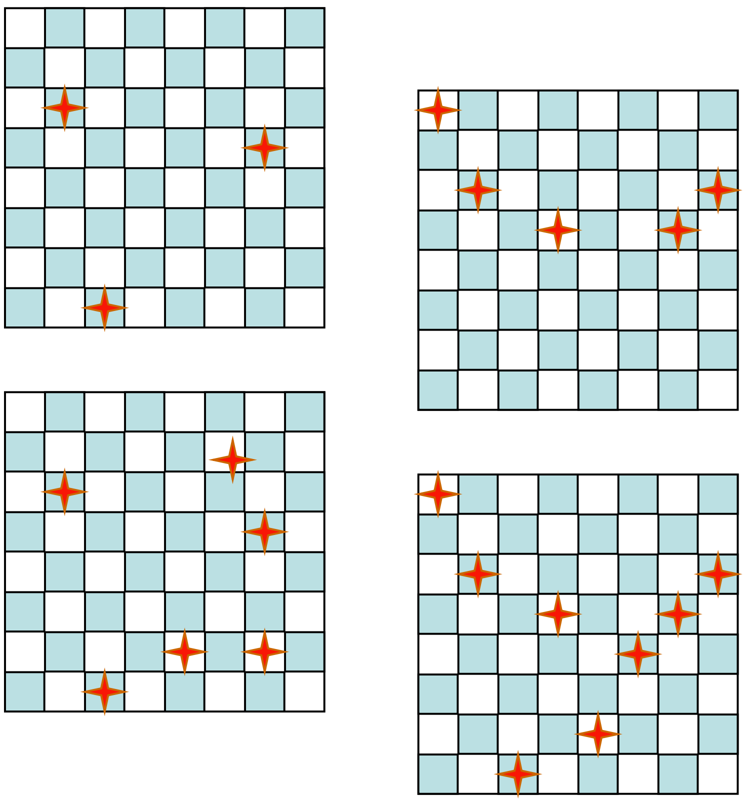
Problèmes à satisfaction de contraintes

Aperçu

- Exemple introductif
- Problèmes à satisfaction de contraintes (PSC)
- PSC comme problème de recherche
- Algorithme de *"backtracking"*
- Heuristiques générales

Certains transparents présentés dans ce chapitre s'inspirent de ceux du Prof. J-C. Latombe (Université de Stanford) et ont été adaptés à notre contexte, ceci avec l'autorisation de l'auteur.

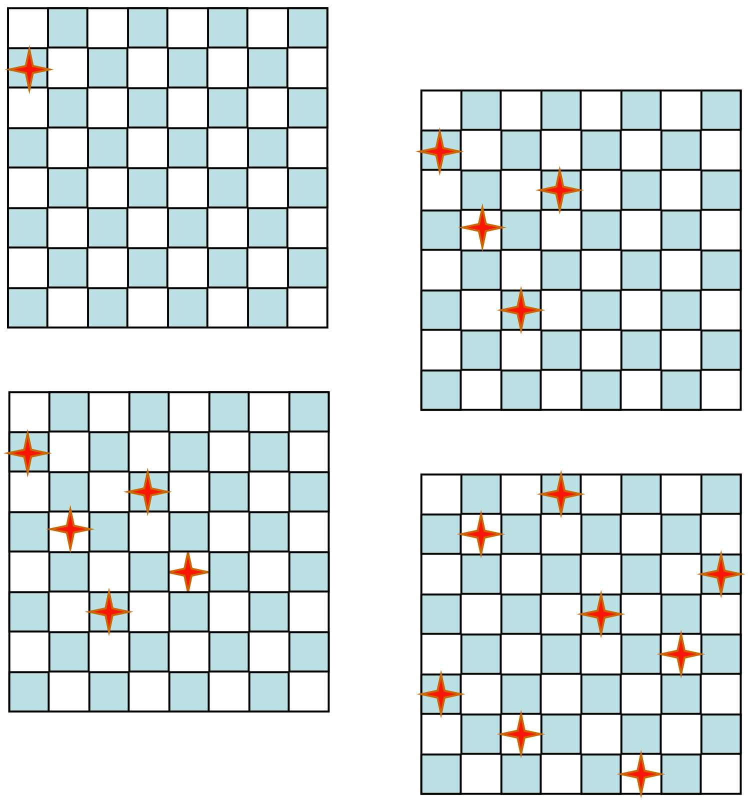
Exemple introductif: 8-reines - 1ère formulation



- **Etats:** tous les arrangements de 0, 1, 2, ..., ou 8 reines sur le damier
- **État initial** : 0 reine sur le damier
- **Fonction successeur** : chacun des successeurs est obtenu en ajoutant une reine sur une case vide
- **Coût d'un arc:** sans importance
- **Test-solution:** 8 reines sur le damier, sans attaque mutuelle

→ 64x63x...x53 ~ 3×10^{14} états

Exemple introductif: 8-reines - 2ème formulation



→ 2,057 états

- **Etats:** tous les arrangements de k = 0, 1, 2, ..., ou 8 reines dans les k colonnes de gauche sans attaque mutuelle
- **Etat initial:** 0 reine sur le damier
- **Fonction successeur:** chaque successeur est obtenu en ajoutant une reine dans une case vide de la 1ère colonne de gauche disponible sans attaque mutuelle
- **Coût d'un arc:** sans importance
- **Test-solution:** 8 reines placées sur le damier

De quoi a-t-on besoin ?

- Fonctions "successeur" et le test-solution ne suffisent plus

- Il faut aussi:
  - le moyen de propager les contraintes imposées par la position d'une reine sur les positions possibles des autres
  - un test d'échec anticipé

→ Représentation explicite des contraintes

→ Algorithmes de propagation de contraintes

Problème à satisfaction de contraintes

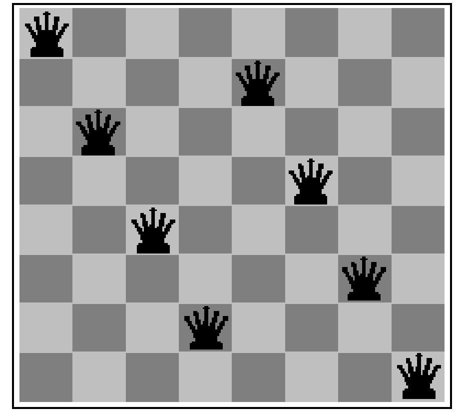
- Un problème à satisfaction de contraintes (PSC) est constitué:
  - d'un ensemble de variables $\{X_1, X_2, \dots , X_n\}$
    - chaque variable X_i ayant un domaine D_i de valeurs possibles
      - en général D_i est discret et fini
  - d'un ensemble de contraintes $\{C_1, C_2, \dots , C_p\}$
    - chaque contrainte C_k concerne un sous-ensemble de variables et spécifie les combinaisons de valeurs permises pour ces variables

- Objectif: assigner une valeur à chaque variable de sorte que toutes les contraintes soient satisfaites.

- Exemples: 8-reines, arithmétique cryptée, coloration de cartes, disposition des éléments sur un circuit VLSI, ordonnancement, ...

8-reines: formulation

- 64 variables X_{ij}, $i = 1$ à 8 et $j = 1$ à 8



- <u>Domaine de chaque variable</u>: $\{ 1, 0 \}$

- <u>Contraintes de la forme</u>:
  - $X_{ij} = 1 \Rightarrow X_{ik} = 0 \quad \forall\, k = 1$ à $8,\ k \neq j$
  - $X_{ij} = 1 \Rightarrow X_{kj} = 0 \quad \forall\, k = 1$ à $8,\ k \neq i$
  - Contraintes semblables pour les diagonales
  - $\sum_{i,j \in [1,8]} X_{ij} = 8$

Contraintes binaires
(chaque contrainte lie seulement 2 variables)

Exemple: coloration de carte

- 7 variables: {*WA, NT, SA, Q, NSW, V, T*}
- Chaque variable a le même domaine {*rouge*, *vert*, *bleu*}
- Contraintes: 2 régions adjacentes doivent être de couleurs différentes
  - *WA≠NT, WA≠SA, NT≠SA, NT≠Q, SA≠Q, SA≠NSW, SA≠V,Q≠NSW, NSW≠V*
  - ou: (*WA, NT*) ∈ {(*rouge,vert*), (*rouge,bleu*), (*vert,rouge*), (*vert,bleu*) …}

Exemple: coloration de carte

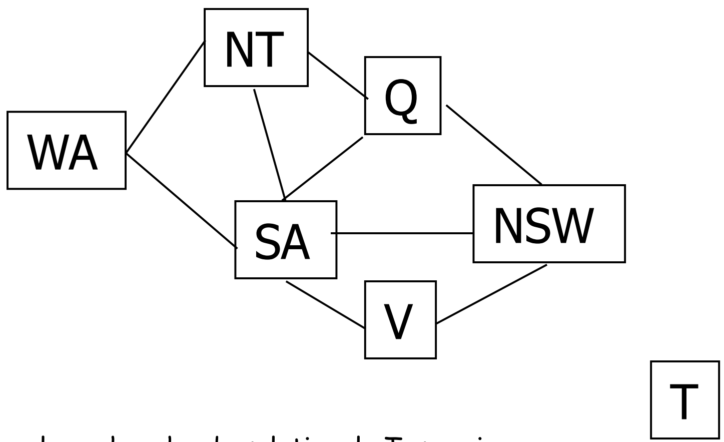
- Les solutions sont des affectations satisfaisant toutes les contraintes, exemple: {*WA=rouge,NT=vert,Q=rouge,NSW=vert,V=rouge,SA=bleu,T=vert*}

PSC fini et infini

- Chaque variable a un domaine fini de valeurs $\rightarrow$ PSC fini
- Quelques (toutes) variables ont des domaines infinis $\rightarrow$ PSC infini
  (cas particulier: programmation linéaire)
  ex., problèmes de programmation linéaire sur des nombres réels:
- On ne traitera que des PSC finis

Graphe de contraintes

- PSC binaire: chaque contrainte concerne au plus deux variables
- Graphe de contraintes
  - les nœuds représentent les variables
  - les arcs représentent les contraintes



Pour accélérer la recherche de solution, la Tasmanie sera considérée comme un sous-problème indépendant

- 2 variables sont adjacentes (ou voisines) si elles sont reliées par un arc
- les algorithmes généraux de PSC utilisent les structures de graphes

Types de contraintes

- Unaire: ne concerne qu'une seule variable
  - exemple: SA ≠ vert
- Binaire: concerne des paires de variables
  - exemple: SA ≠ WA
- Ordre supérieur: concerne 3 variables ou plus
  - exemple: arithmétique cryptée
- Préférence (contrainte "molle"):

  exemple: rouge est meilleur que vert

  souvent représentée par un coût associé à chaque affectation possible

  → *problème d'optimisation de contraintes*

Exemples de PSC réels

- Problèmes d'affectation
  - <u>ex</u>: qui enseigne quel cours?
- Problèmes d'horaires
  - <u>ex</u>: où et quand un enseignant donne-t-il ses cours?
- Configuration matérielle
- Organisation de transport (chemins de fer, compagnies aériennes)
- Ordonnancement de production (atelier)
- Occupation d'espace (architecture)

- Beaucoup de problèmes du monde réel impliquent des variables à valeurs réelles (continues)

PSC comme un problème de recherche

- n variables $X_1, \ldots, X_n$
- Affectation valide :
  $\{X_1 \leftarrow v_1, \ldots, X_k \leftarrow v_k\}, \quad 0 \leq k \leq n,$
  telles que les valeurs $v_1, \ldots, v_k$ satisfassent toutes les contraintes liant les variables $X_1, \ldots, X_k$
- Affectation complète: une pour qui $k = n$
  [Si tous les domaines de variables sont de taille d, il y a $O(d^n)$ affectations complètes]
- Etats: affectations valides
- Etat initial: affectation vide { } , càd $k = 0$
- Successeur d'un état:
  $\{X_1 \leftarrow v_1, \ldots, X_k \leftarrow v_k\} \rightarrow \{X_1 \leftarrow v_1, \ldots, X_k \leftarrow v_k, X_{k+1} \leftarrow v_{k+1}\}$
- Test-solution: $k = n$

- 4 variables $X_1, \ldots, X_4$
- Soit l'assignement valable de N:

  $$A = \{X_1 \leftarrow v_1, X_3 \leftarrow v_3\}$$

- (par exemple) choisir la variable X_4
- Soit $\{v_{4,1}, v_{4,2}, v_{4,3}\}$ le domaine de X_4
- Les successeurs de A sont tous les assignements valables parmi:

  $$\{X_1 \leftarrow v_1, X_3 \leftarrow v_3, X_4 \leftarrow v_{4,1}\}$$
  $$\{X_1 \leftarrow v_1, X_3 \leftarrow v_3, X_4 \leftarrow v_{4,2}\}$$
  $$\{X_1 \leftarrow v_1, X_3 \leftarrow v_3, X_4 \leftarrow v_{4,3}\}$$

Propriété des PSC: commutativité

- L'ordre dans lequel les valeurs sont assignées aux variables est sans importance pour la solution finale
  - [*WA=rouge* suivi de *NT=vert*] identique à [*NT=vert* suivi de *WA=rouge*]

- Donc:
  1. On peut produire les successeurs d'un nœud N en sélectionnant **une** variable X absente de l'assignement A associé à N et en assignant chaque valeur v du domaine de X
     [$\rightarrow$ importante réduction du facteur de branchement]
  2. Il n'est pas nécessaire de mémoriser le chemin menant à un nœud donné

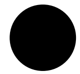
→ Algorithme de recherche par "Backtracking"

Recherche « Backtracking »

- La recherche en profondeur pour des PSC avec affectation d'une seule variable à la fois est appelée recherche "backtracking"
  - c'est l'algorithme non heuristique de base pour les PSC
  - capable de résoudre le problème des n-reines pour $n \approx 25$

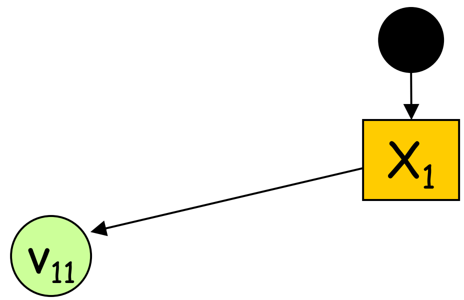
- C'est essentiellement une version simplifiée de l'algorithme de recherche en profondeur utilisant la récursivité

Recherche « backtracking » (3 variables)



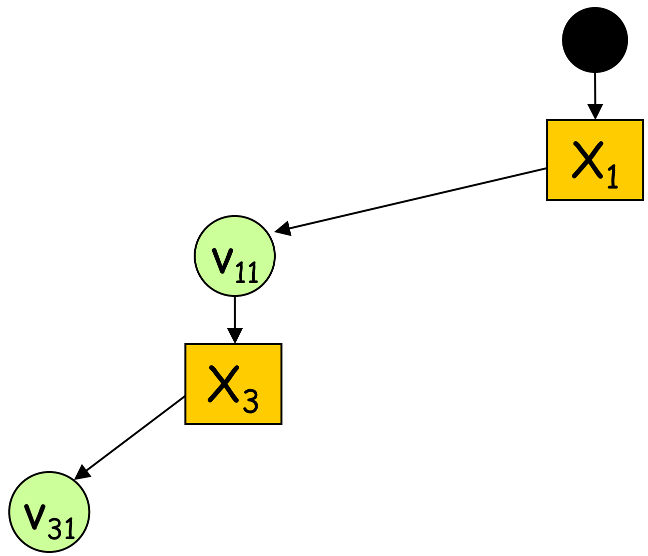
Assignement = {}

Recherche « backtracking » (3 variables)



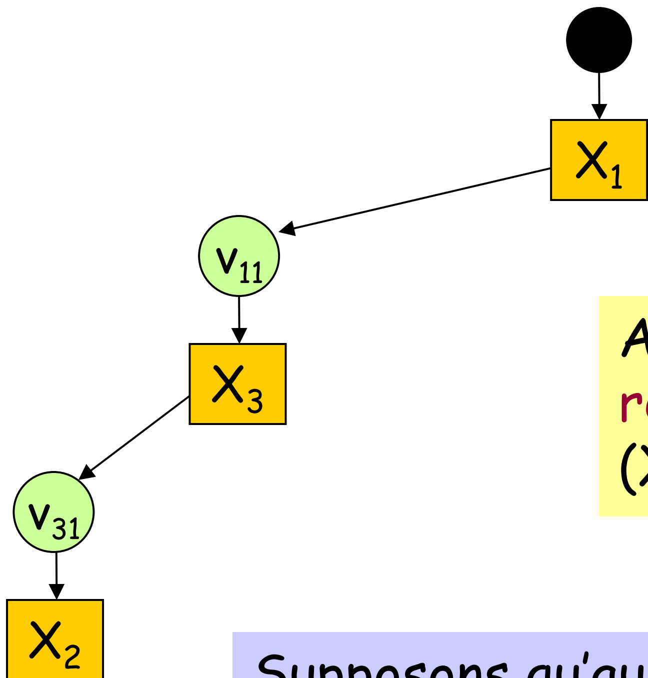
Assignement = $\{(X_1, v_{11})\}$

Recherche « backtracking » (3 variables)



Assignement = $\{(X_1,v_{11}), (X_3,v_{31})\}$

Recherche « backtracking » (3 variables)

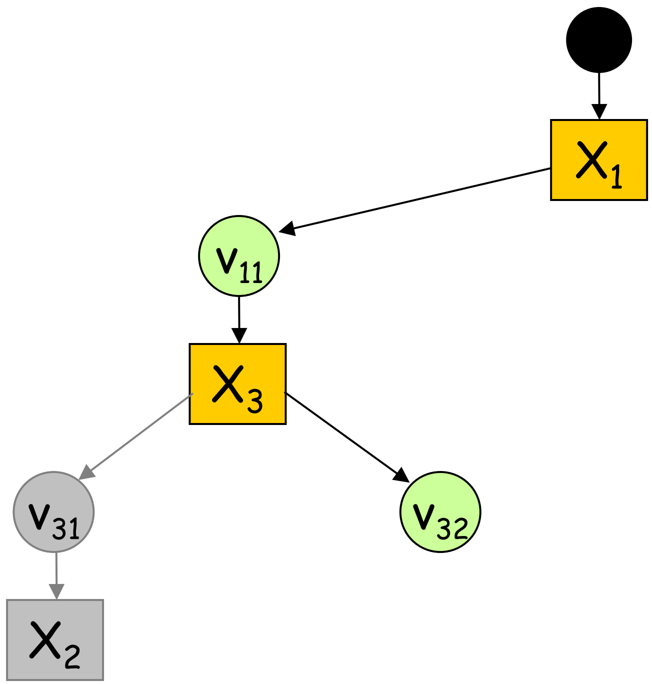


Alors, l'algorithme de recherche revient à la variable précédente (X_3) et essaie une autre valeur

Supposons qu'aucune valeur de X_2 ne produit d'assignement valable

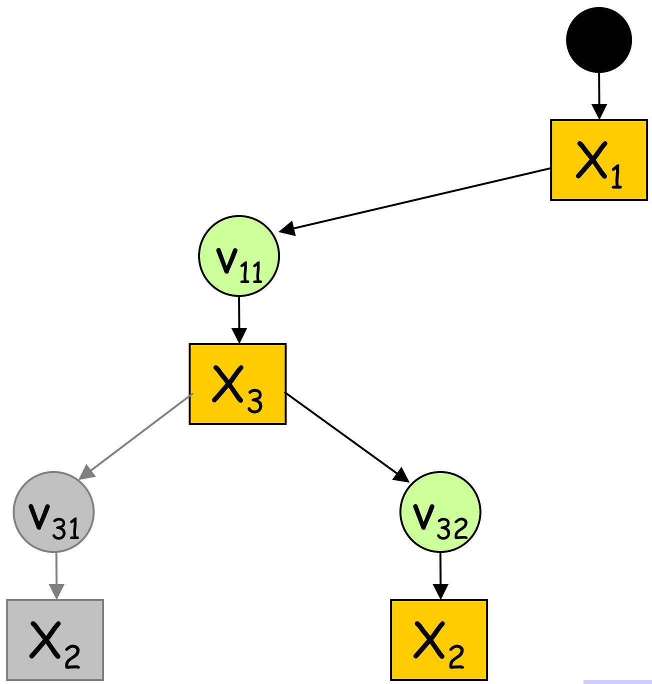
Assignement = $\{(X_1, v_{11}), (X_3, v_{31})\}$

Recherche « backtracking » (3 variables)



Assignement = $\{(X_1, v_{11}), (X_3, v_{32})\}$

Recherche « backtracking » (3 variables)

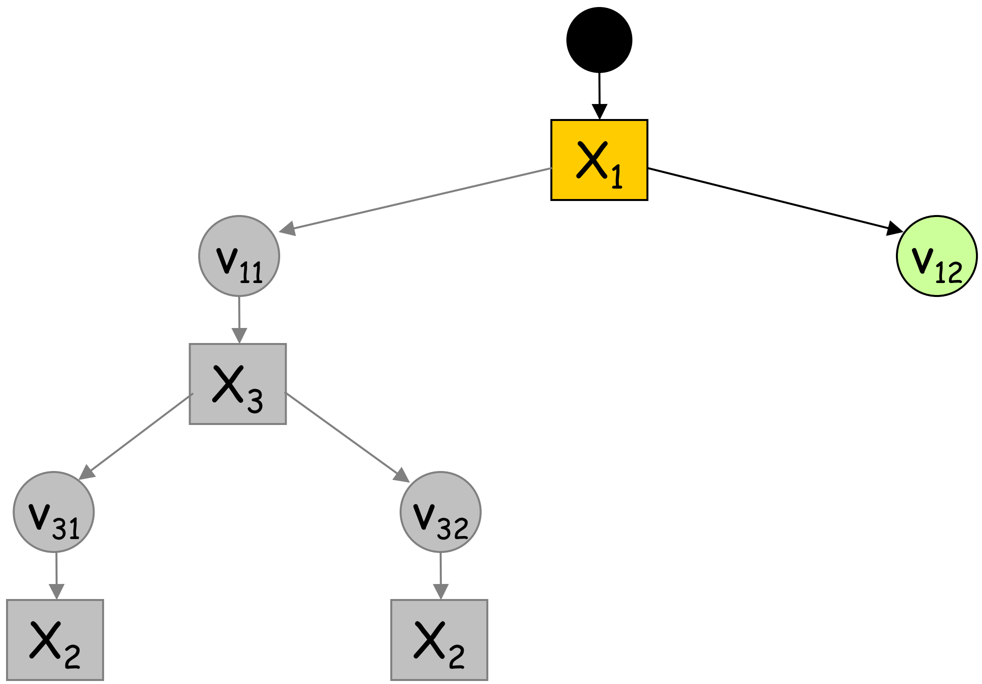


L'algorithme de recherche revient à la variable précédente (X_3) et essaie une autre valeur. Mais supposons que X_3 n'a que deux valeurs possibles. Alors l'algorithme revient à X_1

Supposons à nouveau qu'aucune valeur de X_2 ne produit d'assignement valable

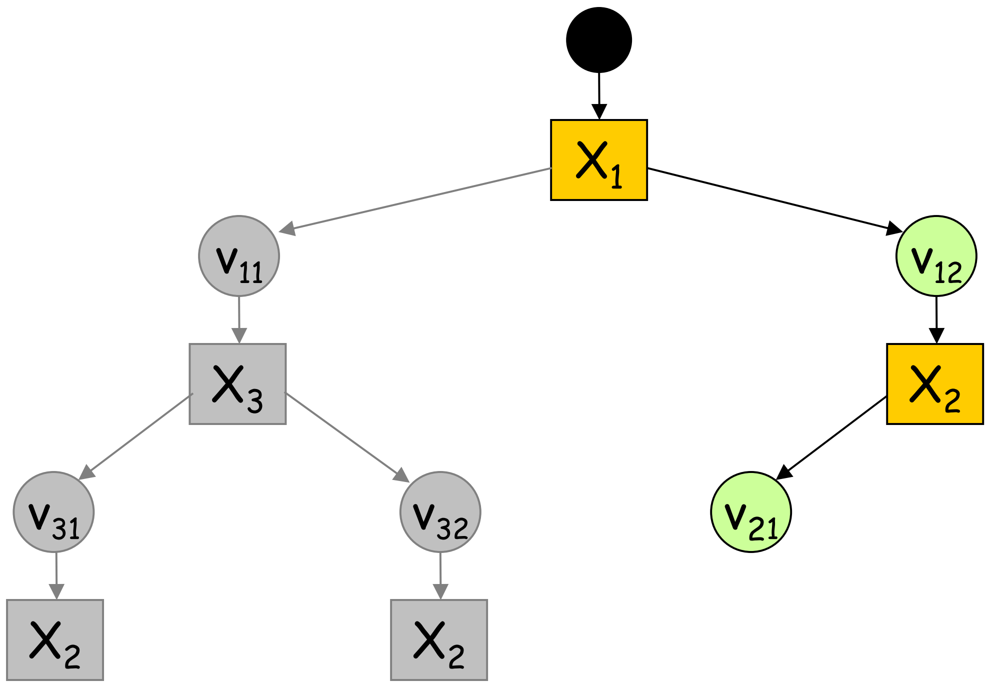
Assignement = $\{(X_1,v_{11}), (X_3,v_{32})\}$

Recherche « backtracking » (3 variables)



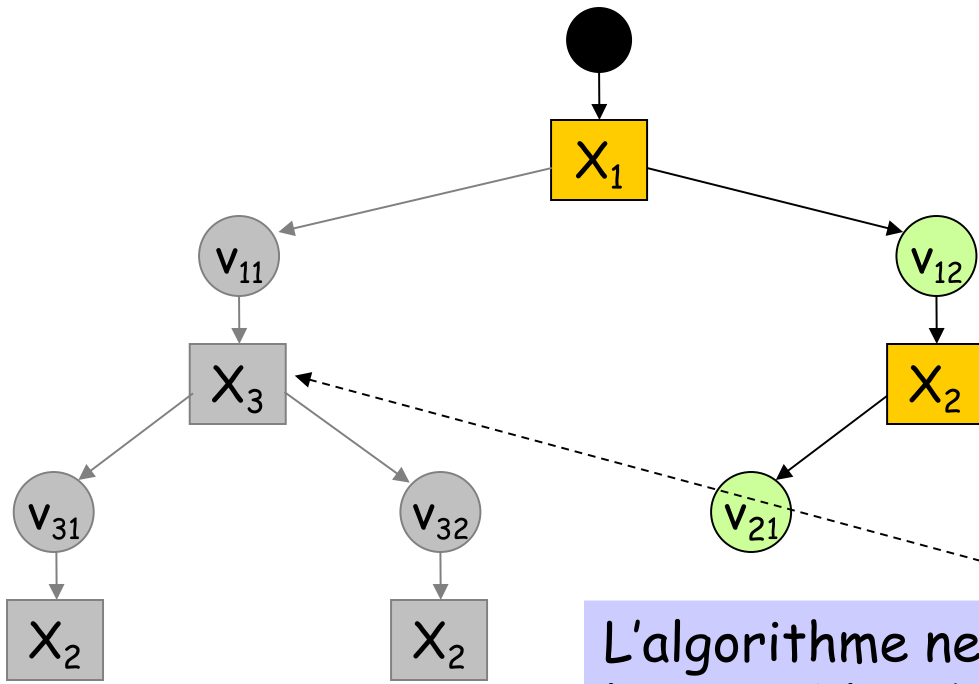
Assignement = $\{(X_1, v_{12})\}$

Recherche « backtracking » (3 variables)



Assignement = $\{(X_1,v_{12}), (X_2,v_{21})\}$

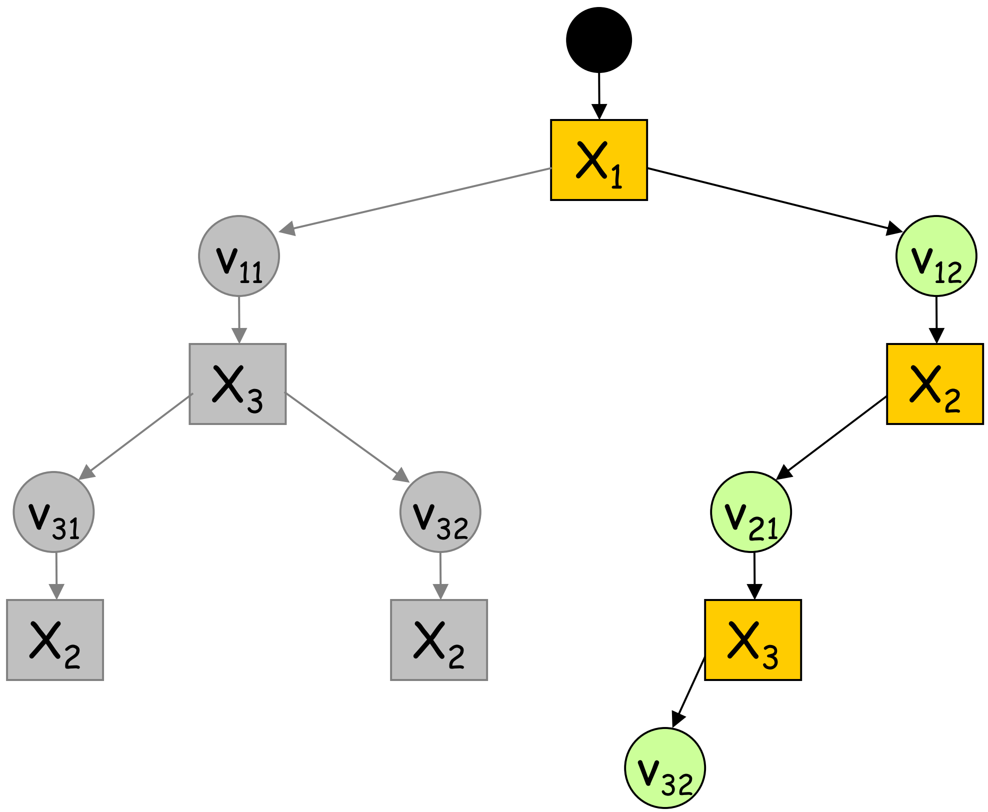
Recherche « backtracking » (3 variables)



L'algorithme ne doit pas considérer les variables dans le même ordre dans ce sous-arbre et dans l'autre

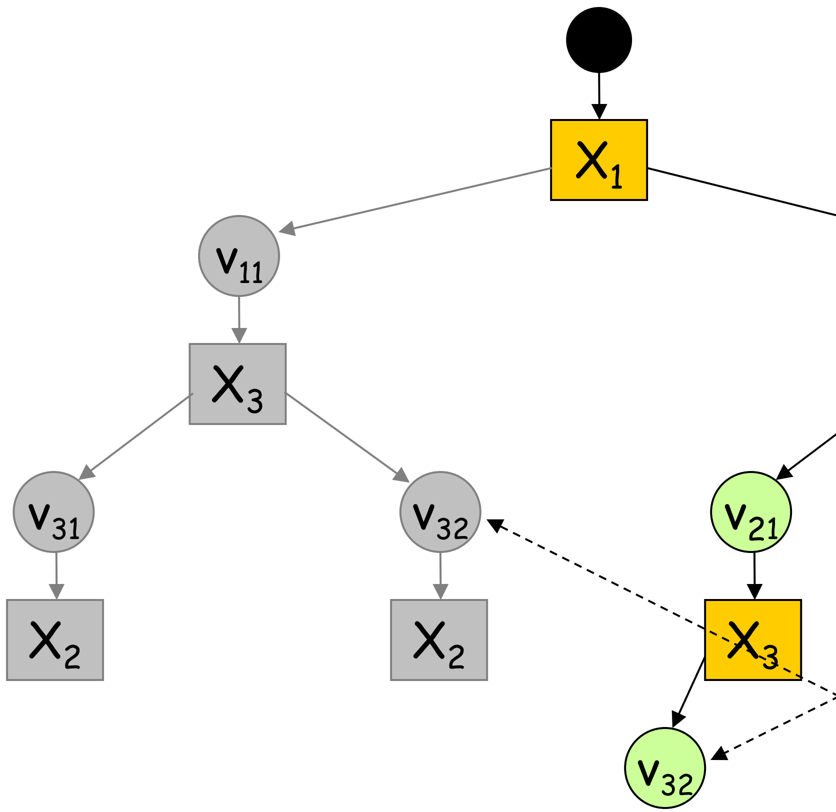
Assignement = $\{(X_1,v_{12}), (X_2,v_{21})\}$

Recherche « backtracking » (3 variables)



Assignement = $\{(X_1, v_{12}), (X_2, v_{21}), (X_3, v_{32})\}$

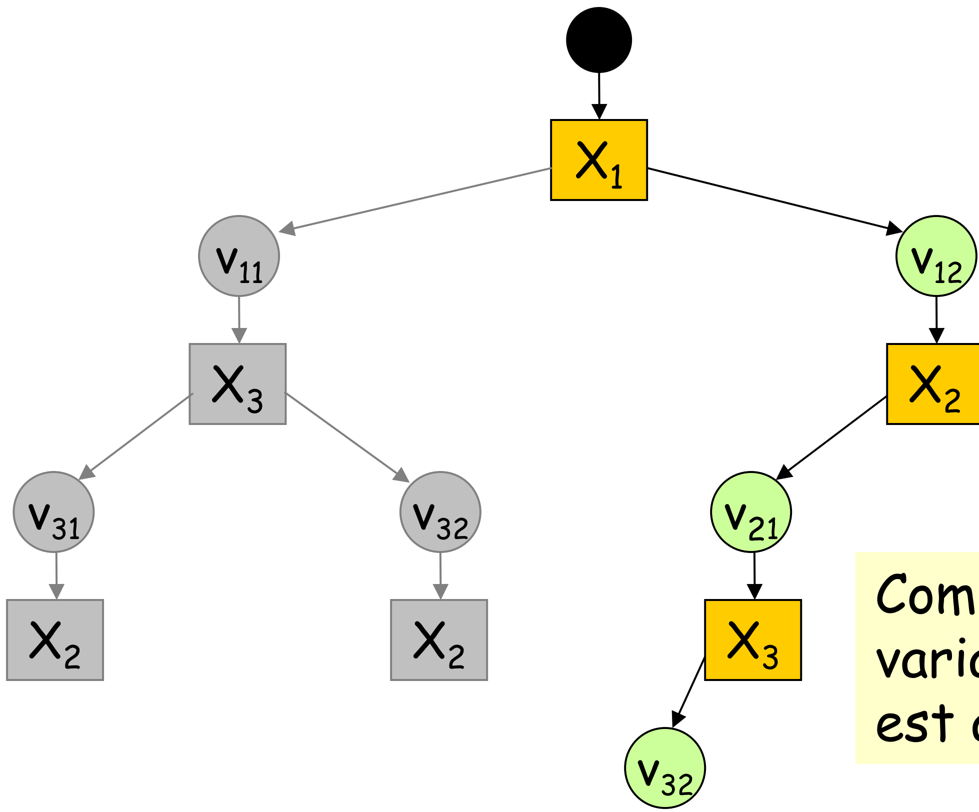
Recherche « backtracking » (3 variables)



L'algorithme ne doit pas considérer les valeurs de X_3 dans le même ordre dans ce sous-arbre

Assignement = $\{(X_1,v_{12}), (X_2,v_{21}), (X_3,v_{32})\}$

Recherche « backtracking » (3 variables)



Comme il n'y a que trois variables, l'assignement est alors complet

Assignement = $\{(X_1,v_{12}), (X_2,v_{21}), (X_3,v_{32})\}$

Algorithme de « backtracking »

PSC-BACKTRACKING(*A*)

1. Si assignement *A* est complet alors retourner *A*
2. X ← sélectionner une variable absente de *A*
3. D ← sélectionner un ordre sur le domaine de X
4. Pour chaque valeur v dans D faire
    a. Add (X←v) à *A*
    b. Si *A* est valide alors
        i. résultat ← PSC-BACKTRACKING(*A*)
        ii. Si résultat ≠ échec alors retourner résultat
5. Retourner échec

Appel: PSC-BACKTRACKING({})

Exemple: coloration de carte

{ }

WA=rouge

WA=vert

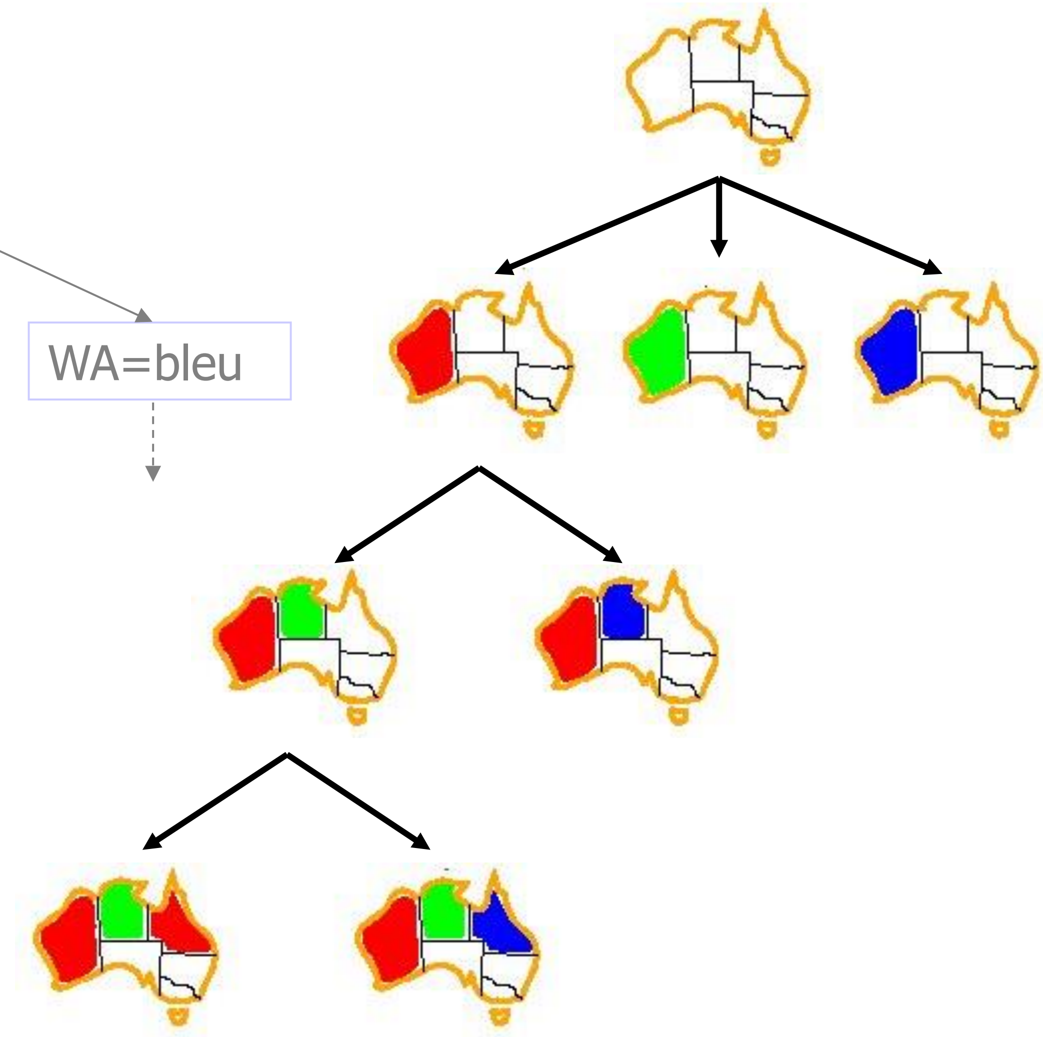
WA=bleu

WA=rouge
NT=vert

WA=rouge
NT=bleu

WA=rouge
NT=vert
Q=rouge

WA=rouge
NT=vert
Q=bleu



Questions

PSC-BACKTRACKING(A)

1. Si assignement A est complet alors retourner A
2. X ← **sélectionner** une variable absente de A
3. D ← **sélectionner** un ordre sur le domaine de X
4. Pour chaque valeur v dans D faire
   a. Add (X←v) à A
   b. Si A est valide alors
      i. résultat ← PSC-BACKTRACKING(A)
      ii. Si résultat ≠ échec alors retourner résultat
5. Retourner échec

Questions

1) Quelle prochaine variable X doit recevoir une valeur ?

1) Dans quel ordre les valeurs de X doivent-elles être assignées?

Questions

1) Quelle prochaine variable X doit recevoir une valeur ?
   L'assignement courant peut ne pas mener à une quelconque solution, mais l'algorithme ne le sait pas encore. Sélectionner la bonne variable X peut aider à trouver la contradiction plus rapidement

1) Dans quel ordre les valeurs de X doivent-elles être assignées?

Questions

1) Quelle prochaine variable X doit recevoir une valeur ?
   L'assignement courant peut ne pas mener à une quelconque solution, mais l'algorithme ne le sait pas encore. Sélectionner la bonne variable X peut aider à trouver la contradiction plus rapidement

1) Dans quel ordre les valeurs de X doivent-elles être assignées?
   L'assignement peut faire partie de la solution. Sélectionner la bonne valeur à assigner à X peut aider à trouver la solution plus rapidement

Questions

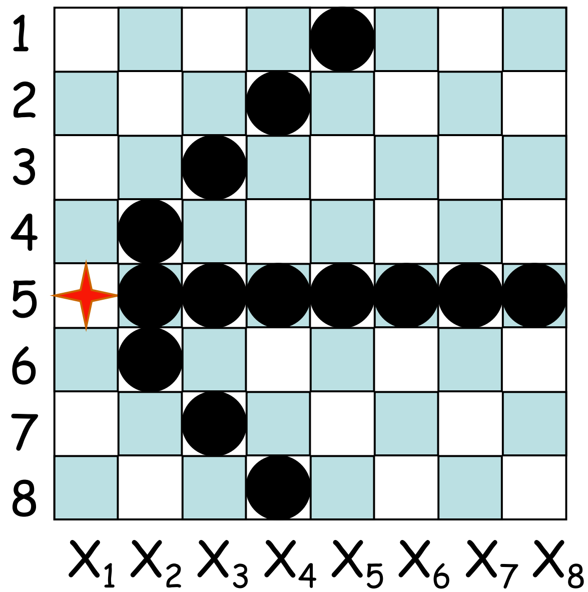
1) Quelle prochaine variable X doit recevoir une valeur ?
   L'assignement courant peut ne pas mener à une quelconque solution, mais l'algorithme ne le sait pas encore. Sélectionner la bonne variable à assigner peut aider à trouver la contradiction plus rapidement

1) Dans quel ordre les valeurs de X doivent-elles être assignées?
   L'assignement peut faire partie de la solution. Sélectionner la bonne valeur à assigner à X peut aider à trouver la solution plus rapidement

Plus sur ces questions prochainement ...

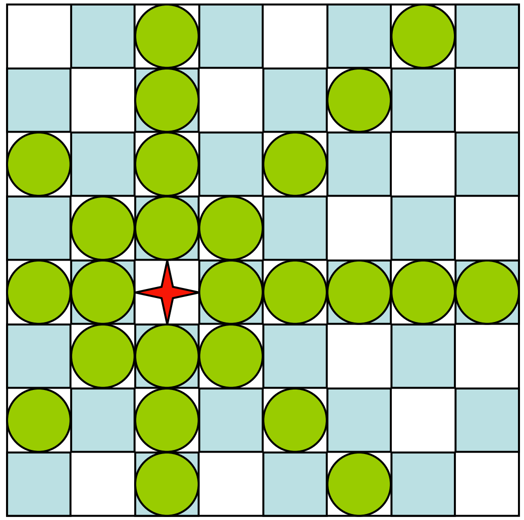
Forward checking

Une technique simple de propagation de contraintes:



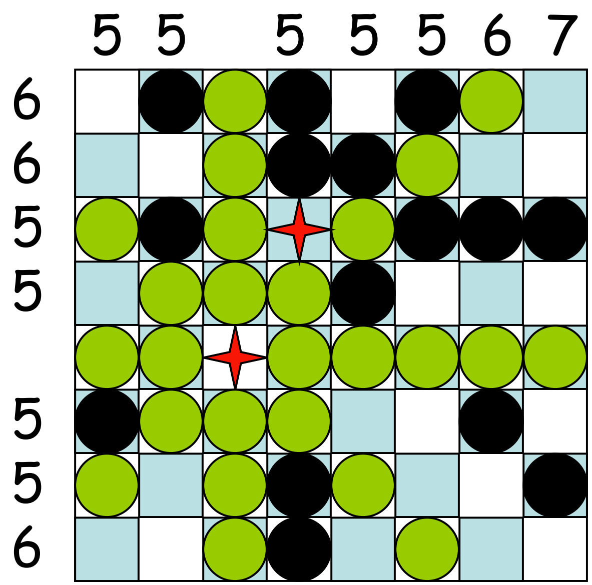
Assigner la valeur 5 à X_1 implique éliminer des valeurs des domaines de X_2, X_3, ..., X_8

Forward checking pour les 8 reines



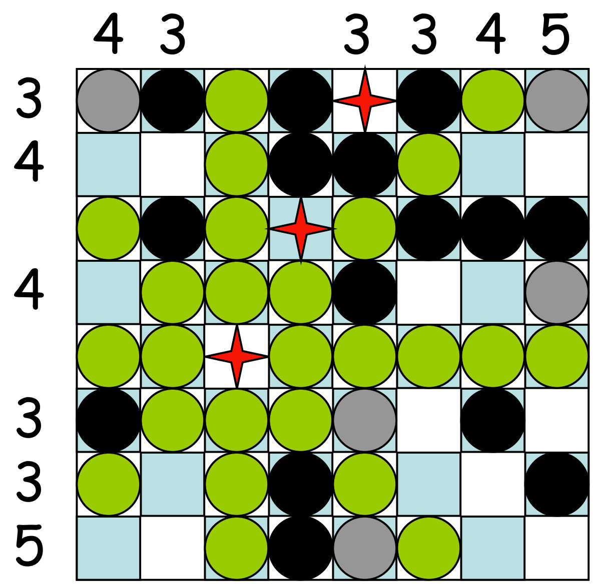
- Placer une reine dans une case
- Éliminer les cases attaquées pour de futures considérations

Forward checking pour les 8 reines



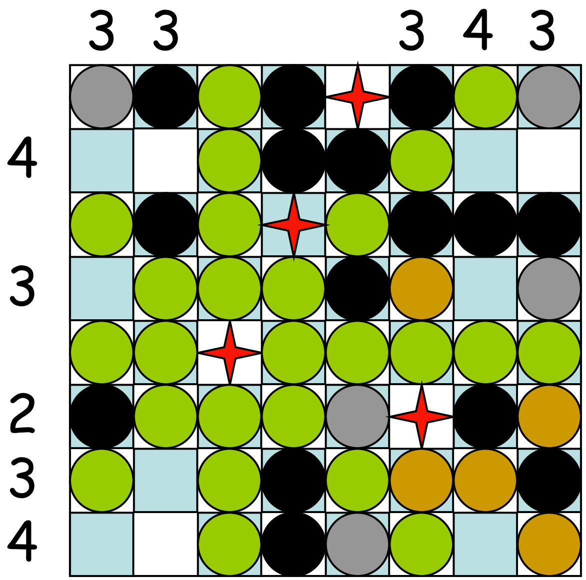
- Compter le nombre de cases libres d'attaques dans chaque ligne et colonne
- Placer une reine dans une ligne ou une colonne ayant le plus petit nombre
- Éliminer les cases attaquées pour de futures considérations

Forward checking pour les 8 reines

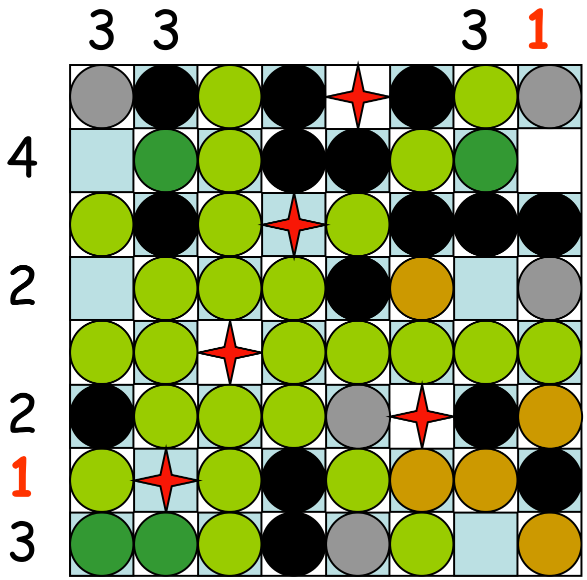


- Répéter

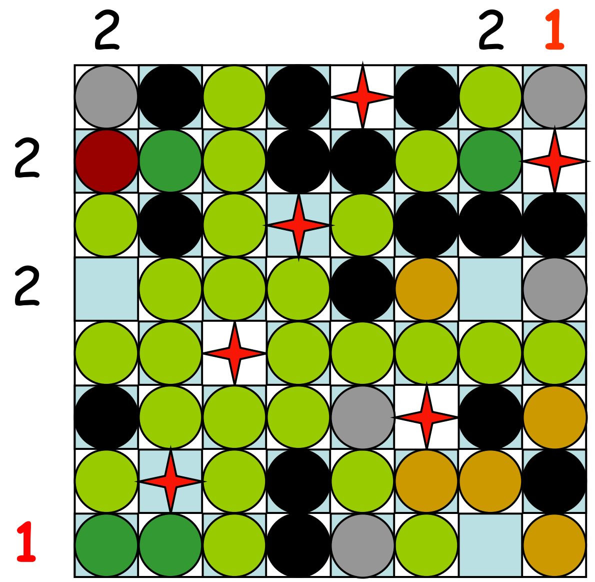
Forward checking pour les 8 reines



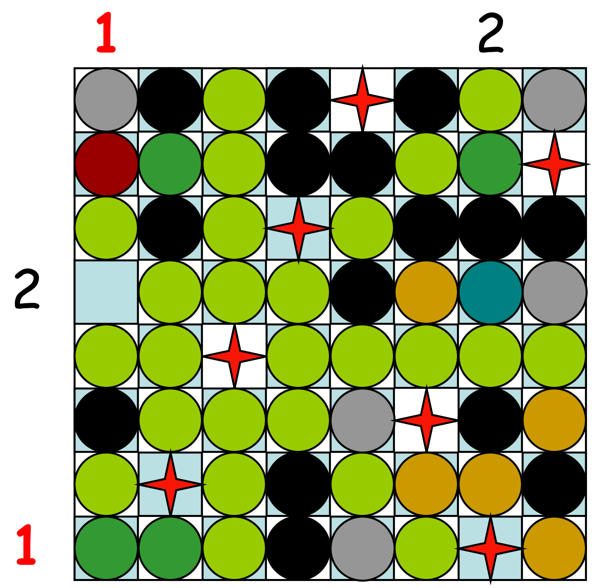
Forward checking pour les 8 reines



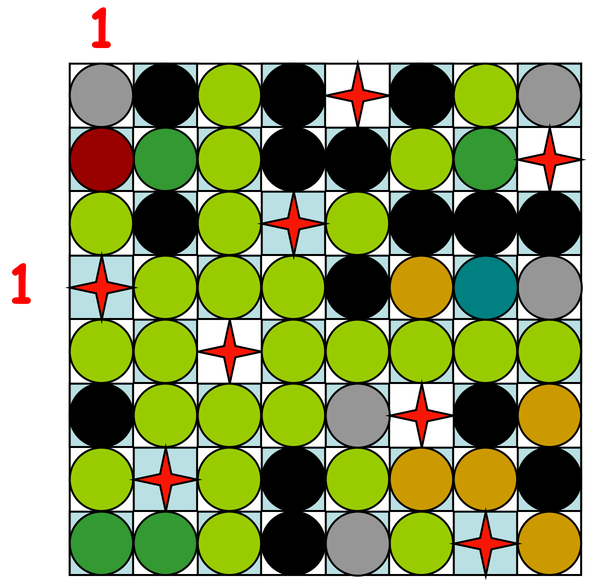
Forward checking pour les 8 reines



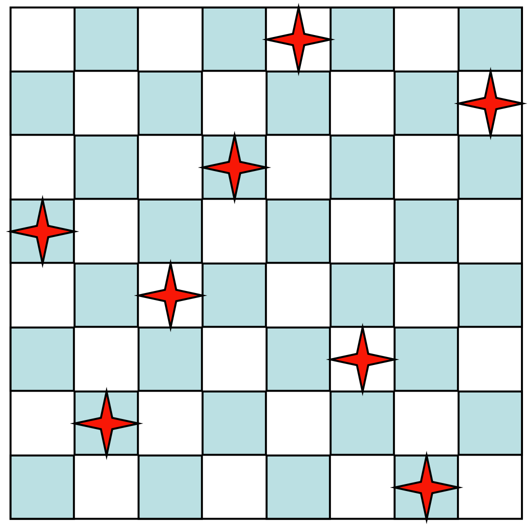
Forward checking pour les 8 reines



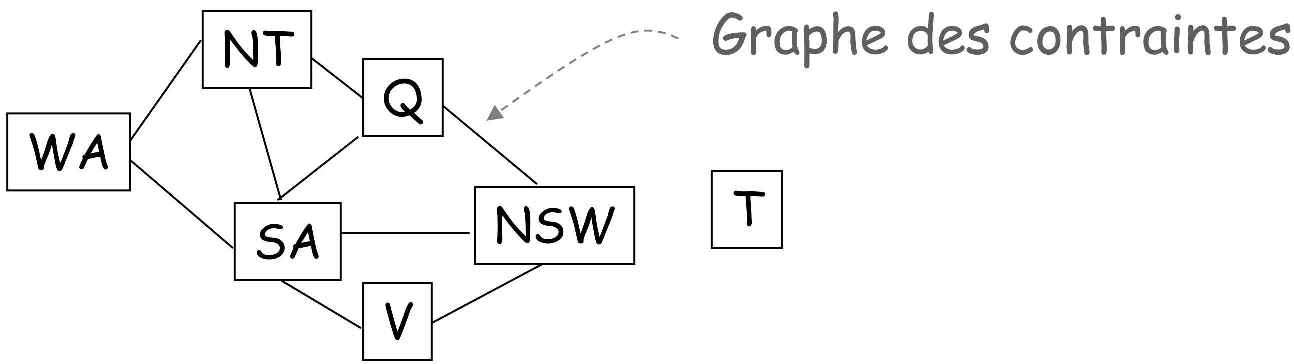
Forward checking pour les 8 reines



Forward checking pour les 8 reines

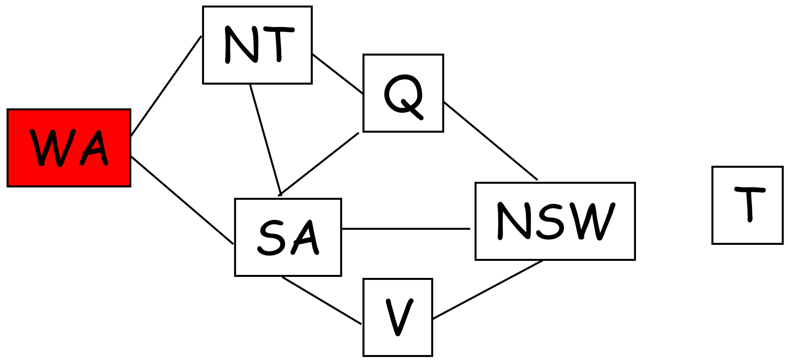


Forward checking - coloration de cartes



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB

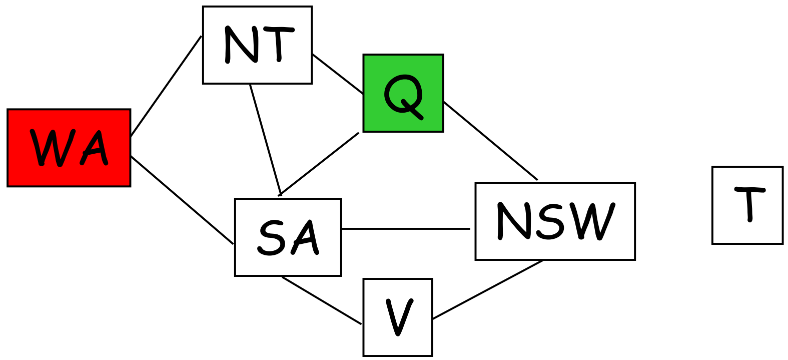
Forward checking - coloration de cartes



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB

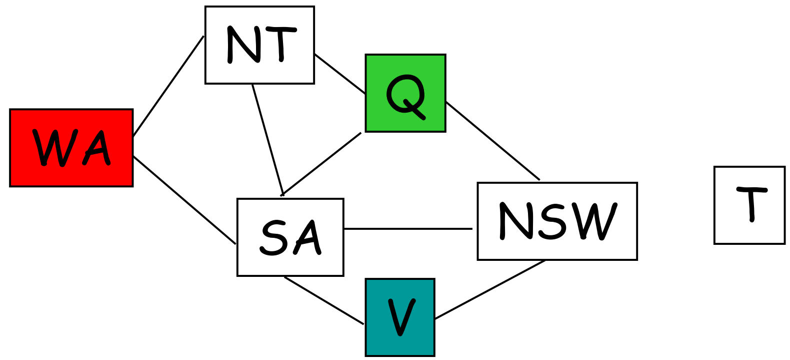
Forward checking élimine la valeur Rouge pour NT et pour SA

Forward checking - coloration de cartes



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	~~G~~B	G	R~~G~~B	RGB	~~G~~B	RGB

Forward checking - coloration de cartes



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	~~RB~~	B	~~B~~	RGB

Forward checking - coloration de cartes

Ensemble vide: l'assignement courant
$\{(WA \leftarrow R), (Q \leftarrow G), (V \leftarrow B)\}$
ne mène pas à la solution

WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	R~~B~~	B	~~B~~	RGB

Forward checking - forme générale

Chaque fois qu'une paire ($X \leftarrow v$) est ajoutée à un assignement A faire:

- Pour chaque variable Y absente de A faire:
  - Pour chaque contrainte C liant Y aux variables de A faire:
    - Eliminer toutes les valeurs des domaines de Y qui ne satisfont pas C

Algorithme de backtracking modifié

PSC-BACKTRACKING(A, var-domaines)

1. Si assignement A est complet alors retourner A
2. X ← sélectionner une variable absente de A
3. D ← sélectionner un ordre sur le domaine de X
4. Pour chaque valeur v dans D faire
   a. Add (X←v) à A
   b. var-domaines ← forward checking(var-domaines, X, v, A)
   c. Si aucune variable a un domaine vide alors
      i. résultat ← PSC-BACKTRACKING(A, var-domaines)
      ii. Si résultat ≠ échec alors retourner résultat
   d. Enlever (X←v) de A
5. Retourner échec

Algorithme de backtracking modifié

PSC-BACKTRACKING(A, var-domaines)

1. Si assignement A est complet alors retourner A
2. X ← sélectionner une variable absente de A
3. D ← sélectionner un ordre sur le domaine de X
4. Pour chaque valeur v dans D faire
    a. Add (X←v) à A → Plus besoin de vérifier que A est valide
    b. var-domaines ← forward checking(var-domaines, X, v, A)
    c. Si aucune variable a un domaine vide alors
        i. résultat ← PSC-BACKTRACKING(A, var-domaines)
        ii. Si résultat ≠ échec alors retourner résultat
    d. Enlever (X←v) de A
5. Retourner échec

Algorithme de backtracking modifié

PSC-BACKTRACKING(A, var-domaines)

1. Si assignement A est complet alors retourner A
2. X ← sélectionner une variable absente de A
3. D ← sélectionner un ordre sur le domaine de X
4. Pour chaque valeur v dans D faire
    a. Add (X←v) à A
    b. var-domaines ← forward checking(var-domaines, X, v, A)
    c. Si aucune variable a un domaine vide alors
        i. résultat ← PSC-BACKTRACKING(A, var-domaines)
        ii. Si résultat ≠ échec alors retourner résultat
    d. Enlever (X←v) d
5. Retourner échec

Besoin de transmettre les domaines de variables modifiés

Algorithme de backtracking modifié

PSC-BACKTRACKING(A, var-domaines)

1. Si assignement A est complet alors retourner A
2. X ← **sélectionner** une variable absente de A
3. D ← **sélectionner** un ordre sur le domaine de X
4. Pour chaque valeur v dans D faire
    a. Add (X←v) à A
    b. var-domaines ← forward checking(var-domaines, X, v, A)
    c. Si aucune variable a un domaine vide alors
        i. résultat ← PSC-BACKTRACKING(A, var-domaines)
        ii. Si résultat ≠ échec alors retourner résultat
    d. Enlever (X←v) de A
5. Retourner échec

1) Quelle variable suivant X_i devrait recevoir une valeur ?
   → heuristique de la variable la plus contrainte
   → heuristique de la variable la plus contraignante

1) Dans quel ordre ses valeurs doivent-elles être assignées?
   → heuristique de la valeur la moins contraignante

Ces heuristiques peuvent être déroutantes

Mais garder à l'esprit que **toutes** les variables finiront par recevoir une valeur, alors que **une** seule valeur d'un domaine doit être assignée à chaque variable

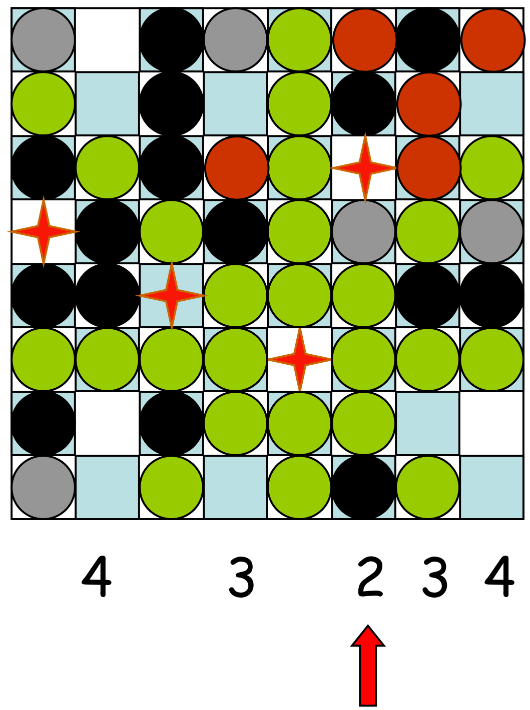
Heuristique de la variable la plus contrainte

1) Quelle variable suivant X_i devrait recevoir une valeur ?

sélectionner la variable ayant le plus petit domaine restant

[Objectif: minimiser le facteur de branchement]

8-reines

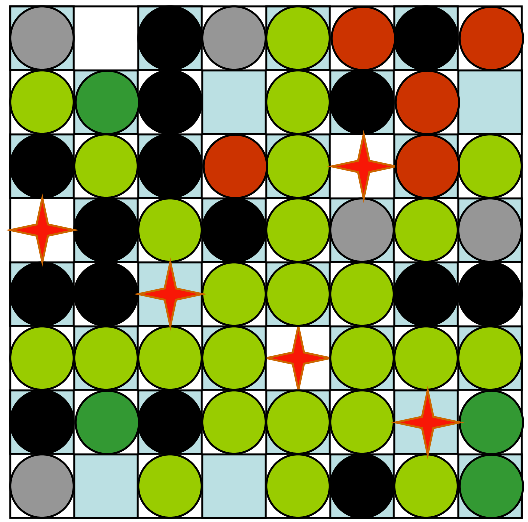


Forward checking

Nouvel assignement

Nombres de valeurs pour chaque variable non-assignée

8-reines



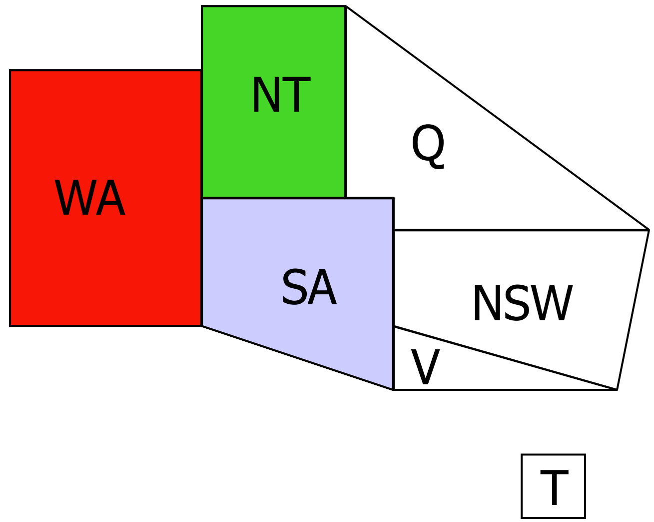
4 2 1 3

Forward checking

Nouvel assignement

Nombres de valeurs pour chaque variable non-assignée

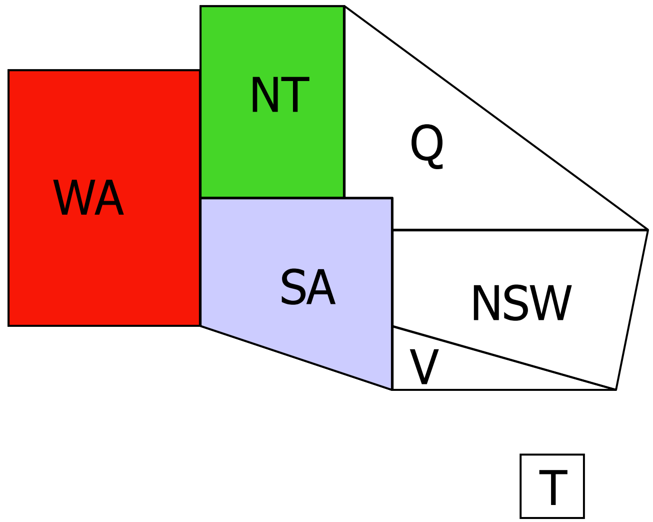
Coloration de cartes



Problèmes analogues :

- Affecter des fréquences différentes à des cellules voisines dans un réseau de téléphone mobile GSM
- Problème d'incompatibilité. Comment faire cohabiter des personnes ou des animaux en tenant compte de leur incompatibilité ?
- La résolution du Sudoku peut se ramener à un problème de coloration de graphe

Heuristique de la variable la plus contrainte



- Taille du domaine restant de SA = 1 (valeur Bleu)
- Taille du domaine restant de Q = 2
- Tailles des domaines de NSW, V et T = 3

→ Sélectionner SA

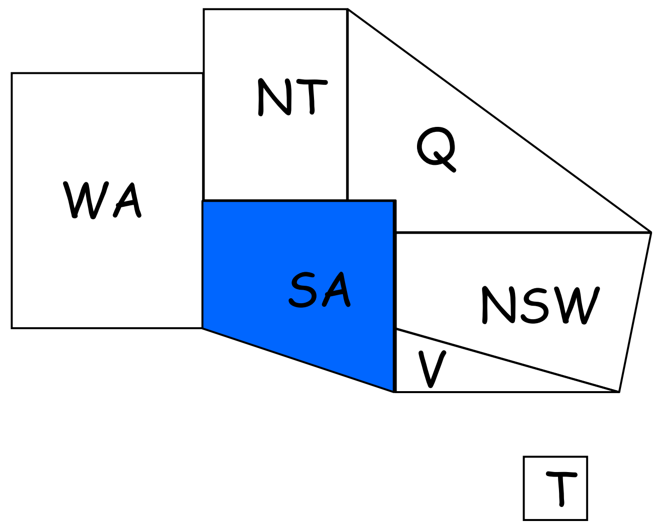
Heuristique de la variable la plus contraignante

1) Quelle variable suivant X_i devrait recevoir une valeur ?

Parmi les variables ayant les domaines le plus petits, choisir celle qui apparait dans le plus grand nombre de contraintes sur des variables non encore assignées

[Objectif: augmenter le nombre d'éliminations futures de valeurs pour réduire le facteur de branchement]

Coloration de cartes



- Avant toute assignation de valeurs, toutes les variables ont des domaines de taille 3, mais SA est impliquée dans plus de contraintes (5) que n'importe quelle autre variable

→ Sélectionner SA et lui assigner une valeur (e.g., Bleu)

Heuristique de la valeur la moins contraignante

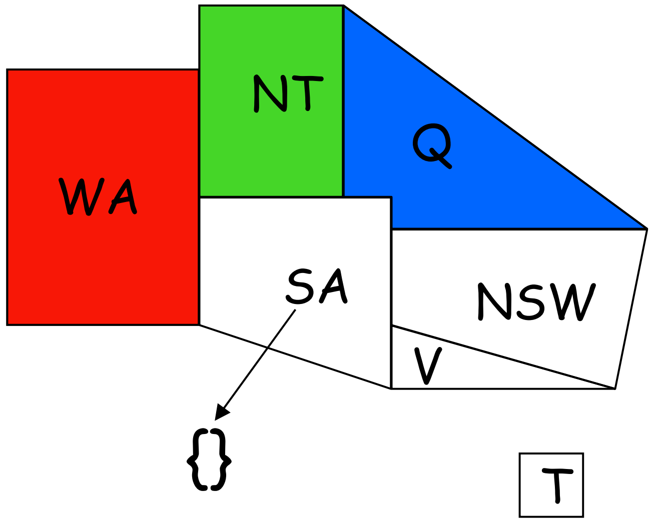
1) Dans quel ordre les valeurs de X doivent-elles être assignées?

Sélectionner la valeur de X qui élimine le plus petit nombre de valeurs des domaines des variables non encore assignées

[Argument: comme une seule valeur doit être assignée à X, choisir en premier la moins contraignante, car elle est celle qui peut le plus probablement produire un assignement valide]

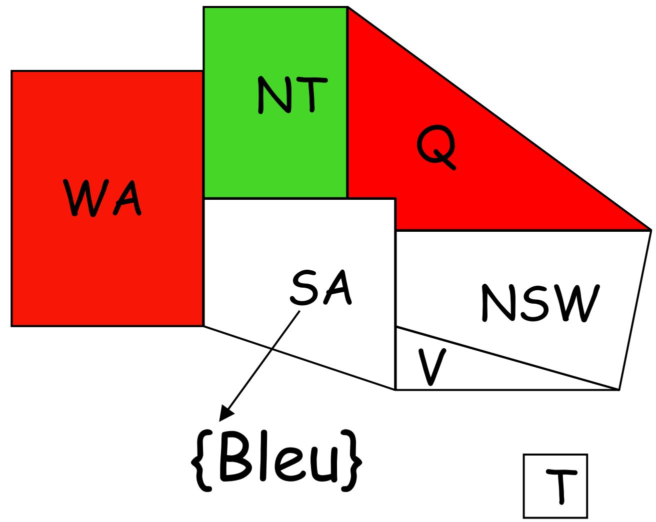
[Note: utiliser cette heuristique demande de faire un "forward-checking" pour chaque valeur, pas seulement pour la valeur sélectionnée]

Coloration de cartes



- Le domaine de Q a 2 valeurs restantes: Bleu et Rouge
- Assigner Bleu à Q laisserait 0 valeur pour SA, alors qu'assigner Rouge laisserait 1 valeur

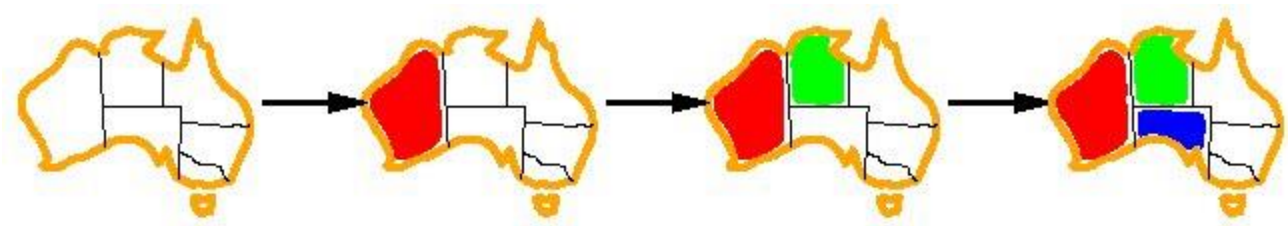
Coloration de cartes



- Le domaine de Q a 2 valeurs restantes: Bleu et Rouge
- Assigner Bleu à Q laisserait 0 valeur pour SA, alors qu'assigner Rouge laisserait 1 valeur
  → Donc assigner Rouge à Q

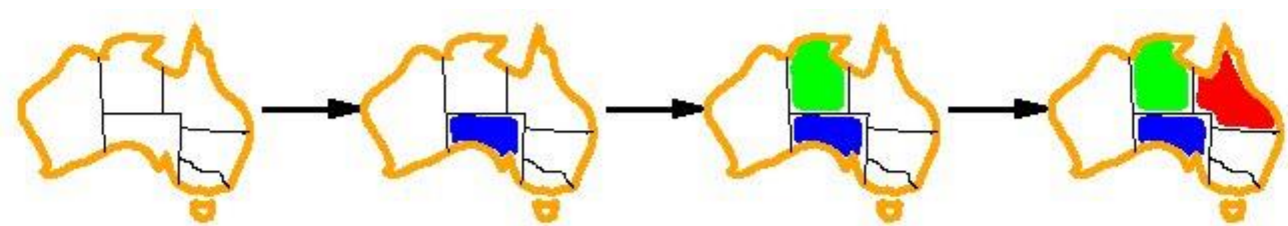
Résumé

- Heuristique de la variable la plus contrainte
  - sélectionner la variable avec le plus petit nombre de valeurs possibles



but: réduire le facteur de branchement

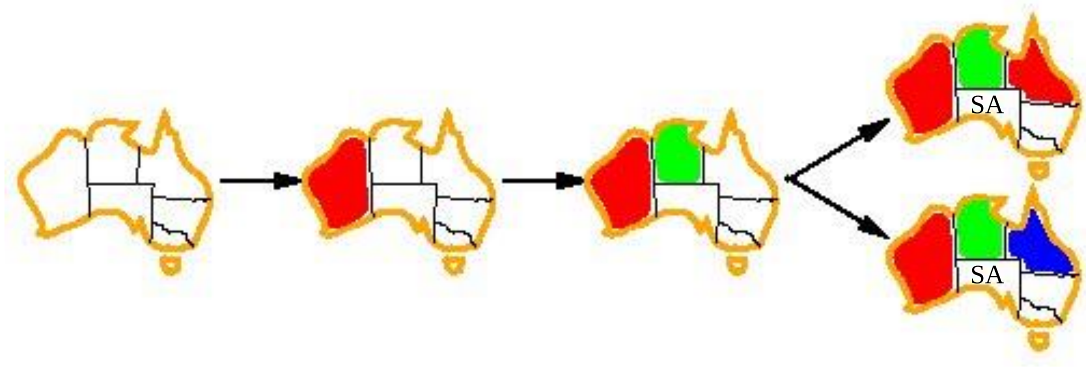
- Heuristique de la variable la plus contraignante
  - sélectionner la variable qui est impliquée dans le plus grand nombre de contraintes sur les variables non encore assignées



but: minimiser le nombre de valeurs restantes possibles

Résumé

- Heuristique de la valeur la moins contraignante
  - préférer la valeur qui laisse le plus de valeurs possibles pour les autres variables non encore assignées



autorise 1 valeur pour SA (bleu)

autorise 0 valeur pour SA

- Une combinaison de ces différentes heuristiques rend le problème des 1000-reines praticable

Algorithme Backtracking modifié

PSC-BACKTRACKING(A, var-domaines)

1. Si assignement A est complet alors retourner A
2. X ← **sélectionner** une variable absente de A
3. D ← **sélectionner** un ordre sur le domaine de X
4. Pour chaque valeur v dans D faire
   a. Add (X←v) à A
   b. var-domaines ← forward checking(var-domaines, X, v, A)
   c. Si aucune variable a une domaine vide alors
      i. résultat ← PSC-BACKTRACKING(A, var-domaines)
      ii. Si résultat ≠ échec alors retourner résultat
5. Retourner échec

1) Heuristique variable-plus-contrainte
2) Heuristique variable-plus-contraignante

1) Heuristique valeur-moins-contraignante

1) Sélectionner la variable ayant le plus petit domaine restant
2) Sélectionner la variable apparaissant dans le plus grand nombre de contraintes sur des variables absentes de l'assignement courant

Applications

- Les techniques des PSC permettent de résoudre des problèmes complexes et sont largement utilisées

- De nombreuses applications telles que:
  - attribution d'équipages à des lignes aériennes
  - gestion d'une flotte de transport
  - horaires de trains, d'avions, etc …
  - ordonnancement et gestion des tâches dans un port marchand
  - conception (en tous genres)
  - procédures/opérations chirurgicales (radiochirurgie)
  - etc …

Références

- Surveys
  - Kumar, AAAI Mag., 1992
  - Dechter et Frost, AAAI Mag. 1999
- Ouvrages
  - Marriott and Stuckey, 1998
  - AIMA, Russell and Norvig, 2nd ed.
- Applications
  - Freuder and Mackworth, 1994
- Conférence
  - Principles and Practice of Constraint Programming (CP)
- Journal
  - *Constraints* (Kluwer Academic Publishers)
- Internet
  - Constraints Archive
    http://www.cs.unh.edu/ccc/archive