Amélioration de *backtracking-search*

- Sans heuristiques, l'algorithme est limité
  - il peut résoudre le problème de 25 reines

- Des **heuristiques générales** peuvent améliorer l'algorithme significativement :
  - choisir judicieusement la prochaine variable (VAR-NON-ASSIGNÉE)
  - choisir judicieusement la prochaine valeur à assigner (VALEURS-ORDONNÉES)
  - détecter les assignations conflictuelles et réduire les domaines (INFÉRENCE)

Algorithme *backtracking-search*

Algorithme BACKTRACKING-SEARCH(*csp*)

1. retourner BACKTRACK({ }, *csp*)

Algorithme BACKTRACK(*assignation, csp*)

1. si *assignation* est complète, retourner *assignation*
2. *X* = VAR-NON-ASSIGNÉE(*assignation, csp*)
3. pour chaque *v* dans VALEURS-ORDONNÉES(*X*, *assignation, csp*)
    4. si COMPATIBLE((*X* = *v*), *assignation, csp*)
        5. ajouter (*X* = *v*) à *assignation*
        6. *csp** = *csp* mais où DOMAINE(*X*, *csp*) est { *v* }
        7. *csp**, *ok* = INFÉRENCE(*csp**)
        8. si *ok* = vrai
            9. *résultat* = BACKTRACK(*assignation*, *csp**)
            10. si *résultat* ≠ faux, retourner *résultat*
        11. enlever (*X* = *v*) de *assignation*
12. retourner faux

choix de prochaine variable

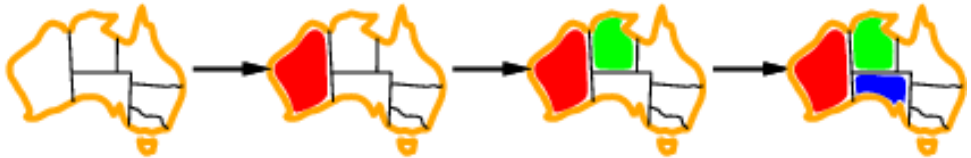
ordre des valeurs à essayer

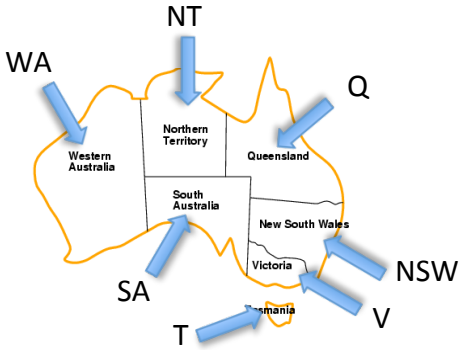
tente de simplifier le problème CSP (si détecte conflit, *ok* = faux)

Choisir l'ordre d'assignation des variables

- À chaque étape, choisir la variable avec le moins de valeurs compatibles restantes
  - c-à-d., la variable « posant le plus de restrictions »
  - appelée ***minimum remaining value*** (MRV) *heuristic*

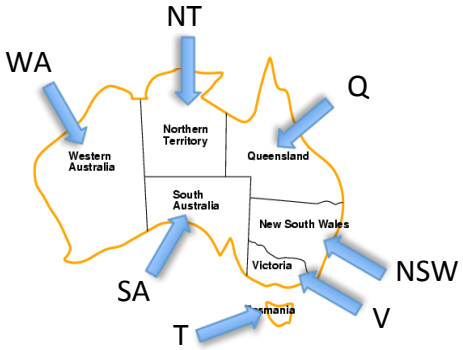
    ou ***most constrained variable*** *heuristic*.
- Illustration :

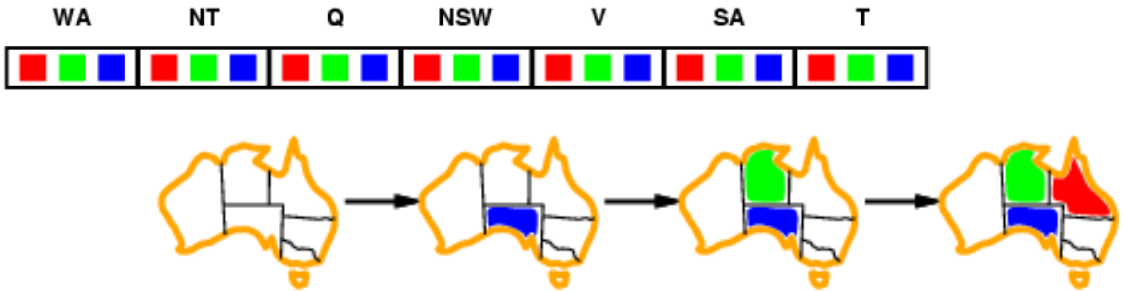




Choisir l'ordre d'assignation des variables

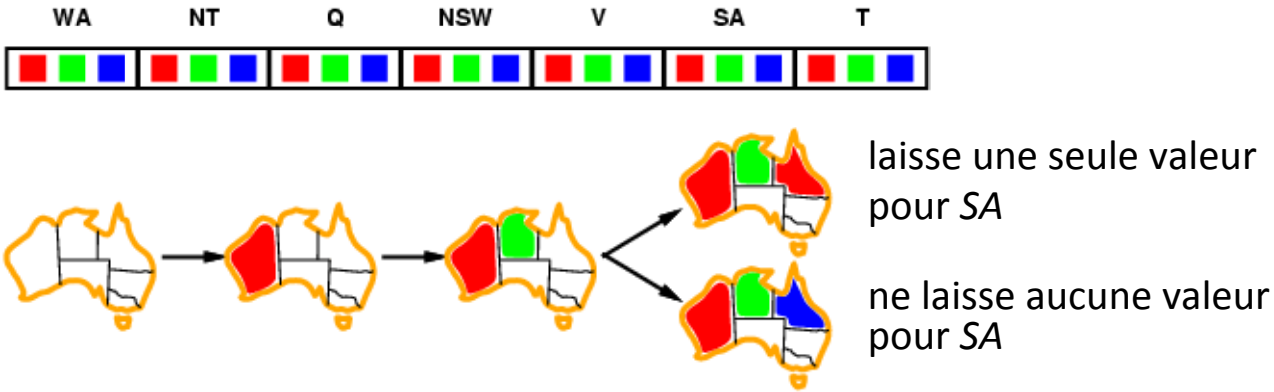
- Si le critère précédent donne des variables avec le même nombre de valeurs compatibles restantes :
  - choisir celle ayant le plus de contraintes impliquant des variables non encore assignées
  - appelée ***degree heuristic***.





Choisir la prochaine valeur à assigner

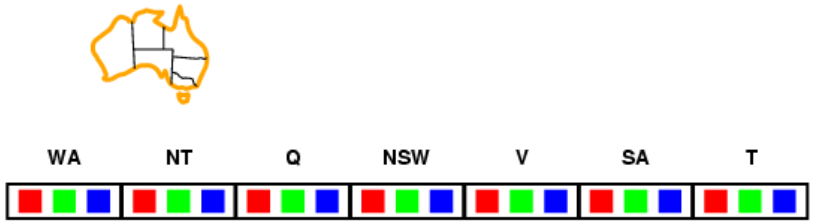
- Pour une variable donnée, choisir une valeur qui invalide le moins de valeurs possibles pour les variables non encore assignées

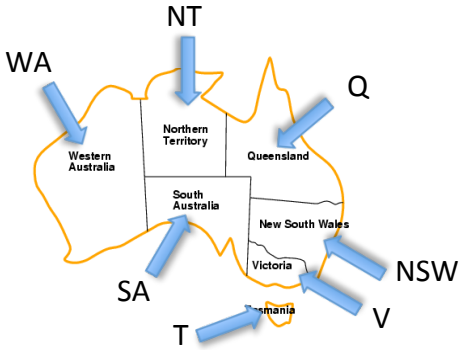


- Ces heuristiques permettent de résoudre un problème de 1000 reines

Détecter les assignations conflictuelles : algorithme *forward checking*

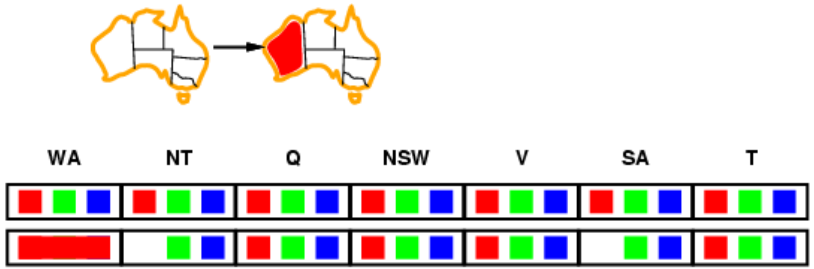
- L'idée de l'inférence *forward checking* (vérification anticipative) est de :
  - vérifier les valeurs compatibles des variables non encore assignées
  - terminer la récursivité (conflit) lorsqu'une variable (non encore assignée) a son ensemble de valeurs compatibles qui devient vide

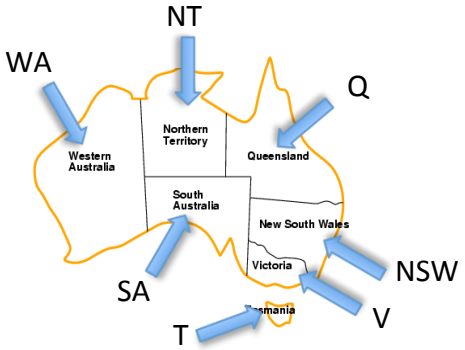




Détecter les assignations conflictuelles : algorithme *forward checking*

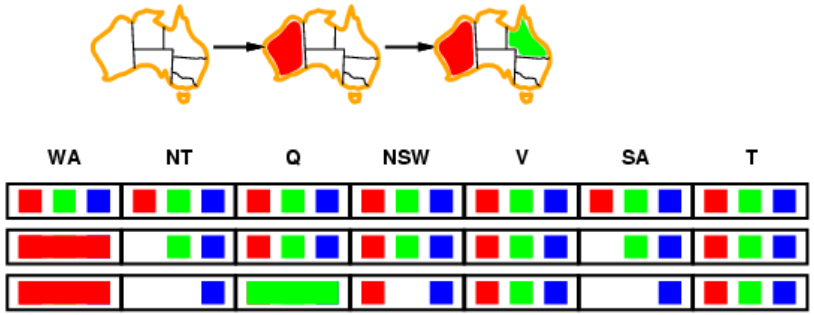
- L'idée de l'inférence *forward checking* (vérification anticipative) est de :
  - vérifier les valeurs compatibles des variables non encore assignées
  - terminer la récursivité (conflit) lorsqu'une variable (non encore assignée) a son ensemble de valeurs compatibles qui devient vide

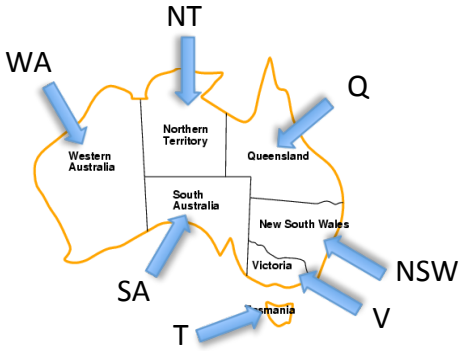




Détecter les assignations conflictuelles : algorithme *forward checking*

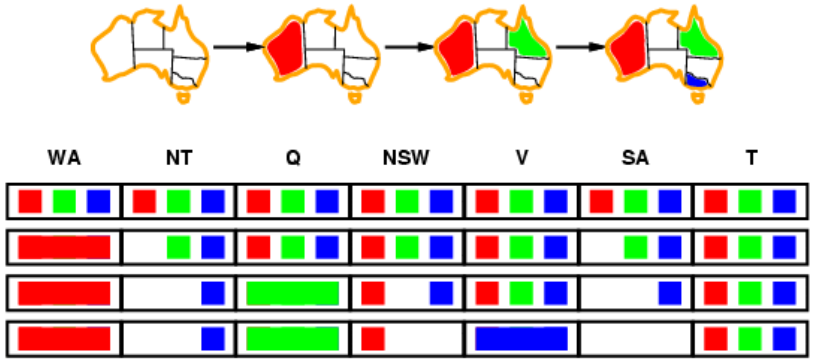
- L'idée de l'inférence *forward checking* (vérification anticipative) est de :
  - vérifier les valeurs compatibles des variables non encore assignées
  - terminer la récursivité (conflit) lorsqu'une variable (non encore assignée) a son ensemble de valeurs compatibles qui devient vide

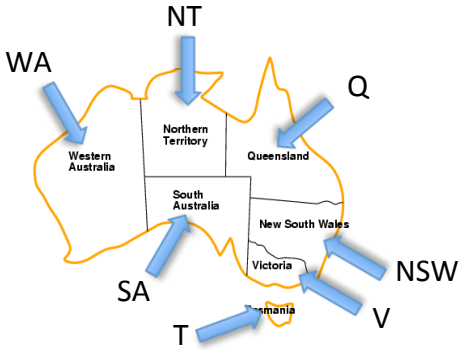




Détecter les assignations conflictuelles : algorithme *forward checking*

- L'idée de l'inférence *forward checking* (vérification anticipative) est de :
  - vérifier les valeurs compatibles des variables non encore assignées
  - terminer la récursivité (conflit) lorsqu'une variable (non encore assignée) a son ensemble de valeurs compatibles qui devient vide





Algorithme *forward checking*

Algorithme FORWARD-CHECKING(X, *csp*)

1. pour chaque X_k dans VOISINS(X, *csp*)
   2. *changé*, *csp* = RÉVISER(X_k, X, *csp*)
   3. si *changé* et DOMAINE(X_k, *csp*) est vide, retourner (void, faux)
3. retourner (*csp*, vrai)

doit spécifier la variable impliquée

on suppose que *csp* est passé par copie

Algorithme RÉVISER(X_i, X_j, *csp*) // *réduit le domaine de X_i en fonction de celui de X_j*

1. *changé* = faux
2. pour chaque *x* dans DOMAINE(X_i, *csp*)
   3. si aucun *y* dans DOMAINE(X_j, *csp*) satisfait contrainte entre X_i et X_j
      4. enlever *x* de DOMAINE(X_i, *csp*) // *ceci change la variable csp*
      5. *changé* = vrai
4. retourner (*changé*, *csp*)