

CHAPITRE 3

PROBLÈMES À SATISFACTION DE CONTRAINTES

©Haythem Ghazouani



DESCRIPTION DES CSP

4

PROBLÈME DE SATISFACTION DE CONTRAINTES (CSP): RÉOLUTION

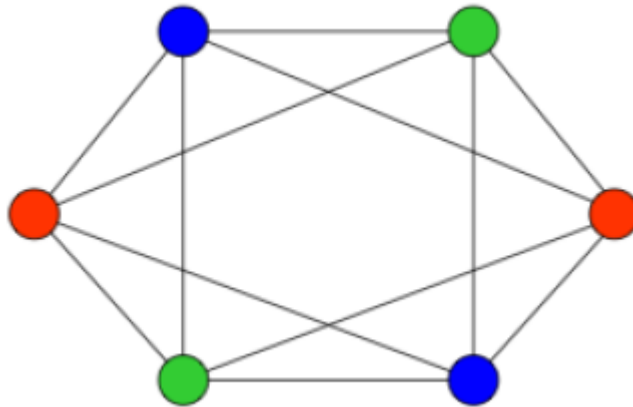
- En raisonnement par contraintes, un modèle est construit en utilisant
 - des **variables**
 - des **domaines** de variables.
 - des **contraintes** entre les variables
- Un tel modèle est appelé **Problème de Satisfaction de contraintes (CSP)**
- Une solution à un CSP est: affecter à chaque variable une valeur de son domaine de manière à satisfaire toutes les contraintes.

POURQUOI EN CSP ?

- La résolution de problèmes de satisfaction de contraintes peut être vue comme un cas particulier de la recherche heuristique
- La structure interne des états (nœuds) a une représentation particulière
 - un état est un ensemble de **variables** avec des **valeurs** correspondantes
 - les transitions entre les états tiennent compte de **contraintes** sur les valeurs possibles des variables
- Sachant cela, on va pouvoir utiliser des **heuristiques générales**, plutôt que des heuristiques spécifiques à une application
- En traduisant un problème sous forme de satisfaction de contraintes, on élimine la difficulté de définir l'heuristique $h(n)$ pour notre application

EXEMPLE 1 : PROBLÈME DE COLORIAGE DE GRAPHE

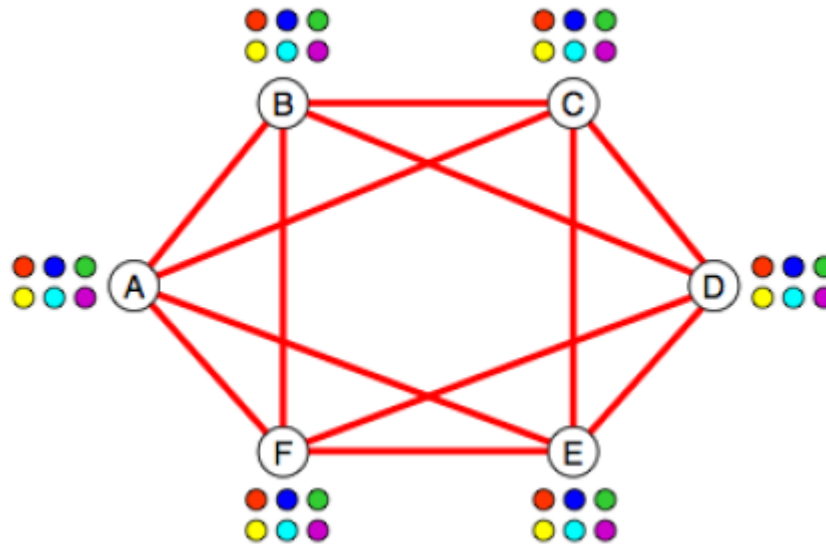
- Coloriage des nœuds d'un graphe:
 - Quel est le nombre **minimal** de couleurs tel que deux nœuds adjacents reçoivent des couleurs différentes.



EXEMPLE 1 : PROBLÈME DE COLORIAGE DE GRAPHE

○ Modèle complet :

- Variables : A, B, C, D, E, F.
- Domaines: {rouge, bleu, vert, jaune, bleu-ciel, violet}
- Contraintes : $A \neq B$, $A \neq C$, $A \neq E$, $A \neq F$, $B \neq F$, $B \neq D$, $B \neq C$, $F \neq D$, $F \neq E$, $C \neq E$, $C \neq D$, $E \neq D$.



DÉFINITION D'UN CSP

- Formellement, *un problème de satisfaction de contraintes* (ou **CSP** pour *Constraint Satisfaction Problem*) est défini par:
 - Un ensemble fini de *variables* X_1, \dots, X_n .
 - Chaque variable X_i a un *domaine* D_i de *valeurs* permises.
 - Un ensemble fini de *contraintes* C_1, \dots, C_m sur les variables.
 - Une contrainte restreint les valeurs pour un sous-ensemble de variables.

DÉFINITION D'UN CSP

- Un *état d'un problème CSP* est défini par une *assignation* de valeurs à certaines variables ou à toutes les variables.
 - $\{X_i=v_i, X_n=v_1, \dots\}$.
- Une assignation qui ne viole aucune contrainte est dite *consistante* ou *légale*.
- Une *assignation* est *complète* si elle concerne toutes les variables.
- Une *solution* à un problème CSP est une *assignation complète* et *consistante*.
- Parfois, la solution doit en plus *maximiser une fonction objectif* donnée.

EXEMPLE 2 : COLORIER UNE CARTE

- On vous donne une carte de l'Australie :



- Et on vous demande d'utiliser seulement trois couleurs (*rouge*, *vert* et *bleu*) de sorte que deux états frontaliers n'aient jamais les mêmes couleurs.
- On peut facilement trouver une solution à ce problème en le formulant comme un problème CSP et en utilisant des algorithmes généraux pour CSP.

EXEMPLE 2: COLORIER UNE CARTE

- Formulation du problème CSP :

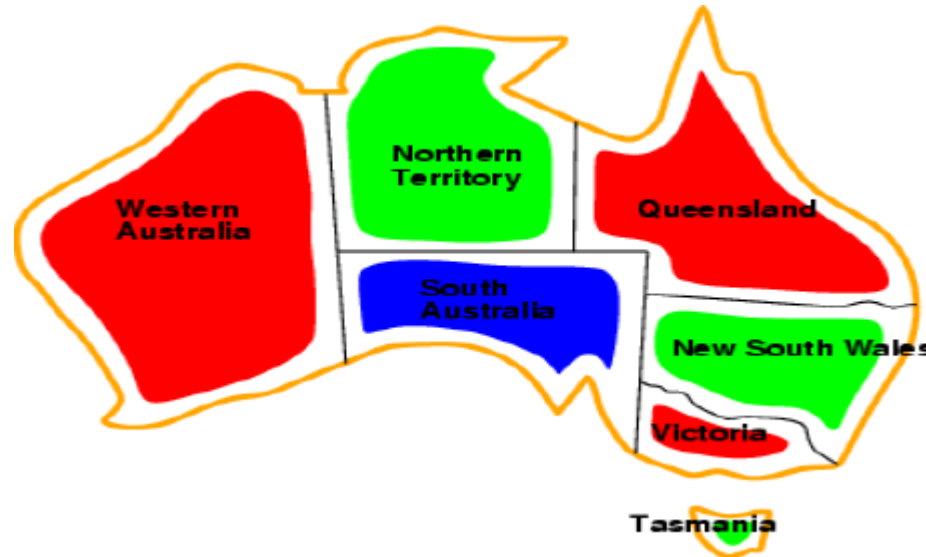


- Les variables sont les états : $V = \{WA, NT, Q, NSW, V, SA, T\}$
- Le domaine de chaque variable est l'ensemble des 3 couleurs : $\{R, G, B\}$



- Contraintes : *Les régions frontalières doivent avoir des couleurs différentes*
 - $WA \neq NT, \dots, NT \neq Q, \dots$

EXEMPLE 2: COLORIER UNE CARTE

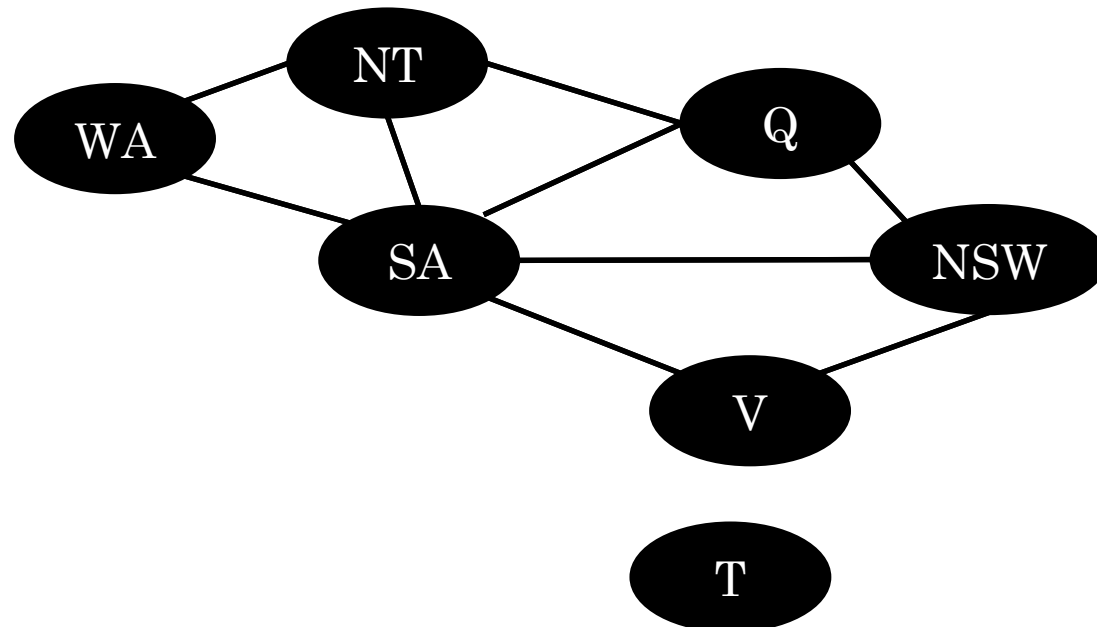


- La solution est complète et consistante

$$\{ WA = R, NT = G, Q = R, NSW = G, V = R, SA = B, T = G \}$$

GRAPHES DE CONTRAINTES

- Pour des problèmes avec des contraintes binaires (c-à-d., entre deux variables), on peut visualiser le problème CSP par un graphe de contraintes.
- Dans un graphe de contraintes: les nœuds sont les variables, et les arcs sont les contraintes



INTÉRÊTS DES CSP

- La fonction successeur et le test de l'état final sont génériques
- Les heuristiques sont génériques
- Le graphe des contraintes peut simplifier le processus de recherche.

FORMULATION DES CSP's

○ Formulation incrémentale

- Etat initial: aucune variable affectée
- Fonction successeur: affecter une valeur à n'importe quelle variable non encore affectée, à condition que celle là ne génère pas de conflit.
- Test de l'état final: affectation courante complète
- Coût du chemin: un coût constant à toutes les étapes.

○ Formulation par états complets: le chemin menant à la solution n'a pas d'importance.

EXEMPLE 3: LE PROBLÈME DES 4 REINES

- Placer 4 reines sur un échiquier de 4 lignes et 4 colonnes de façon à ce qu'aucune reine ne soit attaquée
- 2 reines sont attaquées si elles se trouvent sur une même diagonale, une même ligne ou une même colonne de l'échiquier



CSP (4 REINES)

- Variables ? (les inconnues du problème)
- Domaine ? (les valeurs possibles pour les variables)
- Contraintes ?



CSP (4 REINES): MODÉLISATION

- **Variables:** associer chaque reine i à deux variables L_i (ligne) et C_i (Colonne)

- **Domaine:** $\{1,2,3,4\}$

$$D(L1) = D(L2) = D(L3) = D(L4) = D(C1) = D(C2) = D(C3) = D(C4) = \{1,2,3,4\}$$

CSP (4 REINES): MODÉLISATION

- Les reines doivent être sur des lignes différentes.

$$\text{Clig} = \{L1 \neq L2, L1 \neq L3, L1 \neq L4, L2 \neq L3, L2 \neq L4, L3 \neq L4\}$$

- Les reines doivent être sur des colonnes différentes.

$$\text{Ccol} = \{C1 \neq C2, C1 \neq C3, C1 \neq C4, C2 \neq C3, C2 \neq C4, C3 \neq C4\}$$

- Les reines doivent être sur des diagonales montantes différentes.

$$\text{Cdm} = \{C1+L1 \neq C2+L2, C1+L1 \neq C3+L3, C1+L1 \neq C4+L4, C2+L2 \neq C3+L3, C2+L2 \neq C4+L4, C3+L3 \neq C4+L4\}$$

- Les reines doivent être sur des diagonales descendantes différentes.

$$\text{Cdd} = \{C1-L1 \neq C2-L2, C1-L1 \neq C3-L3, C1-L1 \neq C4-L4, C2-L2 \neq C3-L3, C2-L2 \neq C4-L4, C3-L3 \neq C4-L4\}$$

CSP (4 REINES): MODÉLISATION

- L'ensemble de contraintes est défini par l'union de ces 4 ensembles

$$C = C_{\text{lig}} \cup C_{\text{col}} \cup C_{\text{dm}} \cup C_{\text{dd}}$$

TYPES DE PROBLÈMES CSP

- CSP avec des domaines finis (et discrets).
- CSP Booléens: les variables sont vraies ou fausses.
- CSP avec des domaines continus (et infinis)
 - Par exemple, problèmes d'ordonnancement avec des contraintes sur les durées.
- CSP avec des contraintes linéaires.
- CSP avec des contraintes non linéaires.
- ...
- Les problèmes CSP sont étudiées de manière approfondies en recherche opérationnelle.

TYPES DES CONTRAINTES

- La contrainte unaire qui restreint la valeur d'une seule variable,
 - Ex: $SA \neq \text{green}$
- La contrainte binaire qui porte sur deux variables
 - Ex: $SA \neq WA$
- Les contraintes d'ordre supérieur impliquent au minimum 3 variables
 - Ex: les énigmes de cryptarithmétique, les 4-reines, etc.

CSP (4-REINES) : TYPES DES CONTRAINTES

○ Contraintes binaires

- Les reines doivent être sur des lignes différentes.

$$C_{lig} = \{L1 \neq L2, L1 \neq L3, L1 \neq L4, L2 \neq L3, L2 \neq L4, L3 \neq L4\}$$

- Les reines doivent être sur des colonnes différentes.

$$C_{col} = \{C1 \neq C2, C1 \neq C3, C1 \neq C4, C2 \neq C3, C2 \neq C4, C3 \neq C4\}$$

○ Contraintes quaternaires

- $C_{dm} = \{C1+L1 \neq C2+L2, C1+L1 \neq C3+L3, C1+L1 \neq C4+L4, C2+L2 \neq C3+L3, C2+L2 \neq C4+L4, C3+L3 \neq C4+L4\}$

- $C_{dd} = \{C1-L1 \neq C2-L2, C1-L1 \neq C3-L3, C1-L1 \neq C4-L4, C2-L2 \neq C3-L3, C2-L2 \neq C4-L4, C3-L3 \neq C4-L4\}$



BACKTRACKING SEARCH

25

ALGORITHME DEPTH-FIRST-SEARCH NAÏVE POUR CSP : EXPLORATION AVEC BACKTRACKING

- On pourrait être tenté d'utiliser la recherche dans un graphe (Algorithme en profondeur d'abord) ou un *depth-first-search naïf* avec les paramètres suivants:
 - **Un état** est une assignation.
 - **État initial** : assignation vide { }
 - **Fonction successeur** : assigne une valeur à une variable non encore assignée, en respectant les contraintes.
 - **But** : Assignation complète et consistante.

En théorie, l'algorithme est général et s'applique à tous les problèmes CSP. La solution doit être complète apparaît à une profondeur n , si nous avons n variables.

- Cependant, ici le chemin à la solution est sans importance.
 - On peut travailler avec des états qui sont des assignations complètes (consistantes ou non).
 - On peut utiliser une méthode de recherche locale (*hill-climbing*, etc.)

LIMITATIONS DE L'APPROCHE PRÉCÉDENTE

- Supposons une recherche en largeur :
 - Le nombre de branches au premier niveau, dans l'arbre est de $n*d$, 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 générés, pour seulement d^n assignations complètes.
- L'algorithme ignore la commutativité des transitions. Par exemple:
 - 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 .
- *Depth-first-search* avec l'assignation d'une seule variable dans un nœud est appelé **backtracking-search** : C'est l'algorithme de base pour résoudre les problèmes CSP.

ILLUSTRATION DE BACKTRACKING-SEARCH

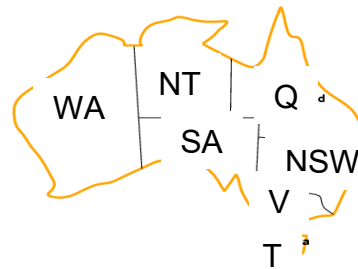


ILLUSTRATION DE BACKTRACKING-SEARCH

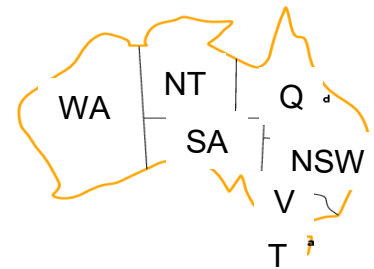
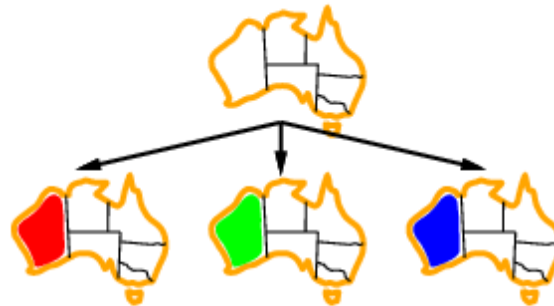


ILLUSTRATION DE BACKTRACKING-SEARCH

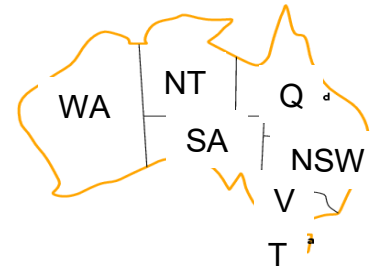
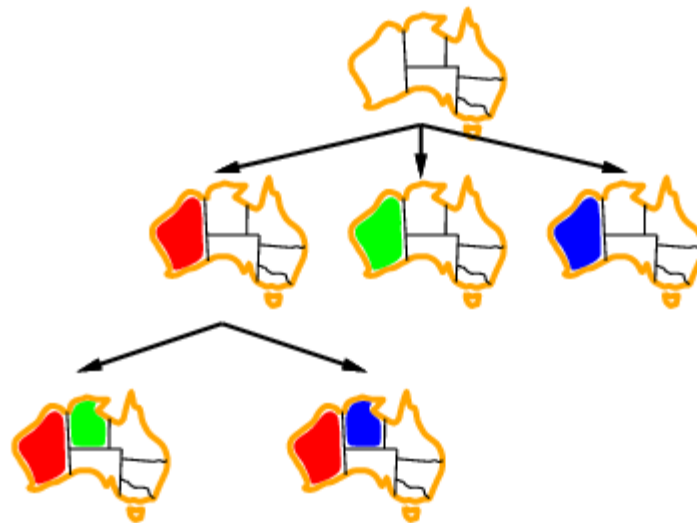
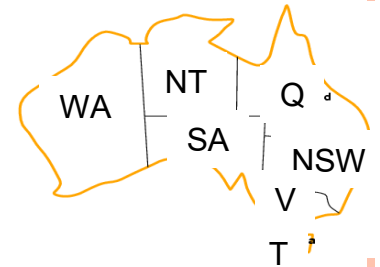


ILLUSTRATION DE BACKTRACKING-SEARCH



BACKTRACKING SEARCH

```
function BACKTRACKING-SEARCH(csp) return a solution or failure
    return BACKTRACK( $\{\}$  , csp)
function BACKTRACK(assignment, csp) return a solution or failure
    if assignment is complete then return assignment
    var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(var, assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment then
            add  $\{var=value\}$  to assignment
            inferences  $\leftarrow$  INFERENCES(csp, var, value) // e.g., AC-3
            if inferences  $\neq$  failure then
                add inferences to assignment
                result  $\leftarrow$  BACKTRACK (assignment, csp)
                if result  $\neq$  failure then return result
            remove  $\{var=value\}$  and inferences from assignment
    return failure
```

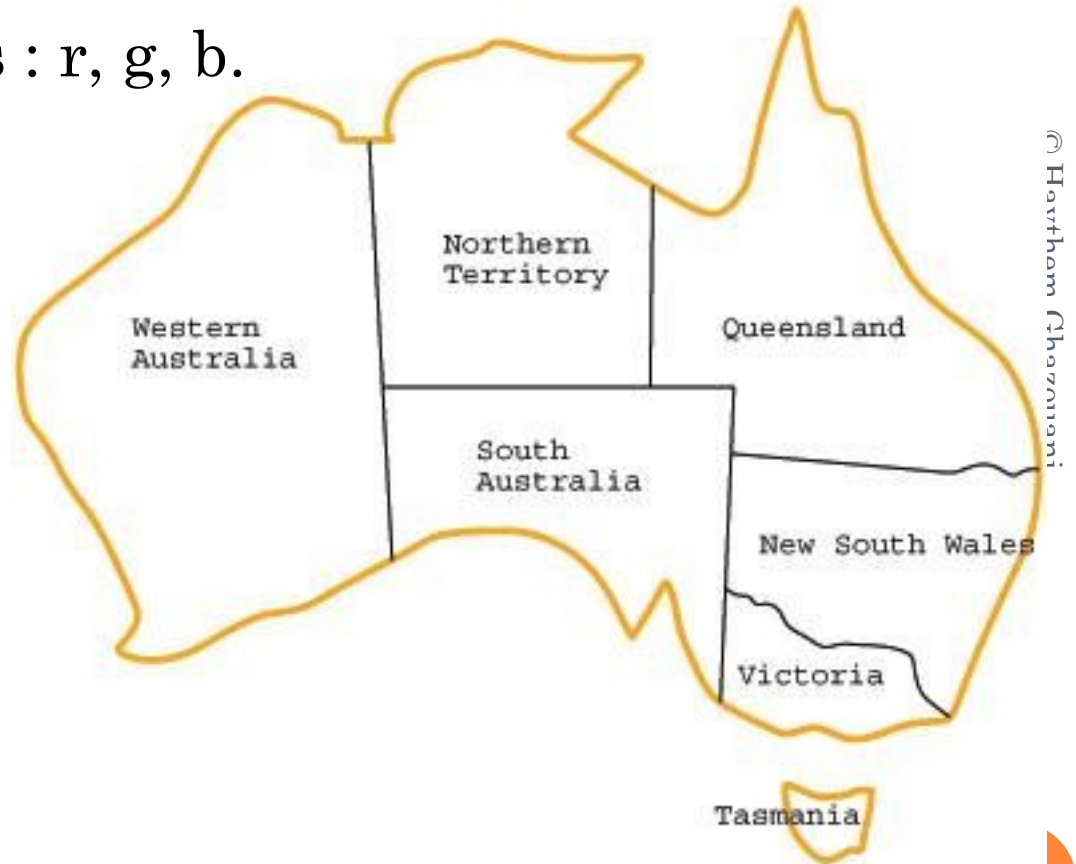
BACKTRACKING (BT) INTELLIGENT

- **BT** visite encore et encore les mêmes régions de l'arbre de recherche, car il a une vision très locale du problème
- Une façon de se débarrasser du problème utilise le **backtracking intelligent** algorithms
 - Backjumping (BJ), Conflict BJ, DB, Graph-based BJ, Learning
- **Backjumping (BJ)** est différent du BT dans ce qui suit:
 - Quand BJ atteint un blocage il ne fait pas un backtrack à la dernière variable immédiate. Il fait un backtrack à la variable la plus profonde dans l'arbre de recherche qui est en conflit avec la variable courante.

BJ vs. BT

- Revenons au problème de coloriage

Avec les trois couleurs : r, g, b.



BT vs. BT

- Considérons ce que fait le BT dans le problème de coloriage
 - Assumons que les variables sont assignées dans l'ordre : Q, NSW, V, T, SA, WA, NT
 - Assumons que nous avons atteint l'affectation partielle :
 $Q = red, NSW = green, V = blue, T = red$
 - En essayant d'affecter une valeur à SA , nous découvrons que toutes les valeurs du domaine ne sont plus permises.
 - Bolcage!
 - BT va effectuer un backtrack pour essayer une nouvelle valeur de la variable T !
 - Pas une bonne idée!

BT vs. BJ

- BJ a une approche plus intelligente pour le backtracking
 - Il nous dit de revenir à une des variables responsable du blocage.
 - L'ensemble de ces variables est appelé **conflict-set**
 - Le conflict-set de SA est $\{Q, NSW, V\}$
 - BJ effectue un saut en arrière vers la variable la plus profonde dans le conflict-set de la variable où le blocage est arrivée.
 - La plus profonde = la plus récemment visitée.
- CBJ, DB, Graph-based BJ, Learning, Backmarking



AMÉLIORATION DU BACKTRACKING SEARCH

1. Most constrained variable
2. Most constraining variable
3. Least constraining value
4. Forward checking
5. Propagation des contraintes
6. Cohérence de arcs

AMÉLIORATION DE BACKTRACKING-SEARCH

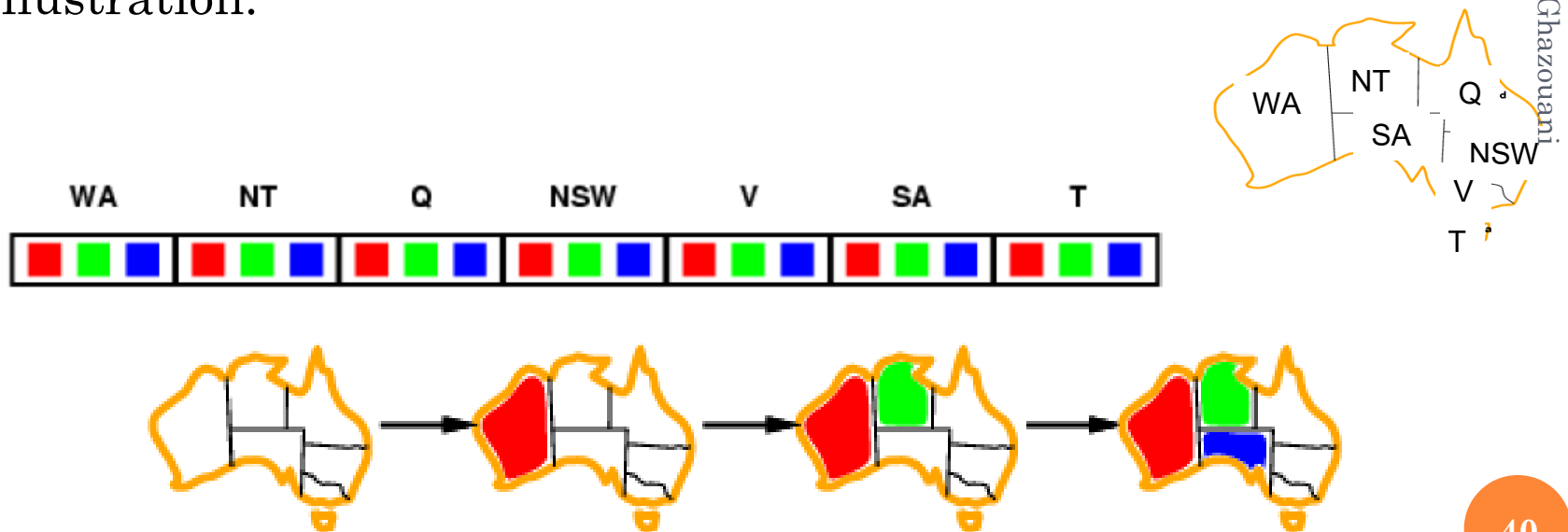
- Sans heuristiques, l'algorithme est limité.
- Des heuristiques générales peuvent améliorer l'algorithme significativement :
 - Choisir judicieusement la prochaine variable:
 - SELECT-UNASSIGNED-VARIABLE
 - Choisir judicieusement la prochaine valeur à assigner:
 - ORDER-DOMAIN-VALUES
 - Faire des inférences pour détecter plus tôt les assignations conflictuels:
 - INFERENCES

AMÉLIORATION DE BACKTRACKING-SEARCH

- *Des méthodes générales qui peuvent donner d'énormes gains de vitesse:*
 - Quelle variable doit être affectée à la prochaine étape?
 - Dans quel ordre ses valeurs devraient être choisies?
 - Peut on détecter les inévitables échecs plutôt?
- Heuristiques:
 1. Most constrained variable
 2. Most constraining variable
 3. Least constraining value
 4. Forward checking

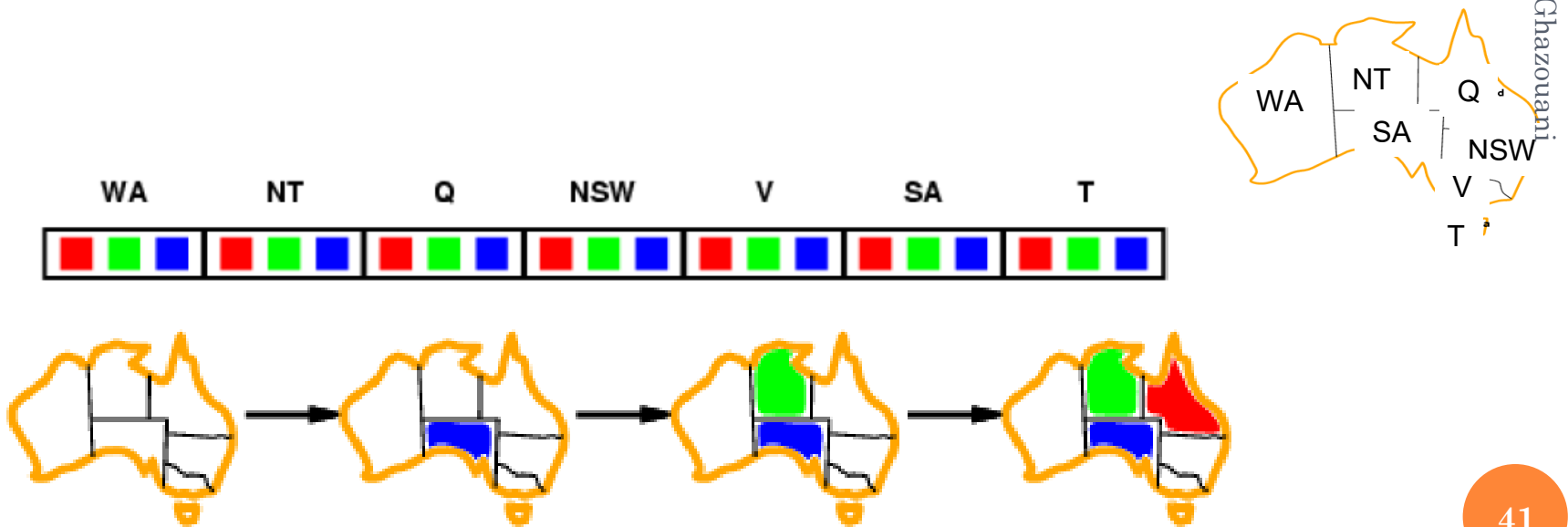
HEURISTIQUE 1: MOST CONSTRAINED VARIABLE

- À chaque étape, choisir la variable avec le moins de valeurs consistantes restantes.
 - C-à-d., la variable « posant le plus de restrictions ».
 - Appelé aussi *Minimum Remaining Value (MRV)*.
- Illustration:



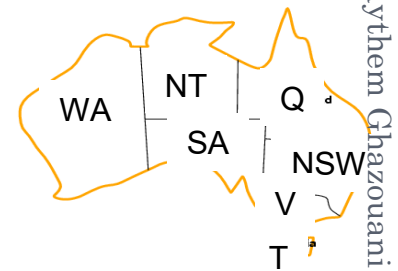
HEURISTIQUE 2: MOST CONSTRAINING VARIABLE

- À chaque étape, choisir la variable la plus contraignante.
 - C-à-d., la variable « avec le plus grand nombre de contraintes ».
 - Appelée aussi « **Degree heuristic** ».
- Illustration:



HEURISTIQUE 3: LEAST CONSTRAINING VALUE

- Pour une variable donnée, choisir une valeur qui invalide le moins de valeurs possibles pour les variables non encore assignées.

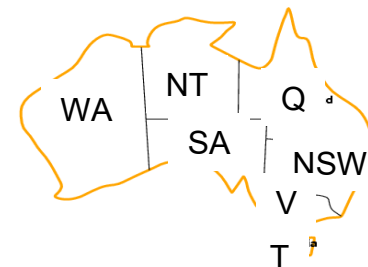


Laisse une seule
valeur pour SA

Ne laisse aucune
valeur pour SA

HEURISTIQUE 4 : *FORWARD CHECKING*

- L'idée de *forward-checking* (vérification avant) est :
 - Chaque fois qu'une variable est assignée, vérifier la cohérence de toutes les variables en contrainte avec elle.
 - Attention: si la valeur est assignée à X, on vérifie juste la cohérence des variables Y telle que il y a une contrainte impliquant X et Y. Par contre, si ce faisant, Y est modifié, on ne vérifiera pas l'impact de cette modification sur les contraintes impliquant Y!
- **Exemple:** Considérons le même CSP. Voici les domaines initiaux.



WA

NT

Q

NSW

V

SA

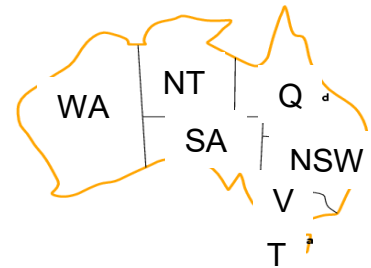
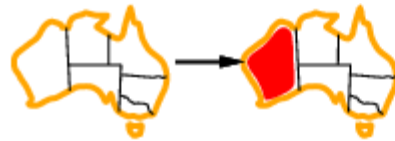
T

Domaines initiaux



FORWARD-CHECKING INFERENCE

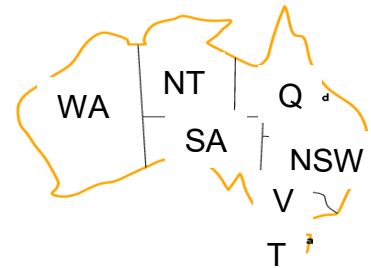
- Supposons que l'on choisisse au départ la variable WA (première étape de la récursivité de *backtracking-search*). Considérons l'assignation WA=Rouge. On voit ici le résultat de *forward-checking*.



	WA	NT	Q	NSW	V	SA	T
Domaines initiaux	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>
Après WA=Red	<div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>

ALGORITHME *FORWARD CHECKING*

- Supposons maintenant que l'on choisisse la variable Q à la prochaine étape de la récursivité de *backtracking-search*. Considérons l'assignation Q=Vert. On voit ici le résultat de *forward-checking*.



© Haythem Ghazouani

Domaines initiaux

WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

Après WA=Red

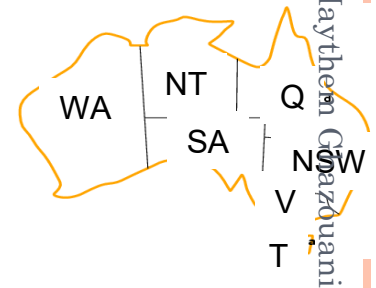
WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

Après Q=Green

WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

ALGORITHME *FORWARD CHECKING*

- Supposons maintenant que l'on choisisse la variable V à la prochaine étape de la récursivité de *backtracking-search*. Considérons l'assignation V=Bleu. On voit ici le résultat de *forward-checking*.

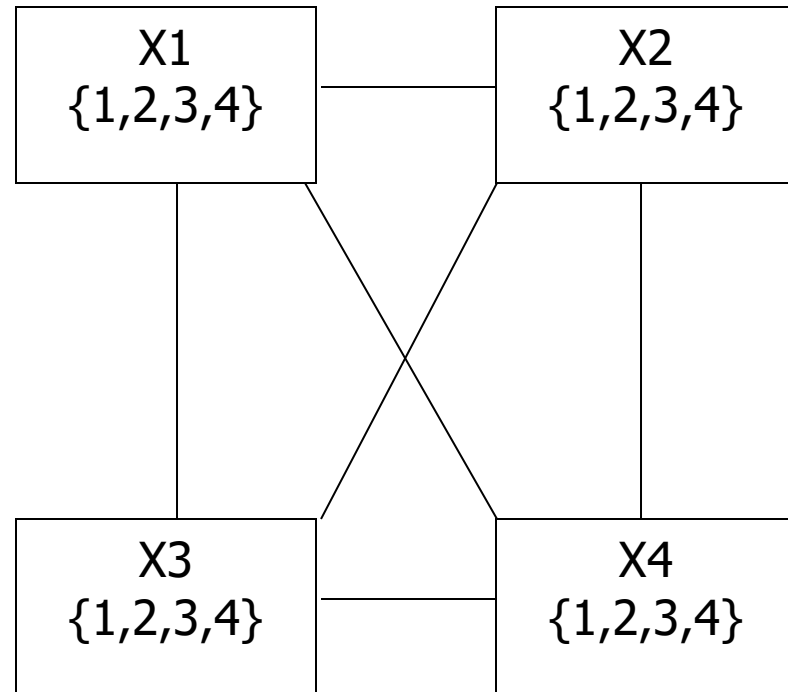


© Haythem G. Azouani








	WA	NT	Q	NSW	V	SA	T
Domaines initiaux	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>
Après WA=Red	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>
Après Q=Green	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>
Après V=Blue	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>

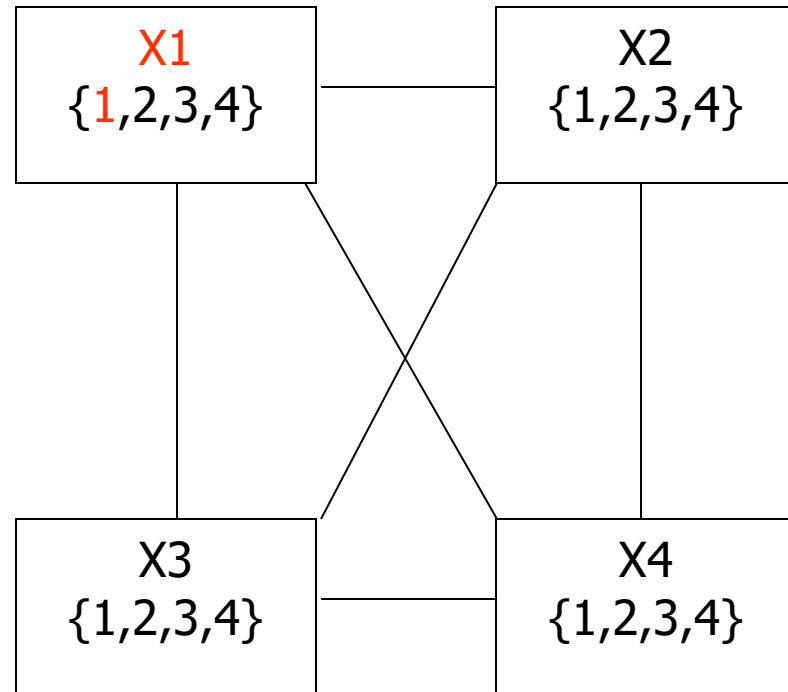
EXEMPLE : PROBLÈME DES 4-REINES

	1	2	3	4
1				
2				
3				
4				










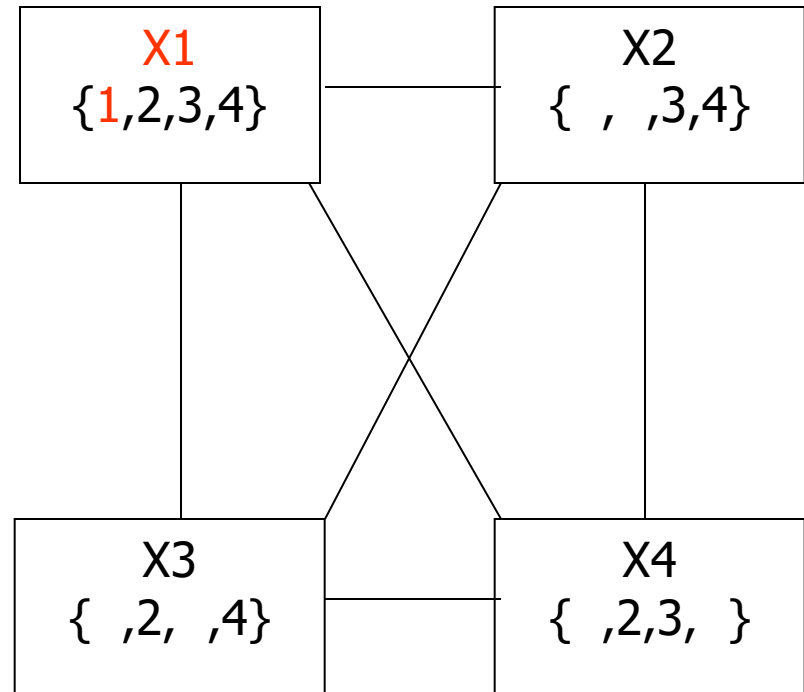
EXEMPLE : PROBLÈME DES 4-REINES

	1	2	3	4
1				
2				
3				
4				



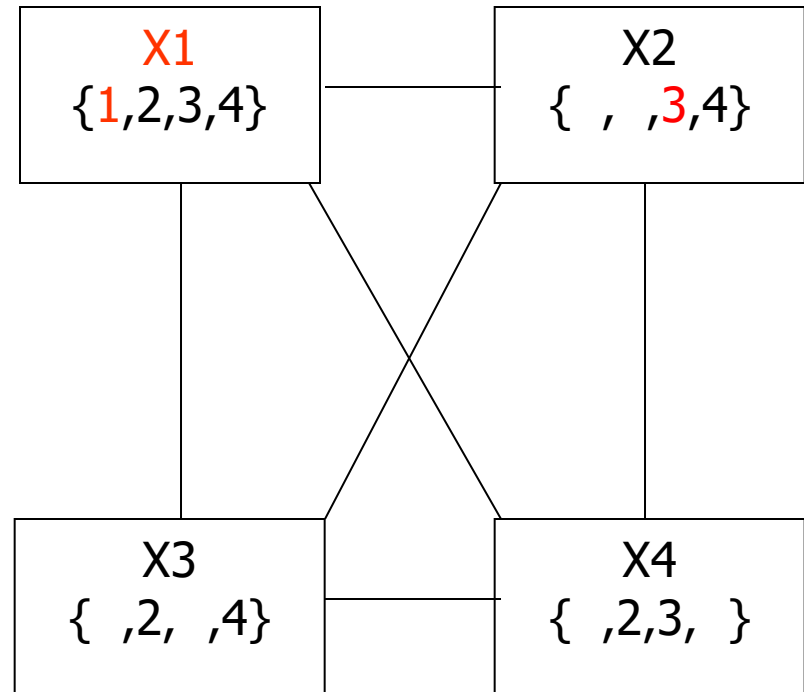
EXEMPLE : PROBLÈME DES 4-REINES

	1	2	3	4
1				
2				
3				
4				



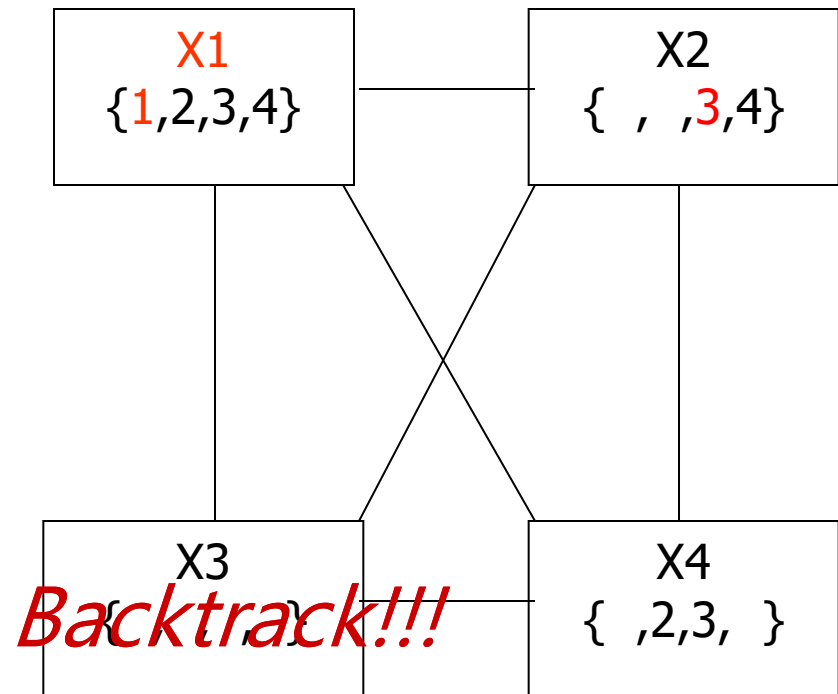
EXEMPLE : PROBLÈME DES 4-REINES

	1	2	3	4
1	★	●	●	●
2	●	●	●	
3		★	●	●
4	●		●	●



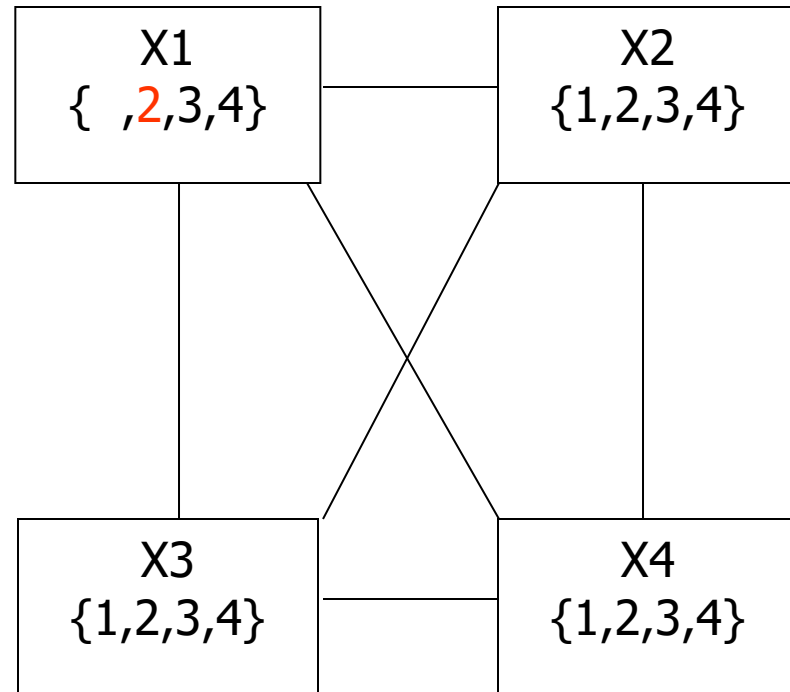
EXEMPLE : PROBLÈME DES 4-REINES

	1	2	3	4
1	★	●	●	●
2	●	●	●	
3		★	●	●
4	●		●	●



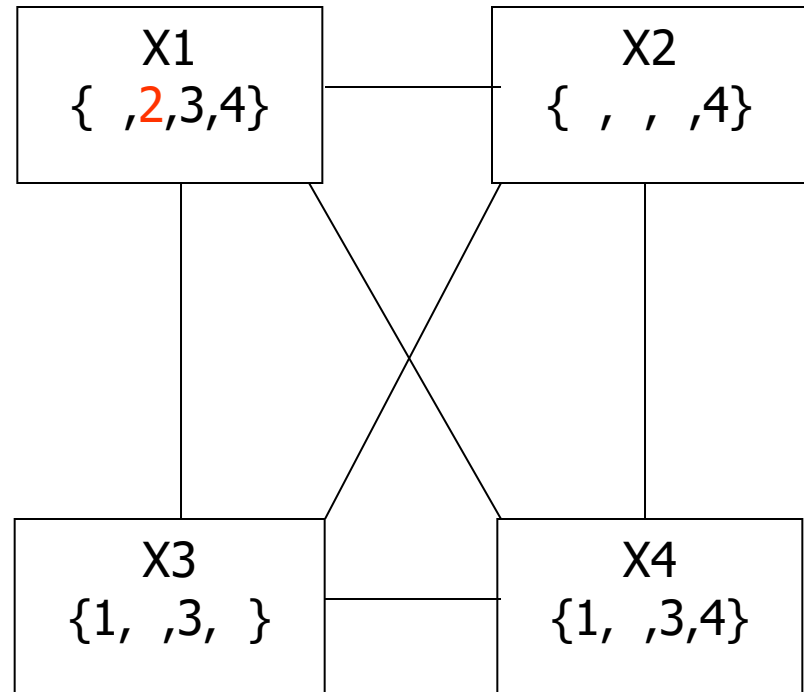
EXEMPLE : PROBLÈME DES 4-REINES

	1	2	3	4
1		●		
2	★	●	●	●
3		●		
4			●	



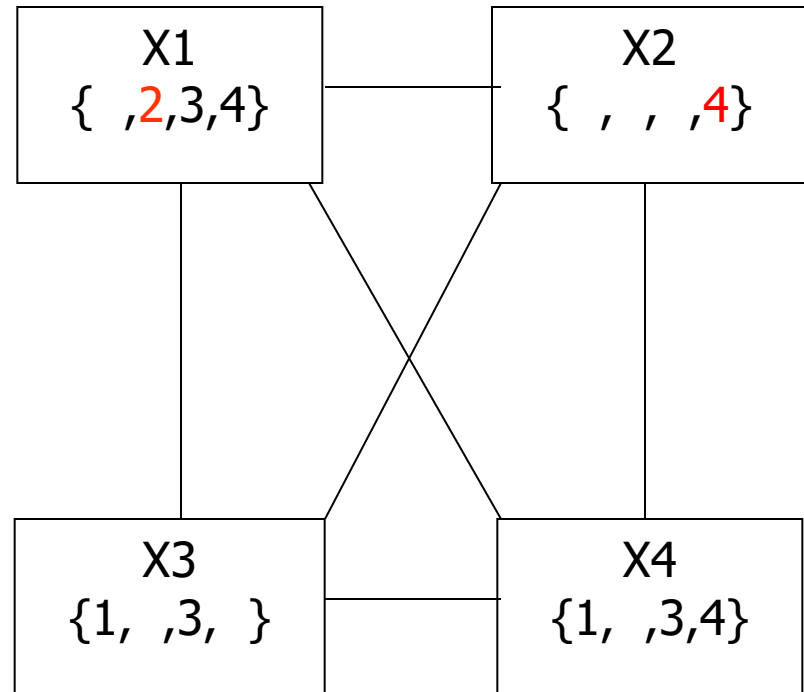
EXEMPLE : PROBLÈME DES 4-REINES

	1	2	3	4
1		●		
2	★	●	●	●
3		●		
4			●	



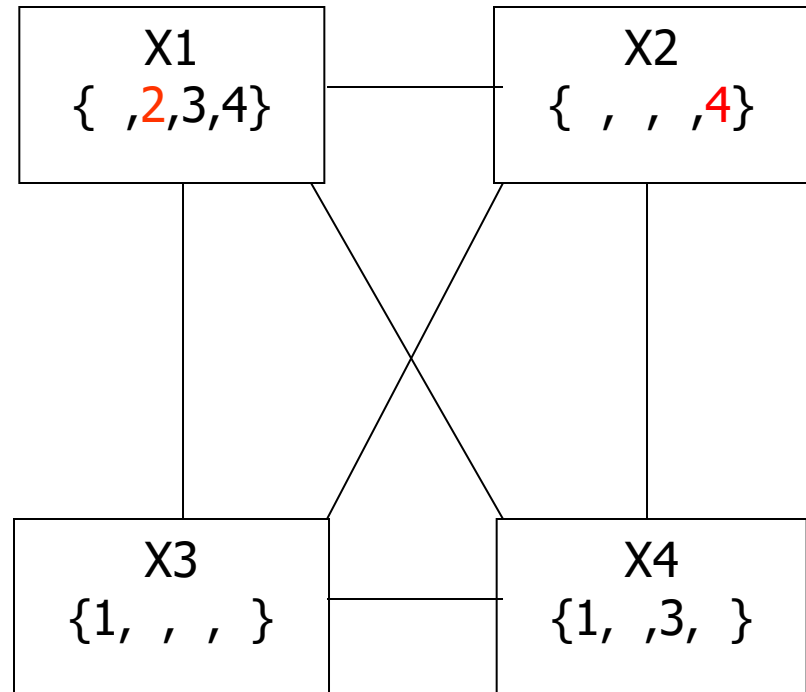
EXEMPLE : PROBLÈME DES 4-REINES

	1	2	3	4
1		●		
2	★	●	●	●
3		●	●	
4		★	●	●



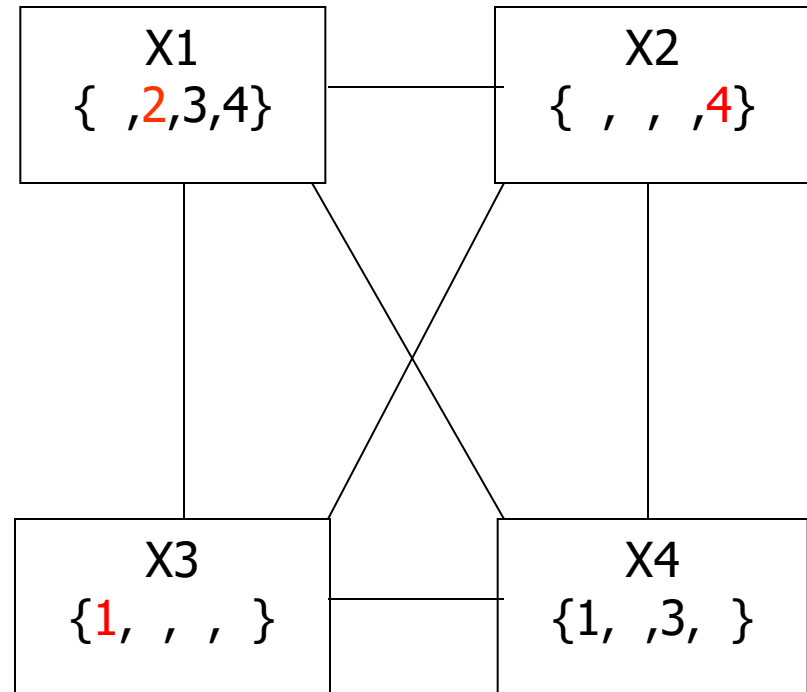
EXEMPLE : PROBLÈME DES 4-REINES

	1	2	3	4
1		●		
2	★	●	●	●
3		●	●	
4		★	●	●



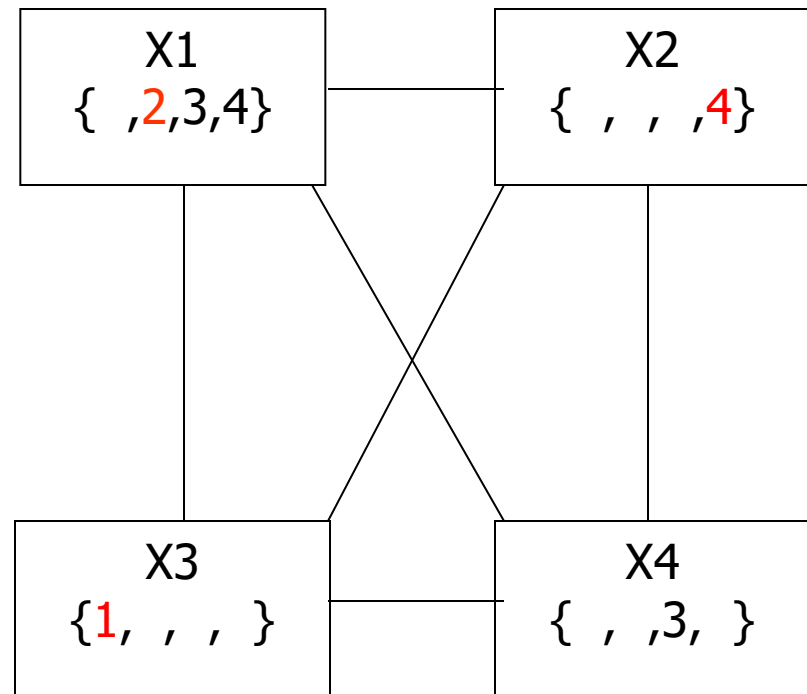
EXEMPLE : PROBLÈME DES 4-REINES

	1	2	3	4
1		●	★	●
2	★	●	●	●
3		●	●	
4		★	●	●



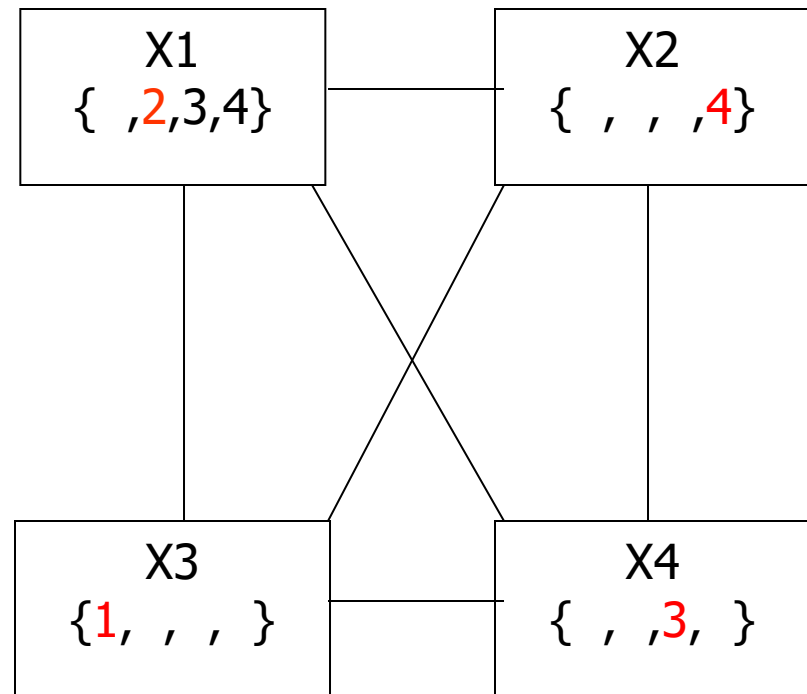
EXEMPLE : PROBLÈME DES 4-REINES

	1	2	3	4
1		●	★	●
2	★	●	●	●
3		●	●	
4		★	●	●

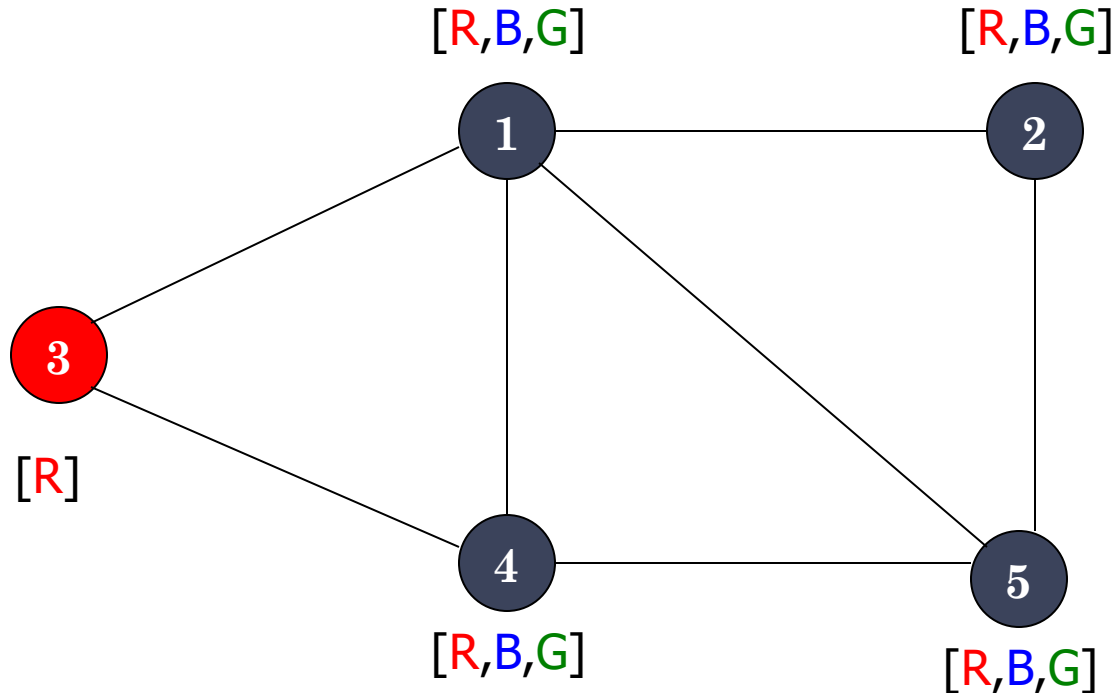


EXEMPLE : PROBLÈME DES 4-REINES

	1	2	3	4
1		●	★	●
2	★	●	●	●
3		●	●	★
4	■	★	●	●



EXERCICE



- Utiliser toutes les heuristiques (MRV, Degree heuristic, Least constraining value) ainsi que la propagation des contraintes pour résoudre ce CSP.

PROPAGATION DE CONTRAINTES

- *Forward checking* propage l'information d'une variable assignée vers les variables en contraintes avec elle, mais ne propagent l'effet des modifications de ces dernières.



	WA	NT	Q	NSW	V	SA	T
Domaines initiaux	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
Après WA=Red	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
Après Q=Green	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>



- Revenons à l'étape de backtracking-search, après que nous ayons choisi la variable Q et assigné la valeur bleu.
 - On voit ici le résultat de forward-checking
 - Forward-checking ne propage pas la modification du domaine NT vers SA pour constater que NT et SA ne peuvent pas être en bleu ensemble!
- La propagation des contraintes permet de vérifier ce type de conflits dans les assignations de variables.

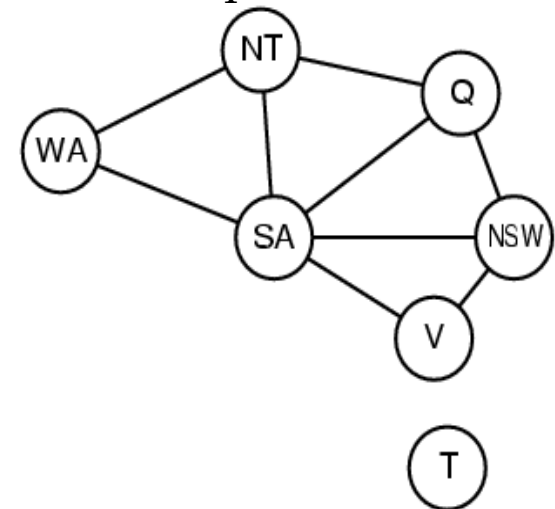
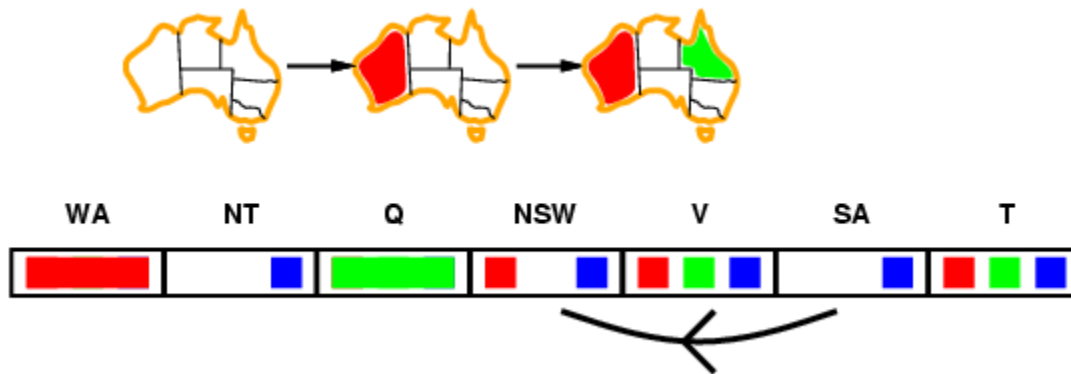
COHÉRENCE DES ARCS (ARC CONSISTENCY)

- *Arc consistency* est la forme de propagation de contraintes la plus simple

- Vérifie la consistance entre les arcs.
- C-à-d., la consistance des contraintes entre deux variables.

- L'arc $X \rightarrow Y$ est consistante si et seulement si

Pour **chaque** valeur x de X il existe **au moins une** valeur permise de y .



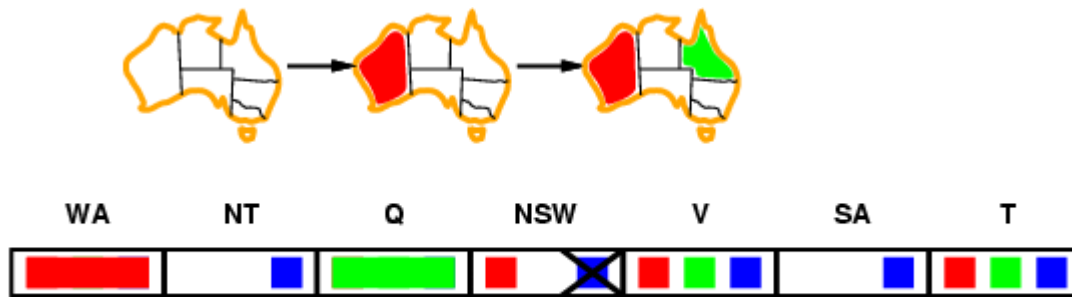
COHÉRENCE DES ARCS (ARC CONSISTENCY)

- *Arc consistency* est la forme de propagation de contraintes la plus simple

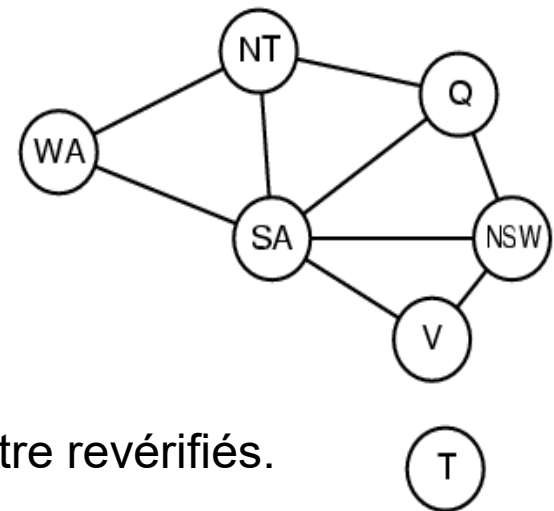
- Vérifie la consistance entre les arcs.
- C-à-d., la consistance des contraintes entre deux variables

- L'arc $X \rightarrow Y$ est consistant si et seulement si

Pour **chaque** valeur x de X il existe **au moins une** valeur permise de y .



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



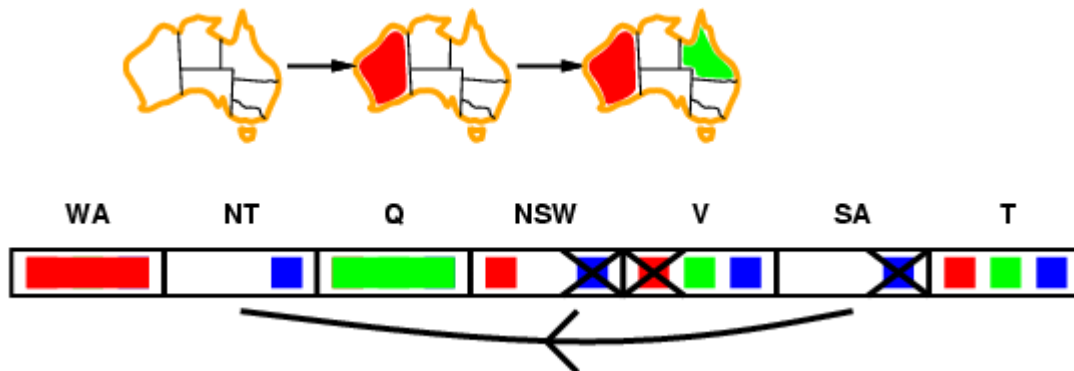
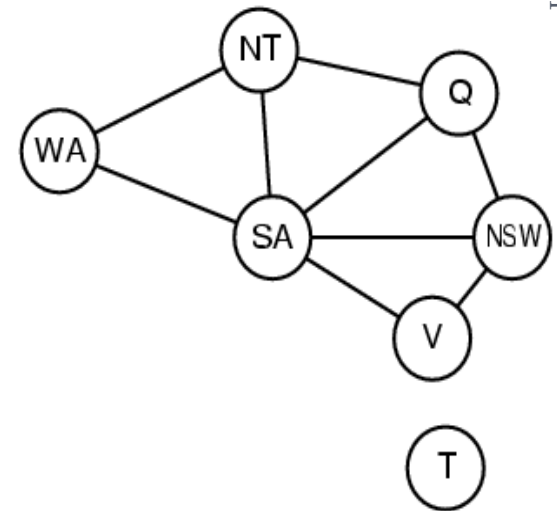
ARC CONSISTENCY (COHÉRENCE DES ARCS)

- *Arc consistency* est la forme de propagation de contraintes la plus simple

- Vérifie la consistance entre les arcs.
- C-à-d., la consistance des contraintes entre deux variables

- L'arc $X \rightarrow Y$ est consistante si et seulement si

Pour **chaque** valeur x de X il existe **au moins une** valeur permise de y .

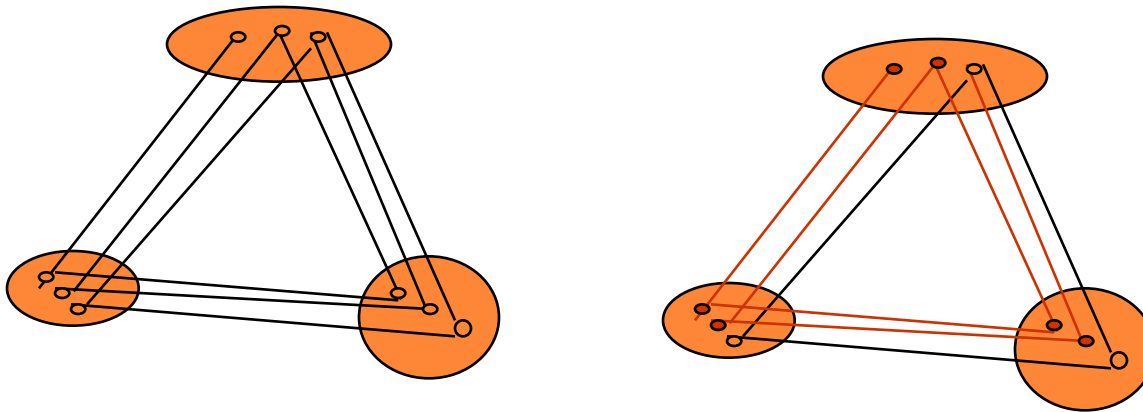


COHÉRENCE DES ARCS

- L'arc-consistance ne détecte pas toutes les inconsistances possibles
- On peut tester la consistance à un niveau supérieur à l'arc-consistance => notion de k-consistance (AC-k)
- Plus les vérifications de consistance sont d'ordre élevé, plus on a une :
 - Augmentation de la complexité en temps
 - Diminution du facteur de branchement

LA CONSISTANCE D'ARC (AC) : AC1

- Filtrage par consistance d'arc : suppression des valeurs qui ne vérifient pas la propriété (AC)



L'ALGORITHME AC1 (MACWORTH 1977)

Algorithme AC-1

répéter

pour chaque contrainte C_k **faire**

 Réviser(C_k)

jusqu'à plus de changement

Réviser(C_k) { $C_k = (X_i, X_j)$ }

pour tout $d_i \in D_i$ **faire**

si $\exists d_j \in D_j$ telle que $(d_i, d_j) \in R_k$

alors supprimer d_i de D_i

pour tout $d_j \in D_j$ **faire**

si $\exists d_i \in D_i$ telle que $(d_i, d_j) \in R_k$

alors supprimer d_j de D_j



L'ALGORITHME AC1 (MACWORTH 1977)

```
procédure Réviser((i,j))  
  changement ← faux  
  pour di ∈ Di faire  
    si ∃ dj ∈ Dj telle que (di,dj) ∈ Rk  
      alors supprimer di de Di  
      changement ← vrai  
    fin si  
  fin pour  
  retourner changement  
fin
```

```
procédure AC-1(G)  
  répéter  
    changement ← faux  
    pour chaque arc (i,j) ∈ G faire  
      changement ← réviser((i,j)) ou changement  
    fin pour  
  jusqu'à non (changement)  
fin AC-1
```



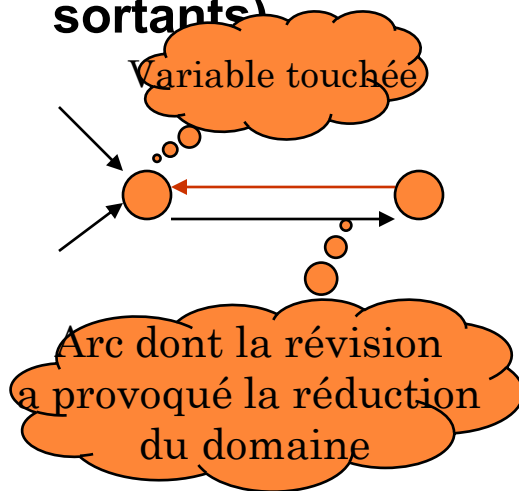
L'ALGORITHME AC1

◦ Quels sont les défauts de AC-1?

- Si un domaine est réduit alors tous les arcs sont testés (révisés) même si le domaine touché n'a aucune influence sur la plupart des arcs

◦ *Quels sont les arcs à reconsidérer ?*

- Les arcs affectés par le filtrage du domaine
i.e., les arcs reliés à la variable touchée (arcs entrants).
- Ignorer les arcs incidents de la variable touchée (arcs sortants)



L'algorithme AC3 [Mackworth 1977]

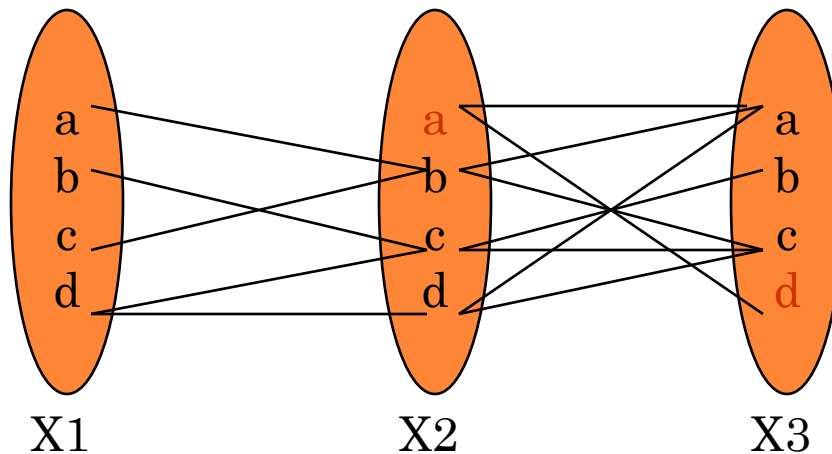
- ♦ utiliser une file pour mémoriser les arcs à (re-)réviser
- ♦ enfiler uniquement les arcs affectés par la réduction des domaines.

```
procedure AC-3(G)
    Q ← {(i,j) | (i,j) ∈ arcs(G), i < j}      % file pour les arcs à réviser
    tant que Q non vide faire
        sélectionner et supprimer (k,m) de Q
        si réviser((k,m)) alors
            Q ← Q ∪ {(i,k) | (i,k) ∈ arcs(G), i ≠ k, i ≠ m}
        fin si
    fin tant que
fin
```

➡ AC3 est l'algorithme le plus utilisé

L'ALGORITHME AC3 : OBSERVATIONS

- Révision d'un arc : de nombreuses paires de valeurs sont testées
- Ces tests sont répétés à chaque fois qu'un arc est révisé



1. Quand l'arc $\langle X2, X1 \rangle$ est révisé, la valeur a est supprimé du domaine de $X2$
2. le domaine de $X3$ doit être exploré pour déterminer si les valeurs a, b, c et d perdent leur support dans $X2$

- **Observation :**
- il n'est pas nécessaire d'examiner les valeurs a, b et c de $X3$ (elles admettent un autre support autre que a dans $X2$)
- L'ensemble support de $a \in D_i$ est l'ensemble $\{ \langle j, b \rangle \mid b \in D_j, (a, b) \in R_{ij} \}$



ARC CONSISTENCY 3 (AC-3)

function AC-3(*csp*) **return** the CSP, possibly with reduced domains

inputs: *csp*, a binary csp with components (X, D, C)

local variables: *queue*, a queue of arcs initially the arcs in *csp*

while *queue* is not empty **do**

$(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue})$

if REVISE(*csp*, X_i , X_j) **then**

if size of $D_i = 0$ **then return** *false*

for each X_k **in** $X_i.\text{NEIGHBORS} - \{X_j\}$ **do**

 add (X_k, X_i) to *queue*

return *true*

function REVISE(*csp*, X_i , X_j) **return** *true* if we revise the domain of X_i

revised \leftarrow *false*

for each x **in** D_i **do**

if no value y in D_j allows (x, y) to satisfy the constraints between X_i and X_j **then**

 delete x from D_i ;

removed \leftarrow *true*

return *revised*

RÉSUMÉ

- Les problèmes CSP sont des problèmes de recherche dans un espace d'assignation de valeurs à des variables.
- *Backtracking-search* = depth-first-search avec une variable assignée par nœud.
- L'ordonnancement des variables et celui de l'assignation des valeurs aux variables jouent un rôle significatif dans la performance.
- *Forward-checking* empêche les assignations qui conduisent à un conflit.
- La propagation des contraintes (par exemple, *arc-consistency*) détecte les inconsistances locales.
- Les méthodes les plus efficaces exploitent la structure du domaine.
- Application surtout à des problèmes impliquant l'ordonnancement des tâches.



CSP's LOCAUX

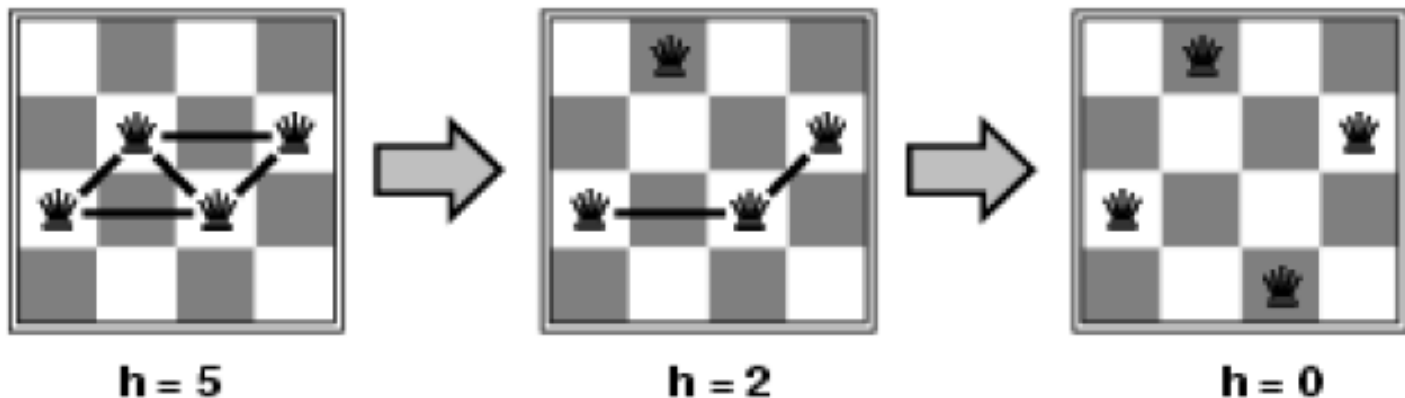
73

EXPLORATION LOCALE POUR LES CSP'S: MIN-CONFLICTS

- Sélection de variable: sélectionner au hasard n'importe quelle variable en conflit
- Sélection de la valeur avec l'heuristique *min-conflicts*:
 - Choisir la valeur qui viole le moins de contraintes possibles i.e., exécuter hill-climbing avec $h(n)$ = nombre total de contraintes violées = nombre de conflits

EXEMPLE: LES 4-REINES

- **Etats**: 4 reines dans 4 colonnes ($4^4 = 256$ états)
- **Actions**: déplacer une reine dans une colonne
- **Test du but**: pas d'attaque
- **Evaluation**: $h(n)$ = nombre d'attaques



CSP ET EXPLORATION LOCALE DANS LE MONDE RÉEL

- L'heuristique *Min-Conflicts* fonctionne bien :
- Pour certains problèmes difficiles (synchronisation des observations de Hubble)
- Pour modifier des paramètres à la volée
 - Ex: Changement des plans de vols dans un aéroport suite au mauvais temps => réaménager la planification avec un minimum de changements

EXERCICE 1

- On s'intéresse à un distributeur automatique de boissons. L'utilisateur insère des pièces de monnaie pour un total de T centimes d'Euros, puis il sélectionne une boisson, dont le prix est de P centimes d'Euros (T et P étant des multiples de 10). Il s'agit alors de calculer la monnaie à rendre, sachant que le distributeur a en réserve $E2$ pièces de 2 €, $E1$ pièces de 1€, $C50$ pièces de 50 centimes, $C20$ pièces de 20 centimes et $C10$ pièces de 10 centimes. **Modélisez ce problème sous la forme d'un CSP.**
- Comment pourrait-on exprimer le fait que l'on souhaite que le distributeur rende le moins de pièces possibles ?

EXERCICE 2: SEND MORE MONEY

- On considère l'addition suivante :

- $$\begin{array}{r} \text{SEND} \\ + \text{MORE} \\ \hline \text{= MONEY} \end{array}$$

où chaque lettre représente un chiffre différent (compris entre 0 et 9). On souhaite connaître la valeur de chaque lettre, sachant que la première lettre de chaque mot représente un chiffre différent de 0.

- Modélisez ce problème sous la forme d'un CSP.