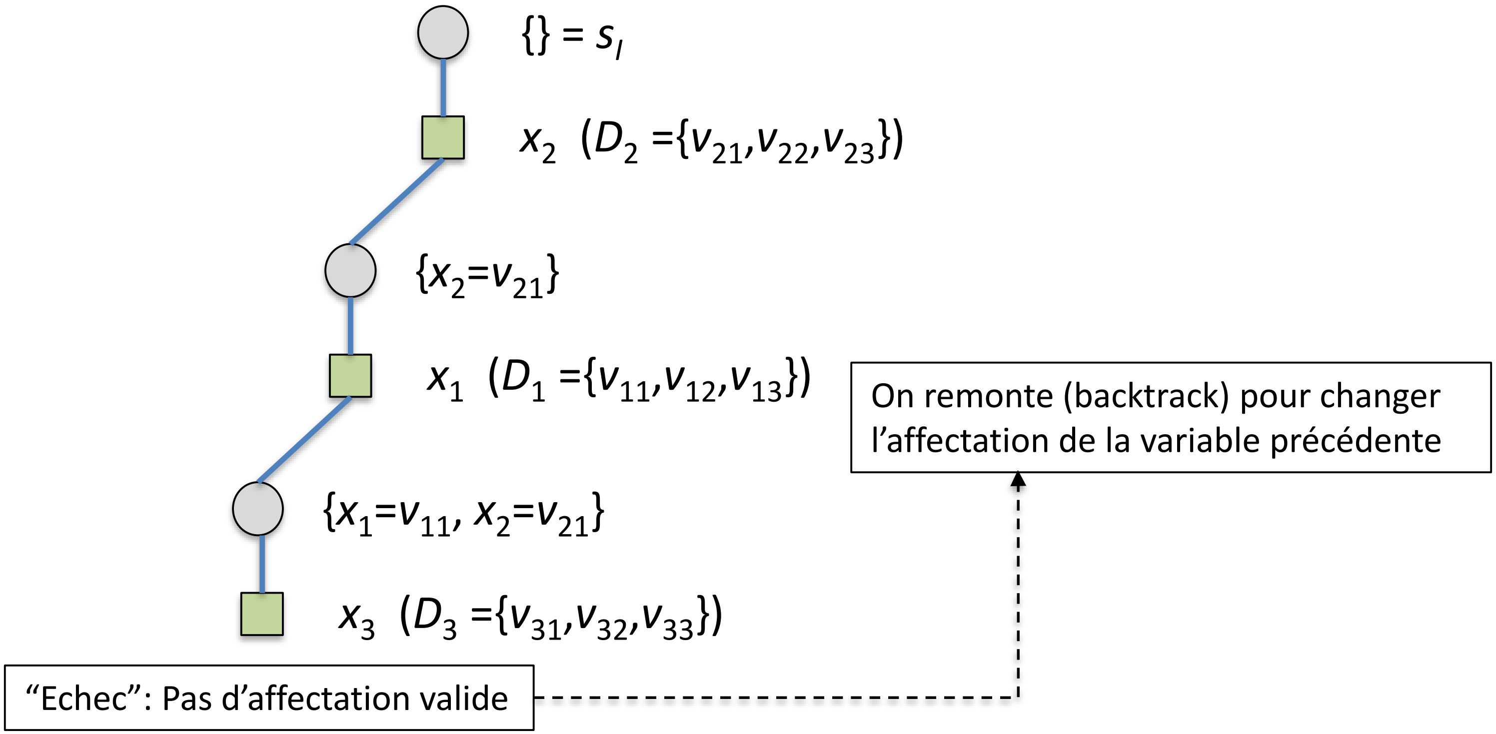
1. Si $A = s_G$ alors retourner A
2. Sélectionner une variable x_p non affectée
3. Pour chaque valeur v_{pi} de D_p faire:
   - Ajouter x_p <- v_{pi} dans A
   - Si A est valide alors retourner PSC_BACKTRACKING(A)
     sinon retourner “echec”

Appel: A <- PSC_BACKTRACKING(s_I)

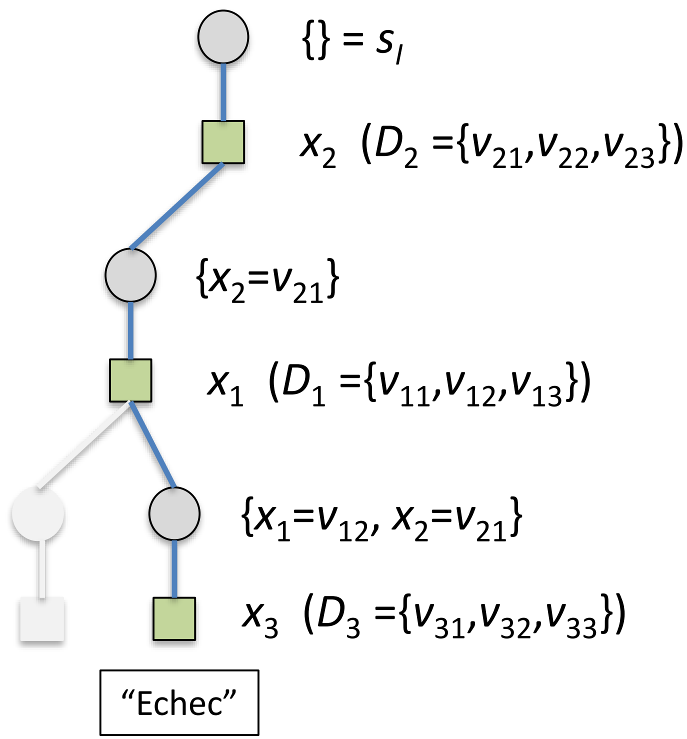
Note: 3. : on suppose un ordre sur les domaines D_p

Exercice: Ecrire la version itérative

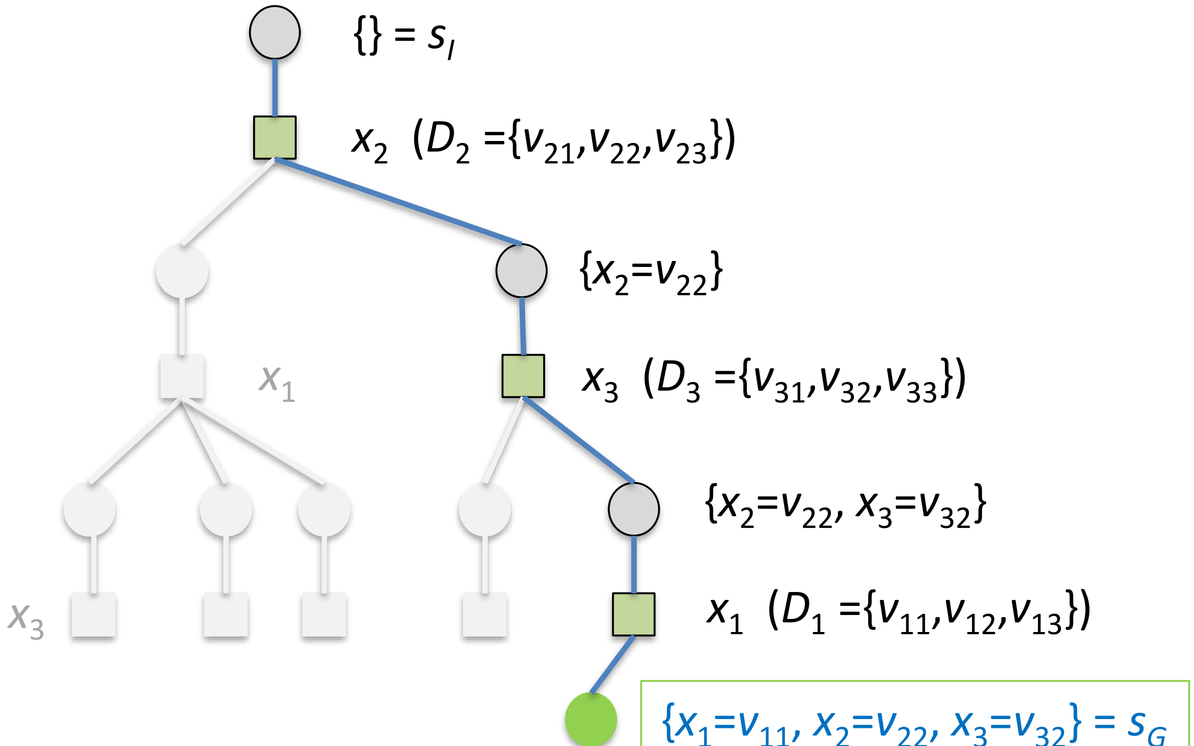
Exemple (3 variables)



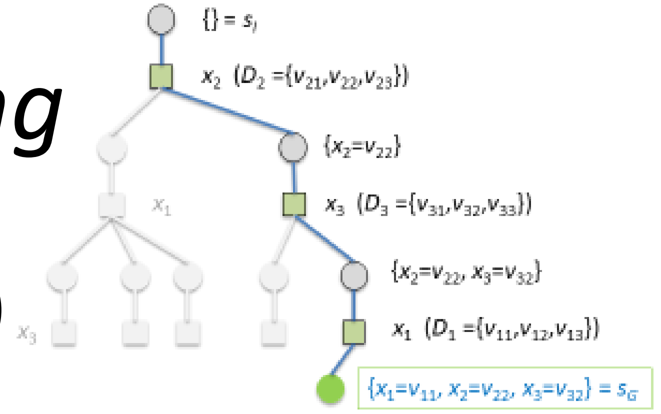
Exemple (3 variables)



Exemple (3 variables)



Algorithme de *Backtracking*



PSC_BACKTRACKING(*A*: affectation)

1. Si $A = s_G$ alors retourner *A*
2. Sélectionner une variable x_p non affectée
3. Pour chaque valeur v_{pi} de D_p faire:
   - Ajouter x_p <- v_{pi} dans *A*
   - Si *A* est valide alors retourner PSC_BACKTRACKING(A)
     sinon retourner “échec”

Aléa:

- Ordre d’affectation des variables (etape 2.)
- Ordre d’affectation des valeurs (etape 3.)

Forward checking

- Affecter une valeur à une variable rend invalides d'autres affectations (par les contraintes)

→On peut prédire les valeurs des variables contraintes et éviter ces affectations

	X		
R	X	X	X
	X		
		X	

x_1=2

<u>Exemple</u>: (4-reines)

X	x_1	x_2	x_3	x_4
D	1 2 3 4	1 2 3 4	1 2 3 4	1 2 3 4
	1	1 2 3 4	1 2 3 4	1 2 3 4
	1	4	1 2 3 4	1 2 3 4
	1	4	2	1 2 3 4

R			X
		R	X
			X
	R		X

Forward checking

- Lors de l'affectation d'une variable, on regarde (en fonction des contraintes) l'impact sur les domaines valides des autres variables non-affectées
- On maintient les domaines valides de l'affectation
- Cela permet de ne pas tester des affectations invalides (boucle 3.)

→ La vérification de validité de l'affectation est faite *a priori* plutôt qu'*a posteriori*

Backtracking adapté

PSC_BACKTRACKING(A: affectation, D: domaines)

1. Si $A = s_G$ alors retourner A
2. Sélectionner une variable x_p non affectée
3. Pour chaque valeur v_{pi} de D_p faire:
   - Ajouter x_p <- v_{pi} dans A
   - $D \leftarrow$ FORWARD_CHECKING(A, D)
   - Si aucun domaine de D n'est vide:
     alors retourner PSC_BACKTRACKING(A, D)
     sinon retourner "échec"

Appel: A <- PSC_BACKTRACKING(s_I, D)

Note: 3. : on suppose un ordre sur les domaines D_p

Exercice: Ecrire la version itérative

Heuristiques

On a toujours les aléas:

- Ordre d'affectation des variables (étape 2.)
  - On diminue le facteur de branchement:
  - Heuristique de la variable la plus contrainte...
    - Variable ayant le plus petit domaine

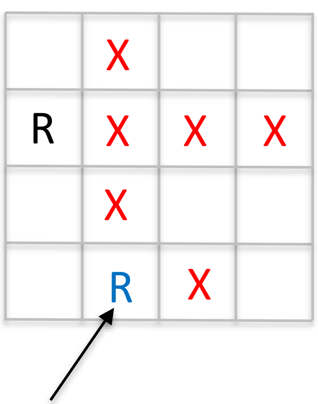
  ...et la plus contraignante
    - Variable apparaissant dans le plus grand nombre de contraintes

- Ordre d'affectation des valeurs (étape 3.)
  - Heuristique de la valeur la moins contraignante
    - Valeur qui élimine le moins de valeurs aux autres
    - Demande un forward-checking pour toutes les valeurs

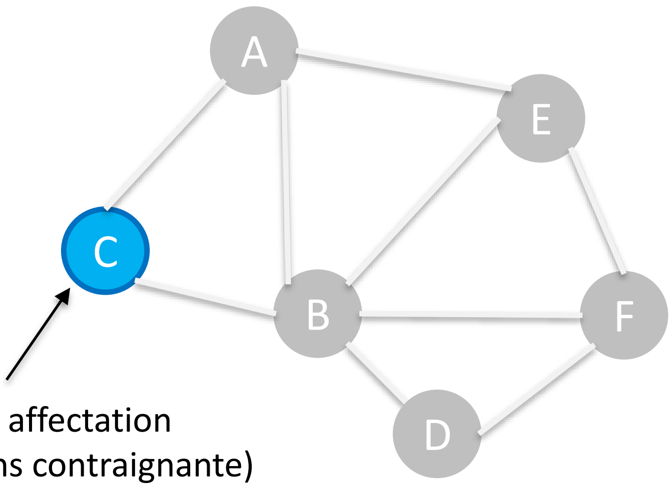
Heuristiques

Heuristique de la variable la plus contrainte et la plus contraignante

Heuristique de la variable la moins contraignante



Nouvelle affectation
(variable la plus contrainte)



Nouvelle affectation
(valeur la moins contraignante)

Backtracking adapté (2)

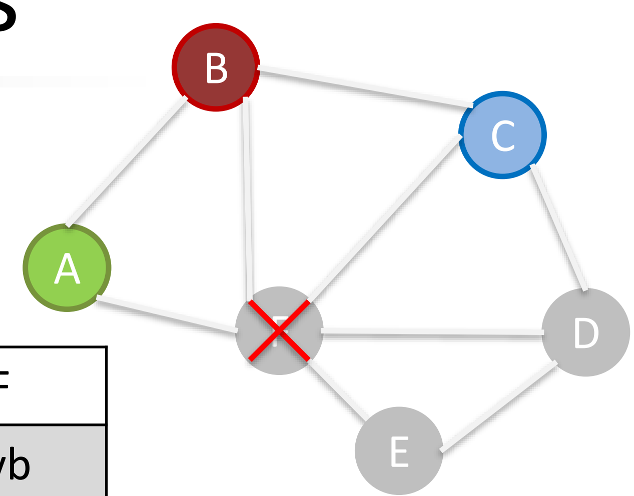
PSC-BACKTRACKING(*A*: affectation, D: domaines)

1. Si $A = s_G$ alors retourner *A*
2. Sélectionner une variable x_p non affectée
   (heuristique sur les variables)
3. Pour chaque valeur v_{pi} de D_p faire:
   (heuristique sur la valeur)
   - Ajouter x_p <- v_{pi} dans *A*
   - $D \leftarrow$ FORWARD_CHECKING(*A, D*)
   - Si aucun domaine de *D* n’est vide:
     alors retourner PSC-BACKTRACKING(A, D)
     sinon retourner “échec”

Appel: *A* <- PSC-BACKTRACKING(s_I, *D*)

Propagation de contraintes

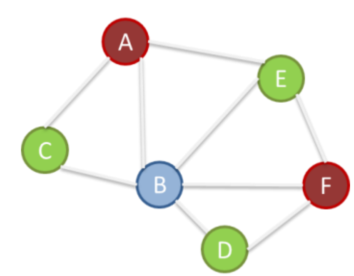
Myopie du *Forward Checking*



X	A	B	C	D	E	F
D	rvb	rvb	rvb	rvb	rvb	rvb
	v	rvb	rvb	rvb	rvb	rvb
	v	rvb	b	rvb	rvb	rvb
	v	r	b	rvb	rvb	~~rvb~~

Contradiction: B et F sont connectés et ont le même domaine
→Echec au prochain pas

Le *Forward Checking* ne le détecte pas
On doit introduire une autre stratégie:
La propagation de contraintes



Consistance

- Anticipation de 1 niveau: pour chaque variable x_i non-affectée dans A on élimine de D_i toute valeur v rendant l'ajout de (x_i <- v) dans A inconsistante avec C: 1-consistance (consistance de nœud)
- Anticipation de 2 niveaux: pour chaque variable x_i non-affectée dans A on élimine de D_i toute valeur v telle que pour (x_i<-v), il existe une variable x_j non affectée pour laquelle aucune affectation (x_j<-w) ne serait 1-consistante : 2-consistance (consistance d'arc)
- Anticipation de 3 niveaux: … 3-consistance (consistance de chemin)
- …

Algorithme AC3 (Arc Consistency, 1977)

C contient des contraintes binaires

```
AC3(D)
pour tout x_p de X faire:
    Q <- {x_q t.q x_p et x_q sont liées par une contrainte de C}
    répéter:
        x_q <- Q.pop()                // voisin de x_p
        si ELIMINER(x_p,x_q)          // garde les paires (v_p, v_q) admissibles
            si D_p est non vide       // alors on propage sinon stop
                alors ajouter dans Q tout x_r différent de x_q et t.q x_r  et x_q sont liées par
                    une contrainte de C
                sinon retourner "échec"
    Tant que Q est non-vide
retourner D
```

```
bool ELIMINER(x_1,x_2)                // maintien de la 2-consistance
changement <- faux
pour tout v_1 de D_1 faire:
    chercher v_2 de D_2 pour satisfaire la contrainte sur x_1 et x_2
    si cet ensemble est vide alors:
        supprimer v_1 de D_1          // on enlève de D_1 les valeurs menant à une contradiction
        changement <- vrai
retourner changement
```

Propriétés de AC3

Algorithme de propagation de contraintes binaires

Complexité (opérations): $O(N.\ m_{max}.L^3)$ avec:

- N variables
- m_{max} contraintes impliquant une variable donnée
- L: tailles moyenne des domaines

La généralisation de AC3 à des contraintes impliquant plus que 2 variables induit une combinatoire sur les variables:

“chercher v_2 de D_2 pour satisfaire la contrainte sur x_1 et x_2”

devient pour (x_1, x_2, x_3):

“chercher v_2 de D_2 et v_3 de D_3 pour satisfaire toute contrainte sur (x_1, x_2) et (x_2, x_3) et (x_1, x_3)”

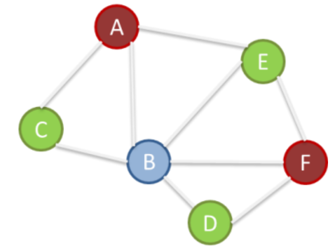
Backtracking adapté (3)

PSC-BACKTRACKING(A: affectation, D: domaines)

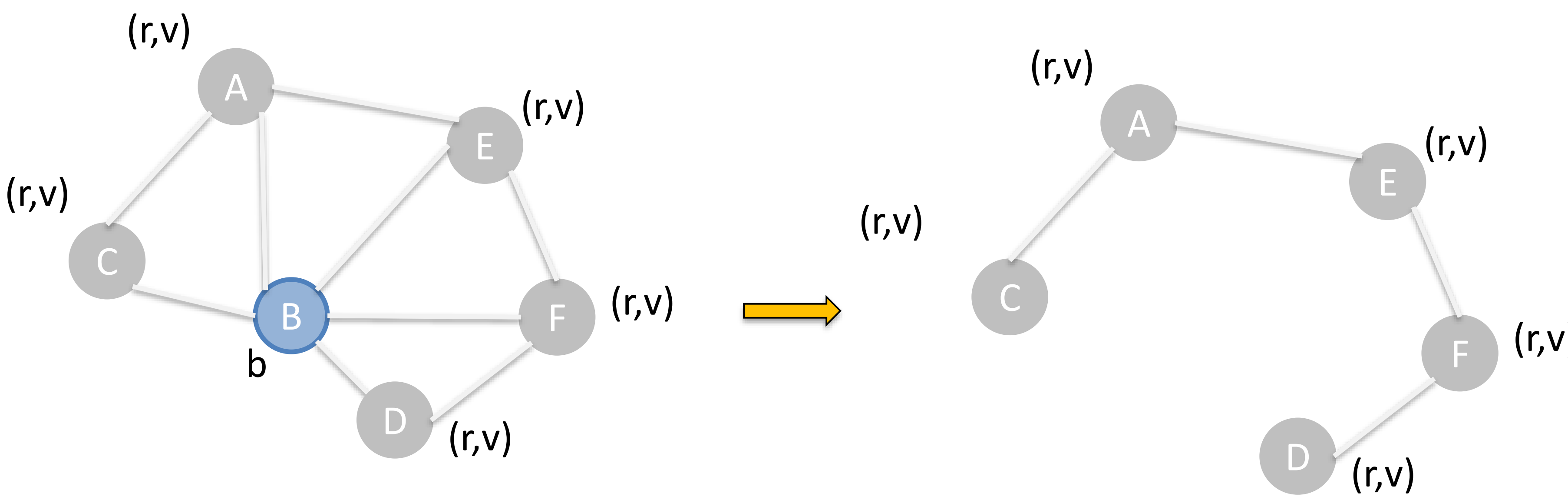
1. Si $A = s_G$ alors retourner A
2. D <- AC3(D)
3. Si une variable a un domaine vide alors echec
4. Sélectionner une variable x_p non affectée
   (heuristique sur les variables)
5. Pour chaque valeur v_{pi} de D_p faire:
   (heuristique sur la valeur)
   – Ajouter x_p <- v_{pi} dans A
   – $D \leftarrow$ FORWARD_CHECKING(A, D)
   – Si aucun domaine de D n'est vide:
     alors retourner PSC-BACKTRACKING(A, D)
     sinon retourner "échec"

Appel: A <- PSC-BACKTRACKING(s_I, D)

Simplification de problème



Quand une variable reçoit une valeur par *backtracking*, on peut propager cette valeur et enlever la variable du graphe:

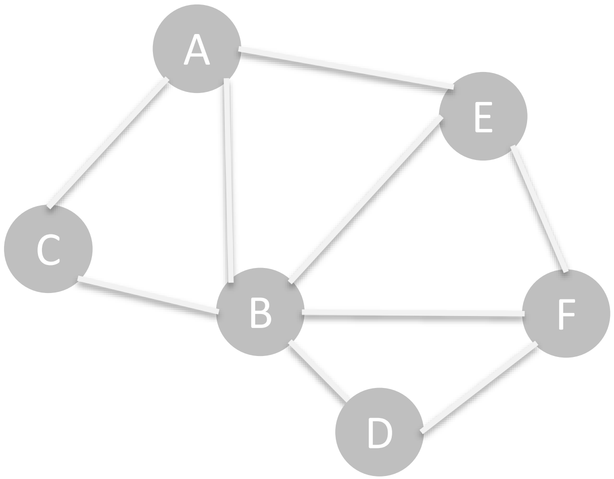


chaine => alternance de couleurs

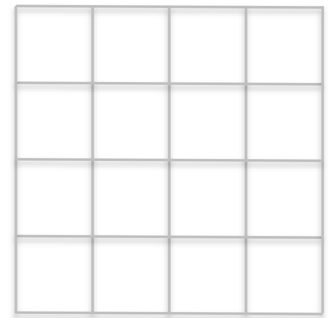
Cela peut simplifier le graphe:

chaîne, arbre, composantes,…

Exemples (à développer)



Coloration (r,v,b)



4-reines (1,2,3,4)

x_1 <- 1
AC3 => change rien
FC
AC3 => $x_2 \neq 3$, $x_3 \neq 2$, $x_3 \neq 4$
D_2 vide => backtrack x_1 <- 2
FC
AC3 => x_2=4, x_3=1, x_4=3

			6
			6
			6
6	6	6	6

Carré magique (1,2,3)

$$
\begin{array}{rcccc}
 & & T & W & O \\
+ & & T & W & O \\
\hline
 & F & O & U & R
\end{array}
$$

(0,1,2,3,4,5,6,7,8,9)

Cas particuliers

- Graphes à multiples composantes connexes
  - Autant de problèmes indépendants

- Arbre de contraintes
  - On ordonne les contraintes de racine aux feuilles, on assigne une valeur à la racine et on propage vers le bas

Résumé

- Tous les problèmes ne se formulent pas directement comme une recherche
- La propagation de contraintes permet de diriger la recherché de solutions
- Les problèmes à contraintes binaires correspondent à un graphe
  - On peut y revenir via l'ajoût de variables
- Ces algorithmes fonctionnent bien si les contraintes ne sont pas trop entremêlées (concernent chacune peu de variables)
  - Lien avec la faisabilité en programmation linéaire