Problèmes de satisfaction de contraintes

M1 Info 2018–2019 *Intelligence Artificielle*

Stéphane Airiau

LAMSADE

DAUPHINE
UNIVERSITÉ PARIS

CSP

Les données :

- un ensemble X de **variables** ;
- le domaine D_V des **valeurs possibles** de chaque variable $V \in X$;
- un ensemble de **contraintes** entre les variables
  (exemple $X_1 \neq X_2$: la valeur de X_1 doit être différente de la valeur de X_2)

Une solution : une affectation de toutes les variables de X avec une valeur de son domaine de façon a ce que toutes les contraintes soient satisfaites.

Domaine

Le domaine d'une variable peut être

- discret : on pourrait énumérer toutes les possibilités pour les contraintes
- infini : un langage de contrainte est nécessaire pour représenter toutes les contraintes sans énumération
- continu : par exemple programmation linéaire
  ⇨ les contraintes sont des égalités ou inégalités de combinaisons linéaires des variables

Contraintes

Les contraintes peuvent être

- unaires : restriction de la variable à des constantes
- binaires : met en relations deux variables. Si toutes les relations sont binaires, on peut représenter le problème avec un graphe.
- complexes : la relation est entre trois variables ou plus
- globales : met en relation toutes les variables
  (par exemple, une contrainte peut être que toutes les variables prennent des valeurs différentes)
- pour des domaines discrets et finis, on peut toujours se rapporter à un problème avec des relations binaires
  (en introduisant des variables auxiliaires).
- certaines contraintes indiquent des préférences
  ex : dans un problème d'emploi du temps, professeur A préfère enseigner le matin et professeur B l'après midi

⇨ on peut utiliser des techniques d'optimisation : problèmes d'optimisation avec des contraintes
ex : programmation linéaire

Résolution d'un CSP

- assigner une valeur à une variable va avoir un impact sur les valeurs possibles pour les autres variables
  ⇨ **propagation**
- essayer des affectations ⇨ **recherche**

⇨ combiner recherche et propagation

Effectuer la recherche : Backtracking (*pas* de propagation)

on va effectuer une recherche en profondeur d'abord où

- à chaque niveau, on cherche à affecter une variable
- on backtrack à chaque fois qu'il n'y a plus de valeurs disponibles pour affecter une des variables restantes

A cause du formalisme d'un CSP, l'algorithme est générique et fonctionnera pour tout problème de CSP !

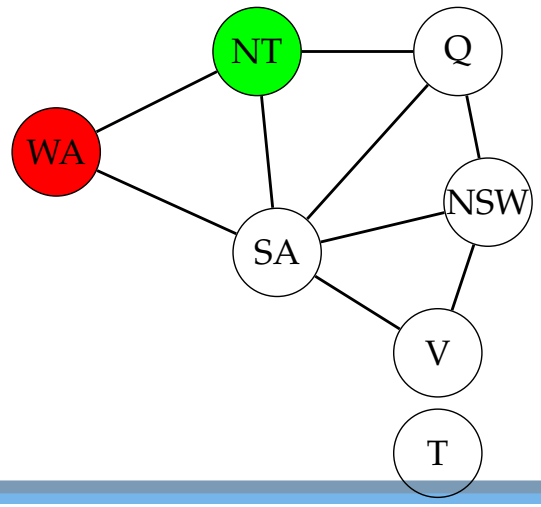
Comment choisir l'ordre de l'examen des variables et des valeurs ?

On peut utiliser des heuristiques "générales"
Pour les *variables*

- choisir la variable qui a le moins de valeurs disponibles
- choisir la variable qui a le plus de contraintes avec des variables non affectées.

Pour les valeurs :

- choisir la valeur qui contraint le moins les voisins



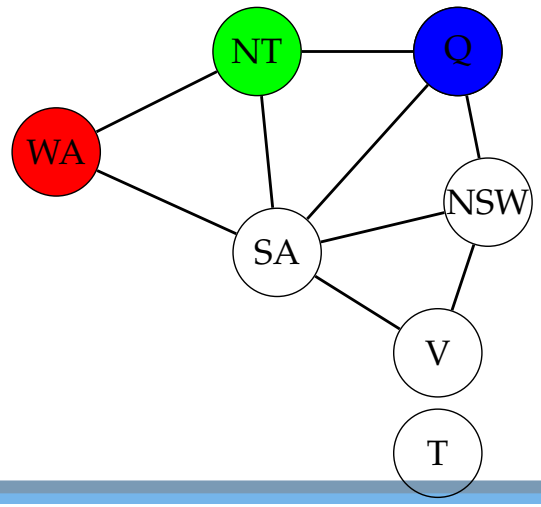
Comment choisir l'ordre de l'examen des variables et des valeurs ?

On peut utiliser des heuristiques "générales"
Pour les *variables*

- choisir la variable qui a le moins de valeurs disponibles
- choisir la variable qui a le plus de contraintes avec des variables non affectées.

Pour les valeurs :

- choisir la valeur qui contraint le moins les voisins



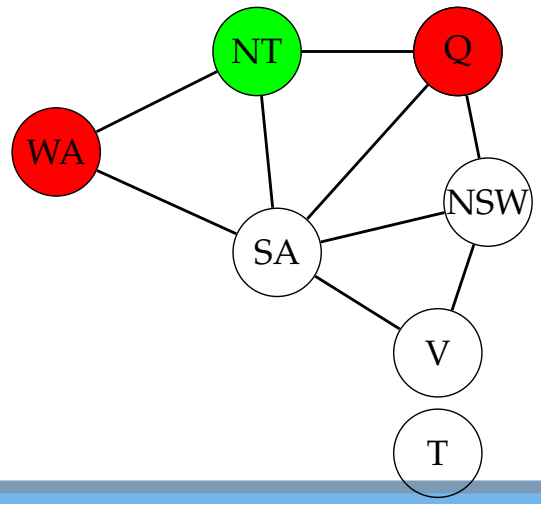
Comment choisir l'ordre de l'examen des variables et des valeurs ?

On peut utiliser des heuristiques "générales"
Pour les *variables*

- choisir la variable qui a le moins de valeurs disponibles
- choisir la variable qui a le plus de contraintes avec des variables non affectées.

Pour les valeurs :

- choisir la valeur qui contraint le moins les voisins



Comment choisir l'ordre de l'examen des variables et des valeurs ?

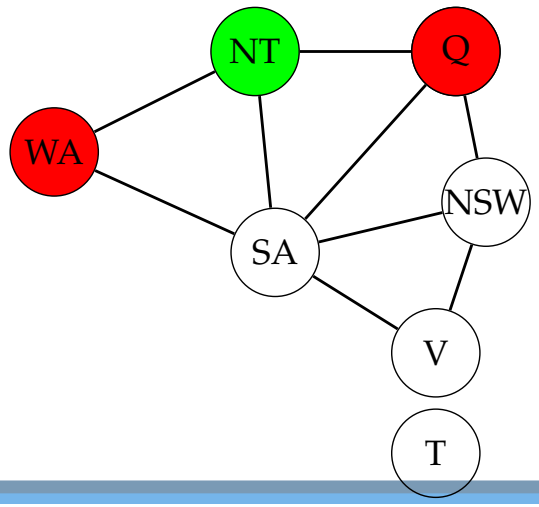
On peut utiliser des heuristiques "générales"
Pour les *variables*

- choisir la variable qui a le moins de valeurs disponibles
- choisir la variable qui a le plus de contraintes avec des variables non affectées.

Pour les valeurs :

- choisir la valeur qui contraint le moins les voisins

  dans l'exemple, on préfère donc rouge à bleu car rouge laisse plus d'option pour `SA`.



Comment choisir l'ordre de l'examen des variables et des valeurs ?

On peut utiliser des heuristiques "générales"
Pour les *variables*

- choisir la variable qui a le moins de valeurs disponibles
- choisir la variable qui a le plus de contraintes avec des variables non affectées.

Pour les valeurs :

- choisir la valeur qui contraint le moins les voisins

Ces heuristiques donnent de bons résultats de manière empirique (pas de preuves que c'est plus rapide).

Ces heuristiques fonctionnent quel que soit le problème à résoudre
⇨ façon de résoudre générale et "intelligente".

Propagation de contraintes

On n'a toujours pas utiliser l'idée de mettre à jour les domaines des variables voisines lorsqu'on a choisi une valeur pour une variable !

Definition (noeud cohérence)

une variable est *"noeud cohérente"* si elle satisfait toutes les contraintes *unaires*.

pré-processing : il suffit de boucler sur chaque variable pour garantir cette cohérence.

Definition (**arc cohérence**)

une variable X_i est *arc cohérente* par rapport à une autre variable X_j si pour toutes valeurs dans D_i, il existe une valeur dans D_j qui satisfait toutes les contraintes entre X_i et X_j.

Exemple :

- contrainte $Y = X^2$
- domaine de X et Y : ce sont des chiffres
  $D_X = D_Y = \{0,1,2,4,5,6,7,8,9\}$

  pour rendre X arc cohérent avec Y ⇨ réduction à $D_X = \{0,1,2,3\}$

  pour rendre Y arc cohérent avec X ⇨ réduction à $D_Y = \{0,1,4,9\}$

Un CSP est arc cohérent si toutes les variables sont arc cohérentes

Propagation de contraintes

N.B. Parfois l'arc cohérence n'aide pas.

ex : coloration de la carte de l'australie.
La contrainte sur le couple (SA, WA) donne les possibilités d'affectation suivantes :
$\{(rouge, vert), (vert, rouge), (rouge, bleu), (bleu, rouge), (bleu, vert), (vert, bleu)\}$.
Mais quel que soit le choix pour SA il y a des valeurs valides pour les autres variables
⇨pas d'effet sur les domaines ici.

Algorithme pour garantir la cohérence

```
1  function AC3 (csp) {
2     Queue Q contenant tous les arcs du csp
3     while (Q≠∅) {
4       (X_i, X_j) ← pop(Q)
5       if (Revise(X_i, X_j)) { // si le domaine de X_i a été réduit
6         if (D(x_i) = ∅) //le CSP n'admet pas de solutions cohérentes!
7           return false
8         else
9           Pour tous les voisins X_k de X_i (sauf X_j)
10            ajouter (X_k, X_i) dans Q
11      }
12    }
13    return true
14 }
```

Algorithme pour garantir la cohérence

```
1  function Revise (csp, X_i, X_j ) {
2      // returns vrai si on a changé le domaine de X_i
2    change ← false
3    for each v_i ∈ D(X_i) {
4      if aucune valeur y de X_j ne permet à (x,y) de satisfaire
4          les contraintes entre X_i et X_j
5        enlève v_i de D(x_i)
6        change ← true
7      }
7    }
8    return change
9  }
```

Algorithme pour garantir la cohérence

```
1  function Revise (variable x_i, constraint c) {
2        returns whether the domain of variable x_i has changed
2     change ← false
3     for each v_i ∈ D(x_i) {
4       if (∄ tuple τ ∈ c with τ(x_i) = v_i) {
5         remove v_i from D(x_i)
6         change ← true
7       }
7     }
8     return change
9  }
```

```
1  function AC3 (csp) {
2     Queue Q ← {(x_i, c) | c ∈ C, x_i ∈ X(c)}
3     while (Q ≠ ∅) {
4       take (x_i, c) from Q
5       if (Revise(x_i, c)) {
6         if (D(x_i) = ∅)
7           return false
8         else
9           Q ← Q ∪ {(x_j, c') | c' ∈ C ∧ c' ≠ c ∧ (x_i, x_j) ∈ X(c')² ∧ j ≠ i}
10      }
11    }
12    return true
12 }
```

arc-cohérence

- après l'exécution de l'algorithme, on a un CSP équivalent mais il peut y avoir moins de valeurs dans chaque domaine

⇨ la recherche pour résoudre le CSP sera sûrement plus facile

- pour des contraintes binaires, la complexité de l'algorithme est $\mathcal{O}(cd^3)$ où d est le nombre de valeurs maximum dans chaque domaine et c est le nombre de contraintes

arc-cohérence n'est pas forcément assez :

Si on a seulement deux couleurs,
• le csp est arc-cohérent
• mais il n'y a pas de solution !
WA, NT et SA se touchant, il faut au moins 3 couleurs !

⇨ arc-cohérence ne peut détecter le problème

autres types de cohérence

Il existe d'autres types de cohérence

- cohérence de chemin
- k-cohérence

⇨ permet de détecter plus rapidement des solutions non valides ou aller plus rapidement à une solution
mais nécessite plus de calculs !

⇨ équilibre entre passer du temps à raisonner sur les contraintes et faire la recherche.

Il existe aussi des méthodes spécialisées pour certains types de contraintes.

ex : contraintes `allDiff`

Forward checking

On va maintenant **allier** la *recherche* et la *propagation* des contraintes.
Principe :

L'algorithme procède comme dans backtrack :

- on effectue une recherche en profondeur d'abord
- un noeud de l'arbre de recherche correspond à l'affectation d'une variable

en plus, à chaque affectation d'une variable X

- on vérifie l'arc-cohérence entre chacun de ses voisins de X et X
- ⇨ on réduit le domaine des voisins de X
- N.B. on ne continue pas avec les voisins des voisins

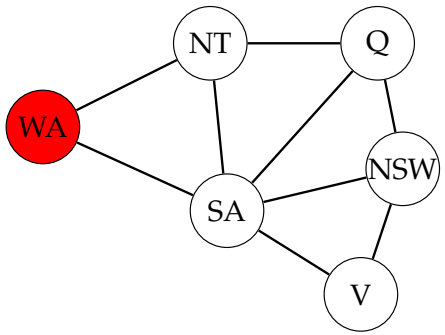
Forward checking

```
1  ForwardChecking (CSP net, Affectation a)
2    si l'affectation a est complète alors retourne a
3    Var ← variable suivante non affectée
4    Pour toutes valeurs val ∈ D(Var)
5       si {Var=val} ne viole aucune contrainte
6           a = a ∪ {Var=val}
7           pour toutes variables X connectée à Var
8               maintenir arc-cohérence de Var et X
9           result ← Backtrack(net, a)
10          si result ≠ échec
11              retourne result
12          sinon retourne échec
```

Forward checking

On peut vérifier l'arc-cohérence pendant la recherche pour éliminer plus vite des solutions partielles.

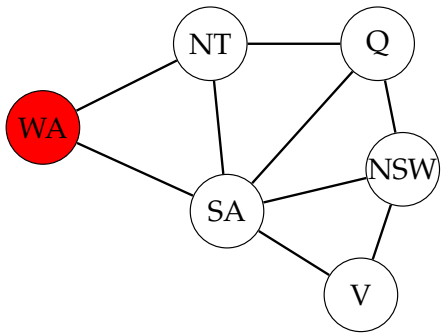
	WA	NT	Q	NSW	V	SA	T
Domaines initiaux	r v b	r v b	r v b	r v b	r v b	r v b	r v b



Forward checking

On peut vérifier l'arc-cohérence pendant la recherche pour éliminer plus vite des solutions partielles.

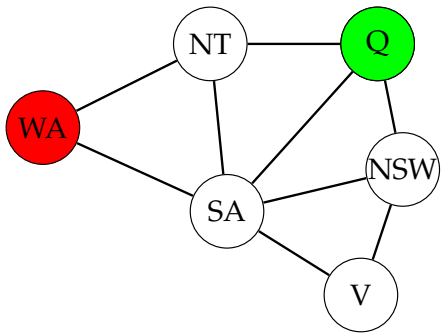
	WA	NT	Q	NSW	V	SA	T
Domaines initiaux	r v b	r v b	r v b	r v b	r v b	r v b	r v b
Affectation WA=r	r	v b	r v b	r v b	r v b	v b	r v b



Forward checking

On peut vérifier l'arc-cohérence pendant la recherche pour éliminer plus vite des solutions partielles.

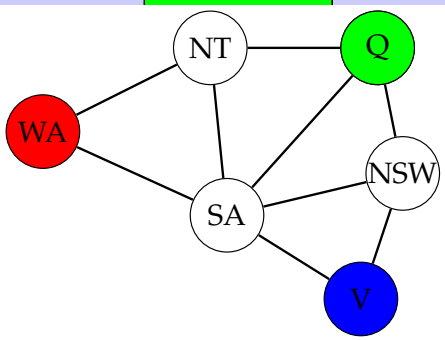
	WA	NT	Q	NSW	V	SA	T
Domaines initiaux	r v b	r v b	r v b	r v b	r v b	r v b	r v b
Affectation WA=r	r	v b	r v b	r v b	r v b	v b	r v b
Affectation Q=v	r	b	v	r b	r v b	b	r v b



Forward checking

On peut vérifier l'arc-cohérence pendant la recherche pour éliminer plus vite des solutions partielles.

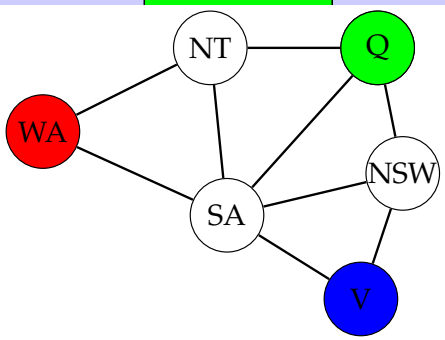
	WA	NT	Q	NSW	V	SA	T
Domaines initiaux	r v b	r v b	r v b	r v b	r v b	r v b	r v b
Affectation WA=r	r	v b	r v b	r v b	r v b	v b	r v b
Affectation Q=v	r	b	v	r b	r v b	b	r v b
Affectation V=b	r	b	v	r	b		r v b



Forward checking

On peut vérifier l'arc-cohérence pendant la recherche pour éliminer plus vite des solutions partielles.

	WA	NT	Q	NSW	V	SA	T
Domaines initiaux	r v b	r v b	r v b	r v b	r v b	r v b	r v b
Affectation WA=r	r	v b	r v b	r v b	r v b	v b	r v b
Affectation Q=v	r	b	v	r b	r v b	b	r v b
Affectation V=b	r	b	v	r	b		r v b



Certaines inconsistances ne sont pas détectées par forward checking (ligne 2, NT et SA n'ont plus qu'une valeur disponible alors qu'ils sont voisins !)
L'algorithme MAC (Maintenir arc cohérence) effectue un propagation récursive des contraintes

Conclusion

- formulation particulière d'un état avec un ensemble de couples valeurs/attributs
- les conditions d'une solution sont représentées par un ensemble de contraintes sur les variables
- la recherche avec backtrack est une technique efficace.
- on a des heuristiques générales (indépendantes du domaine) qui permettent de résoudre plus rapidement un csp.
- d'autres techniques utilisant la décomposition peuvent aussi être efficaces.