Problèmes de satisfaction de contraintes

M1 IDD 2019–2020 *Intelligence Artificielle*

Stéphane Airiau

LAMSADE

DAUPHINE
UNIVERSITÉ PARIS

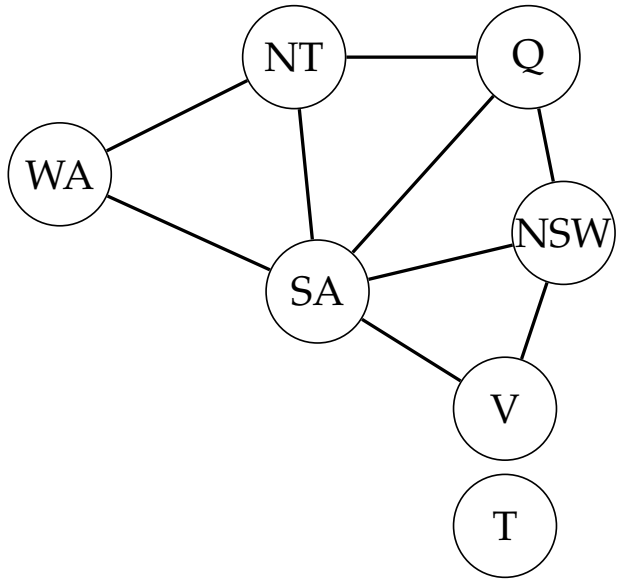
Ce cours suit le chapître 6 de “Artificial Intelligence : A modern Approach”.

Exemple : colorier une carte

On possède une carte d'Australie montrant ses territoires et états. On a trois couleurs à notre disposition Rouge, Bleu et Vert. Notre problème est de colorier chaque état de sorte que deux états voisins n'ont jamais la même couleur.

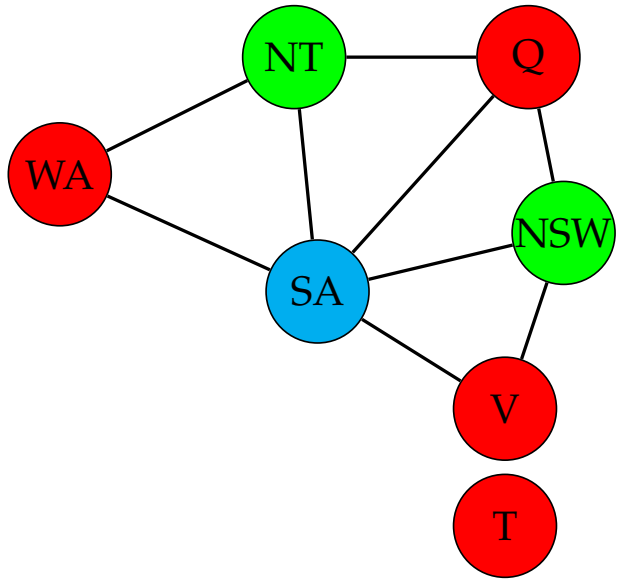
Modèle

- ensemble des variables : $X = \{WA, NT, SA, Q, NSW, V, T\}$
- domaine de chaque variable $D_i = \{rouge, bleu, vert\}$
- ensemble des contraintes $\mathcal{C} = \{WA \neq NT, WA \neq SA, NT \neq SA, SA \neq Q, SA \neq NSW, SA \neq V, NT \neq Q, Q \neq NSW, NSW \neq V\}$
  avec un abus de notation



Modèle

- ensemble des variables : $X = \{WA, NT, SA, Q, NSW, V, T\}$
- domaine de chaque variable $D_i = \{rouge, bleu, vert\}$
- ensemble des contraintes $\mathcal{C} = \{WA \neq NT, WA \neq SA, NT \neq SA, SA \neq Q, SA \neq NSW, SA \neq V, NT \neq Q, Q \neq NSW, NSW \neq V\}$
  avec un abus de notation
- une solution possible : $\{WA = rouge, NT = vert, Q = rouge, NSW = vert, V = rouge, SA = bleu, T = rouge\}$



Pourquoi choisir un tel modèle ?

- utilisation des contraintes pour éliminer un bon nombre d'états

  par exemple dès que $SA = bleu$, aucun noeud voisin ne peut prendre la couleur bleu.

➪ au lieu de devoir considérer $3^5 = 243$ allocations possibles une fois que $SA = bleu$, il ne reste plus que $2^5 = 32$ allocations possibles !

- Dans les recherches dans les espaces d'états, on peut simplement demander "cet état est-il oui ou non solution"

  avec un CSP, si une solution partielle n'est pas solution, on n'a pas besoin de considérer ses complétions

- représentation naturelle pour bons nombres de problèmes intéressants

Autres exemples

Problème des n-reines : On veut disposer n reines sur un échiquier de taille $n \times n$ sans qu'aucune reine ne s'attaque :

- ensemble de variables est l'ensemble des reines : $X = \{R_1, \ldots, R_n\}$
- le domaine de chaque reine est l'ensemble des positions sur l'échiquier
- les contraintes indiquent que les reines ne doivent pas s'attaquer entre elles

On peut aussi réduire le nombre d'états en ajoutant que la reine i est placée sur la colonne i

- ensemble de variables est l'ensemble des reines : $X = \{R_1, \ldots, R_n\}$
- le domaine de chaque reine est l'ensemble des lignes $\mathcal{C} = \{1, \ldots, n\}$
- les contraintes indiquent que les reines ne doivent pas s'attaquer entre elles

autres problèmes : sudoku

Sudoku

- 81 variables
- cases vides : domaine $\{1,2,3,4,5,6,7,8,9\}$
- cases remplies : domaine est un singleton contenant le chiffre
- une contrainte `allDiff` par ligne et par colonne

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

(b)

Un autre problème : problème d'ordonnancement

- on a un ensemble de tâches ⇨ variables $X = \{T_1, \ldots, T_n\}$
- chaque tâcheT_i a une durée d_i
- certaines tâches doivent être effectuées avant d'autres ⇨ contraintes
- objectif : trouver un temps (par exemple en minute) où chaque tâche doit commencer ⇨ domaine

ex 1 : tâche T_1 dure $d_1 = 10$min et doit être effectuée avant tâche T_2

$$T_1 + d_1 \leqslant T_2$$

ex 2 : on veut dire que deux tâches T_3 et T_4 ne peuvent pas avoir lieu en même temps (par exemple car les deux tâches utilisent un outil en commun).

Ici, on veut exprimer que soit on fait T_3 avant T_4 ou l'inverse, mais on ne veut satisfaire qu'une seule des deux contraintes

$$(T_3 + d_3 \leqslant T_4) \vee (T_4 + d_4 \leqslant T_3)$$

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

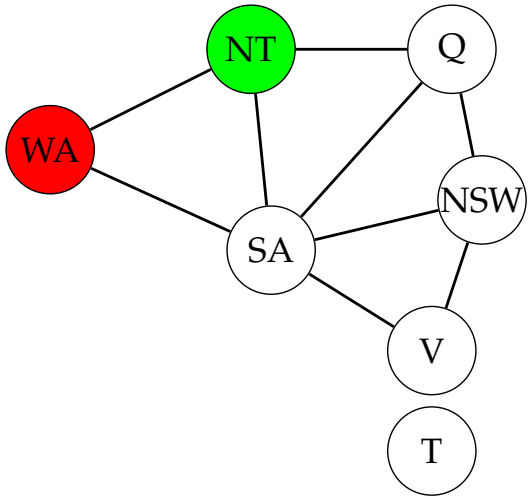
On peut utiliser des heuristiques "générales"
Pour les *variables*

- choisir la variable qui a le moins de valeurs disponibles
  ⇨choisir une variable qui risque forcer rapidement un *échec*
  ⇨élagage rapide
- choisir la variable qui a le plus de contraintes avec des variables non affectées.
  ⇨choisir la variable qui va réduire le facteur de branchement dans le futur.

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

Pour les valeurs :

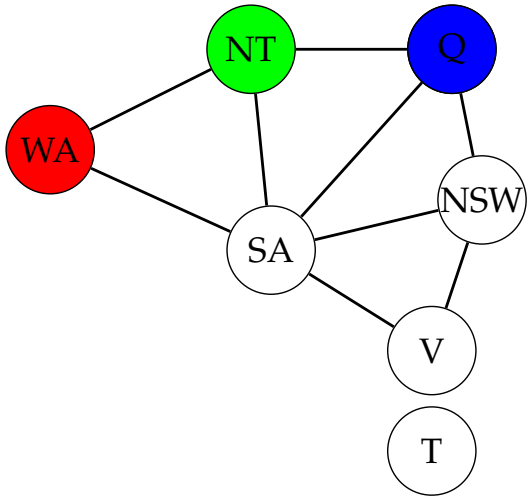
- choisir la valeur qui contraint le moins les voisins
  on cherche une solution, donc on essaye d'utiliser les valeurs les plus prometteuses.



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

Pour les valeurs :

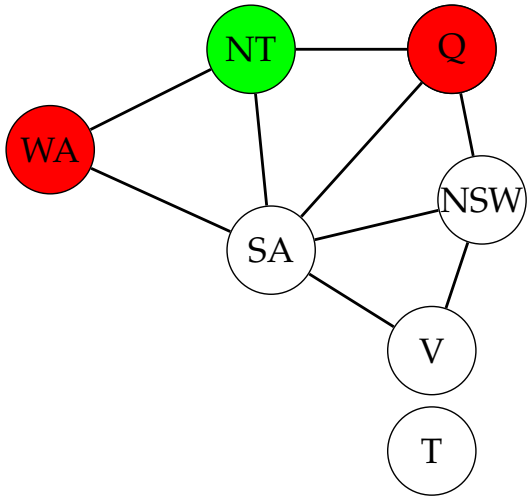
- choisir la valeur qui contraint le moins les voisins
  on cherche une solution, donc on essaye d'utiliser les valeurs les plus prometteuses.



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

Pour les valeurs :

- choisir la valeur qui contraint le moins les voisins
  on cherche une solution, donc on essaye d'utiliser les valeurs les plus prometteuses.

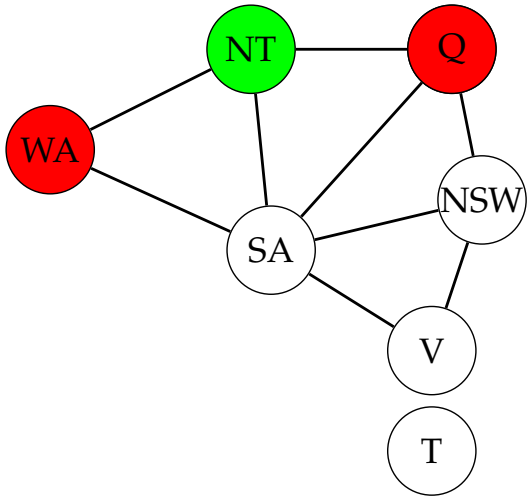


Comment choisir l'ordre de l'examen des valeurs ?

Pour les valeurs :

- choisir la valeur qui contraint le moins les voisins
  on cherche une solution, donc on essaye d'utiliser les valeurs les plus prometteuses.

  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`

autres types de cohérence

cohérence de chemin $\{X_i, X_j\}$ est "chemin-cohérent" par rapport à X_k si pour chaque affectation $X_i = a$ et $X_j = b$ cohérente avec les contraintes sur $\{X_i, X_j\}$, il existe toujours une affectation de X_k cohérente avec les contraintes sur $\{X_i, X_k\}$ et $\{X_j, X_k\}$.

ex : Coloration de la carte avec deux couleur rouge et bleu.
Question : Peut-on rendre $\{WA, SA\}$ chemin-cohérent par rapport à NT ?

- affectation WA=bleu et SA=rouge.

⇨ NT ne peut pas être affecté sans violer une contrainte ($WA \neq NT$ ou $SA \neq NT$).

⇨ on détecte l'impossibilité.

Notion encore plus forte :
k-cohérence un CSP est k-cohérent si pour chaque ensemble de $k-1$ variables et pour chaque affectation cohérente de ces variables, une valeur cohérente peut toujours être trouvée pour la $k^{\text{ième}}$ variable.

Contraintes globales : `allDiff`

On peut avoir des méthodes spécialisées pour certaines contraintes globales.

pour `allDiff` on peut utiliser l'algorithme suivant :

```
1  tant que ( chaque domaine n'est pas vide
                          et
           il y a plus de variables restante que de valeurs restante )
2      on enlève une variable qui a comme domaine un singleton
3      on efface la valeur présente dans le singleton des domaines des autres variables
```

ceci sera plus efficace que tester la n-cohérence du problème !

Autres contraintes spéciales

il existe d'autres types de contraintes globales

- contraintes sur les ressources :
  ex : `auPlus` on peut vouloir dire que la somme des valeurs des variables A, B, C et D soit 10. Si les domaines sont $\{2,3,4,5,6\}$

Autres contraintes spéciales

il existe d'autres types de contraintes globales

- contraintes sur les ressources :
  ex : `auPlus` on peut vouloir dire que la somme des valeurs des variables A, B, C et D soit 10. Si les domaines sont $\{2,3,4,5,6\}$
  on peut enlever 5 et 6 de chaque domaine !

Autres contraintes spéciales

il existe d'autres types de contraintes globales

- contraintes sur les ressources :
  ex : `auPlus` on peut vouloir dire que la somme des valeurs des variables A, B, C et D soit 10. Si les domaines sont $\{2,3,4,5,6\}$
  on peut enlever 5 et 6 de chaque domaine !
- contraintes avec des bornes
  ex : si $A+B=420$ et $D_A=[0,165]$, $D_B=[0,385]$

Autres contraintes spéciales

il existe d'autres types de contraintes globales

- contraintes sur les ressources :
  ex : `auPlus` on peut vouloir dire que la somme des valeurs des variables A, B, C et D soit 10. Si les domaines sont $\{2,3,4,5,6\}$

  on peut enlever 5 et 6 de chaque domaine !
- contraintes avec des bornes
  ex : si $A+B=420$ et $D_A=[0,165]$, $D_B=[0,385]$

  on peut réduire le domaine à $D_A=[35,165]$, $D_B=[255,385]$

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

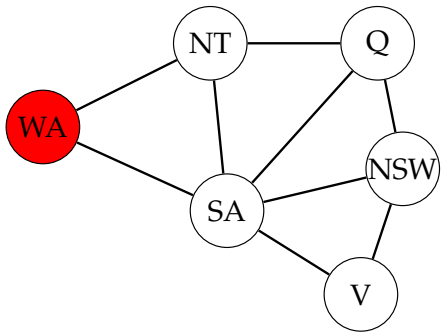
Forward checking

```
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.

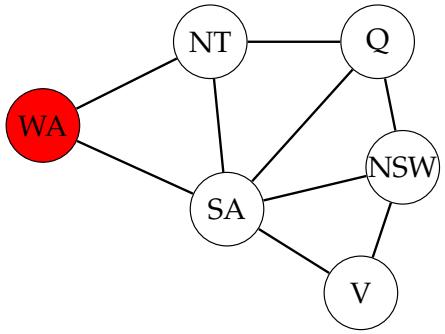
	WA	NT	Q	NSW	V	SA	T
Domaines initiaux	r v b	r v b	r v b	r v b	r v b	r v b	r v b



Forward checking

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

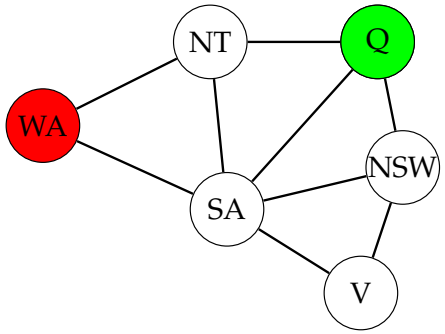
	WA	NT	Q	NSW	V	SA	T
Domaines initiaux	r v b	r v b	r v b	r v b	r v b	r v b	r v b
Affectation WA=r	r	v b	r v b	r v b	r v b	v b	r v b



Forward checking

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

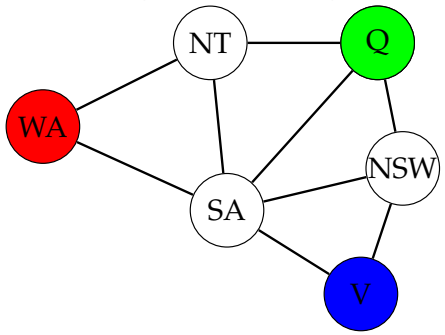
	WA	NT	Q	NSW	V	SA	T
Domaines initiaux	r v b	r v b	r v b	r v b	r v b	r v b	r v b
Affectation WA=r	r	v b	r v b	r v b	r v b	v b	r v b
Affectation Q=v	r	b	v	r b	r v b	b	r v b



Forward checking

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

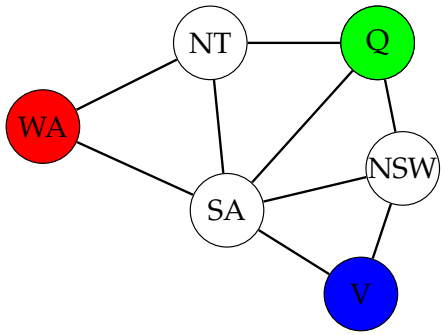
	WA	NT	Q	NSW	V	SA	T
Domaines initiaux	r v b	r v b	r v b	r v b	r v b	r v b	r v b
Affectation WA=r	r	v b	r v b	r v b	r v b	v b	r v b
Affectation Q=v	r	b	v	r b	r v b	b	r v b
Affectation V=b	r	b	v	r	b		r v b



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

Backtrack Intelligent

Lorsqu'on backtrack, on peut revenir à la décision la plus récente (comme dans DFS)

⇨ on peut revenir à une décision qui peut résoudre l'échec qui vient de se produire !

- on conserve un ensemble de conflits.
- on revient sur la décision la plus récente dans l'ensemble des conflits.

ex :

- On utilise l'ordre fixé de visite : `Q`, `NSW`, `V`, `T`, `SA`, `WA`, `NT`

Backtrack Intelligent

Lorsqu'on backtrack, on peut revenir à la décision la plus récente (comme dans DFS)
⇨ on peut revenir à une décision qui peut résoudre l'échec qui vient de se produire !

- on conserve un ensemble de conflits.
- on revient sur la décision la plus récente dans l'ensemble des conflits.

ex :

- On utilise l'ordre fixé de visite : `Q`, `NSW`, `V`, `T`, `SA`, `WA`, `NT`
- On a l'allocation partielle suivante
  `{Q=rouge, NSW=vert, V=bleu, T=rouge}`

Backtrack Intelligent

Lorsqu'on backtrack, on peut revenir à la décision la plus récente (comme dans DFS)
⇨ on peut revenir à une décision qui peut résoudre l'échec qui vient de se produire !

- on conserve un ensemble de conflits.
- on revient sur la décision la plus récente dans l'ensemble des conflits.

ex :

- On utilise l'ordre fixé de visite : `Q`, `NSW`, `V`, `T`, `SA`, `WA`, `NT`
- On a l'allocation partielle suivante
  `{Q=rouge, NSW=vert, V=bleu, T=rouge}`
- On a un problème avec `SA`
  L'ensemble des conflits pour `SA` est `{Q=rouge, NSW=vert, V=bleu}`

Backtrack Intelligent

Lorsqu'on backtrack, on peut revenir à la décision la plus récente (comme dans DFS)
⇨ on peut revenir à une décision qui peut résoudre l'échec qui vient de se produire !

- on conserve un ensemble de conflits.
- on revient sur la décision la plus récente dans l'ensemble des conflits.

ex :

- On utilise l'ordre fixé de visite : `Q`, `NSW`, `V`, `T`, `SA`, `WA`, `NT`
- On a l'allocation partielle suivante
  `{Q=rouge, NSW=vert, V=bleu, T=rouge}`
- On a un problème avec `SA`
  L'ensemble des conflits pour `SA` est `{Q=rouge, NSW=vert, V=bleu}`
- `V` est la dernière variable affectée, on va donc choisir une autre valeur que bleu

Backtrack Intelligent

Lorsqu'on backtrack, on peut revenir à la décision la plus récente (comme dans DFS)
⇨ on peut revenir à une décision qui peut résoudre l'échec qui vient de se produire !

- on conserve un ensemble de conflits.
- on revient sur la décision la plus récente dans l'ensemble des conflits.

ex :

- On utilise l'ordre fixé de visite : `Q`, `NSW`, `V`, `T`, `SA`, `WA`, `NT`
- On a l'allocation partielle suivante
  `{Q=rouge, NSW=vert, V=bleu, T=rouge}`
- On a un problème avec `SA`
  L'ensemble des conflits pour `SA` est `{Q=rouge, NSW=vert, V=bleu}`
- `V` est la dernière variable affectée, on va donc choisir une autre valeur que bleu

L'ensemble des conflits peut être géré pendant l'algorithme de backtrack

Backtrack Intelligent

En fait, ce backtrack intelligent est redondant avec le forward checking ! !
⇨ forward checking éviterait de visiter un état qui n'aurait plus de valeurs possibles !

ex : $\texttt{WA} = \texttt{rouge}, \texttt{NSW} = \texttt{rouge}$

Backtrack Intelligent

En fait, ce backtrack intelligent est redondant avec le forward checking ! !
⇨ forward checking éviterait de visiter un état qui n'aurait plus de valeurs possibles !

ex : `WA` = `rouge`,`NSW` = `rouge`

- On sait que cette affectation partielle est vouée à l'échec.

Backtrack Intelligent

En fait, ce backtrack intelligent est redondant avec le forward checking ! !
⇨ forward checking éviterait de visiter un état qui n'aurait plus de valeurs possibles !

ex : `WA` = `rouge`,`NSW` = `rouge`

- On sait que cette affectation partielle est vouée à l'échec.
- On suppose qu'on tente `T=rouge` et ensuite on tente d'affecter `NT`, `Q`, `V` et `SA`.

Backtrack Intelligent

En fait, ce backtrack intelligent est redondant avec le forward checking ! !
⇨ forward checking éviterait de visiter un état qui n'aurait plus de valeurs possibles !

ex : `WA` = `rouge`,`NSW` = `rouge`

- On sait que cette affectation partielle est vouée à l'échec.
- On suppose qu'on tente `T`=`rouge` et ensuite on tente d'affecter `NT`, `Q`, `V` et `SA`.
- On va essayer toutes les couleurs pour `NT` et on va échouer. Où devons-nous revenir ? `WA`, `NSW`, `T` ? Pourtant `NT` a des valeurs cohérentes avec `WA` et `NSW` !

Backtrack Intelligent

En fait, ce backtrack intelligent est redondant avec le forward checking ! !
⇨ forward checking éviterait de visiter un état qui n'aurait plus de valeurs possibles !

ex : `WA` = `rouge`,`NSW` = `rouge`

- On sait que cette affectation partielle est vouée à l'échec.
- On suppose qu'on tente `T=rouge` et ensuite on tente d'affecter `NT`, `Q`, `V` et `SA`.
- On va essayer toutes les couleurs pour `NT` et on va échouer. Où devons-nous revenir ? `WA`, `NSW`, `T` ? Pourtant `NT` a des valeurs cohérentes avec `WA` et `NSW` !
- `NT` ainsi que les variables suivantes ont *ensemble* un conflit. On va donc combiner des conflits

Backtrack Intelligent

En fait, ce backtrack intelligent est redondant avec le forward checking ! !
⇨ forward checking éviterait de visiter un état qui n'aurait plus de valeurs possibles !

ex : `WA` = `rouge`,`NSW` = `rouge`

- On sait que cette affectation partielle est vouée à l'échec.
- On suppose qu'on tente `T=rouge` et ensuite on tente d'affecter `NT`, `Q`, `V` et `SA`.
- On va essayer toutes les couleurs pour `NT` et on va échouer. Où devons-nous revenir ? `WA`, `NSW`, `T` ? Pourtant `NT` a des valeurs cohérentes avec `WA` et `NSW` !
- `NT` ainsi que les variables suivantes ont *ensemble* un conflit. On va donc combiner des conflits
- Soit X_j la variable courante et $\texttt{conf}(X_j)$ son ensemble de conflits. Si toutes les valeurs de X_j échouent on revient vers la dernière variable X_i affectée dans $\texttt{conf}(X_j)$ et

$$\texttt{conf}(X_i) \leftarrow \texttt{conf}(X_i) \cup \texttt{conf}(X_j) \setminus \{X_i\}$$

Apprentissage de contraintes

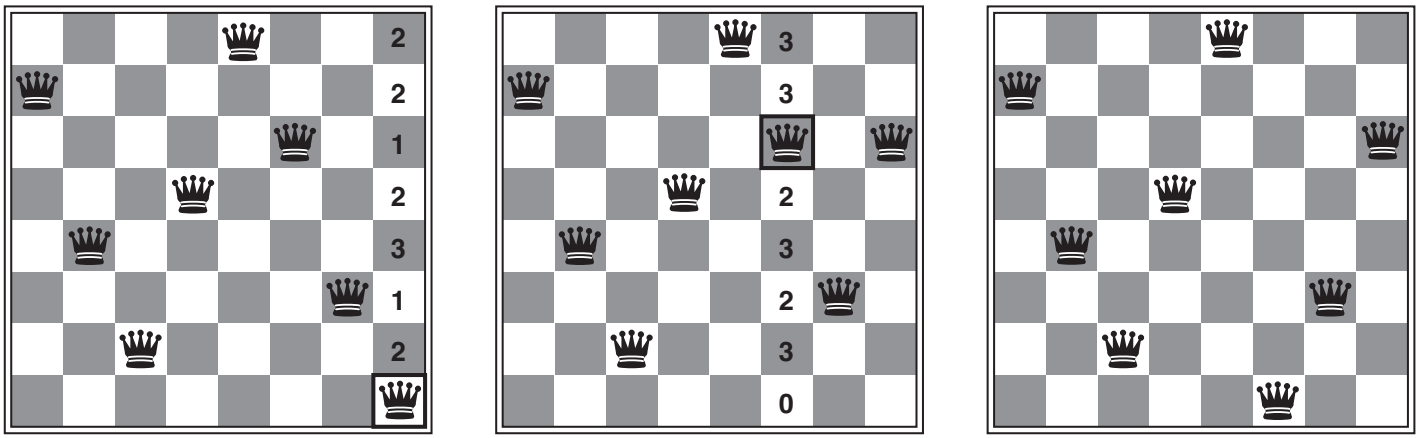
Lorsqu'on arrive dans une impasse, un sous ensemble de l'ensemble des conflits est responsable de cette impasse.
⇨on peut se rappeler de l'ensemble de conflits afin de trouver l'ensemble minimum de variables qui cause le problème.

apprendre des "no good" pour éviter de répéter ces états dans la suite de la recherche.

Un approche différente : la recherche locale

Au lieu de construire une solution en affectant une à une les variables, on peut partir d'une affectation complète (qui n'est pas cohérente) et essayer de modifier l'affectation.

- Quelle variable corriger ? Elle peut être choisie au hasard
- Quelle valeur choisir ? *heuristique du minimum de conflit* : choisir la valeur qui a le minimum de conflits avec les autres variables.



Pour de nombreux CSP, cette stratégie est efficace. Elle a notamment permis de réduire le calcul du planning du telescope Hubble de 3 semaines à 10 minutes !

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.