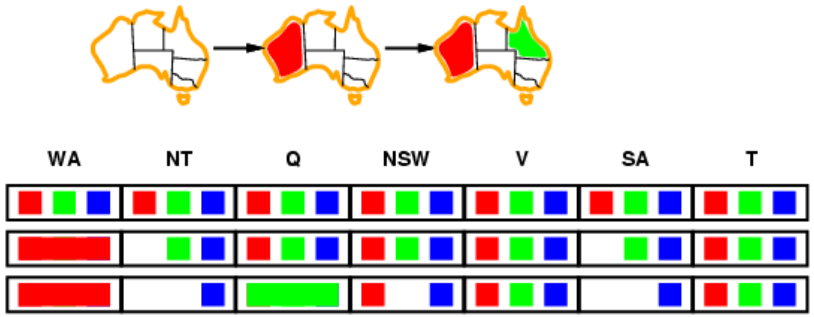
Amélioration de *backtracking-search*

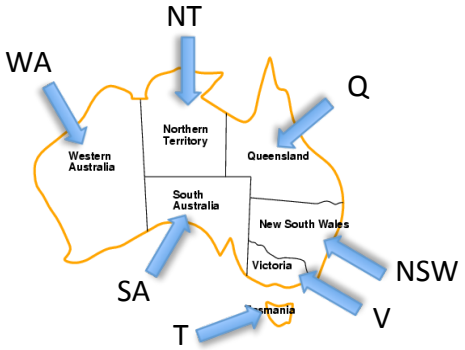
- Sans heuristiques, l'algorithme est limité
  - il peut résoudre le problème de 25 reines

- 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)

Détecter les assignations conflictuelles : algorithme *forward checking*

- *Forward checking* propage l'information des variables assignées vers les variables non assignées, mais ne détecte pas les conflits locaux entre ces variables :

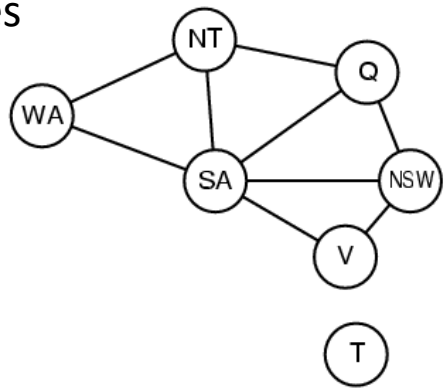


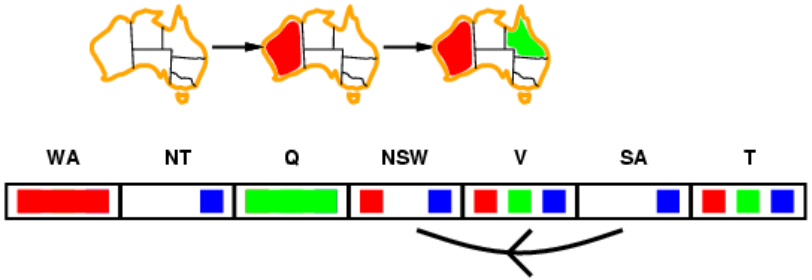


- *NT* et *SA* ne peuvent pas être bleues ensemble!
- L'inférence par propagation des contraintes permet de vérifier les contraintes localement

Arc consistency

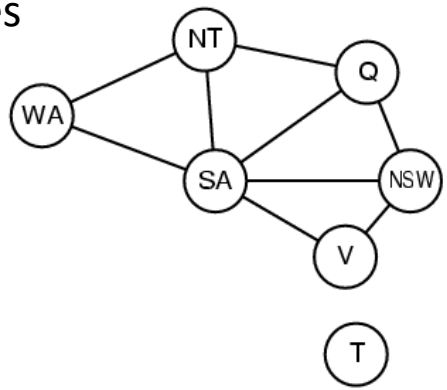
- *Arc consistency* est la forme de propagation de contraintes la plus simple
  - vérifie la compatibilité entre les arcs
    c-à-d., la compatibilité des contraintes entre deux variables
- L'arc *X* → *Y* est compatible si et seulement si
  - pour chaque valeur *x* de *X* il existe au moins
    une valeur permise *y* de *Y*

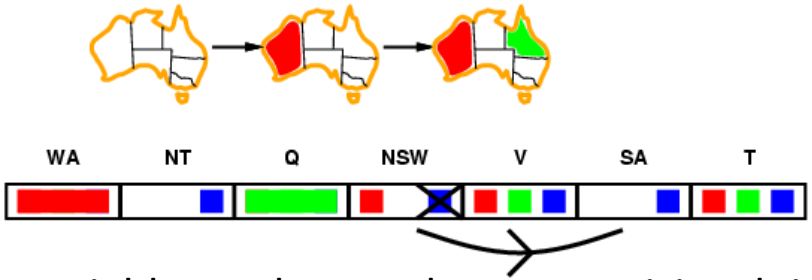




Arc consistency

- *Arc consistency* est la forme de propagation de contraintes la plus simple
  - vérifie la compatibilité entre les arcs
    c-à-d., la compatibilité des contraintes entre deux variables
- L'arc *X* → *Y* est compatible si et seulement si
  - pour chaque valeur *x* de *X* il existe au moins
    une valeur permise *y* de *Y*

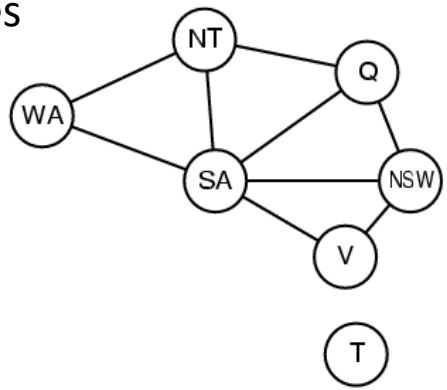


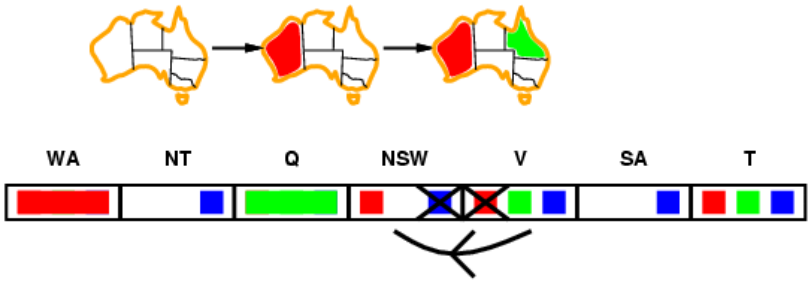


Si une variable perd une valeur, ses voisins doivent être revérifiés

Arc consistency

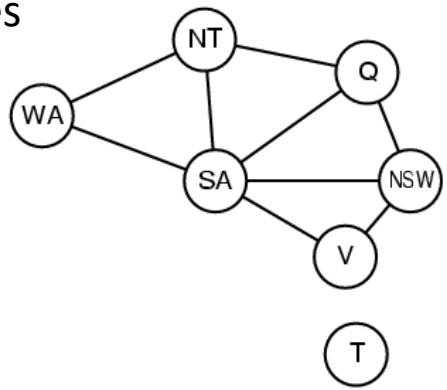
- *Arc consistency* est la forme de propagation de contraintes la plus simple
  - vérifie la compatibilité entre les arcs
    c-à-d., la compatibilité des contraintes entre deux variables
- L'arc *X* → *Y* est compatible si et seulement si
  - pour chaque valeur *x* de *X* il existe au moins
    une valeur permise *y* de *Y*

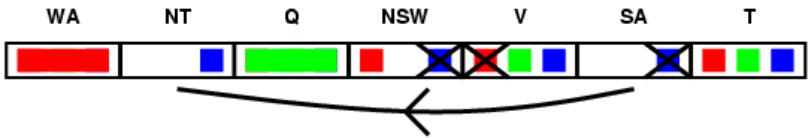




Arc consistency

- *Arc consistency* est la forme de propagation de contraintes la plus simple
  - vérifie la compatibilité entre les arcs
    c-à-d., la compatibilité des contraintes entre deux variables
- L'arc *X* → *Y* est compatible si et seulement si
  - pour chaque valeur *x* de *X* il existe au moins
    une valeur permise *y* de *Y*





Algorithme *Arc consistency* (AC-3)

Algorithme AC-3(*csp*) // *retourne le CSP simplifié et un booléen vrai si pas de conflit*

1. *file_arcs* = ARCS-DU-CSP(*csp*)
2. tant que *file_arcs* n'est pas vide
3. (X_i, X_j) = POP(*file_arc*) // *retire premier arc de la file*
4. *changé*, *csp* = RÉVISER(X_i, X_j, *csp*) // *vérifie la compatibilité de l'arc*
5. si *changé*
6. si DOMAINE(X_i, *csp*) est vide, retourner (void, faux)
7. pour chaque X_k dans VOISINS(X_i, *csp*)
8. si $X_k \neq X_j$, ajouter (X_k, X_i) dans *file_arcs*
8. retourner (*csp*, vrai)

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*)

Algorithme *Arc consistency* (AC-3)

- Appliqué au début de *backtracking-search* et/ou juste après chaque nouvelle assignation de valeur à une variable

- Complexité : O($C\ D^3$) dans le pire cas, où C est le nombre de contraintes
  - complexité de RÉVISER : O(D^2)
  - on a O(C) arcs, qui peuvent être réinsérés dans la file O(D) fois par RÉVISER
  - RÉVISER peut donc être appelé O($C\ D$), pour une complexité globale de O($C\ D^3$)

- Une meilleure version en O($C\ D^2$) dans le pire cas existe : AC-4
  - par contre AC-3 est en moyenne plus efficace

- Exploiter la structure du domaine (Section 6.5)
  - certains graphes de contraintes ont une structure « simple » qui peut être exploitée (ex. : un arbre)
  - peut améliorer le temps de calcul exponentiellement