Problèmes de satisfaction de contraintes

M1 Miage 2016–2017 *Intelligence Artificielle*

Stéphane Airiau

LAMSADE

DAUPHINE
UNIVERSITÉ PARIS

Effectuer la recherche : Backtracking

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 variable

A cause du formalisme d'un CSP, on peut utiliser l'algorithme tel que pour tout CSP.

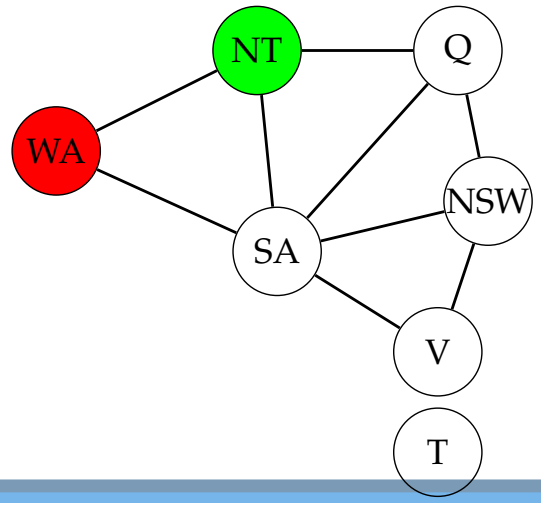
Comment choisir l'ordre de l'examen des variables et des valeurs ?

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



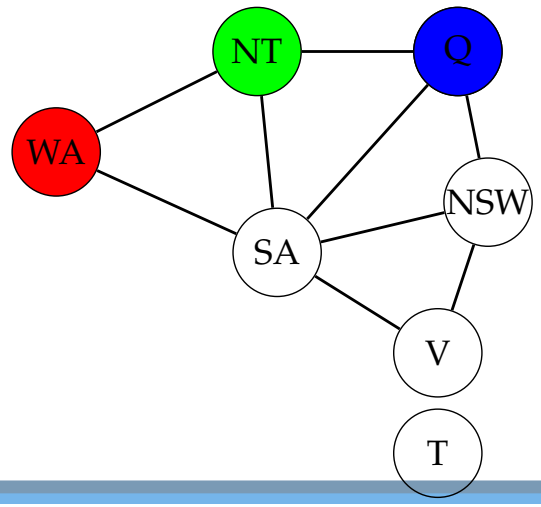
Comment choisir l'ordre de l'examen des variables et des valeurs ?

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



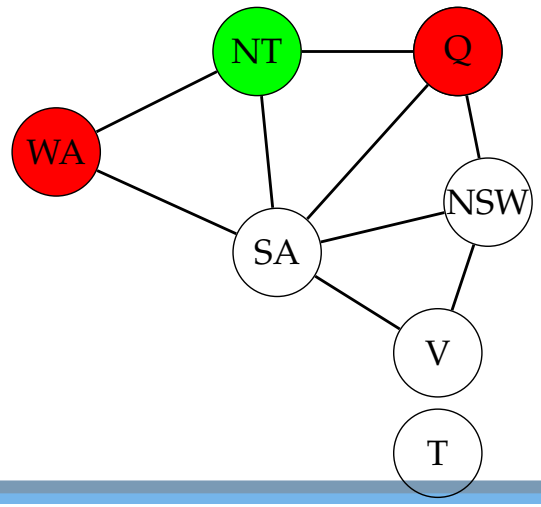
Comment choisir l'ordre de l'examen des variables et des valeurs ?

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 ?

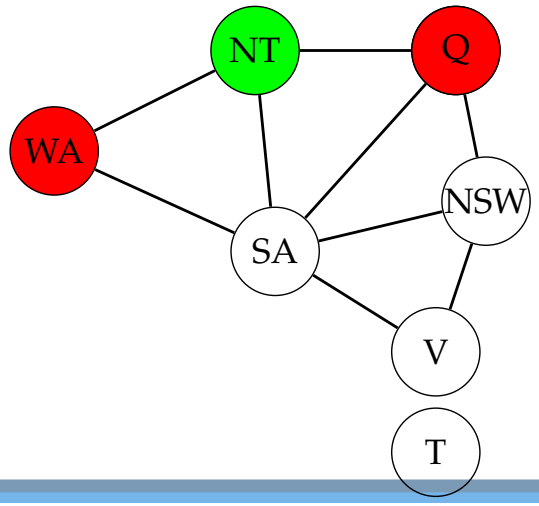
On peut utiliser des heuristiques "générales"
Pour les *variables*

- 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`.



Différents niveaux de propagation

- noeud cohérence : une variable est *"noeud cohérente"* si elle satisfait toutes les contraintes *unaires*.
  il suffit de boucler sur chaque variable pour garantir cette cohérence.
- **arc cohérence** : une variable est *arc cohérente* si toutes les valeurs associées à la variable satisfont toutes les contraintes *binaires*.
  ex :
  - 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 que X soit arc cohérent, réduction à $D_X = \{0,1,2,3\}$
    pour que Y soit arc cohérent, réduction à $D_Y = \{0,1,4,9\}$
- le CSP est arc cohérent si toutes les variables sont arc cohérentes

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 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

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 valuer 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 $\{2,3,4,5,6\}$

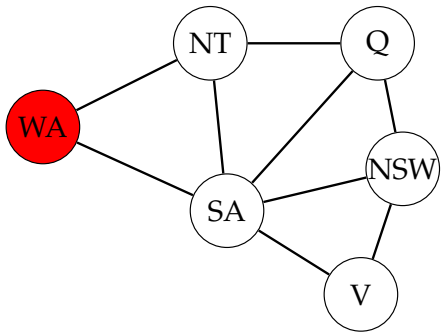
  on peut enlever 5 et 6 de chaque domaine !
- contraintes avec des bornes
  ex : si $A+B=420$ et $D_A=[0,165]$, $D_B=[0,385]$

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

Forward checking

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

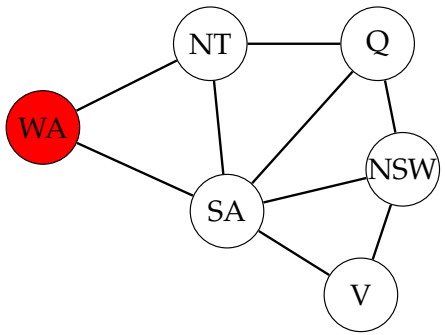
	WA	NT	Q	NSW	V	SA	T
Domaines initiaux	r v b	r v b	r v b	r v b	r v b	r v b	r v b



Forward checking

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

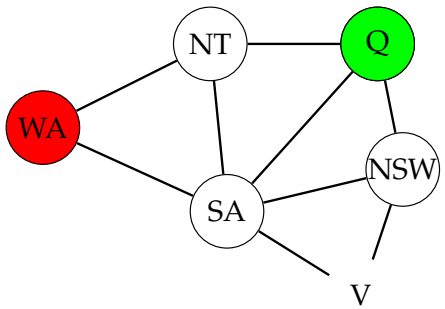
	WA	NT	Q	NSW	V	SA	T
Domaines initiaux	r v b	r v b	r v b	r v b	r v b	r v b	r v b
Affectation WA=r	r	v b	r v b	r v b	r v b	v b	r v b



Forward checking

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

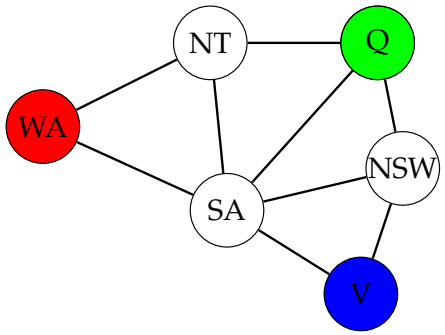
	WA	NT	Q	NSW	V	SA	T
Domaines initiaux	r v b	r v b	r v b	r v b	r v b	r v b	r v b
Affectation WA=r	r	v b	r v b	r v b	r v b	v b	r v b
Affectation Q=v	r	b	v	r b	r v b	b	r v b



Forward checking

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

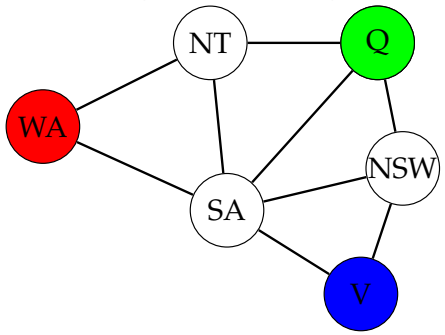
	WA	NT	Q	NSW	V	SA	T
Domaines initiaux	r v b	r v b	r v b	r v b	r v b	r v b	r v b
Affectation WA=r	r	v b	r v b	r v b	r v b	v b	r v b
Affectation Q=v	r	b	v	r b	r v b	b	r v b
Affectation V=b	r	b	v	r	b		r v b



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 inconsistance ne sont pas détectée par le 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

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`

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

- 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

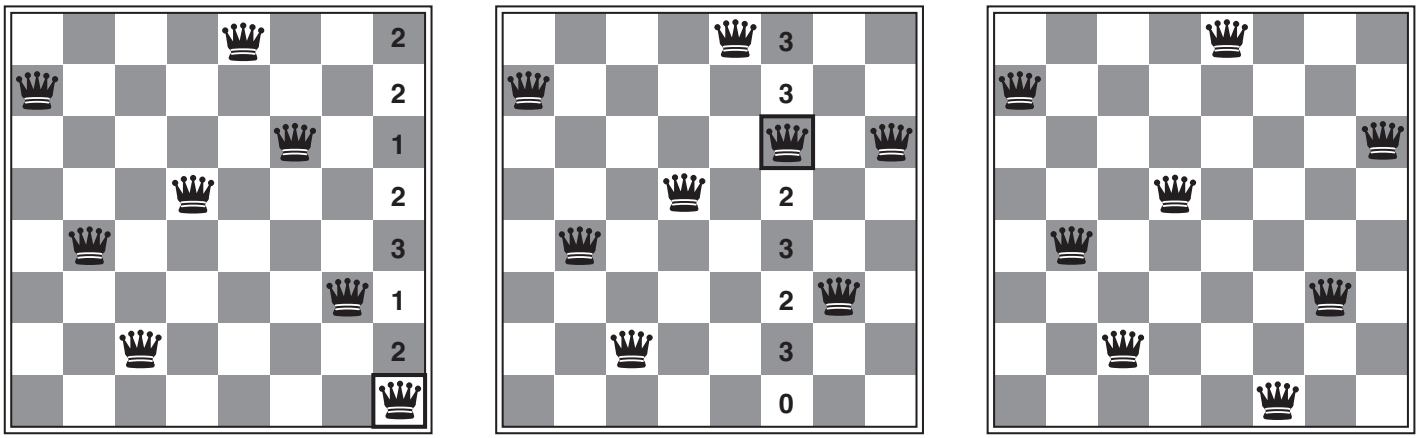
Lorsqu'on arrive dans une impasse, un sous ensemble de l'ensemble des conflits est responsable de cette impasse.
⇨on peut se rappeler de l'ensemble de conflits afin de trouver l'ensemble minimum de variables qui cause le problème.

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.
- la recherche locale avec l'heuristique de conflits minimum est une autre approche efficace
- d'autres techniques utilisant la décomposition peuvent aussi être efficaces.