Problème de satisfaction de contraintes

- Formellement, un problème de satisfaction de contraintes (ou CSP pour *Constraint Satisfaction Problem*) est défini par :
  - un **ensemble fini de variables** $V = \{ X_1, \ldots, X_N \}$
    - » chaque variable X_i a un **domaine** D_i de valeurs possibles
  - un **ensemble fini de contraintes** $C_1, \ldots, C_M$ sur les variables.
    - » une contrainte restreint les valeurs pour un sous-ensemble de variables

- Un **état (nœud)** d'un problème CSP est défini par une **assignation de valeurs** $\{X_i=v_i, X_j=v_j, \ldots\}$ à certaines variables ou à toutes les variables
  - une assignation qui viole aucune contrainte est dite **compatible** ou **légale**
  - une assignation est **complète** si elle concerne toutes les variables
  - une solution à un problème CSP est
    une assignation **complète et compatible**

Algorithme *Depth-First-Search* pour CSP

- On peut utiliser la recherche dans un graphe avec les paramètres suivants :
  - un état est une assignation
  - état initial : assignation vide { }
  - fonction de transition : assigne une valeur à une variable non encore assignée
  - fonction but : retourne vrai si l'assignation est complète et compatible

- L'algorithme est général et s'applique à tous les problèmes CSP
- Comme la solution doit être complète,
  elle apparaît à une profondeur *N*

Limitations de l'approche précédente

- Taille de l'arbre de recherche :
  - le nombre de branches au premier niveau, dans l'arbre est de $N*D$ (D est la taille du domaine), parce que nous avons N variables, chacune pouvant prendre D valeurs
  - au prochain niveau, on a (N-1) D successeurs pour chaque nœud
  - ainsi de suite jusqu'au niveau N
  - cela donne $N!*D^N$ nœuds, pour seulement D^N assignations complètes

- L'algorithme ignore la **commutativité** des transitions :
  - $SA=R$ suivi de $WA=B$ est équivalent à $WA=B$ suivi de $SA=R$
  - si on tient compte de la commutativité, le nombre de nœuds générés est D^N

- **Idée 1 :** considérer **une seule variable** à assigner
  à chaque niveau

Hugo Larochelle et Froduald Kabanza

Limitations de l'approche précédente

- Inutile de continuer à assigner des variables à un état s'il y a déjà des contraintes qui sont violées

- **Idée 2 : reculer (*backtrack*)** lorsqu'aucune nouvelle assignation compatible est possible

- Le ***backtracking-search*** est le résultat de la combinaison de ces deux idées
  - c'est l'algorithme de base pour résoudre les problèmes CSP

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

information sur les variables, domaines et contraintes du problème CSP

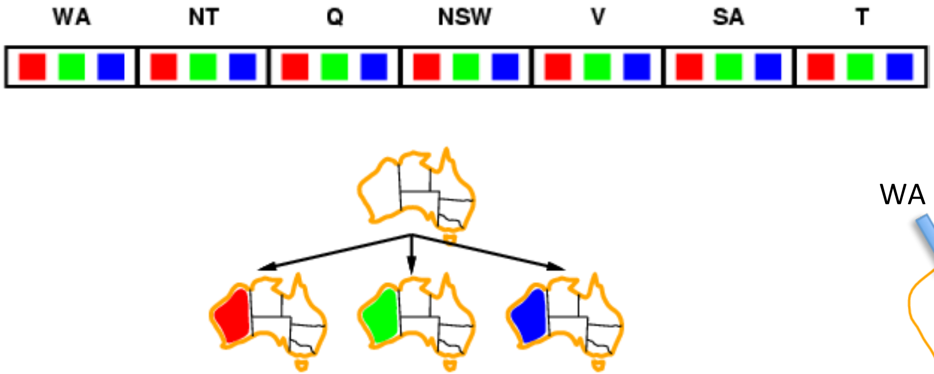
assignation de variables à des valeurs

choix de prochaine variable

ordre des valeurs à essayer

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

Illustration de *backtracking-search*



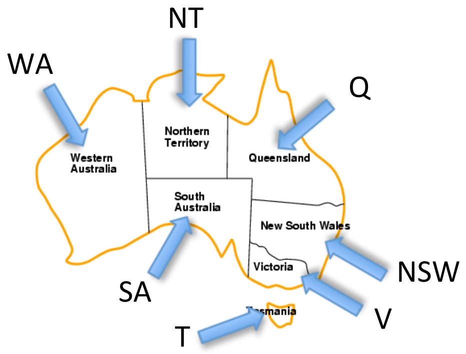
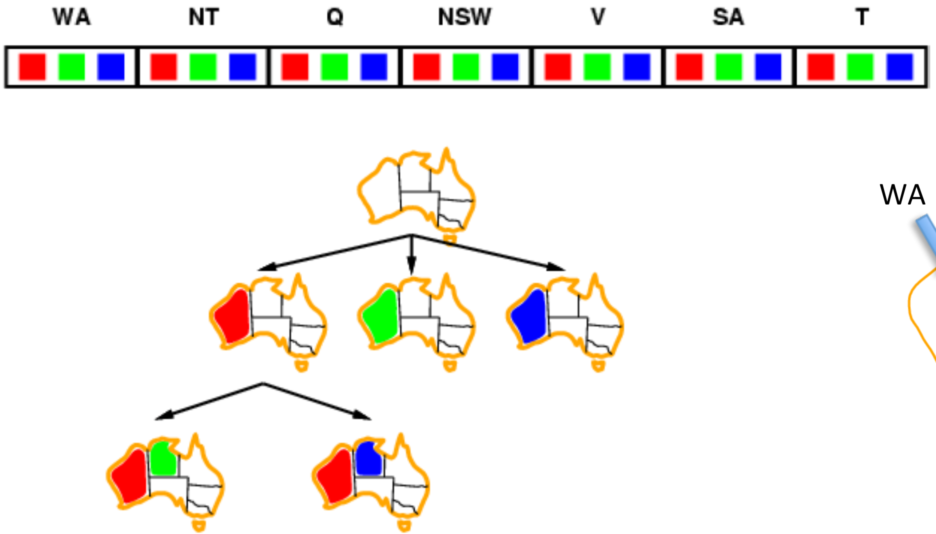


Illustration de *backtracking-search*



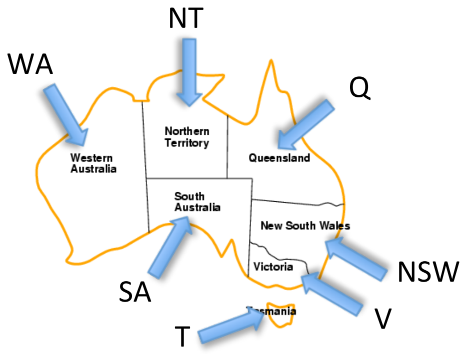


Illustration de *backtracking-search*