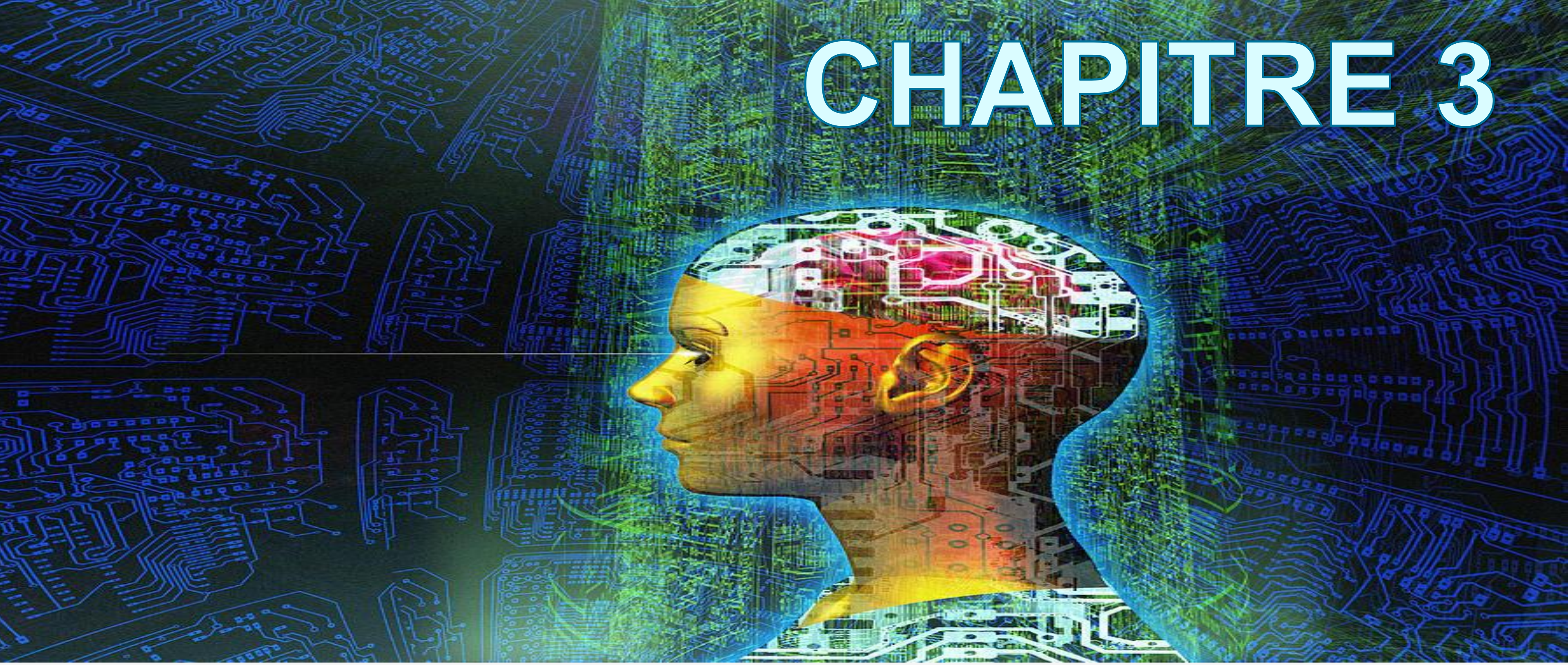


CHAPITRE 3

Problèmes à satisfaction de contraintes (CSP)

2020-2021

Plan

- Description des CSPs.
- Propagation de contraintes.
- Exploration avec retour arrière (backtracking).
- Exploration locale.

Description des CSP

Exploration d'un espace d'états versus CSP

- Résolution par exploration :
  - Exploration d'un espace d'états.
  - États pouvant être évalués par des heuristiques spécifiques.
  - Ces états sont atomiques (indivisibles).
- CSP :
  - Utilise une représentation factorisée des états par le biais de variables.
  - Un problème est résolu lorsque chaque variable a une valeur satisfaisant les contraintes.
  - Les algorithmes d'exploration CSP tirent profit de la structure des états et utilisent des heuristiques générales plus efficaces (élimine rapidement des états) pour résoudre des problèmes.

Problème de satisfaction de contraintes (CSP)

- Un problème de satisfaction de contraintes a trois composants :
  - *X* : un ensemble de variables.
  - *D* : un ensemble de domaines (valeurs permises) $\{D_1,\ldots,D_n\}$, un pour chaque variable.
  - *C* : un ensemble de contraintes spécifiant les combinaisons autorisées de valeurs prises par les variables, chaque contraintes fait participer certaines variables Xi.
- Exemple : si X1 et X2 ont le domaine {A,B}, une contrainte qui stipule les deux variables ont des valeurs différentes peut être $\langle(X1,X2), X1 \neq X2\rangle$.

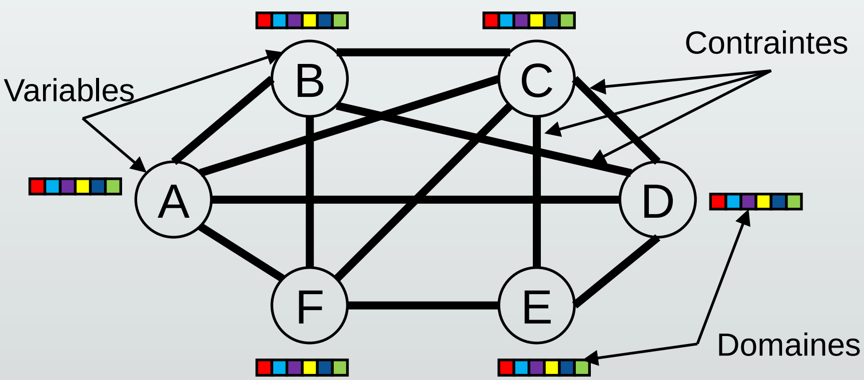
Problème de satisfaction de contraintes (CSP)

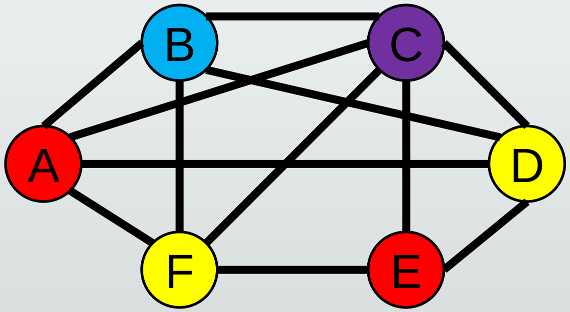
- Un état est une assignation de valeurs $\{v_1,\ldots,v_k\}$ admissibles dans D_i à certaines ou à toutes les variables X_i de l'état : $\{X_i=v_i, X_j=v_j,\ldots\}$.
- Une assignation consistante, cohérente ou légale est une assignation qui ne viole aucune contrainte.
- Une assignation complète affecte des valeurs à toutes les variables.
- Une assignation partielle attribue des valeurs qu'à certaines variables.
- Une solution à un problème CSP est une assignation complète et consistante.

Exemple 1 : coloration de graphes

- Trouver le nombre minimal de couleurs de telle sorte qu'aucuns nœuds adjacents n'aient la même couleur.

- Formulation du problème CSP :
  - X={A,B,C,D,E,F}.
  - D={rouge, jaune, bleu-foncé, bleu-ciel, vert, violet}.
  - C={A≠B, A≠C, A≠D, A≠F, B≠C, B≠D, B≠F, C≠F, C≠D, C≠E, D≠E, E≠F}





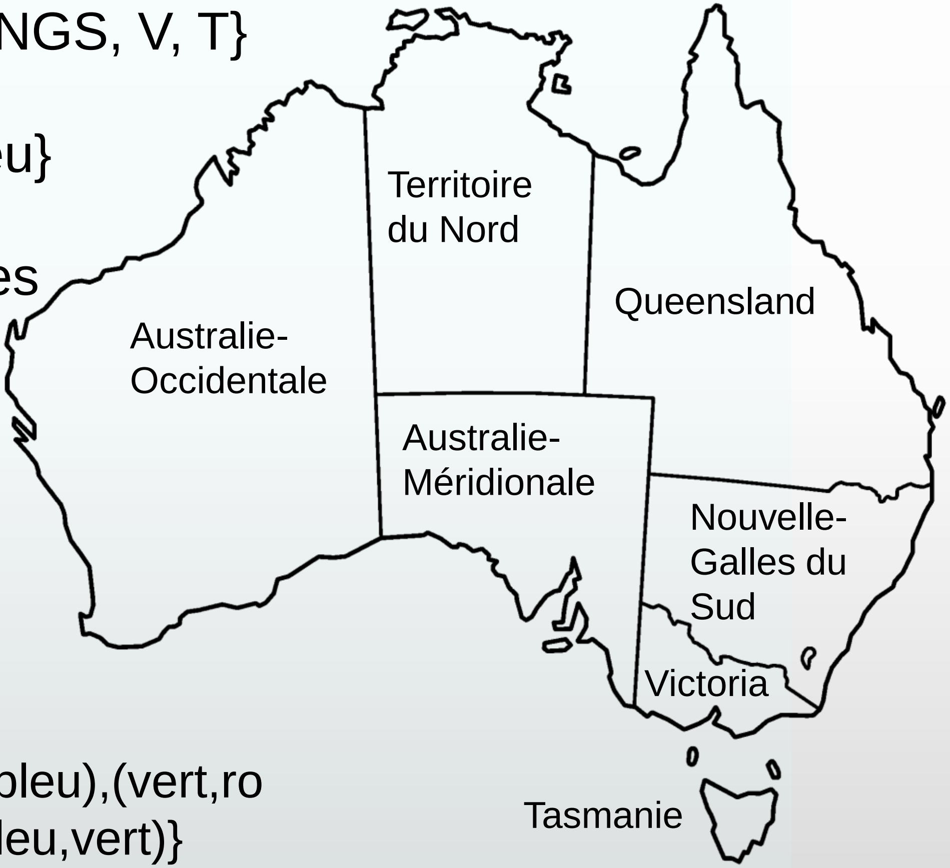
Une solution

Exemple 2 : coloration de carte

- Colorier la carte de l'Australie :
  - Utiliser uniquement 3 couleurs : bleu, rouge et vert.
  - Aucune région n'ait la même couleur qu'une de ses voisines.

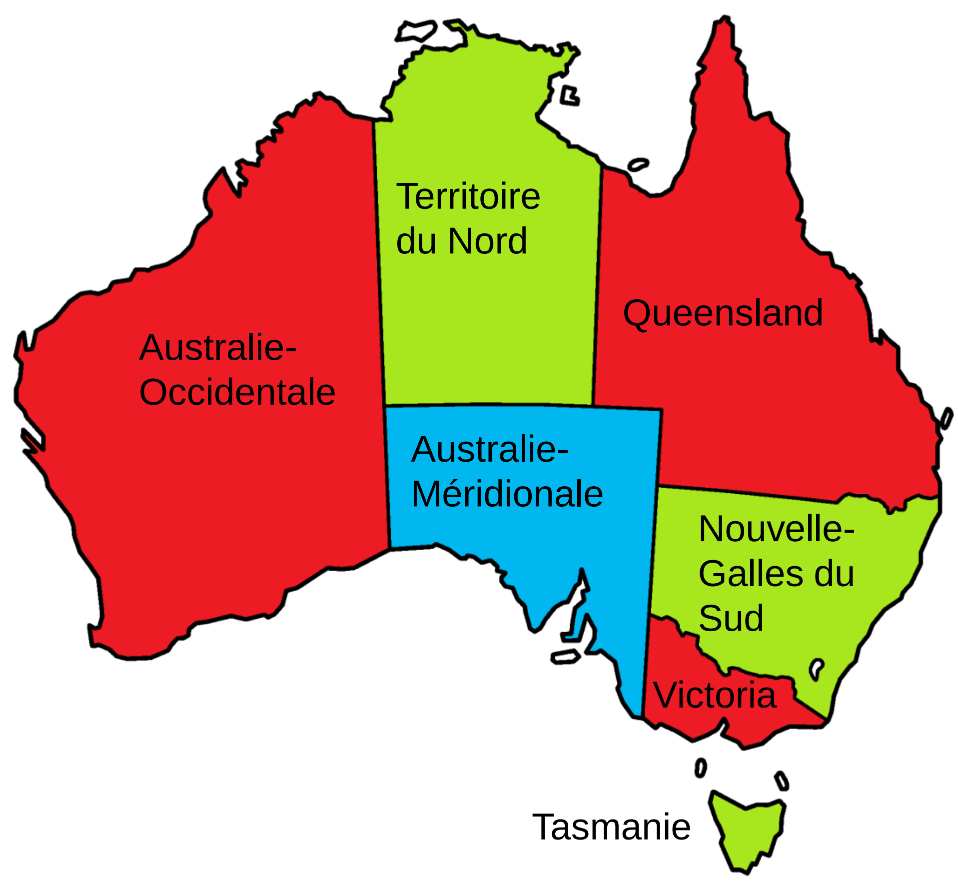
Exemple 2 : coloration de carte

- Variables X = {AO,AM, TN, Q, NGS, V, T}
- Domaines D_i = {rouge, vert, bleu}
- Contraintes : régions adjacentes doivent avoir des couleurs différentes :
  - C ={AM≠AO ,AM≠TN, AM ≠Q, AM ≠NGS, AM ≠V, AO ≠TN, TN ≠ Q, Q ≠ NGS, NGS ≠V}
  - AM ≠AO peut aussi s'écrire (AM,AO)∈{(rouge,vert),(rouge,bleu),(vert,rouge),(vert,bleu),(bleu,rouge),(bleu,vert)}



Exemple 2 : coloration de carte

- Une solution possible est une affectation qui satisfait toutes les contraintes :
  - *{AO=rouge, TN=vert, Q=rouge, NGS=vert,V=rouge, AM=bleu,T=rouge}*.
- Elle est complète et cohérente.



Exemple 3 : 4-Reines

- Variables : deux variables pour chaque reine i (C_i et L_i) correspondant à ses coordonnées et ayant {1, 2, …, n} pour domaine.

- Les contraintes :

- Colonnes : $C_i \neq C_j$ $\forall$ i,j $\in$ {1, 2, …, n} avec i $\neq$ j.

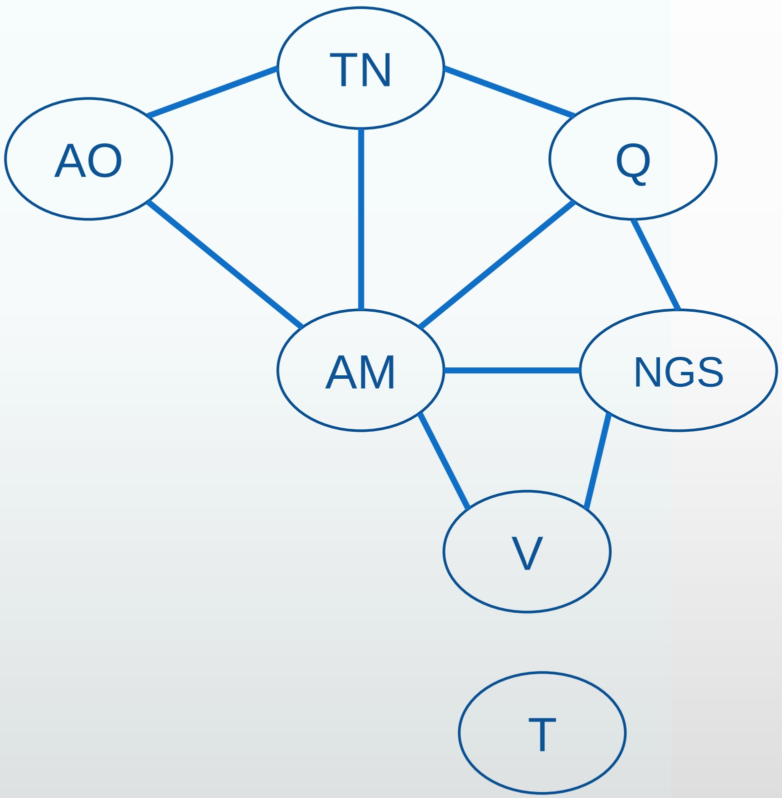
- Lignes : $L_i \neq L_j$ $\forall$ i,j $\in$ {1, 2, …, n} avec i $\neq$ j.

- Diagonales montantes : $L_i + (C_j - C_i) \neq L_j$ $\forall$ i,j $\in$ {1, 2, …, n} avec i $\neq$ j.

- Diagonales descendantes : $C_i + (L_j - L_i) \neq C_j$ $\forall$ i,j $\in$ {1, 2, …, n} avec i $\neq$ j.

Graphe de contraintes

- Un CSP peut être représenté comme un graphe de contraintes.
- Les nœuds sont les variables et les arcs relient les variables participantes à une contrainte.



Variantes du CSP

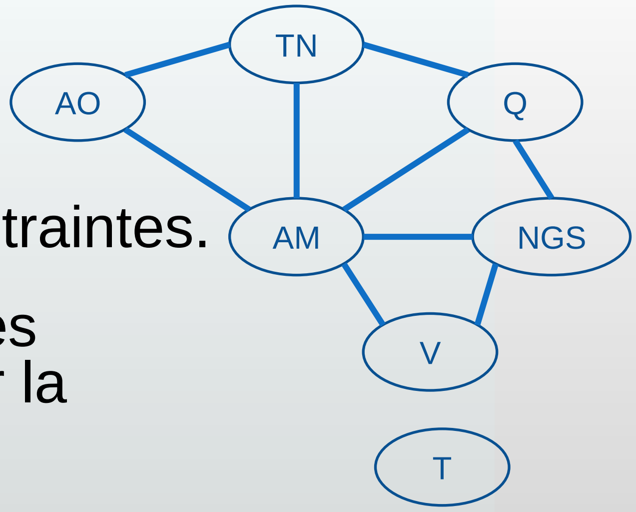
- Variables discrètes (domaines discrets) :
  - Domaines finis : pour *n* variables avec la taille maximale pour toute variable est *d*, il existe O(d^n) affectations complètes :
    - Exemples : CSP booléens, coloration de carte, problème des 8 reines (chaque reine Ri a une valeur dans {1,2,3,4,5,6,7,8}).
  - Domaines infinis : comme les entiers et chaînes de caractères :
    - Exemples : ordonnancement avec des variables indiquant les dates de début et de fin.
    - Une contrainte ne peut être décrite par une énumération, on a besoin d'un langage de contraintes, la tache T_1 qui dure 2 jours doit précéder la tâche T_2 : $T_1+2 \leq T_2$.
    - Contraintes linéaires : décidable.
    - Contraintes non linéaires : indécidable .

Variantes du CSP

- Variables continues (problèmes vus en RO) :
  - Exemple : synchronisation des expériences sur le télescope spatial Hubble utilise des variables à valeurs continues (temps) soumises à des contraintes astronomiques.
  - Catégorie la plus connue : programmation linéaire, où les contraintes sont des égalités ou inégalités linéaires et sont résolus en un temps polynomial par rapport au nombre de variables.

Types de contraintes

- Contrainte unaire : restreint la valeur d’une seule variable :
  - Exemple: AM≠*vert*.
- Contrainte binaire : porte sur deux variables :
  - Exemple : AM ≠ NGS.
  - Un CSP binaire ne contient que des contraintes binaires.
  - Représentable par un graphe de contraintes.
  - Des algorithmes généraux utilisent les graphes de contrainte pour accélérer la recherche.

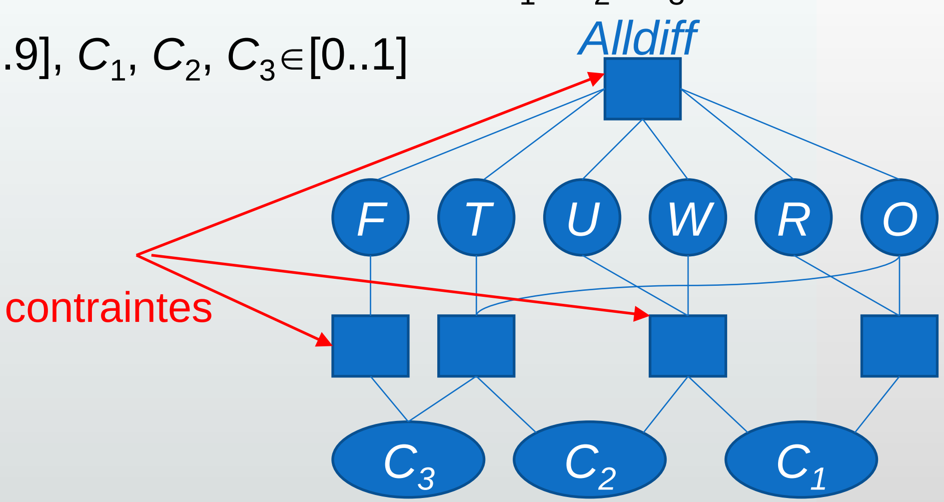


Types de contraintes

- **Contrainte d'ordre supérieur** : impliquent au minimum 3 variables (contrainte globale fait intervenir un nombre arbitraire de variables) :
  - Sudoku : *Alldiff*.
  - Énigmes cryptarithmétiques :

$$\begin{array}{rcccc} & C_3 & C_2 & C_1 & \\ & & T & W & O \\ + & & T & W & O \\ \hline & F & O & U & R \end{array}$$

    - Variables : F,T,U,W,R,O *et des variables de retenue*, C_1, C_2, C_3
    - Domaines : $F,T,U,W,R,O \in [0..9]$, $C_1, C_2, C_3 \in [0..1]$
    - Contraintes :
      - $Alldiff(F,T,U,W,R,O)$
      - $O+O=R+10 \cdot C_1$
      - $C_1+W+W=U+10 \cdot C_2$
      - $C_2+T+T=O+10 \cdot C_3$
      - $C_3=F$



hypergraphe

Types de contraintes

- Contrainte de préférence :
  - Ne sont pas absolues : un professeur ne peut pas donner un cours dans deux endroits différents en même temps est une contrainte absolue.
  - Indique quelle solution est préférée : le professeur X préfère enseigner le matin alors que le professeur Y préfère l’après midi.
  - Un emploi dans lequel le professeur X enseigne à 15h est une solution bien qu’elle soit non optimale.
  - Souvent écrites sous forme de coûts des assignations des variables qu’on cherche à optimiser : l’attribution au professeur X un créneau horaire l’après midi coûte 2 points à la fonction objectif alors qu’un créneau le matin coûte 1 point.
  - Se sont des problèmes d’optimisation de contraintes (ou COP pour *Constraint Optimization Problem*)

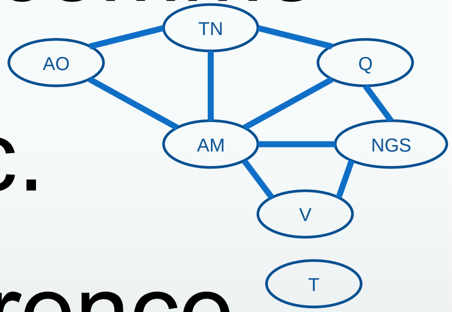
Propagation de contraintes

Propagation de contraintes

- Exploration standard dans un espace d'états : uniquement de l'exploration.
- Propagation de contraintes :
  - Réduire le nombre de valeurs légales d'une variable (elle peut engendrer la réduction de valeurs légales pour une autre variable).
  - Peut être combinée à l'exploration ou comme prétraitement de celle-ci.

Principe de la propagation de contraintes

- Idée : cohérence locale.
- Traiter chaque variable comme un nœud et chaque contrainte comme un arc.



- L'application de la cohérence locale dans chaque partie du graphe élimine les valeurs incohérentes.

Types de cohérence locale

- Cohérence de nœuds.
- Cohérence d’arc.
- Cohérence de chemin.
- K-cohérences.
- Contraintes globales.

Cohérence de nœuds

- Une variable Xi est dite nœud-cohérente si toutes les valeurs dans son domaine satisfont ses contraintes unaires :
  - Si les australiens méridionaux détestent le vert, la cohérence du nœud AM est obtenue par l'élimination de la couleur verte de son domaine, ce qui donne {rouge,bleu}.
- Un CSP est dit nœud-cohérent si chacune de ses variables est nœud-cohérente.
- Une fois la cohérence de nœuds est obtenue, toutes les contraintes unaires peuvent être éliminées.
- Il est aussi possible de transformer toutes les contraintes n-aires en contraintes binaires (uniquement des résolveurs de CSP pour des contraintes binaires seront considérés).

Cohérence d'arcs

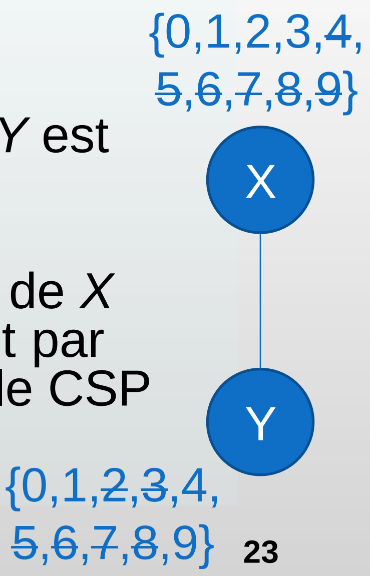
- Une variable dans un CSP est dite arc-cohérente si chaque valeur de son domaine satisfait ses contraintes binaires.

- Formellement, une variable X_i est arc-cohérente par rapport à une variable X_j si pour chaque valeur v de son domaine D_i, il existe au moins une valeur w dans D_j tel que (v,w) satisfait la contrainte (arc) (X_i,X_j), on note $X_i \rightarrow X_j$.

- Un CSP est arc-cohérent si chaque variable X_i est arc-cohérente avec tout autre variable X_j.

- Ex. Pour la contrainte $Y=X^2$, avec le domaine de X et de Y est {0,…,9}, la contrainte est [(X,Y),{(0,0),(1,1), (2,4),(3,9)}].

- Pour rendre X arc-cohérent relativement à Y, le domaine de X doit être réduit à {0,1,2,3} et pour que Y soit arc-cohérent par rapport à X, le domaine de Y doit être réduit à {0,1,4,9} : le CSP est donc arc-cohérent.



Algorithme AC-3 pour la cohérence d'arcs

- Lorsqu'une incohérence d'arc est détectée, les valeurs ayant créer le conflit sont éliminées du domaine de la variable source.

Réellement c'est une liste.

Même si X_k a été traité, il faut l'ajouter à la file, puisque la modification de D_i peut entrainer des réductions dans les domaines des D_k.

fonction AC-3(*csp*) **retourne** faux si une incohérence est détectée, vrai autrement
  entrées: *csp*, un CSP binaire avec les composantes (X, D, C)
  variables locales: *file*, une file d'arcs, initialement tous ceux dans *csp*

  tant que *file* n'est pas vide **faire**
    $(X_i, X_j) \leftarrow$ ENLEVER-PREMIER(*file*)
    si RÉVISER(*csp*, X_i, X_j) **alors**
      si taille de $D_i = 0$ **alors retourner** *faux*
      pour chaque X_k **dans** X_i.VOISINS - $\{X_j\}$ **faire**
        ajouter (X_k, X_i) à *file*
  retourner *vrai*

---

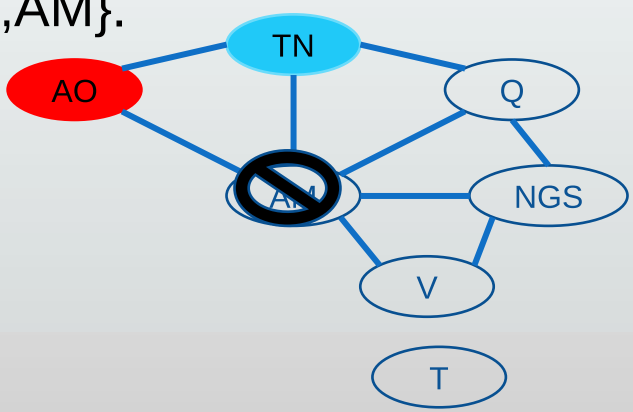
fonction RÉVISER(*csp*, X_i, X_j) **retourne** vrai si et seulement si le domaine de X_i a été mis à jour
  modifié $\leftarrow$ *faux*
  pour chaque x **dans** D_i **faire**
    si aucune valeur y de D_j ne permet à (x,y) de satisfaire la contrainte entre X_i et X_j **alors**
      enlever x de D_i
      modifié $\leftarrow$ *vrai*
  retourner *modifié*

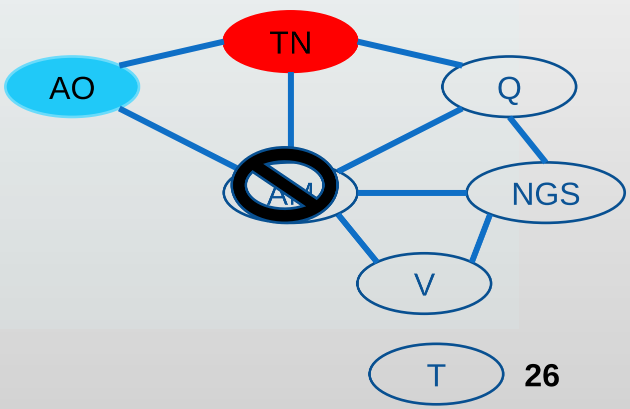
Cohérence de chemin

- La cohérence d'arc n'est pas toujours utile puisqu'elle ne réalise pas assez d'inférences.
- Ex. Carte de l'Australie avec 2 couleurs {*rouge*,*bleu*} : toutes les variables sont arc-cohérentes, mais aucune solution n'est possible !
- La cohérence de chemin utilise des contraintes implicites inférées à partir de triplets de variables (algo PC-2).
- $\{X_i,X_j\}$ est chemin-cohérent relativement à X_m si pour chaque assignation cohérente (satisfaisant la contrainte sur $\{X_i,X_j\}$) $\{X_i=a,X_j=b\}$ il existe encore une valeur cohérente pour X_m (satisfaisant $\{X_i,X_m\}$ et $\{X_m,X_j\}$).

Cohérence de chemin

- Pour l’Australie avec deux couleurs (bleu et rouge), en imposant la cohérence de chemin entre n’importe quel triplet connecté par des arcs (ex. {AO, AM} et TN), on peut détecter une incohérence inhérente.
  - Assignations cohérentes pour AO et AM : {AO=*rouge*,AM=*bleu*} et {AO=*bleu*,AM=*rouge*}.
  - Par conséquent il n’y a plus d’assignation pour TN et pour garantir la cohérence on doit éliminer les deux assignations pour {AO,AM}.





K-cohérence forte

- L'algorithme de cohérence de chemin comme PC-2 ne peut pas résoudre certains problèmes tel que le coloriage de cartes nécessitant 4 couleurs.
- Un graphe est fortement *k*-cohérent s'il est *k*, (*k-1*), …, *2* et *1*-cohérent.
- Si un CSP avec *n* variables est *n*-cohérent, il peut être résolu sans retour-arrière.
- Mais attention à la complexité !

Contraintes globales

- <u>Contrainte Alldiff</u> : toutes les variables impliquées doivent être différentes :
  - S'il existe plus de variables que de valeurs, le CSP est non solvable.
  - Si une variable a uniquement une valeur, l'assigner et supprimer cette valeur du domaine des autres variables : détection rapide d'incohérence.
  - Ex. carte de l'Australie avec l'assignation {AO=rouge, NGS=rouge} : AM, TN et Q sont reliées par la contrainte Alldiff, l'application de AC-3 (cohérence d'arc binaire) est moins efficace pour détecter l'incohérence qu'avec l'application de la contrainte globale Alldiff.

Contraintes globales

- Contrainte de ressource (Atmost) : limite l'allocation des ressources. Ex. 10 personnes pour 4 tâches avec P1, P2, P3 et P4 des variables : Pi nombre de personnes affectées à la tâche i $\Rightarrow$ Atmost(10,P1,P2,P3,P4)
  - Incohérence si la somme des valeurs minimales des domaines des Pi >10 : exemple Di={3,4,5,6}.
  - On peut obtenir la cohérence en éliminant les valeurs maximales de certains domaines si elles sont non cohérentes avec les valeurs minimales des autres domaines : pour Di ={2,3,4,5,6} les valeurs 5 et 6 doivent être éliminés de tous les domaines.
- Dans le cas où les domaines sont très grands, ils sont représentés par des bornes inf et sup et gérés par propagation de limites, on parle alors de limite-cohérente.

Propagation de contraintes : Exemple de Sudoku avec AC-3

- La grille de sudoku peut être vue comme un CSP avec 81 variables : une pour chaque case (Ex. A1).
- Le domaine des cases vides est : {1,2,3,4,5,6,7,8,9}.
- Le domaine de chaque case remplie est la valeur affichée.

	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		

Propagation de contraintes : Exemple de Sudoku avec AC-3

- Il existe 27 contraintes différentes *AllDiff*, une pour chaque ligne, chaque colonne, et chaque zone de 9 cases :

$Alldiff(A1, A2, A3, A4, A5, A6, A7, A8, A9)$
$Alldiff(B1, B2, B3, B4, B5, B6, B7, B8, B9)$
$\cdots$
$Alldiff(A1, B1, C1, D1, E1, F1, G1, H1, I1)$
$Alldiff(A2, B2, C2, D2, E2, F2, G2, H2, I2)$
$\cdots$
$Alldiff(A1, A2, A3, B1, B2, B3, C1, C2, C3)$
$Alldiff(A4, A5, A6, B4, B5, B6, C4, C5, C6)$
$\cdots$

	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		

Propagation de contraintes : Exemple de Sudoku avec AC-3

- Pour appliquer AC-3, il faut travailler sur des contraintes binaires :
  - *A1≠A2, A1 ≠A3,…A1 ≠A9,A2 ≠A3,…A2 ≠A9,… A8 ≠A9*
  - *B1≠B2, B1 ≠B3,…B1 ≠B9,B2 ≠B3,…B2 ≠B9,… B8 ≠B9*
  - *……*

	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		

Propagation de contraintes : Exemple de Sudoku avec AC-3

- Considérons E6 :
  - Le domaine initial est {1,2,3,4,5,6,7,8,9}
  - La contrainte sur la boite exclus 1,2,7 et 8 du domaine de E6.
  - La contrainte sur la colonne exclus 3,5,6,9.
  - La contrainte sur la ligne n'exclut aucune valeur.
  - Ceci laisse une seule valeur pour E6 : 4.

	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		

Propagation de contraintes : Exemple de Sudoku avec AC-3

- Considérons E6 :
  - Le domaine initial est {1,2,3,4,5,6,7,8,9}
  - La contrainte sur la boite exclus 1,2,7 et 8 du domaine de E6.
  - La contrainte sur la colonne exclus 3,5,6,9.
  - La contrainte sur le carré n'exclut aucune valeur.
  - Ceci laisse une seule valeur pour E6 : 4.

	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					4			8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

Propagation de contraintes : Exemple de Sudoku avec AC-3

- Considérons I6 :
  - Le domaine initial est {1,2,3,4,5,6,7,8,9}
  - La contrainte sur ligne exclut 1,3,5.
  - La contrainte sur la colonne exclut 2,4,6,8,9.
  - La contrainte sur la boite n’exclut aucune valeur.
  - Ceci laisse une seule valeur pour E6 : 7.

	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					**4**			8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

Propagation de contraintes : Exemple de Sudoku avec AC-3

- Considérons I6 :
  - Le domaine initial est {1,2,3,4,5,6,7,8,9}
  - La contrainte sur ligne exclut 1,3,5.
  - La contrainte sur la colonne exclut 2,4,6,8,9.
  - La contrainte sur la boite n'exclut aucune valeur.
  - Ceci laisse une seule valeur pour E6 : 7.

	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					4			8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1	7	3		

Propagation de contraintes : Exemple de Sudoku avec AC-3

- Considérons A6 :
  - Le domaine initial est {1,2,3,4,5,6,7,8,9}
  - La contrainte sur ligne exclut 2,3,6.
  - La contrainte sur la colonne exclut 4,5,7,8,9.
  - La contrainte sur la boite n’exclut aucune valeur.
  - Ceci laisse une seule valeur pour E6 : 1.

	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					4			8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1	7	3		

Propagation de contraintes : Exemple de Sudoku avec AC-3

- Considérons A6 :
  - Le domaine initial est {1,2,3,4,5,6,7,8,9}
  - La contrainte sur ligne exclut 2,3,6.
  - La contrainte sur la colonne exclut 4,5,7,8,9.
  - La contrainte sur la boite n’exclut aucune valeur.
  - Ceci laisse une seule valeur pour E6 : 1.

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

Propagation de contraintes : Exemple de Sudoku avec AC-3

- En continuant l'application de AC-3 pour la cohérence d'arcs, nous pouvons résoudre le Sudoku complet !
- Tous les Sudokus ne peuvent pas être résolus par la cohérence d'arcs seule, et c'est tant mieux !
- Ceux un peu plus difficiles peuvent être résolus par PC-2, mais pour un coût plus élevé.
- Pour les grilles les plus difficiles il faut appliquer des inférences plus astucieuses : « triplets nus ».

	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

Limites de l'inférence sur les contraintes

- Certains problèmes, comme le Sudoku, peuvent être résolus par inférence sur des contraintes.
- Mais plusieurs ne peuvent pas être résolus par cette inférence seule : ne suffit pas pour trouver toujours une solution complète.

Résolution pour les CSP

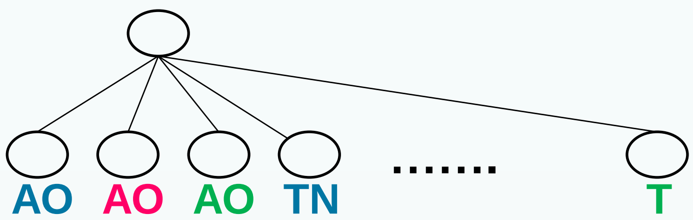
- Recherche standard dans un graphe : exploration avec retour arrière.
- Recherche locale.

Exploration avec retour arrière (backtracking)

CSP par la méthode d’exploration standard

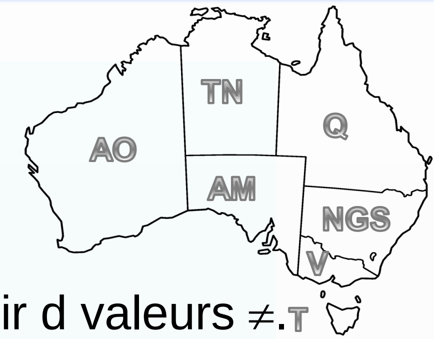
- Formulation incrémentale : la même pour tous les CSP.
- Espace d’états : assignations partielles de variables.
- État initial : une assignation vide.
- Fonction successeur : affecte une valeur à n’importe quelle variable non assignée pourvu quelle ne viole pas une contrainte.
- Etat but : une assignation complète et cohérente.
- Coût du chemin : coût constant pour chaque étape (en général non pertinent).
- Toute solution apparait à une profondeur n, avec n est le nombre de variables.

CSP par la méthode d'exploration standard

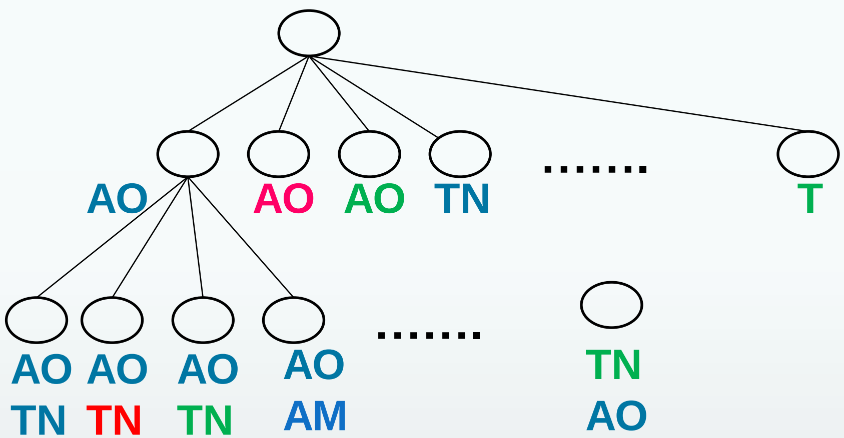


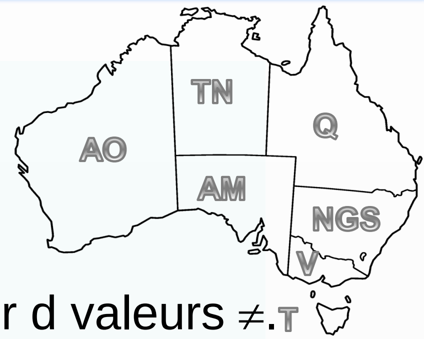
n variables pouvant avoir d valeurs $\neq$.

Facteur de branchement $b = n \times d$



CSP par la méthode d'exploration standard

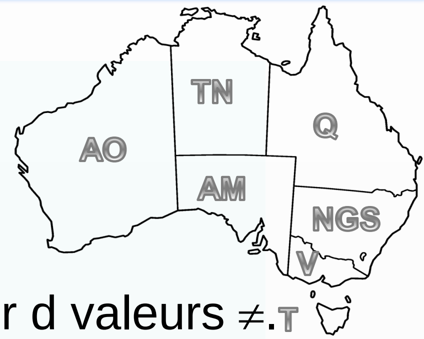


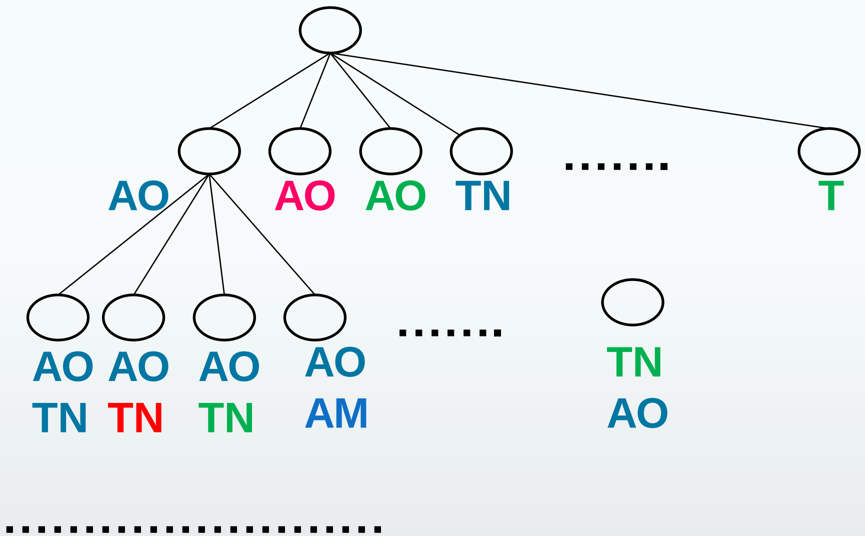


n variables pouvant avoir d valeurs $\neq$.
Facteur de branchement $b=n \times d$

n-1 variables pouvant avoir d valeurs $\neq$.
Facteur de branchement $b=(n-1) \times d$

CSP par la méthode d’exploration standard



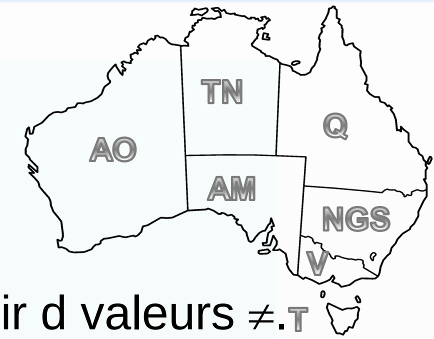


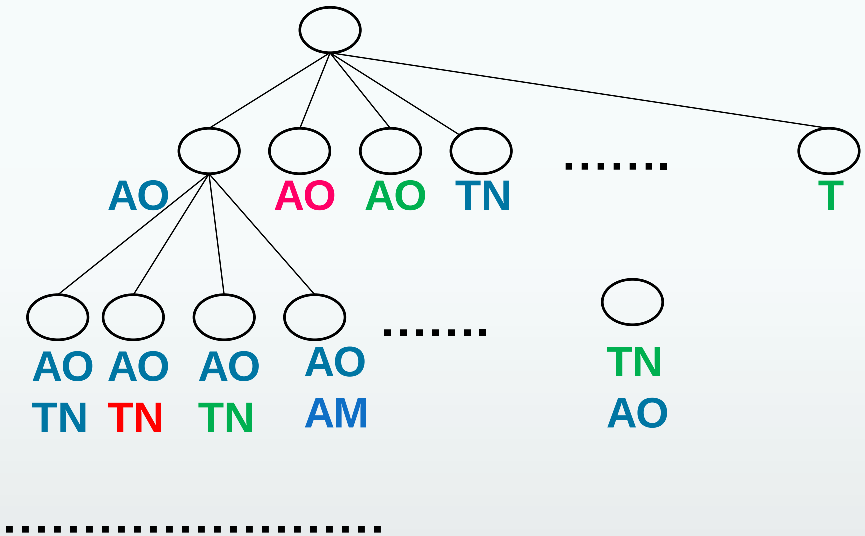
n variables pouvant avoir d valeurs $\neq$.
Facteur de branchement $b=n \times d$

n-1 variables pouvant avoir d valeurs $\neq$.
Facteur de branchement $b=(n-1) \times d$

Il y aura $n! \times d^n$ feuilles mais uniquement d^n assignations complètes !

CSP par la méthode d'exploration standard





n variables pouvant avoir d valeurs $\neq$.
Facteur de branchement $b=n\times d$

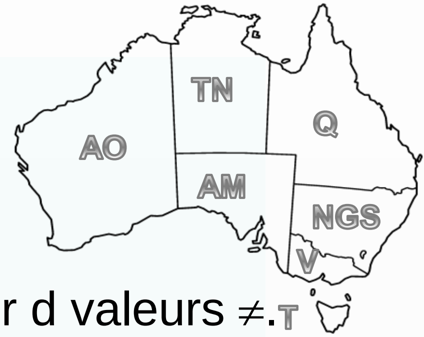
n-1 variables pouvant avoir d valeurs $\neq$.
Facteur de branchement $b=(n-1)\times d$

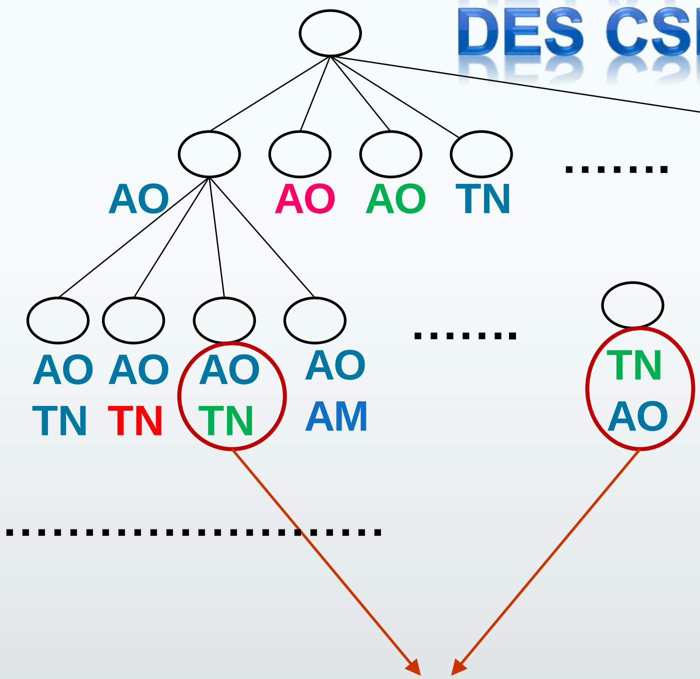
Il y aura $n! \times d^n$ feuilles mais uniquement d^n assignations complètes !

Une exploration en largeur d'abord ou en profondeur limitée seraient totalement inefficace.

CSP par la méthode d'exploration standard (largeur d'abord)

SOLUTION : TIRER PROFIT DE LA COMMUTATIVITÉ DES CSP





Répétition : les CSP sont commutatifs.

n variables pouvant avoir d valeurs $\neq$.
Facteur de branchement : $n \times d$

n-1 variables pouvant avoir d valeurs $\neq$.
Facteur de branchement : $(n \times d) \times (n-1) \times d$

Il y a aura $n! \times d^n$ feuilles mais uniquement d^n assignations complètes !

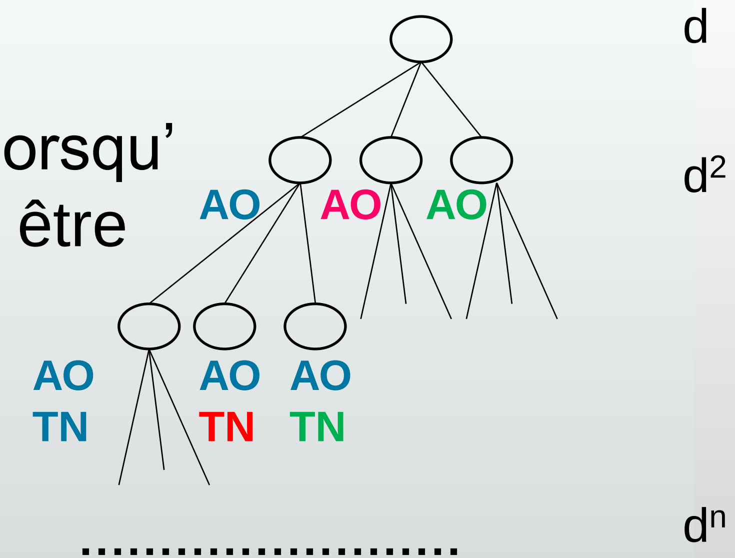
Une exploration en largeur d'abord ou en profondeur limitée seraient totalement inefficace.

Quoi faire ?

- Tirer profit de la commutativité des CSP (AO=*bleu*,TN=*vert*) c'est la même que (TN=*vert*, AO=*bleu*).
- Il suffit d'assigner une seule variable à chaque nœud : facteur de branchement =d.
- Une recherche en profondeur d'abord en considérant <u>une seule variable à chaque nœud</u> est appelée exploration avec retour arrière (*backtracking search*).

Exploration avec retour arrière (*backtracking search*)

- C'est l'algorithme de base d'une exploration non informée pour les CSPs.
- Basé sur une exploration en profondeur d'abord avec l'assignation d'une seule variable du nœud à la fois.
- Il a recours au retour arrière lorsqu' aucune valeur légale ne peut être assignée à une variable.
- $b=d$ et il existe d^n feuilles.



Algorithme de l'exploration avec retour arrière

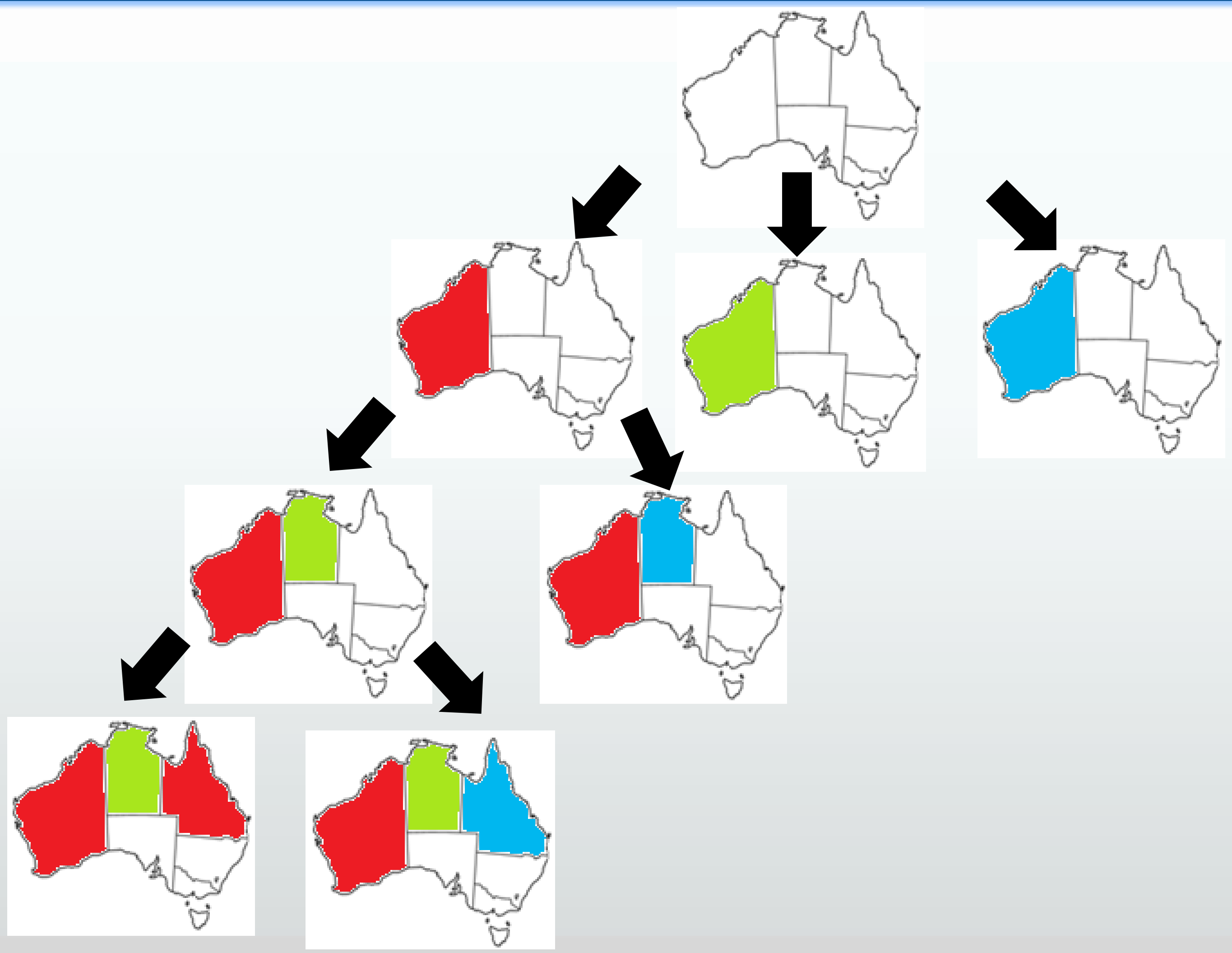
fonction EXPLORATION-BACKTRACKING(*csp*) **retourne** une solution, ou échec
  retourner BACKTRACK({}, *csp*)

fonction BACKTRACK(*assignation*, *csp*) **retourne** une solution, ou échec
  si *assignation* est complète **alors retourner** *assignation*
  var ← SÉLECTIONNER-VARIABLE-NON-ASSIGNÉE(*csp*)
  pour chaque *valeur* **dans** ORDONNER-VALEURS-DOMAINE(*var*, *assignation*, *csp*) **faire**
    si *valeur* est cohérente avec *assignation* **alors**
      ajouter {*var* = *valeur*} à *assignation*
      inférences ← INFÉRENCE(*csp*, *var*, *valeur*)
      si *inférences* ≠ *échec* **alors**
        ajouter *inférences* à *assignation*
        résultat ← BACKTRACK(*assignation*, *csp*)
        si *résultat* ≠ *échec* **alors**
          retourner *résultat*
    enlever {*var* = *valeur*} et *inférences* de *assignation*
  retourner *échec*

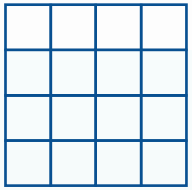
Optionnelle, elle permet d'imposer une cohérence d'arc, de chemin, …

Ces fonctions peuvent être modifiées pour développer des heuristiques (voir suite du cours)

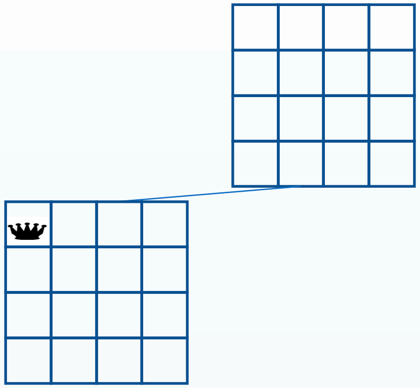
Exemple : coloration d'une carte avec retour arrière



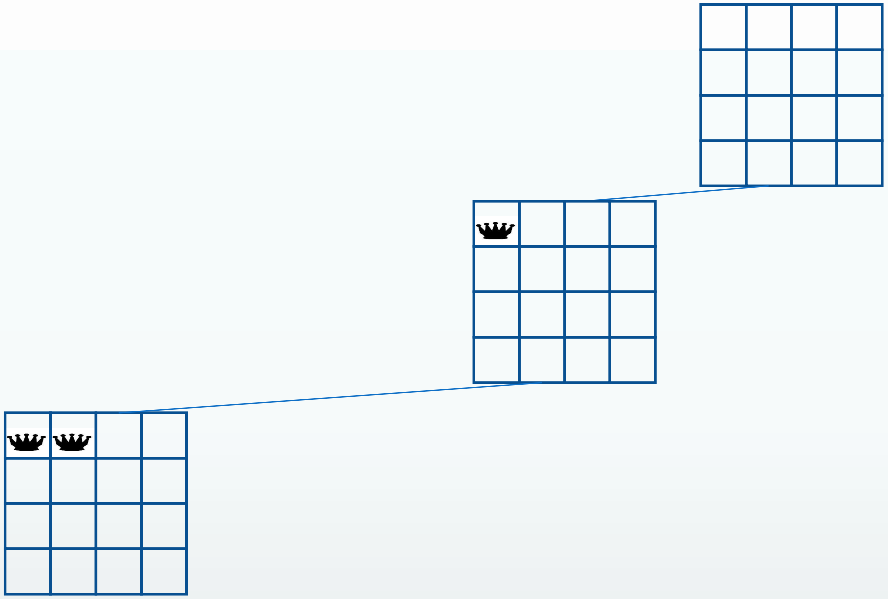
Exemple : résolution des 4 reines avec retour arrière



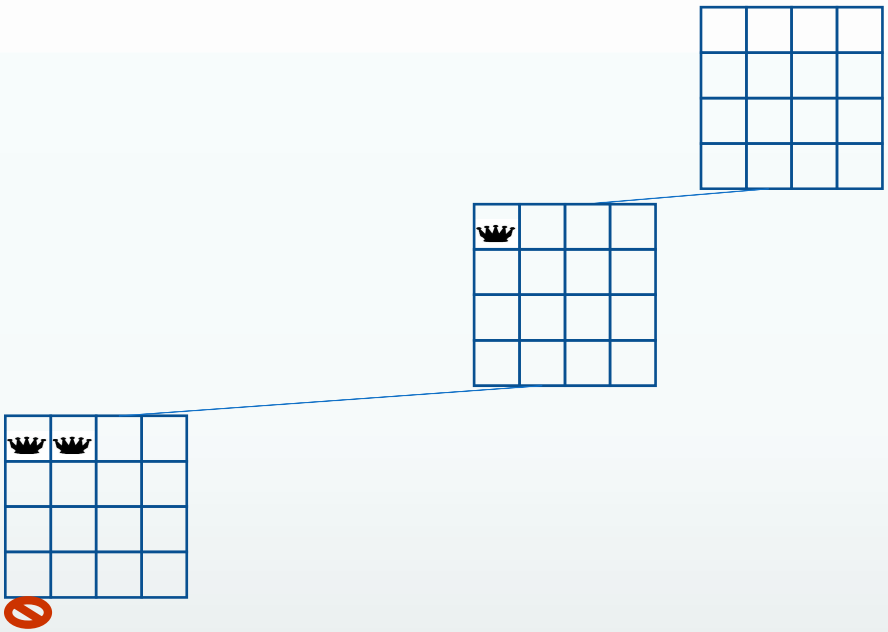
Exemple : résolution des 4 reines avec retour arrière



Exemple : résolution des 4 reines avec retour arrière

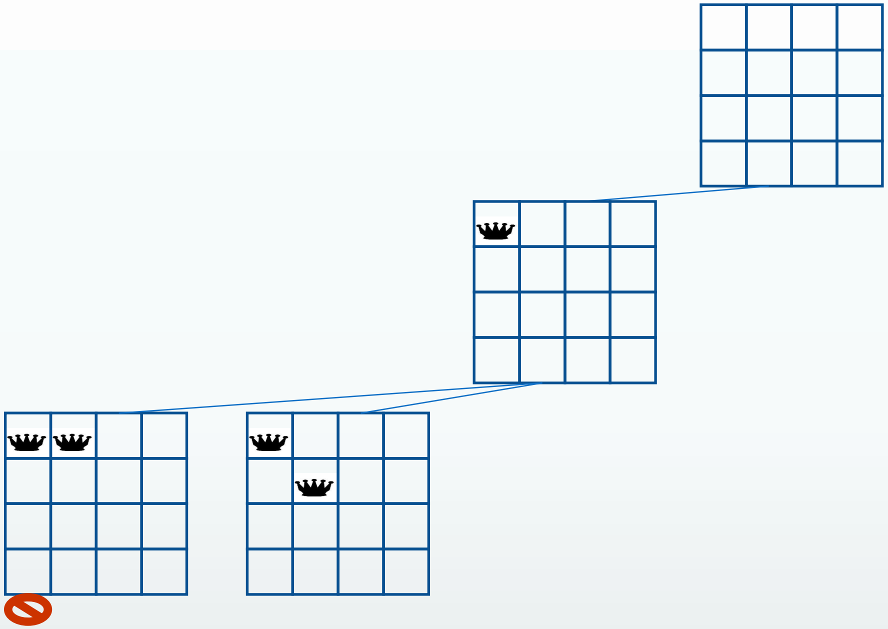


Exemple : résolution des 4 reines avec retour arrière



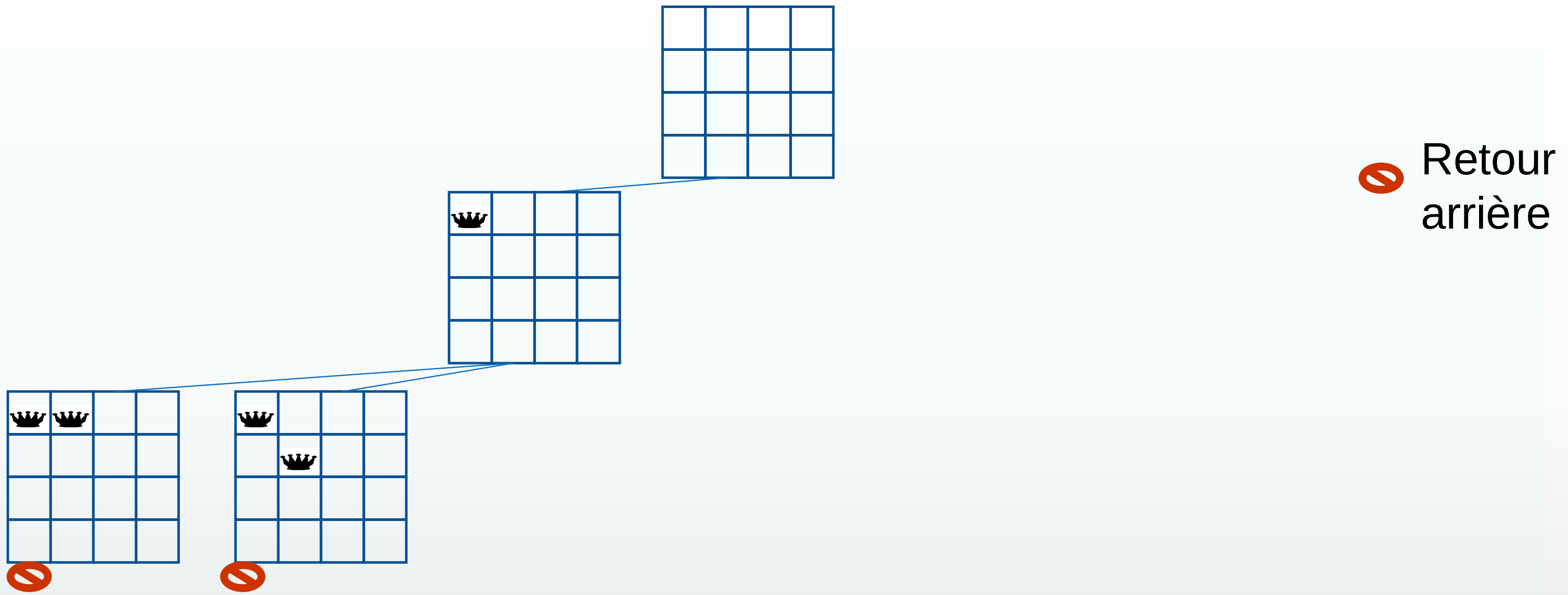
Retour arrière

Exemple : résolution des 4 reines avec retour arrière

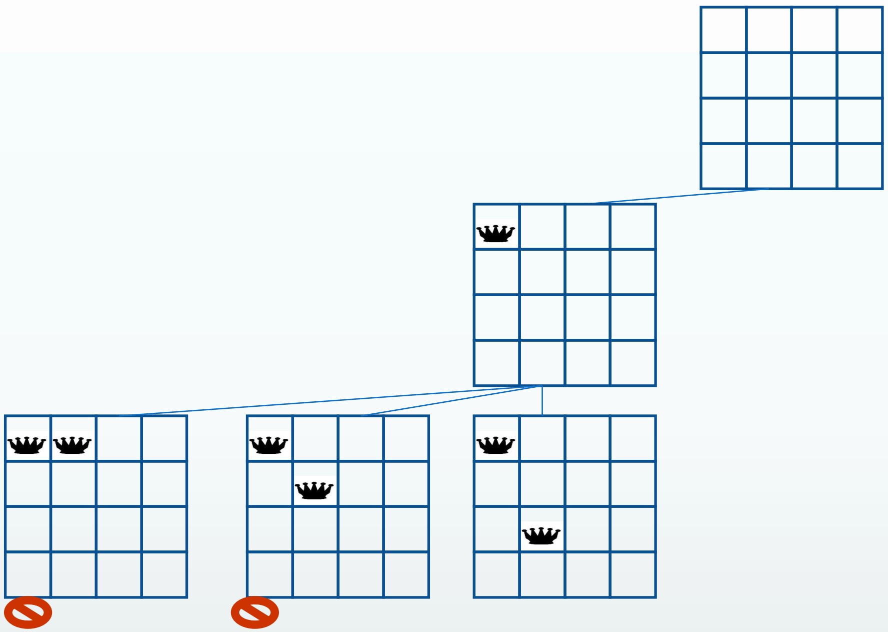


Retour arrière

Exemple : résolution des 4 reines avec retour arrière

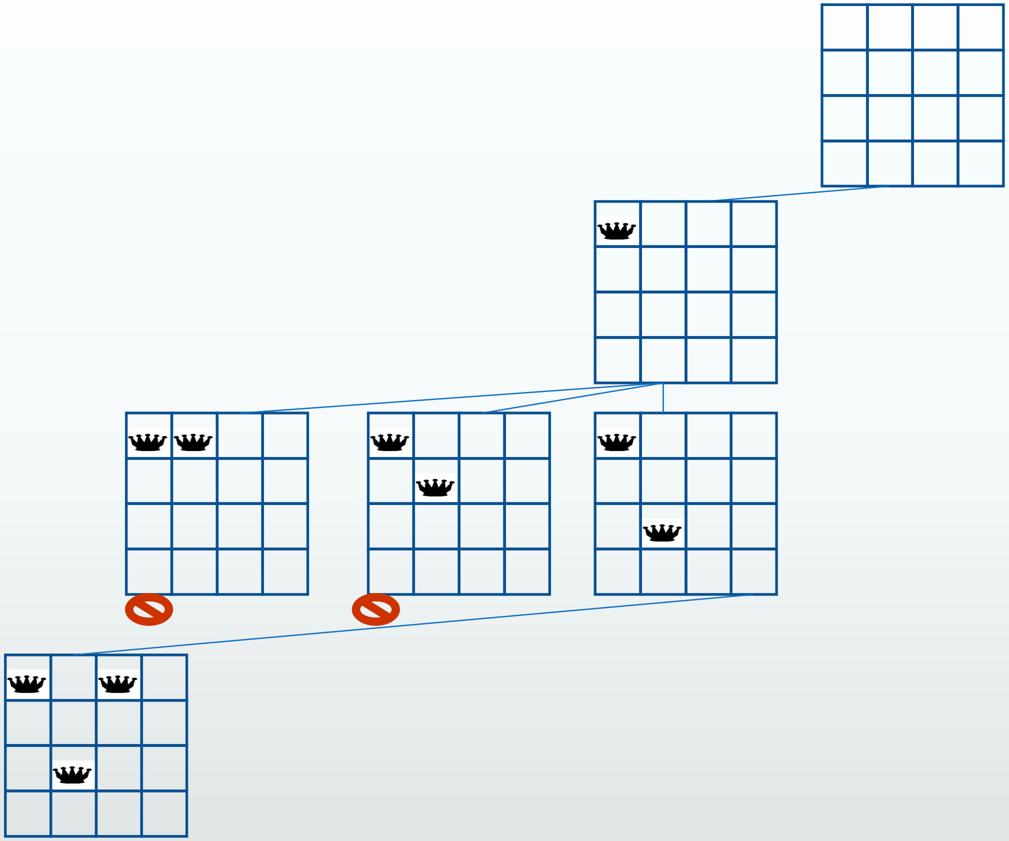


Exemple : résolution des 4 reines avec retour arrière



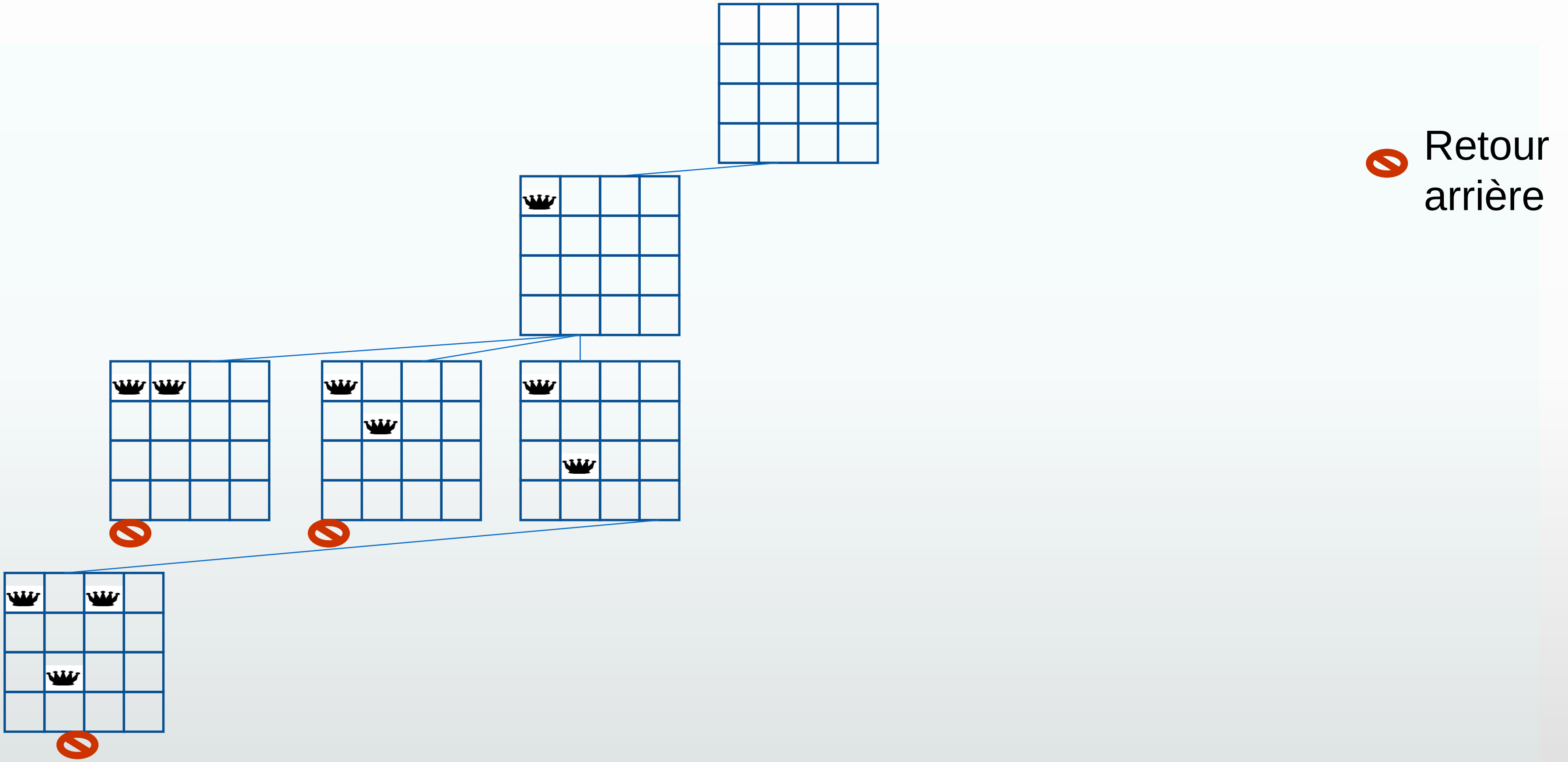
Retour arrière

Exemple : résolution des 4 reines avec retour arrière

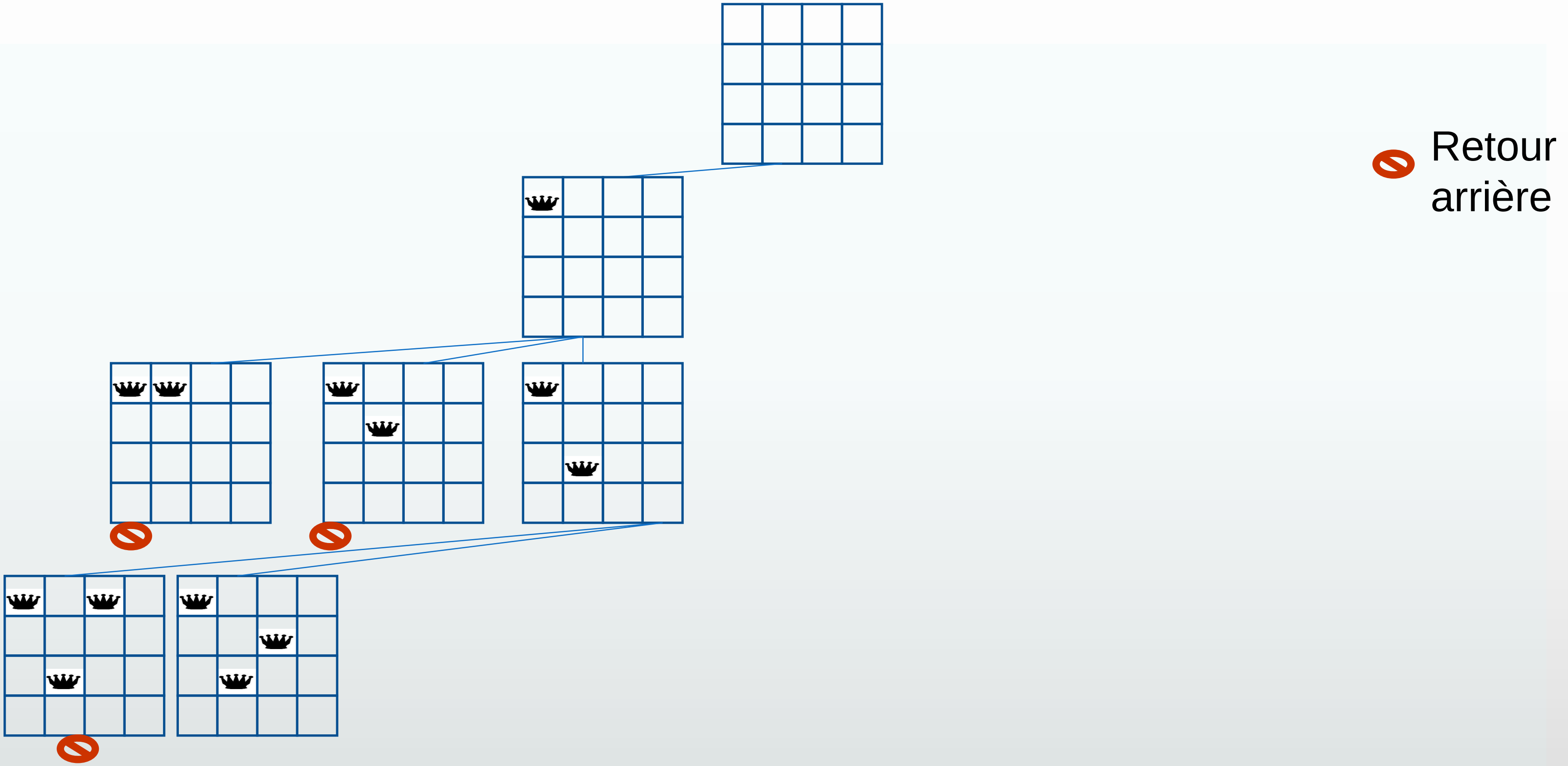


Retour arrière

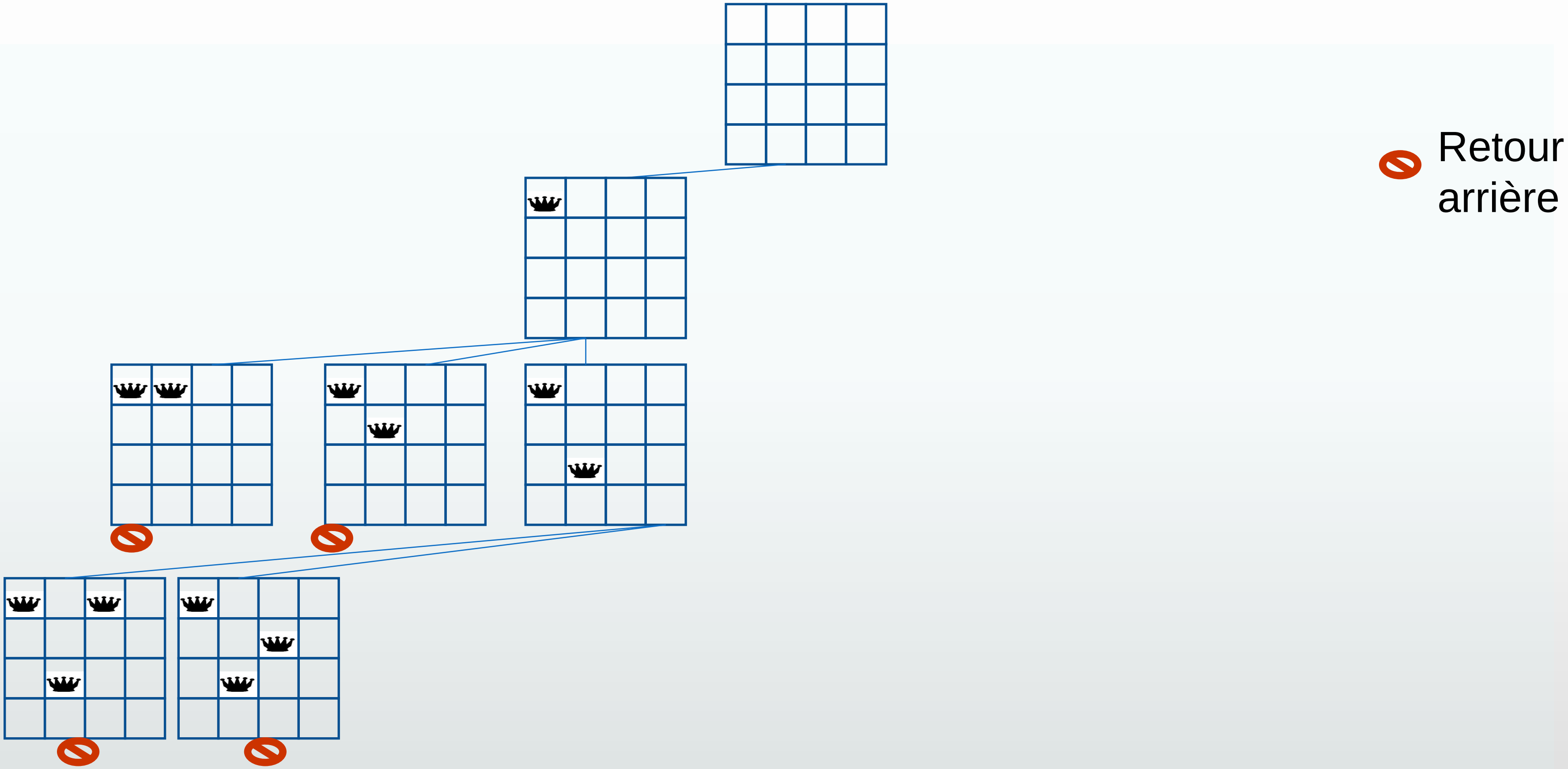
Exemple : résolution des 4 reines avec retour arrière



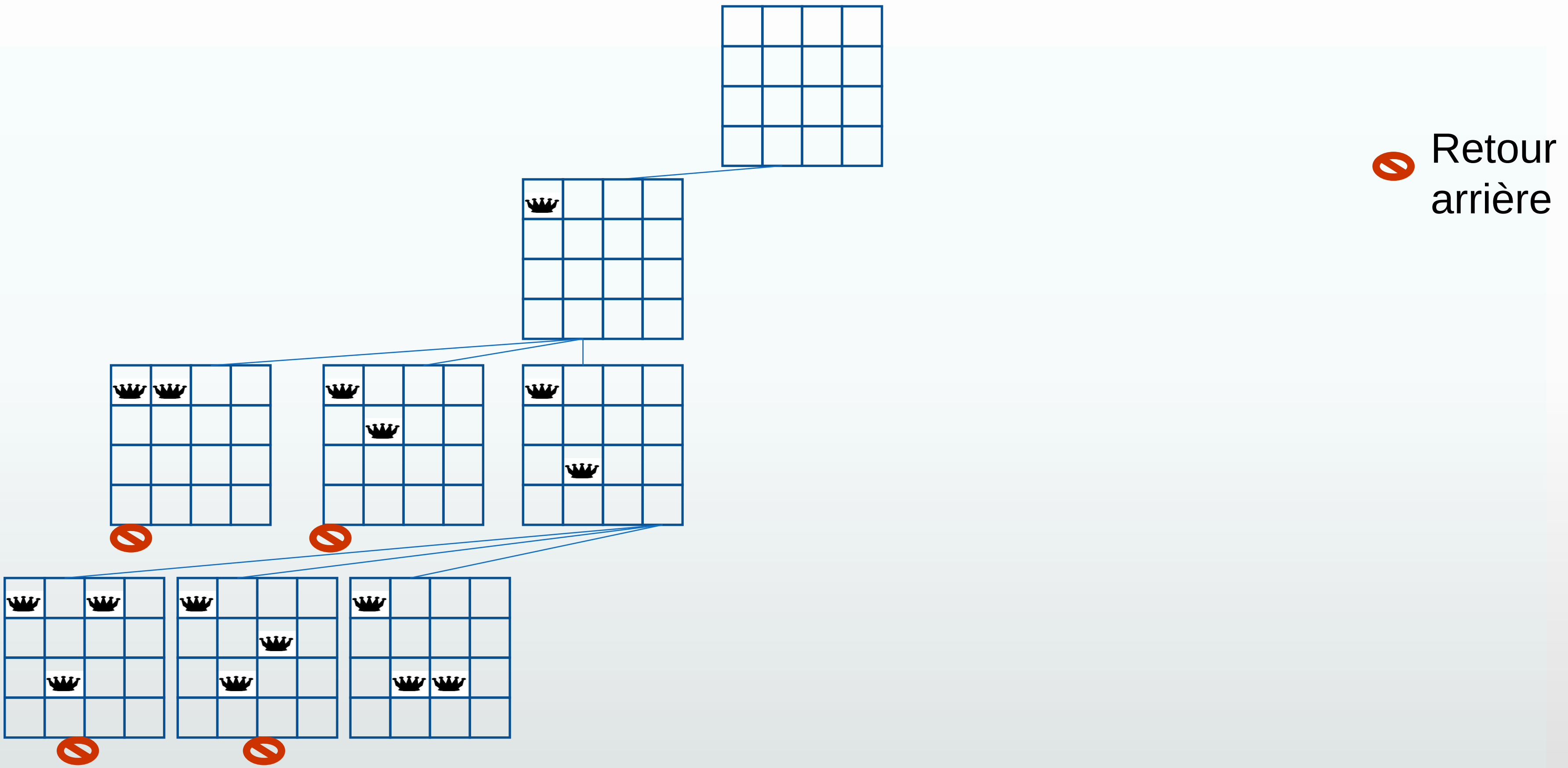
Exemple : résolution des 4 reines avec retour arrière



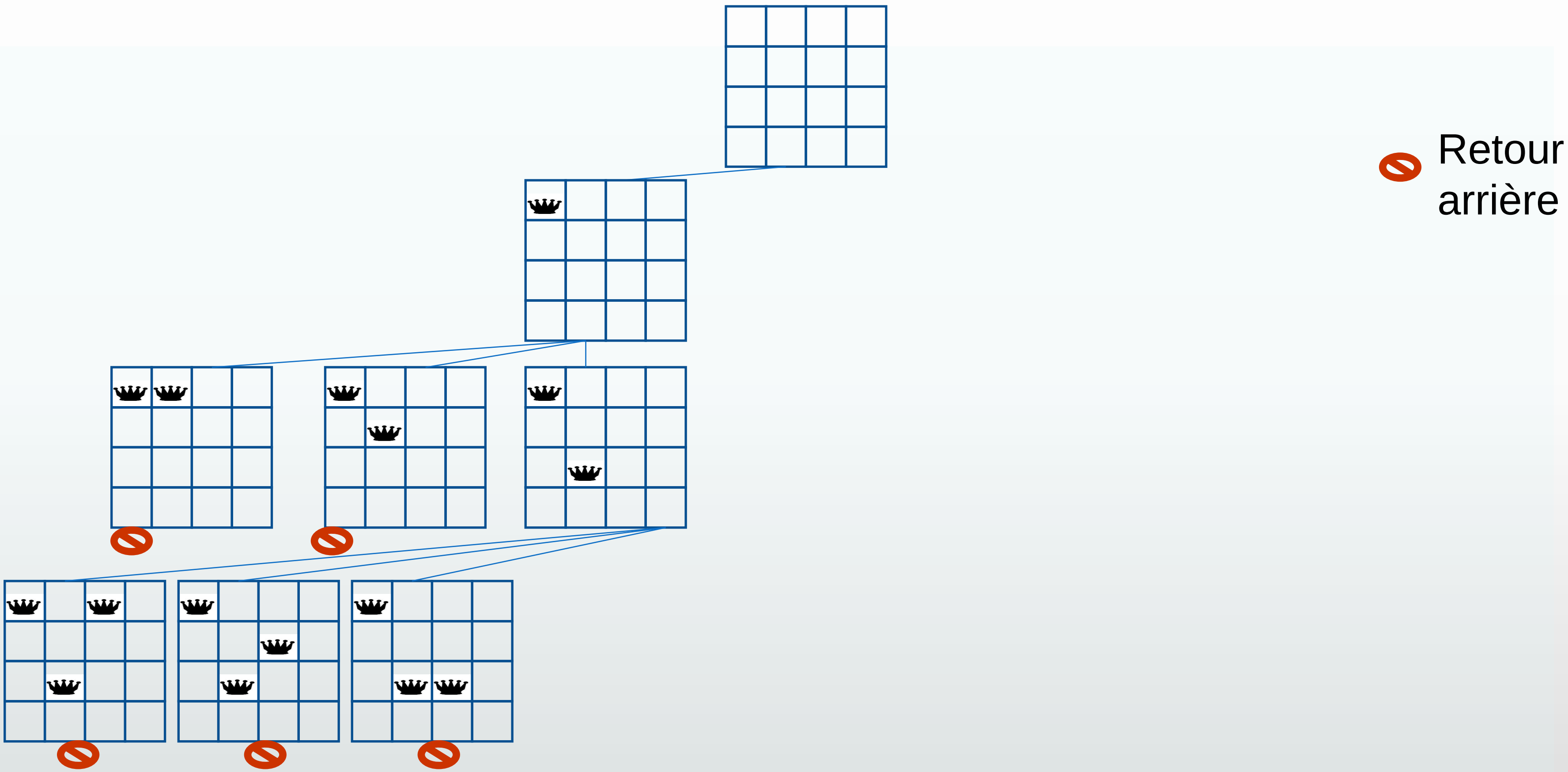
Exemple : résolution des 4 reines avec retour arrière



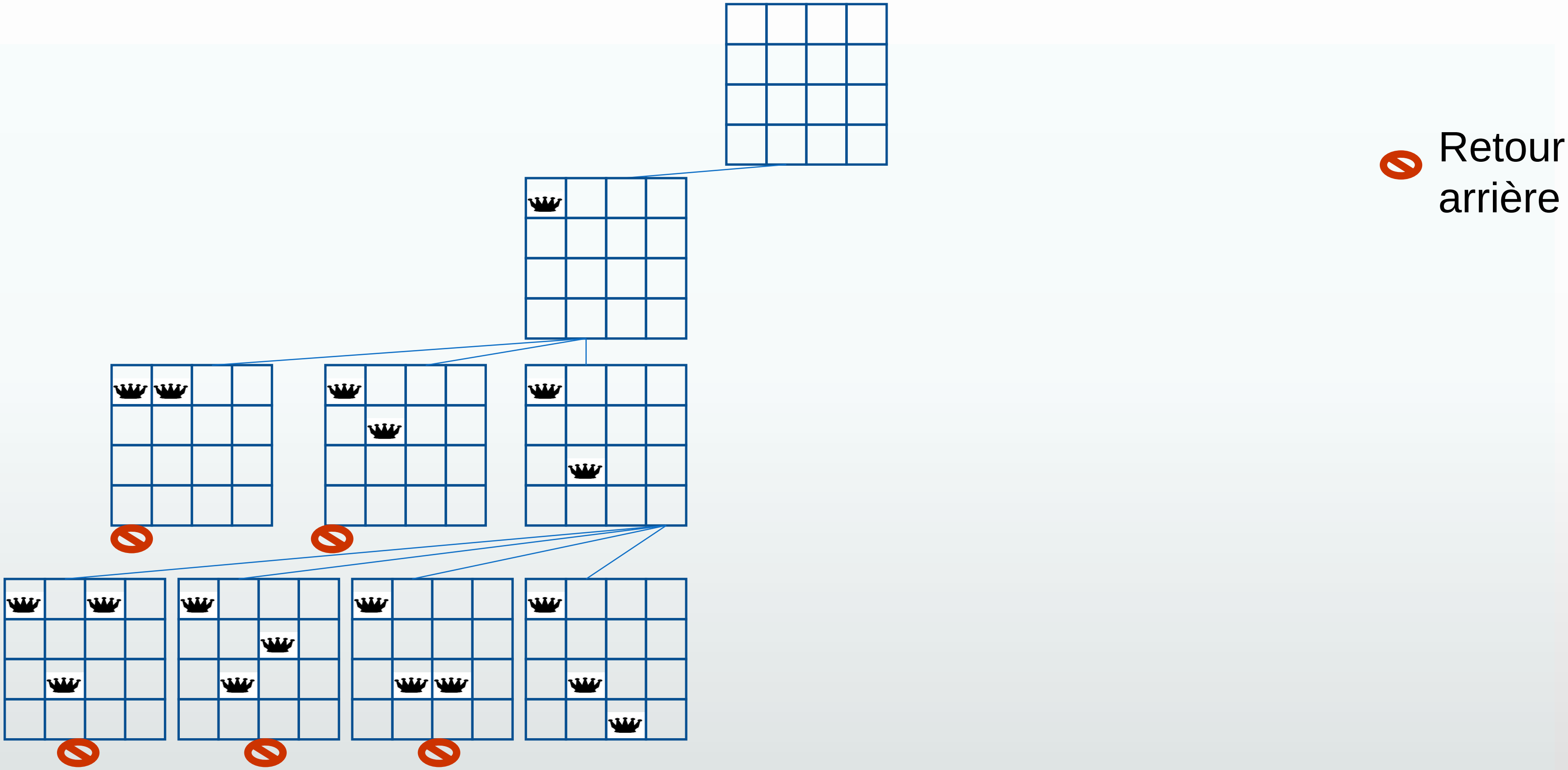
Exemple : résolution des 4 reines avec retour arrière



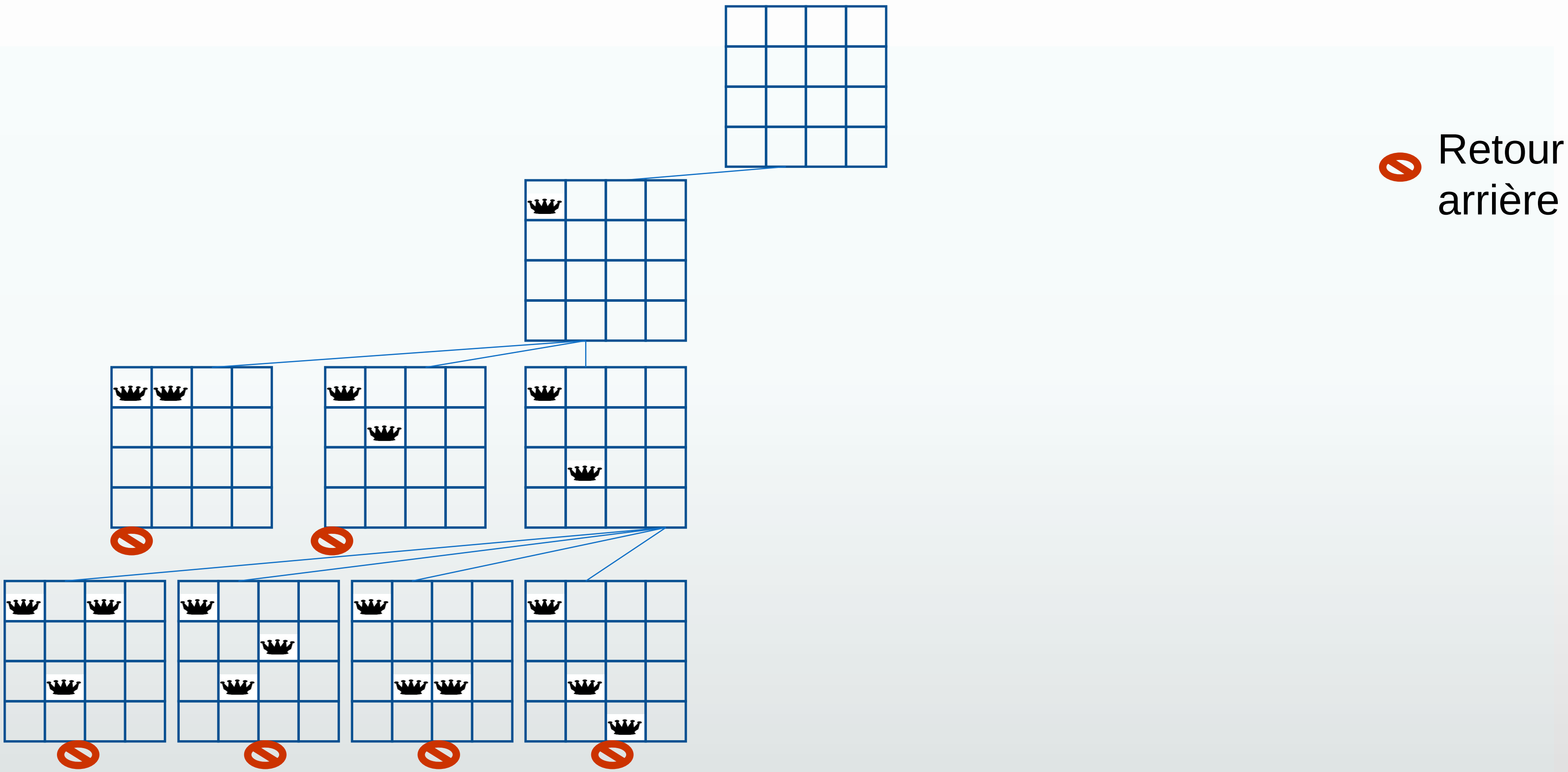
Exemple : résolution des 4 reines avec retour arrière



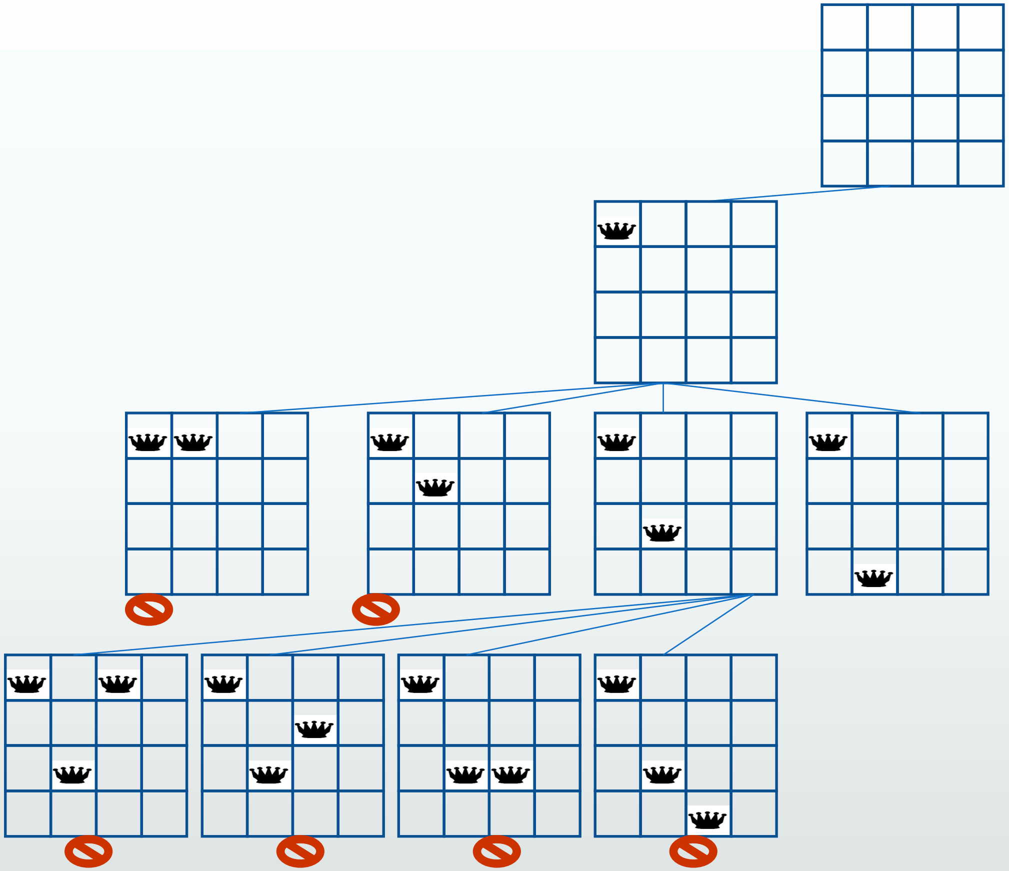
Exemple : résolution des 4 reines avec retour arrière



Exemple : résolution des 4 reines avec retour arrière

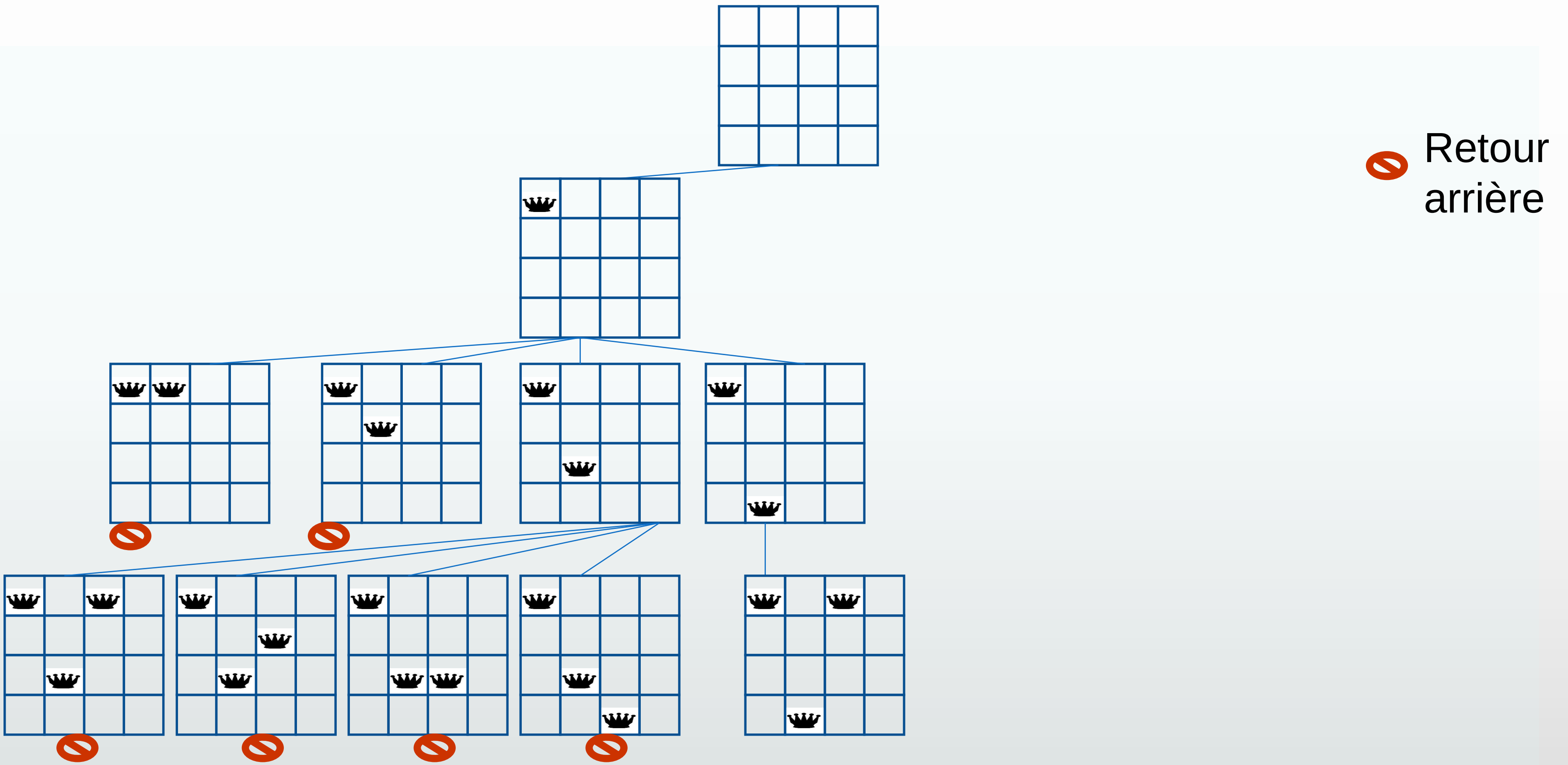


Exemple : résolution des 4 reines avec retour arrière

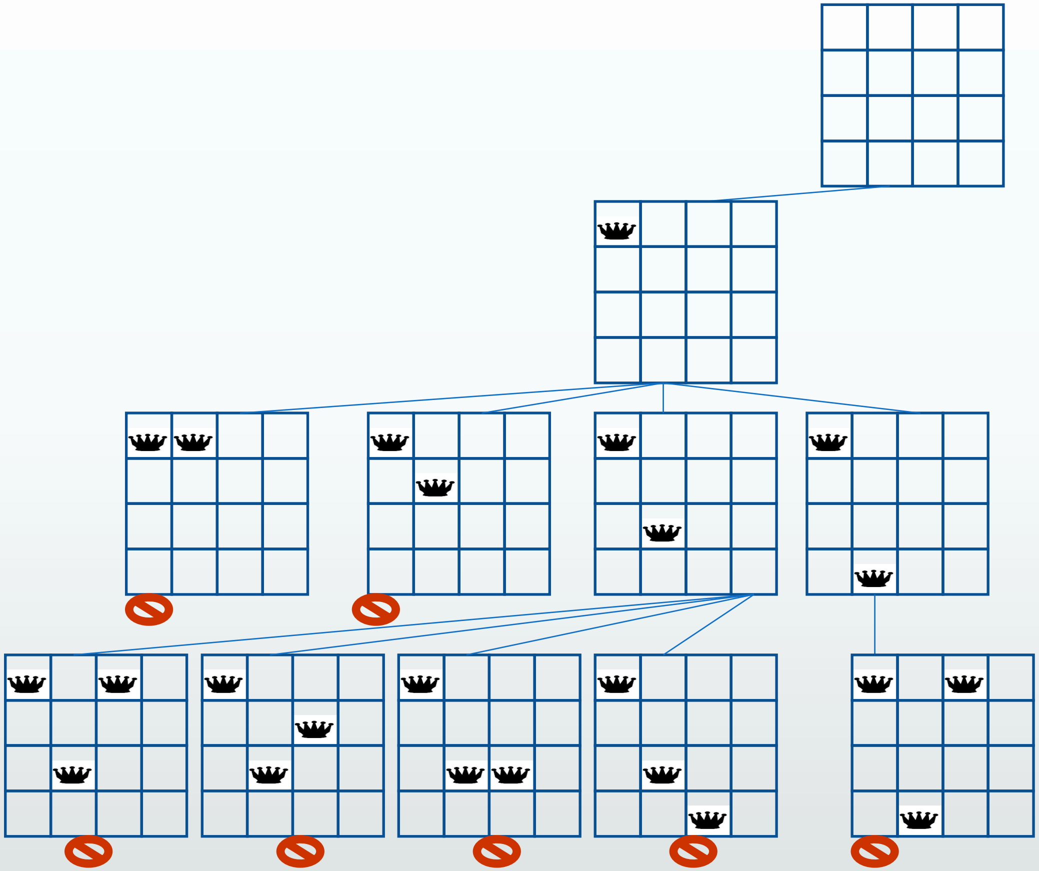



Retour arrière

Exemple : résolution des 4 reines avec retour arrière

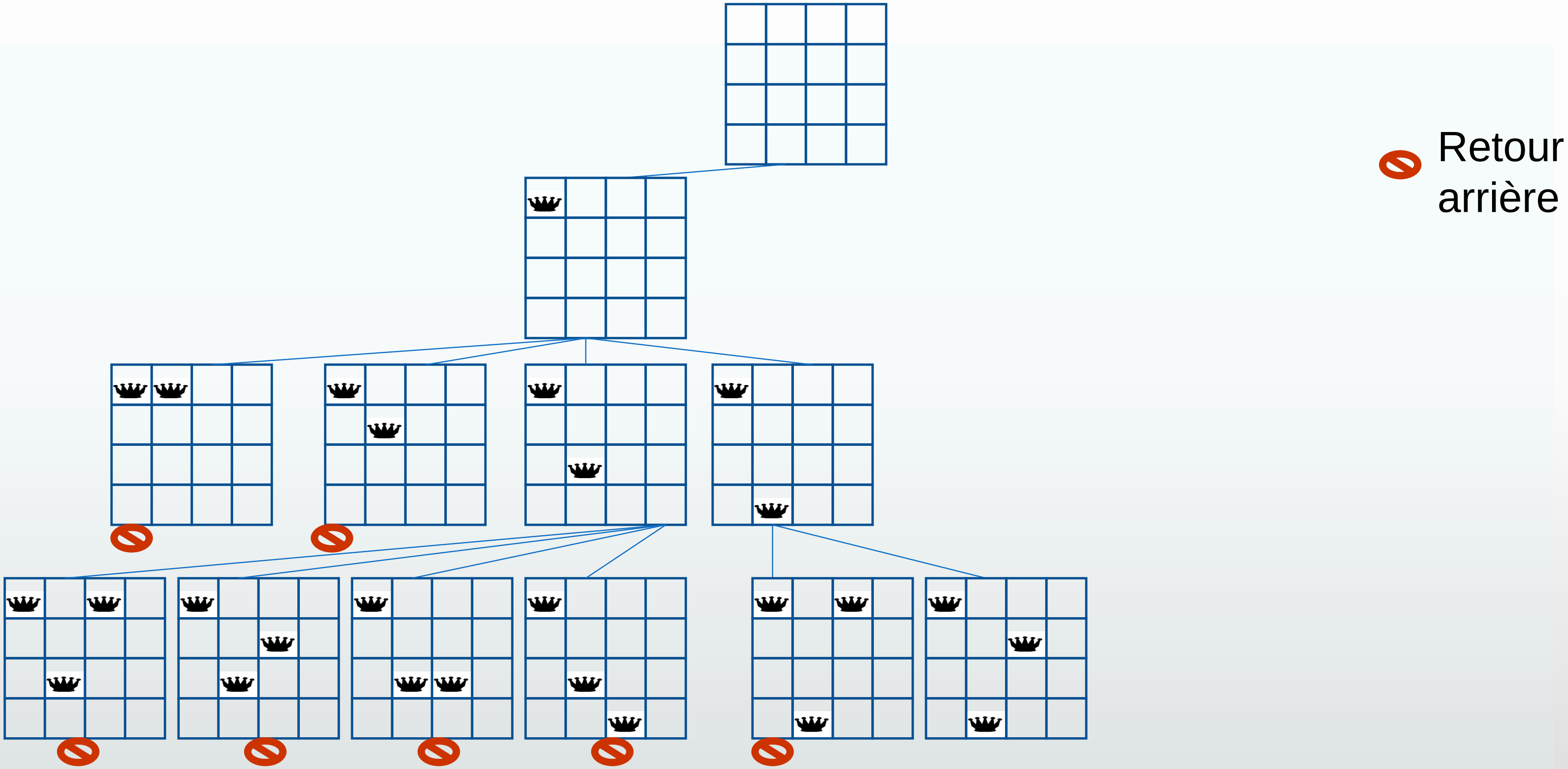


Exemple : résolution des 4 reines avec retour arrière

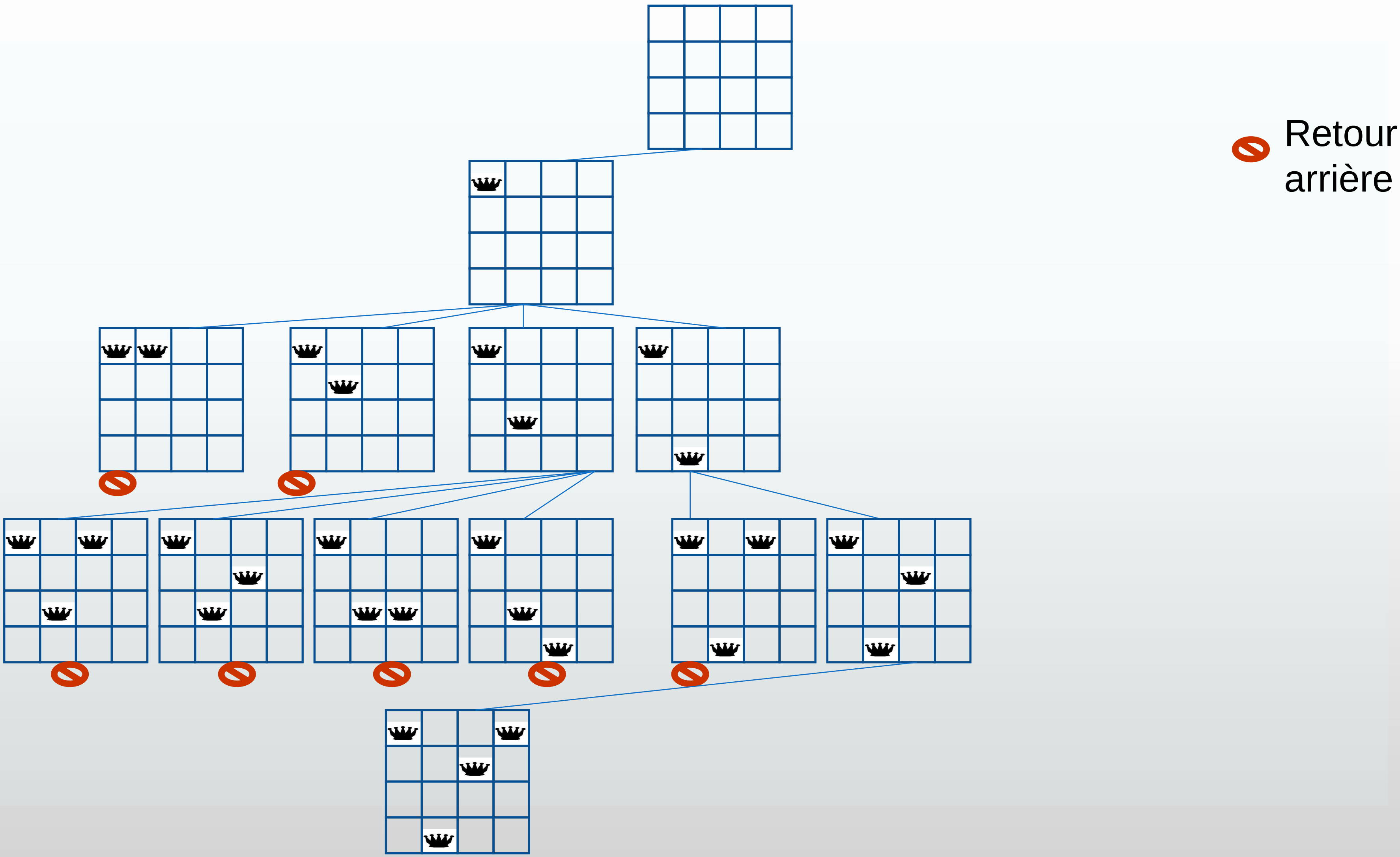


Retour arrière

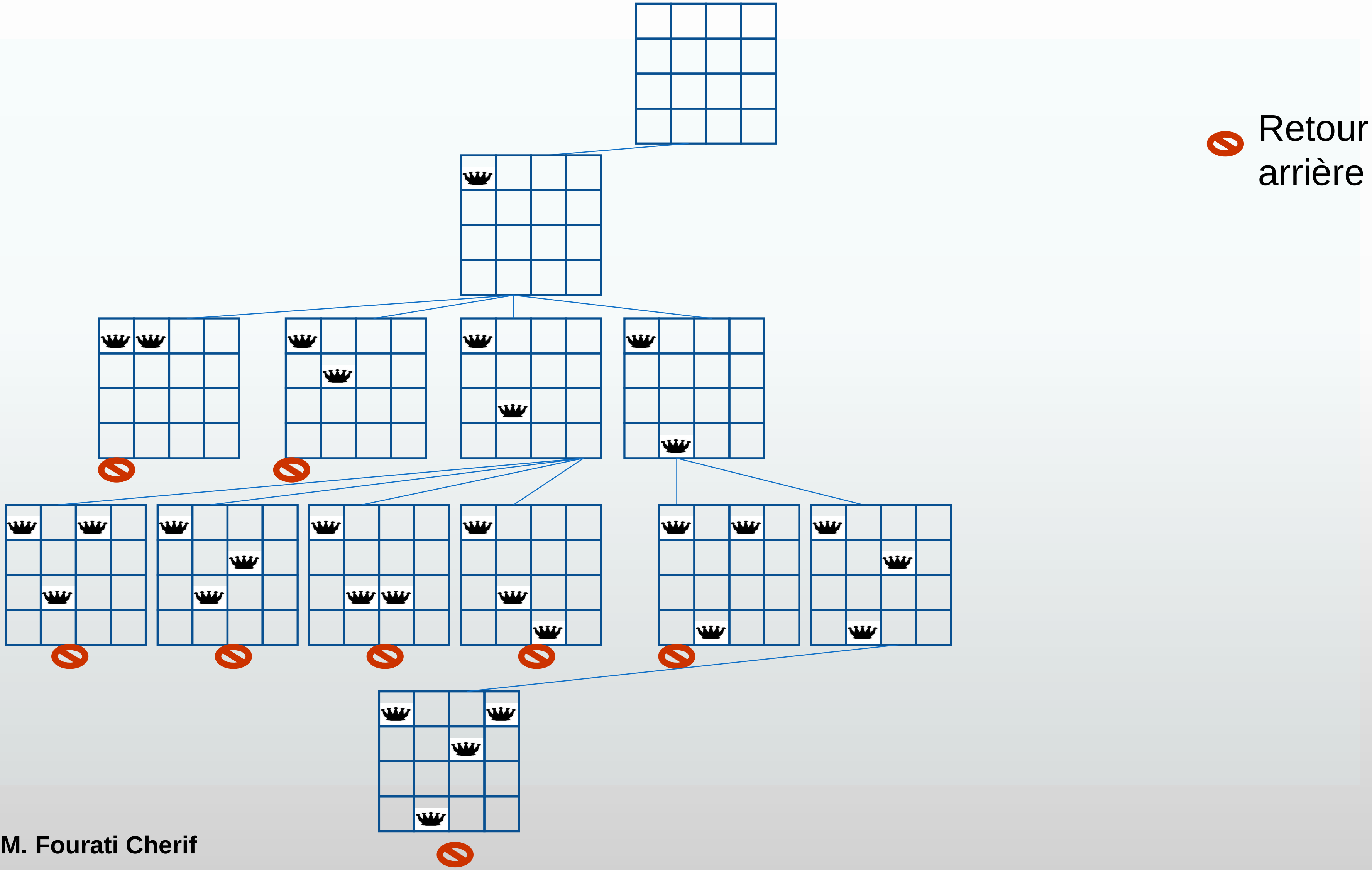
Exemple : résolution des 4 reines avec retour arrière



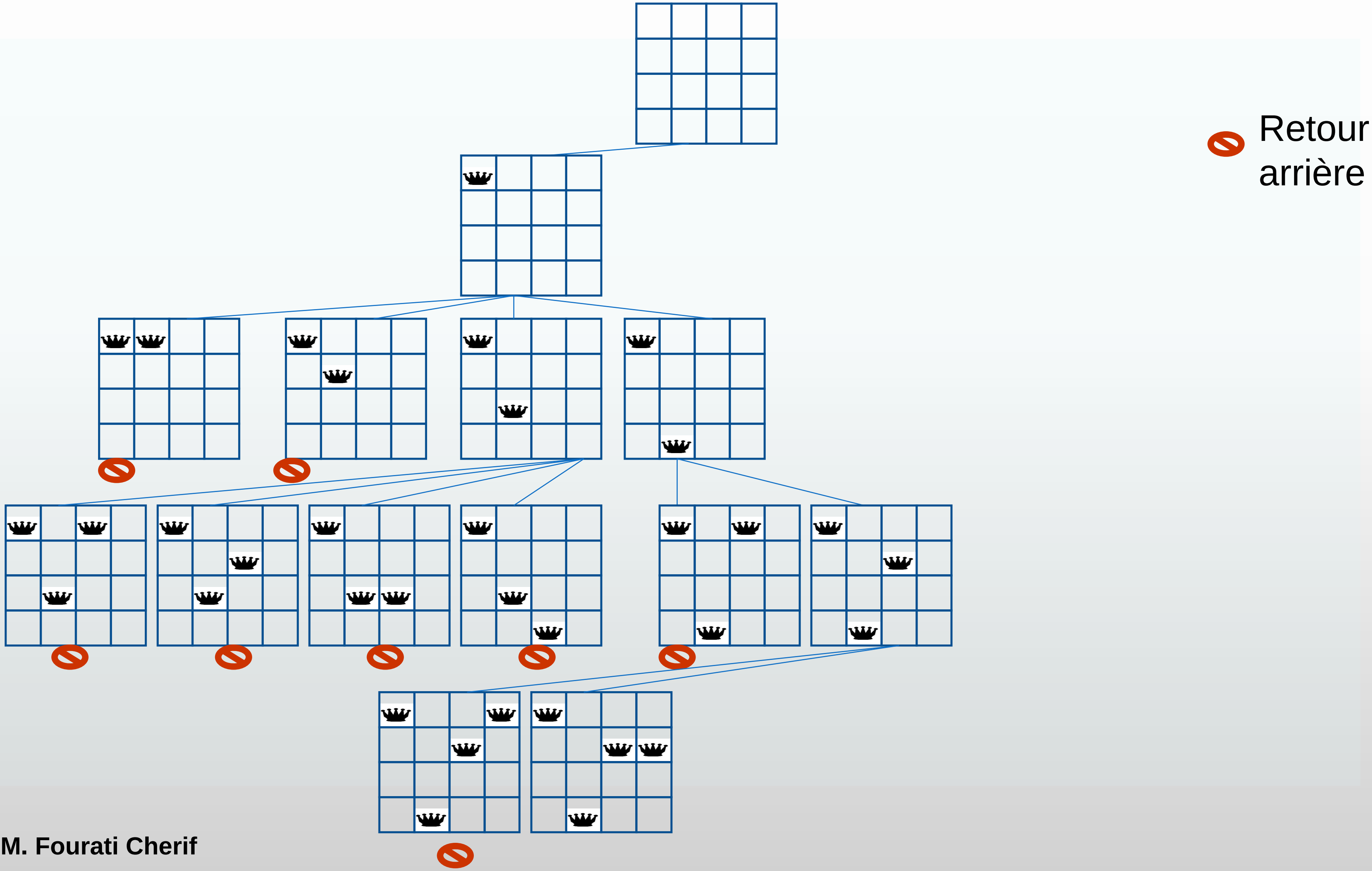
Exemple : résolution des 4 reines avec retour arrière



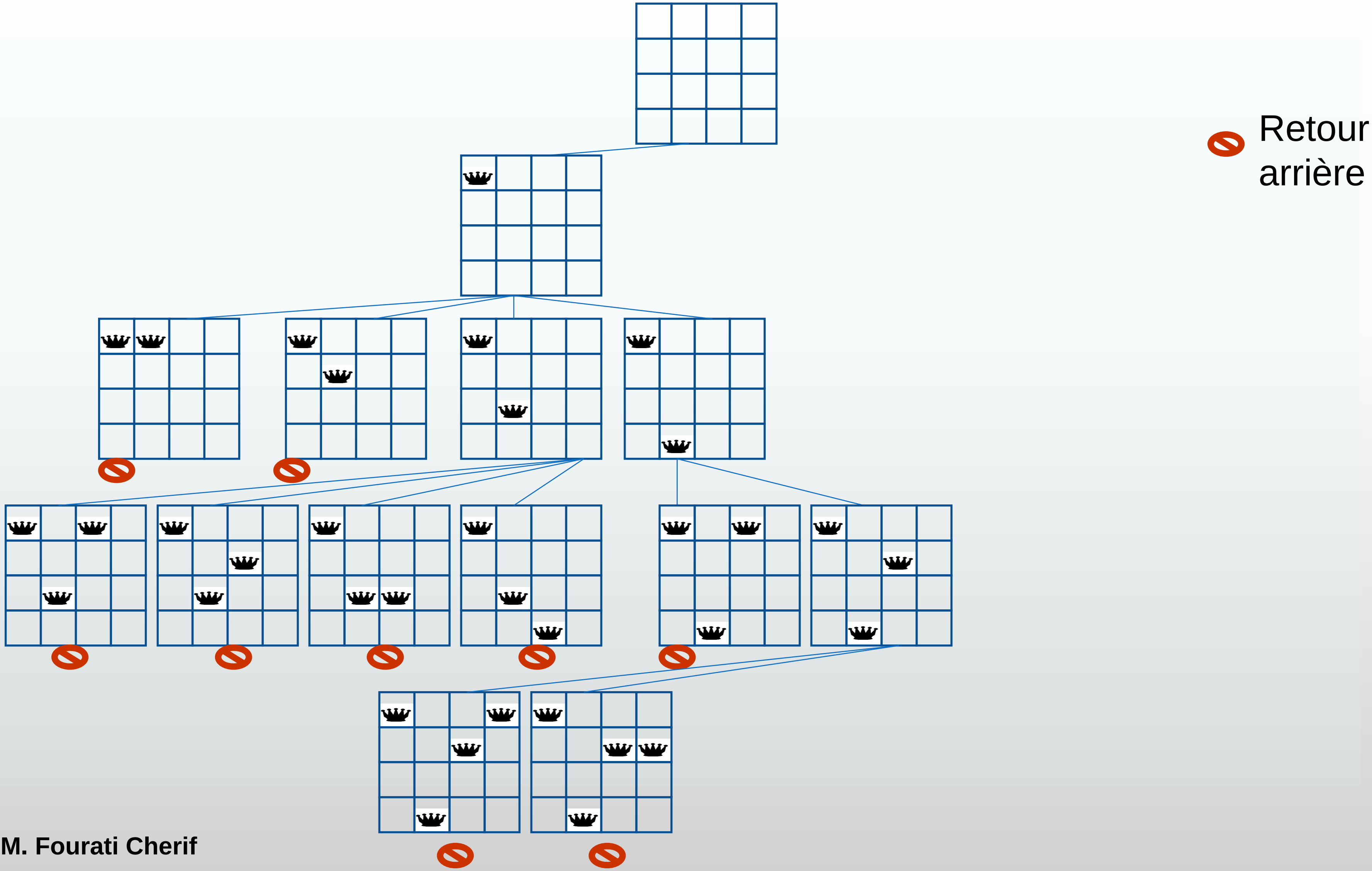
Exemple : résolution des 4 reines avec retour arrière



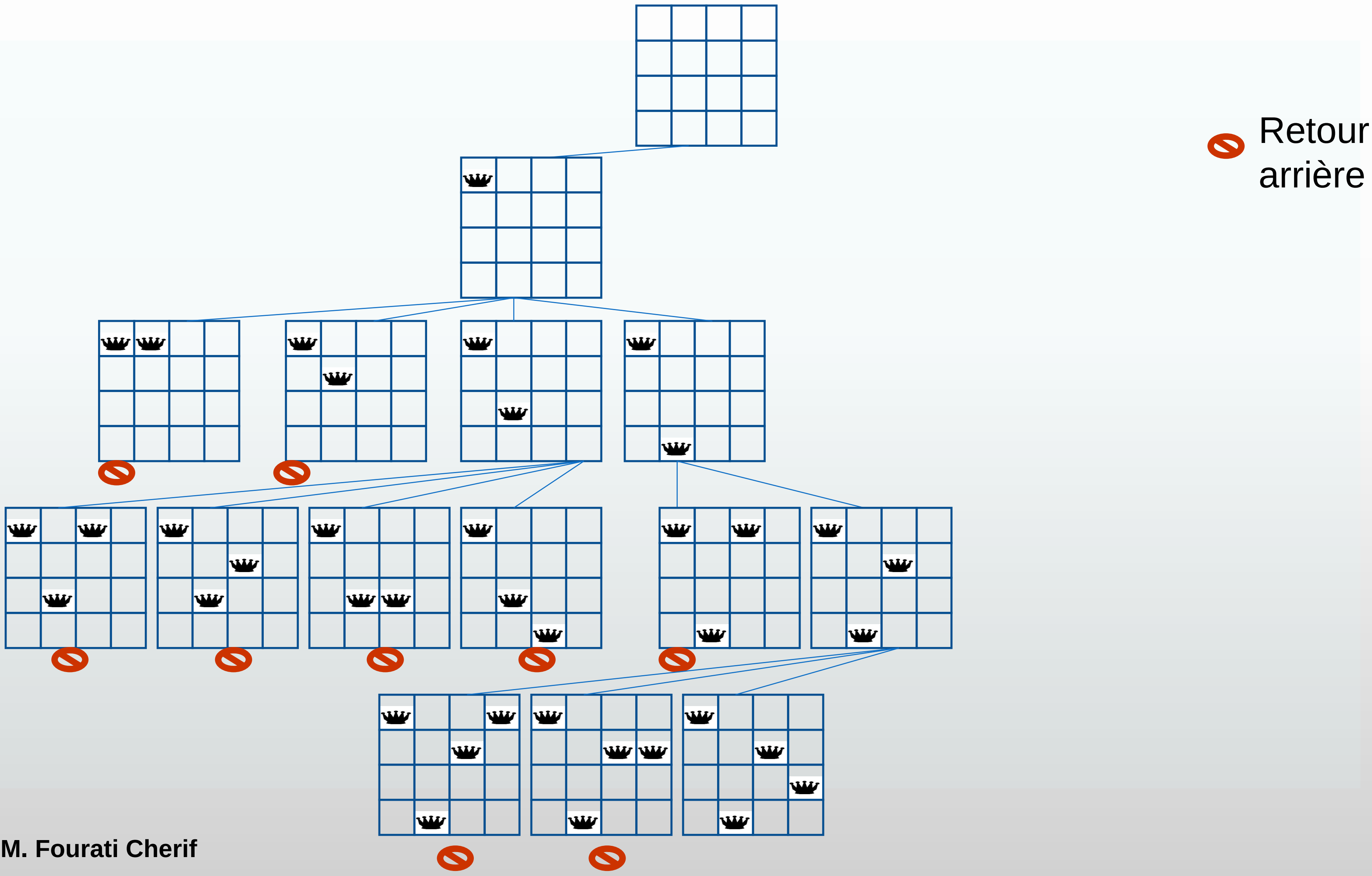
Exemple : résolution des 4 reines avec retour arrière



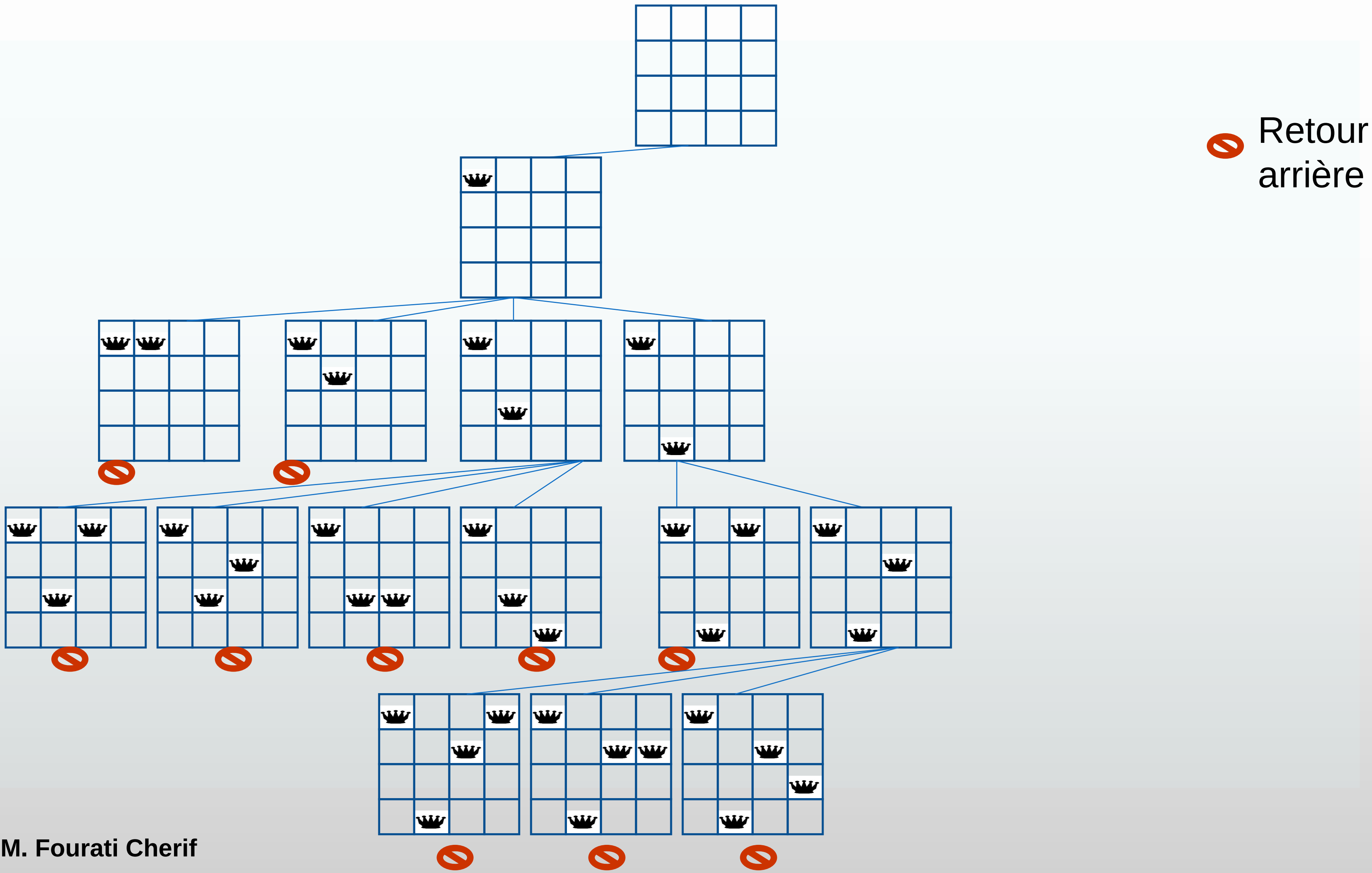
Exemple : résolution des 4 reines avec retour arrière



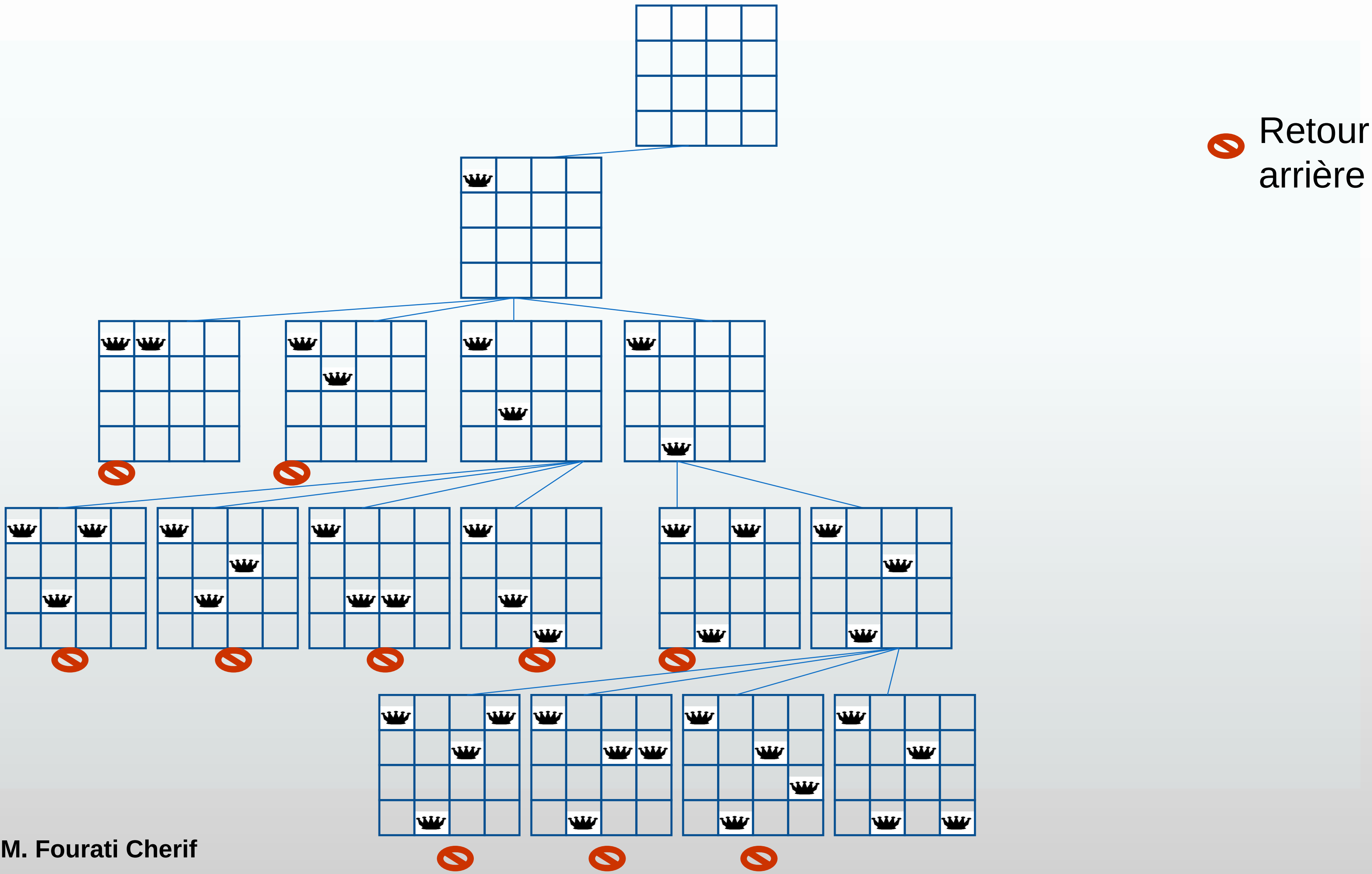
Exemple : résolution des 4 reines avec retour arrière



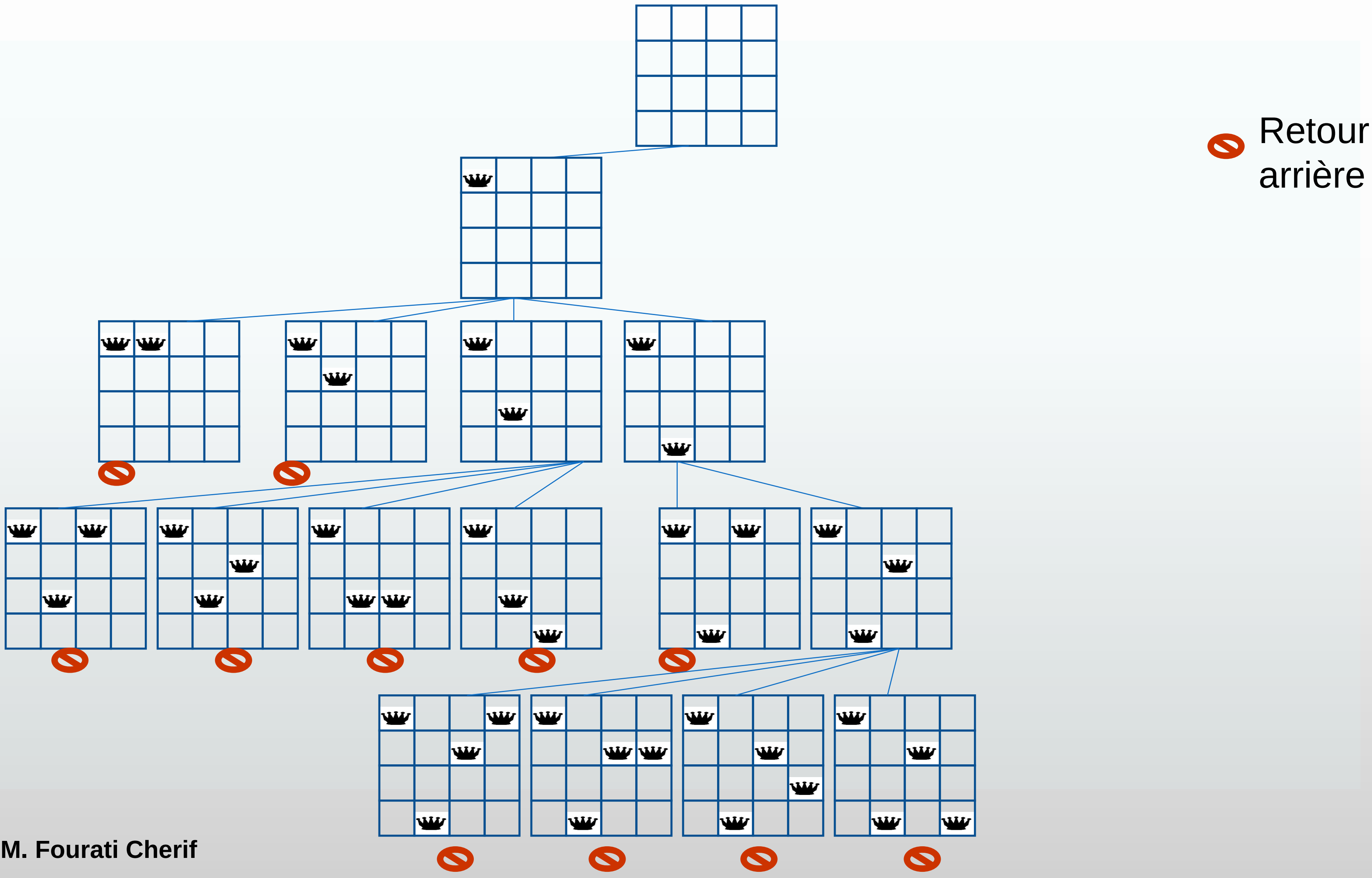
Exemple : résolution des 4 reines avec retour arrière



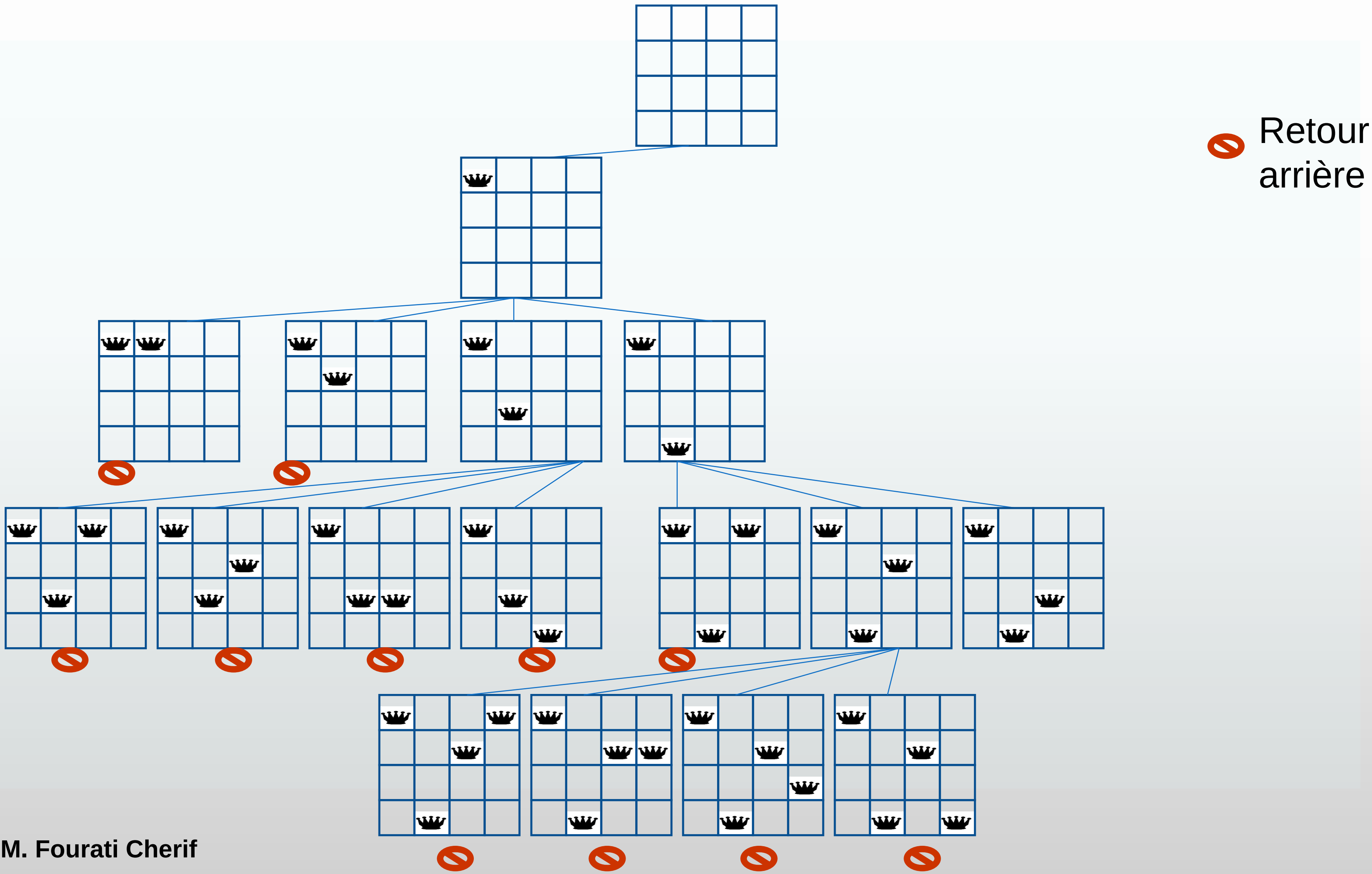
Exemple : résolution des 4 reines avec retour arrière



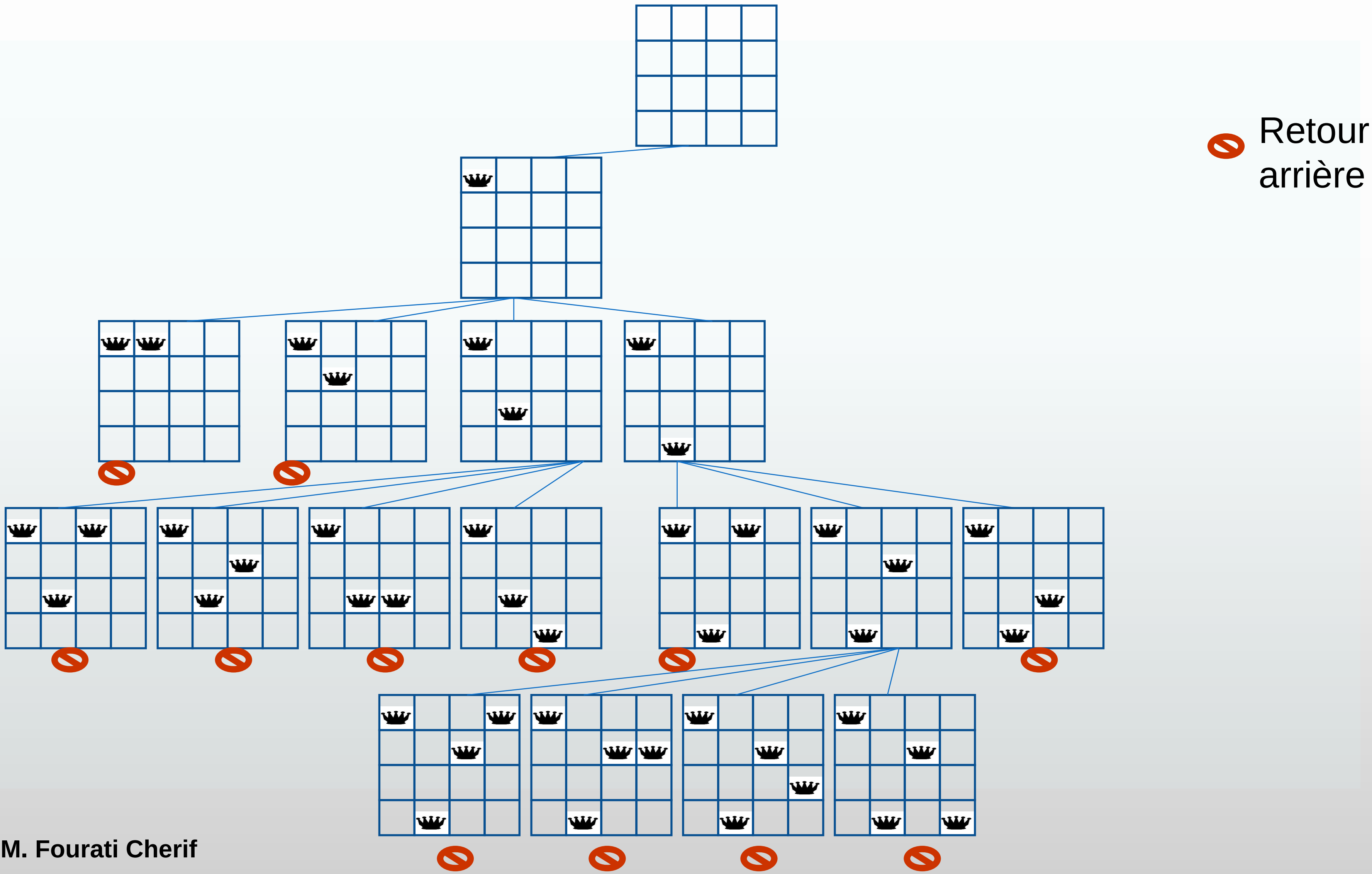
Exemple : résolution des 4 reines avec retour arrière



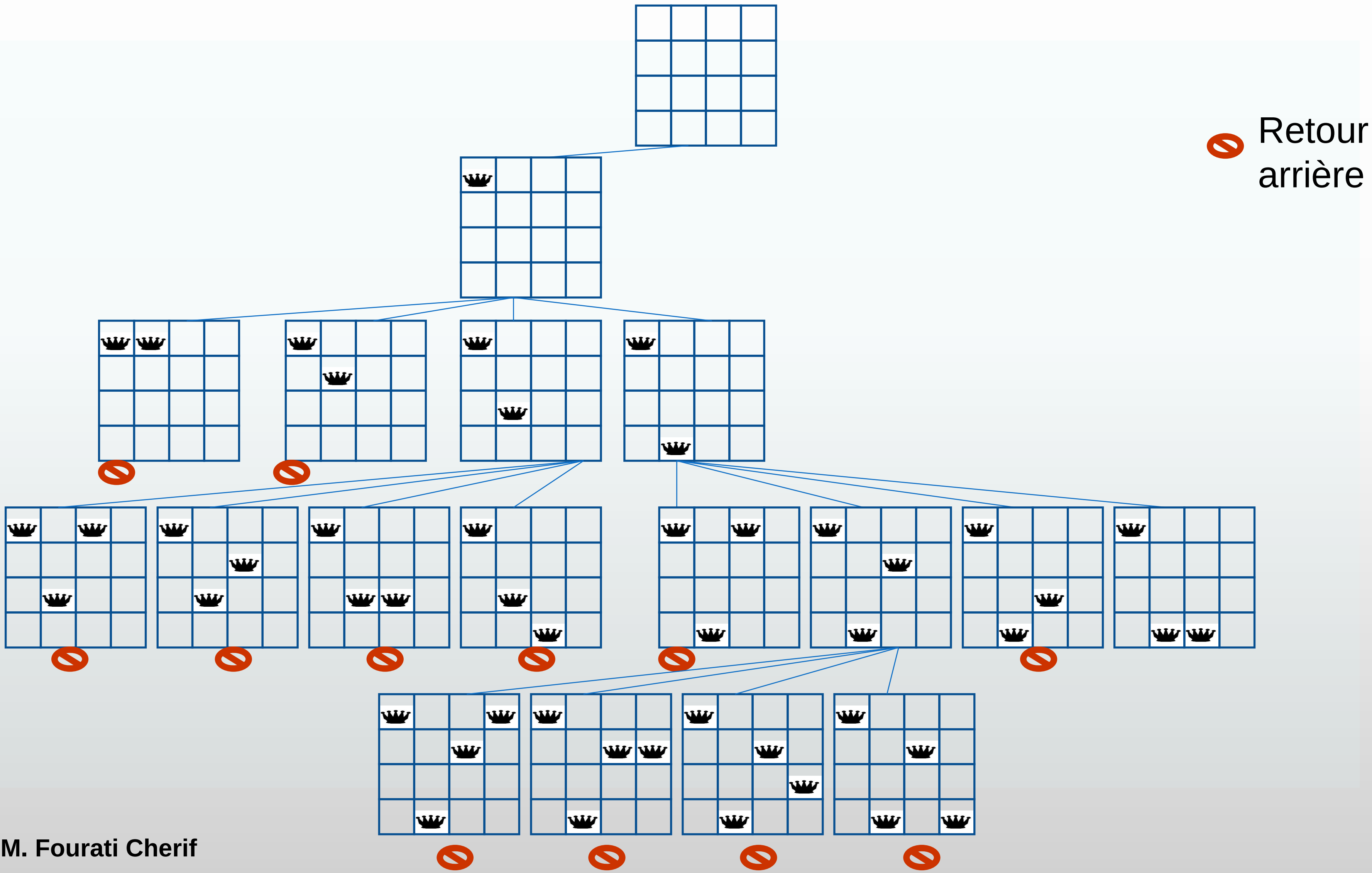
Exemple : résolution des 4 reines avec retour arrière



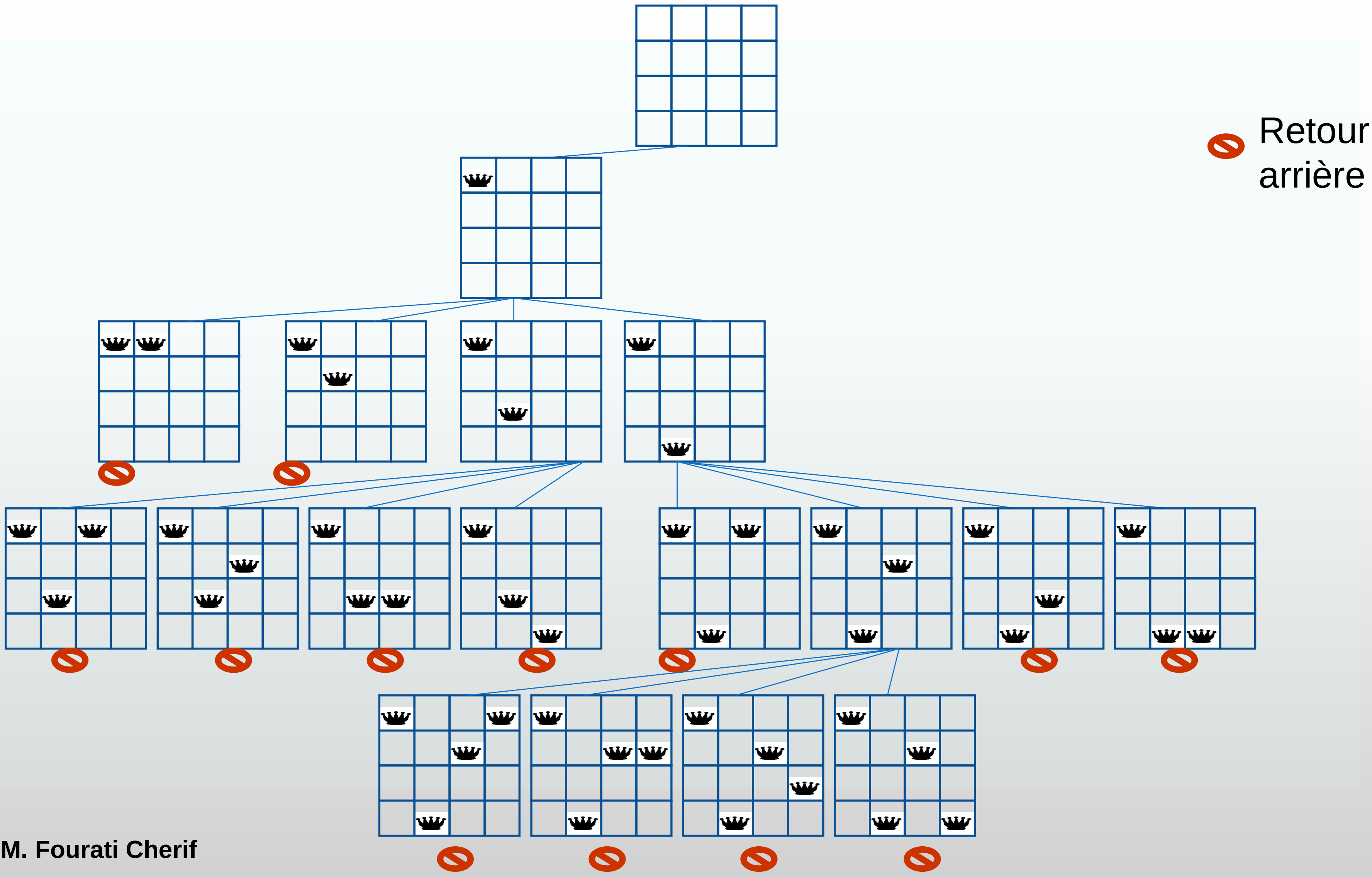
Exemple : résolution des 4 reines avec retour arrière



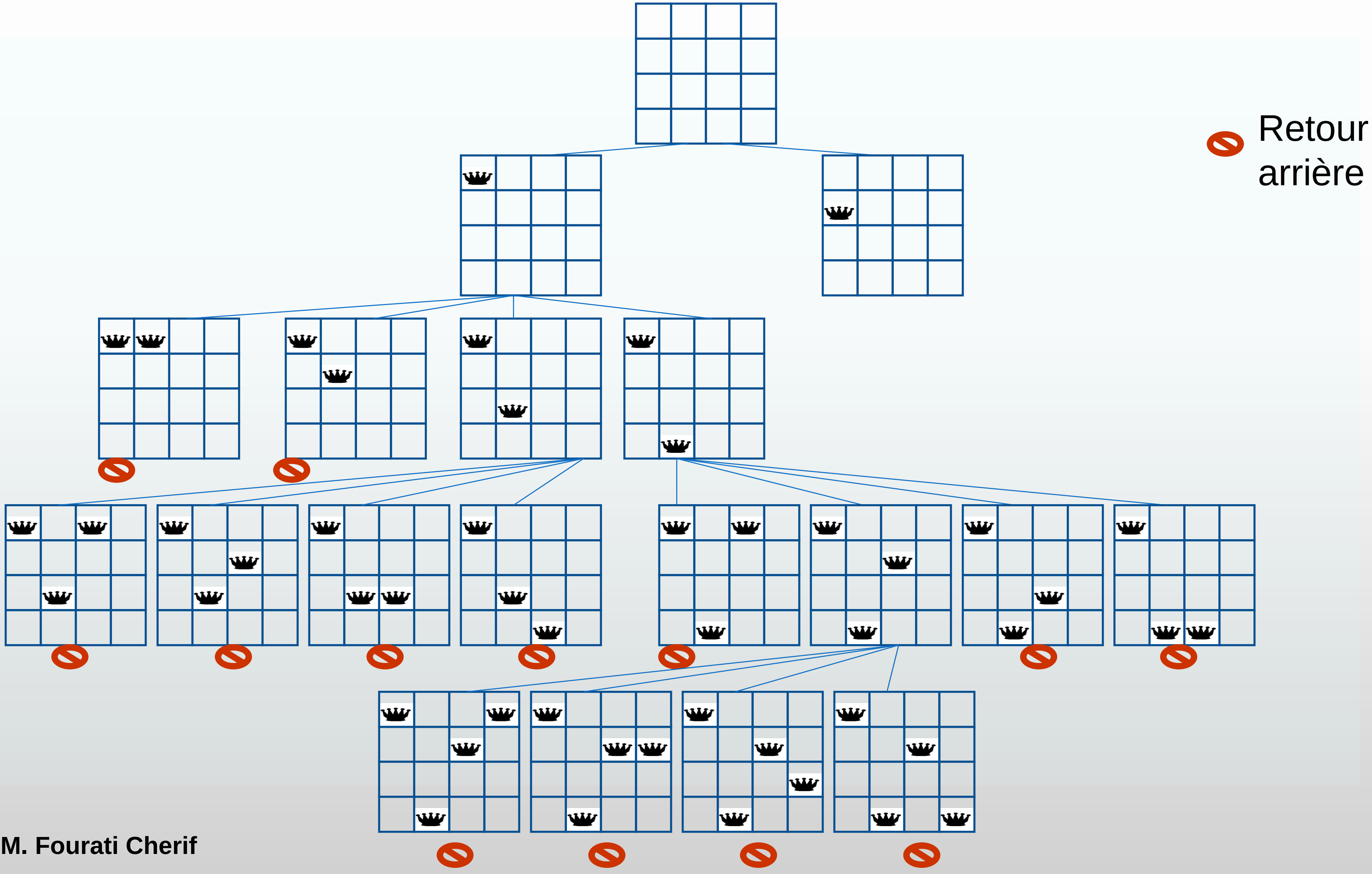
Exemple : résolution des 4 reines avec retour arrière



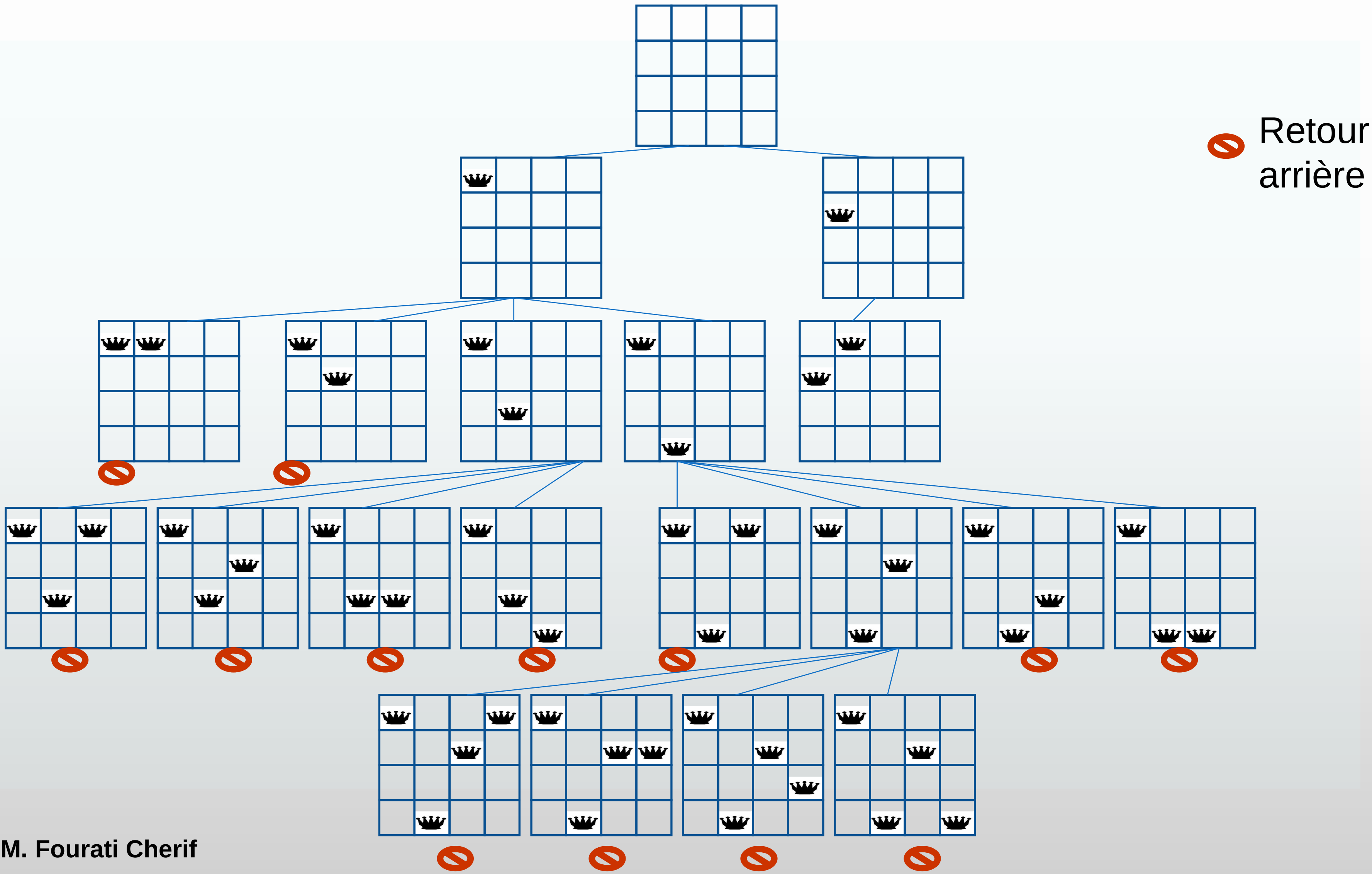
Exemple : résolution des 4 reines avec retour arrière



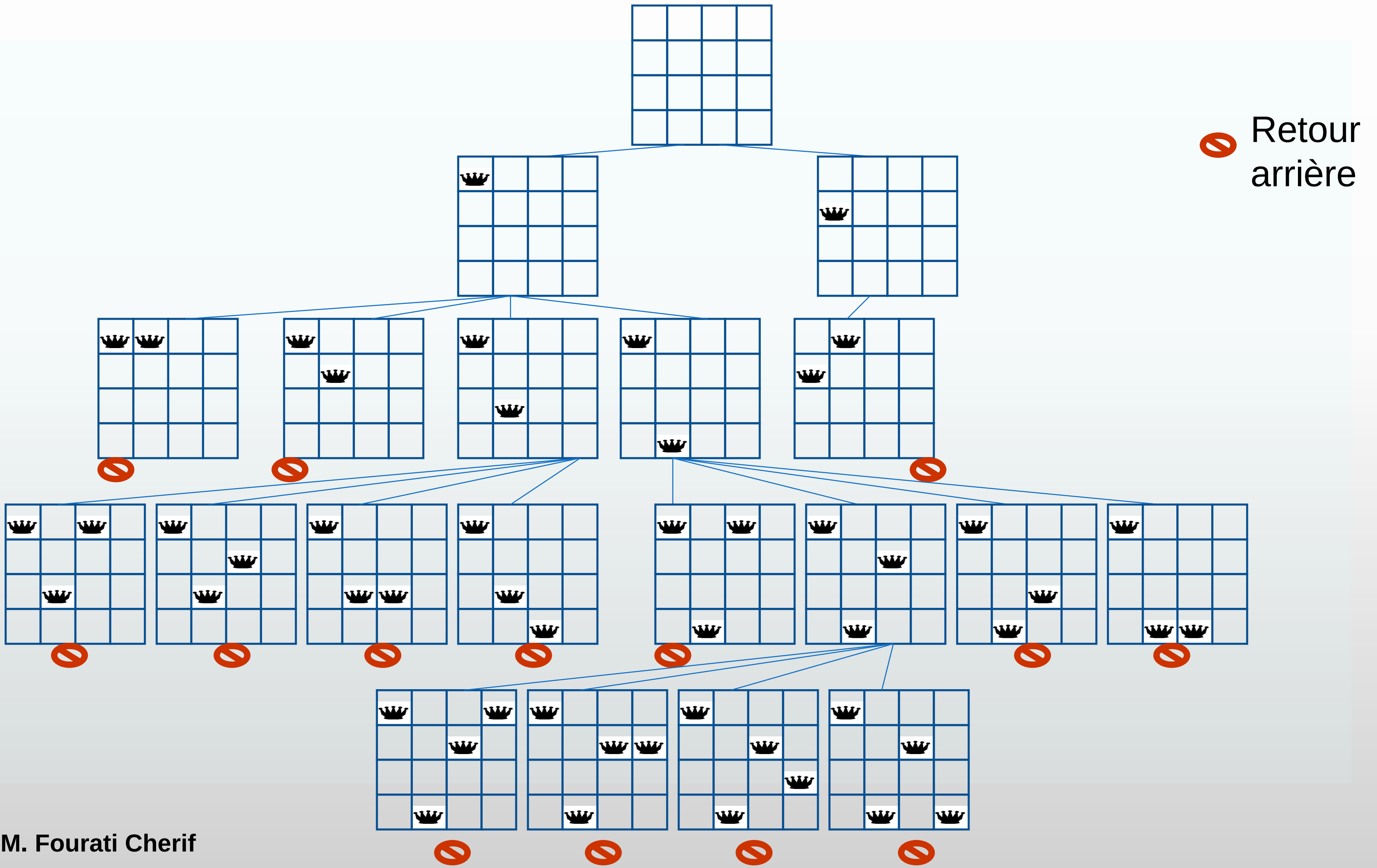
Exemple : résolution des 4 reines avec retour arrière



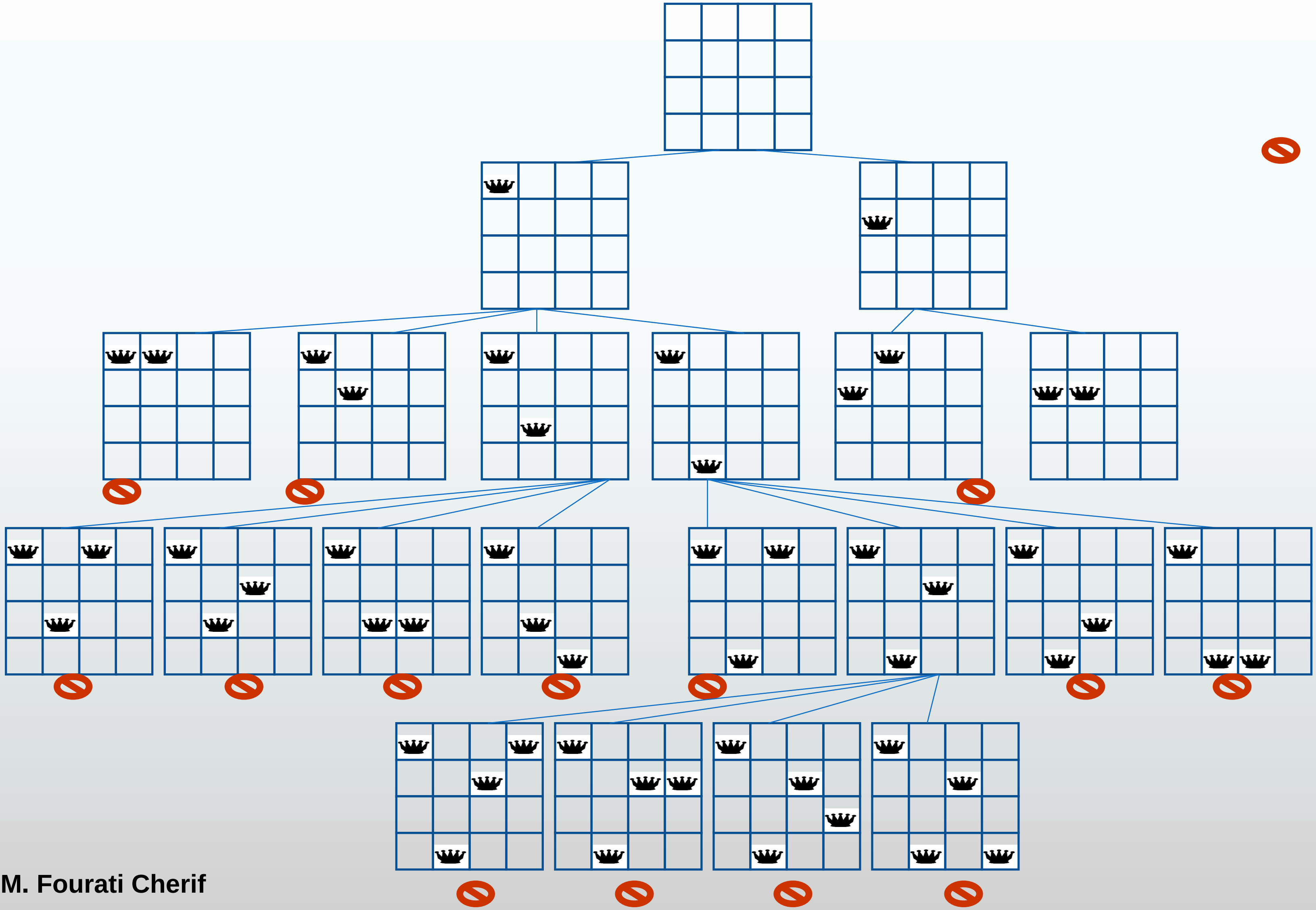
Exemple : résolution des 4 reines avec retour arrière



Exemple : résolution des 4 reines avec retour arrière

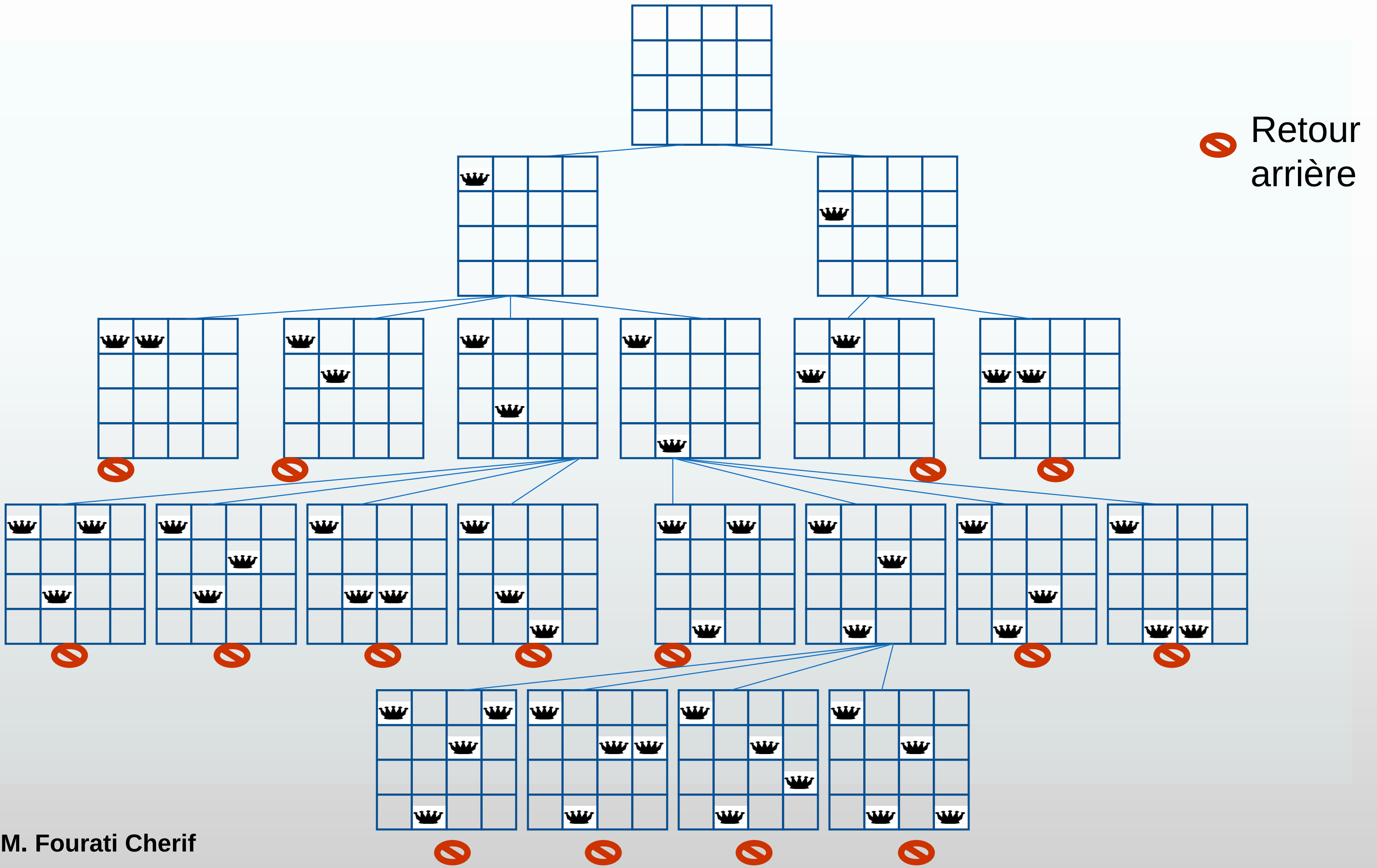


Exemple : résolution des 4 reines avec retour arrière

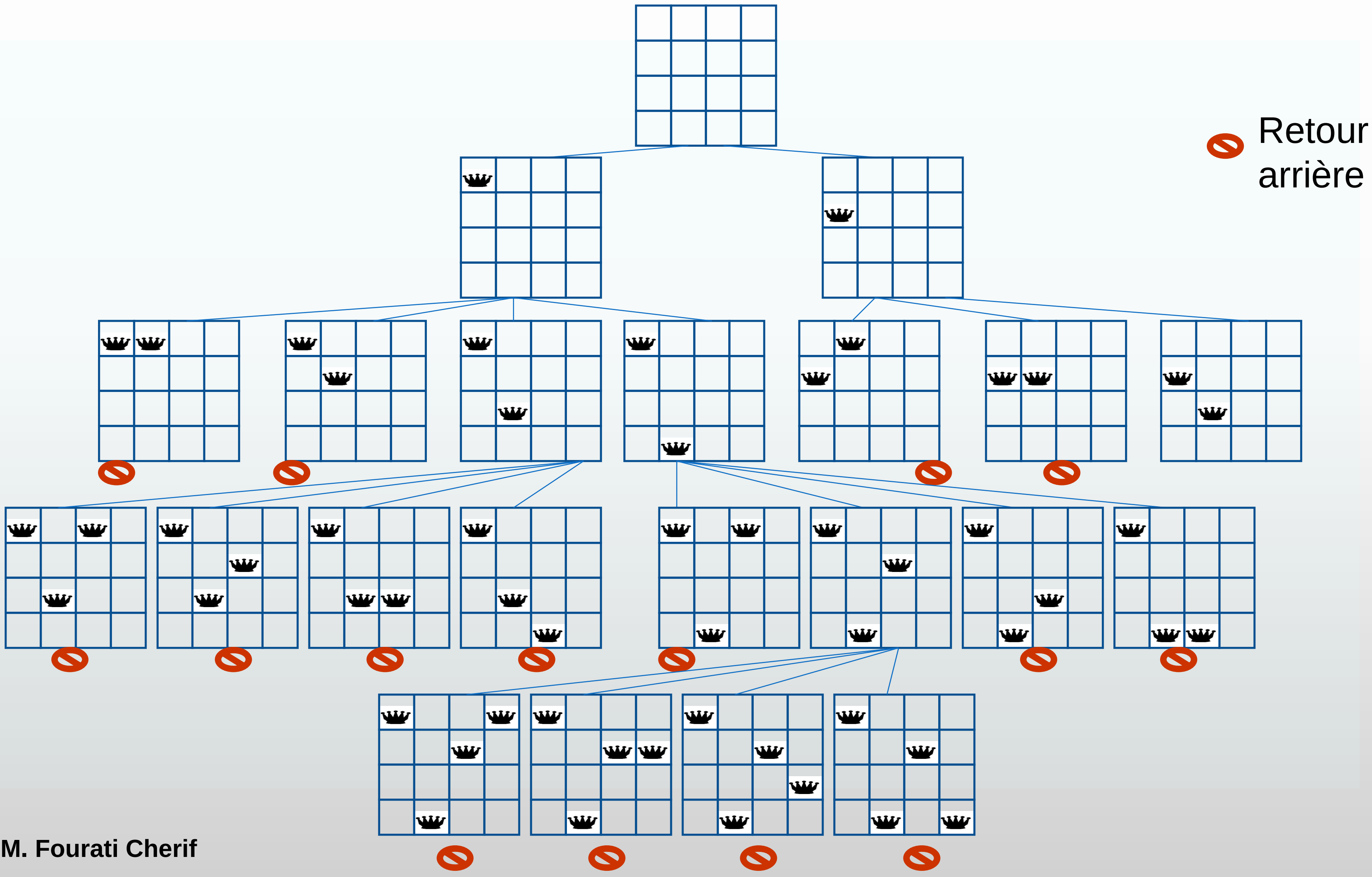


Retour arrière

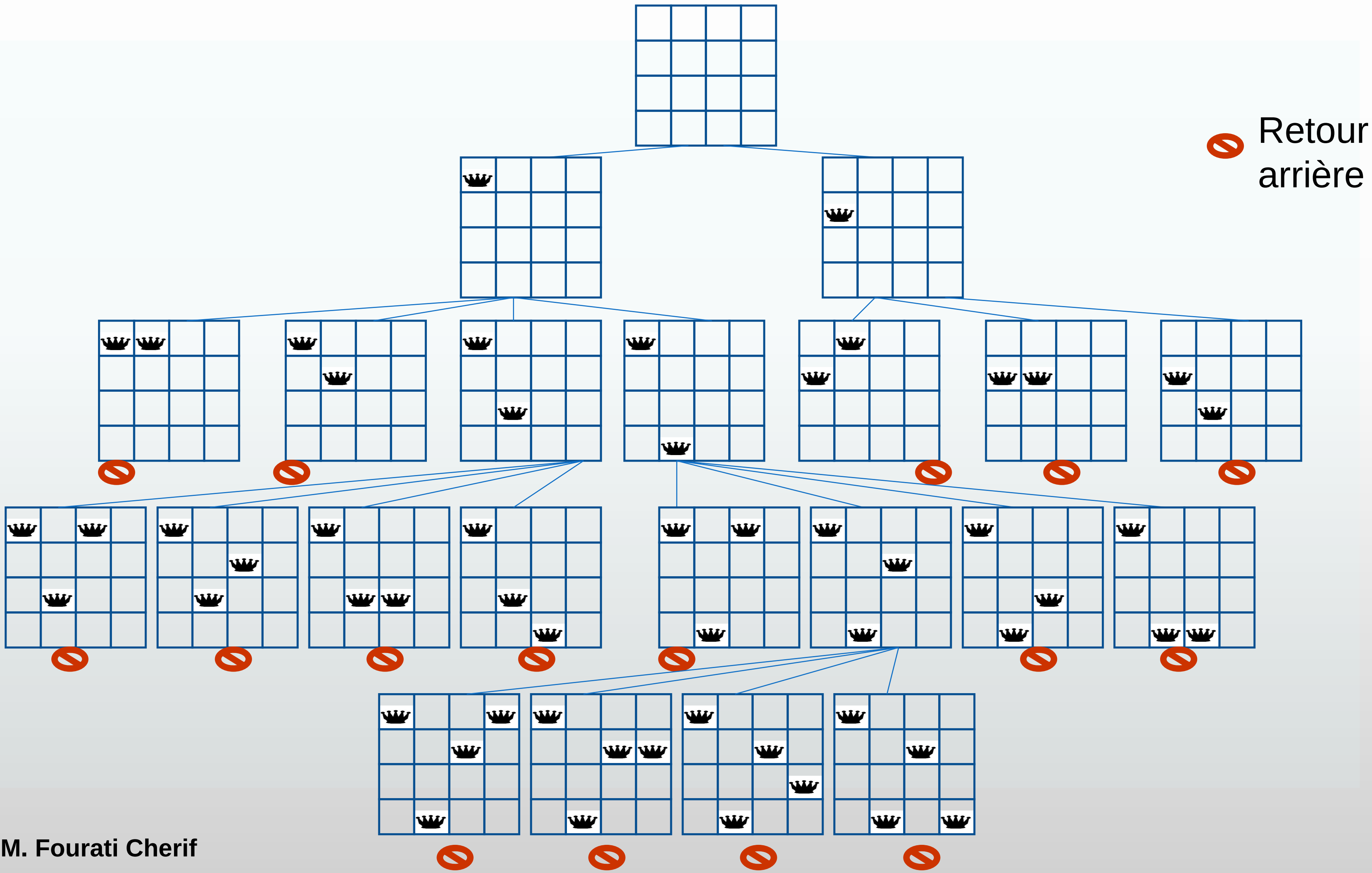
Exemple : résolution des 4 reines avec retour arrière



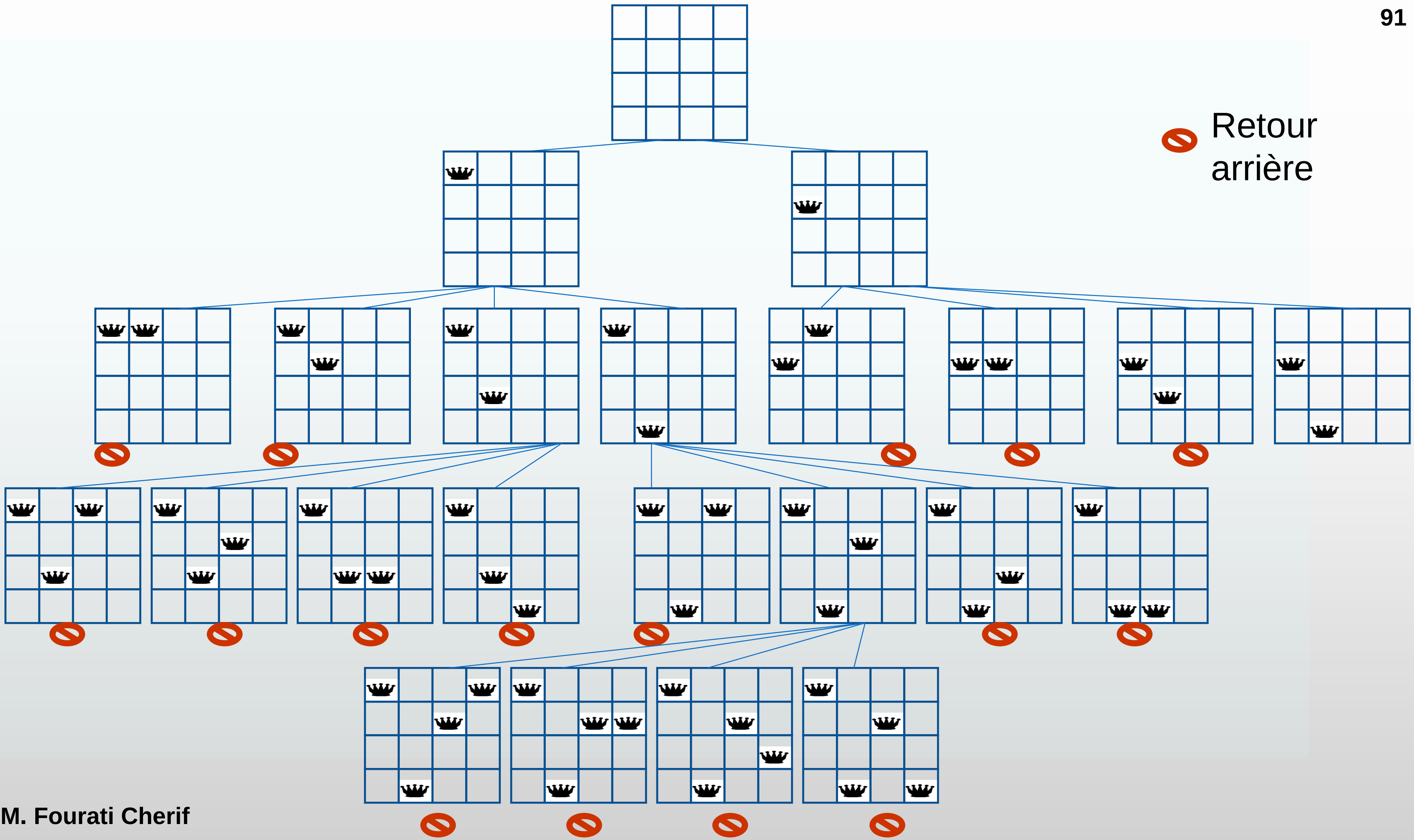
Exemple : résolution des 4 reines avec retour arrière



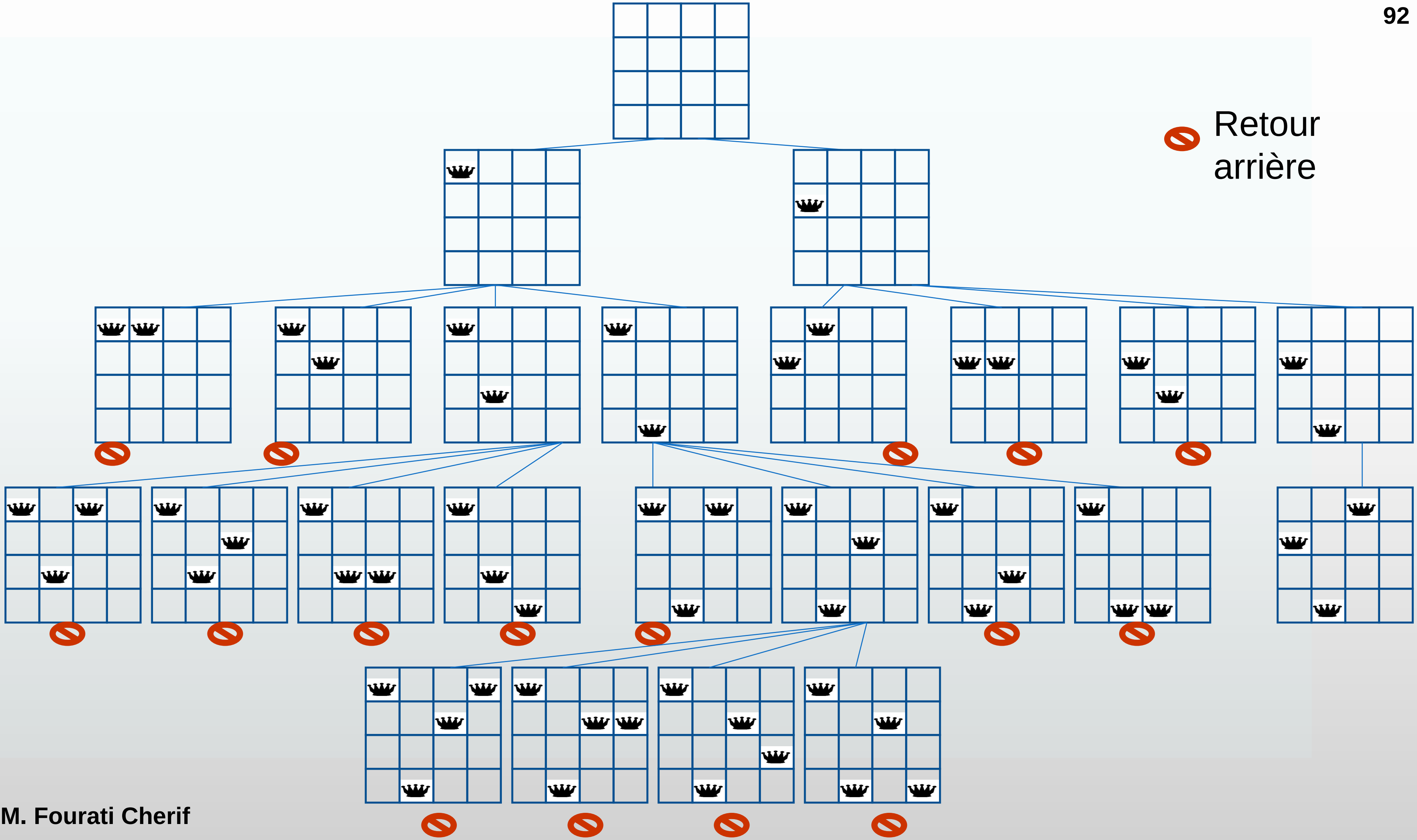
Exemple : résolution des 4 reines avec retour arrière



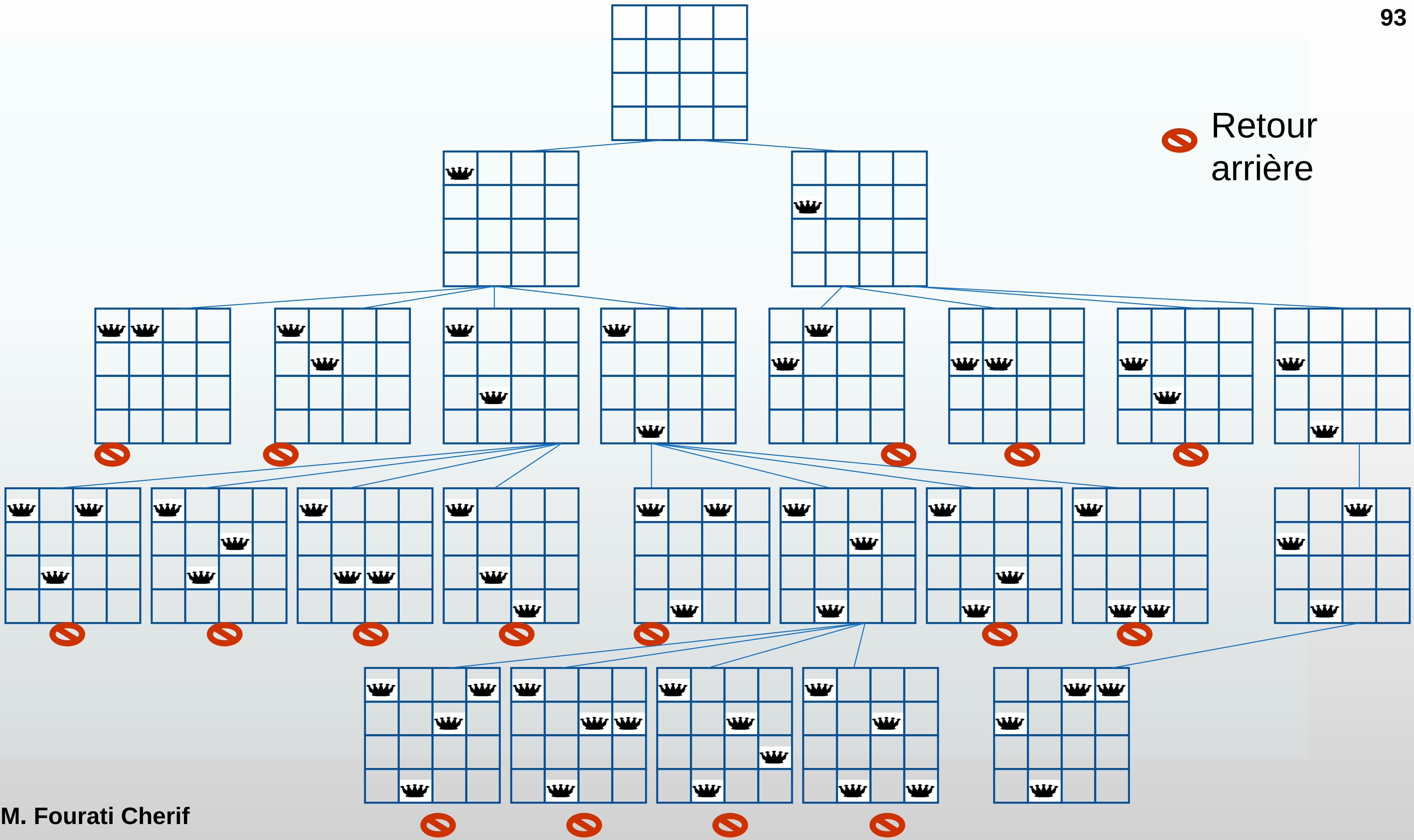
Exemple : résolution des 4 reines avec retour arrière



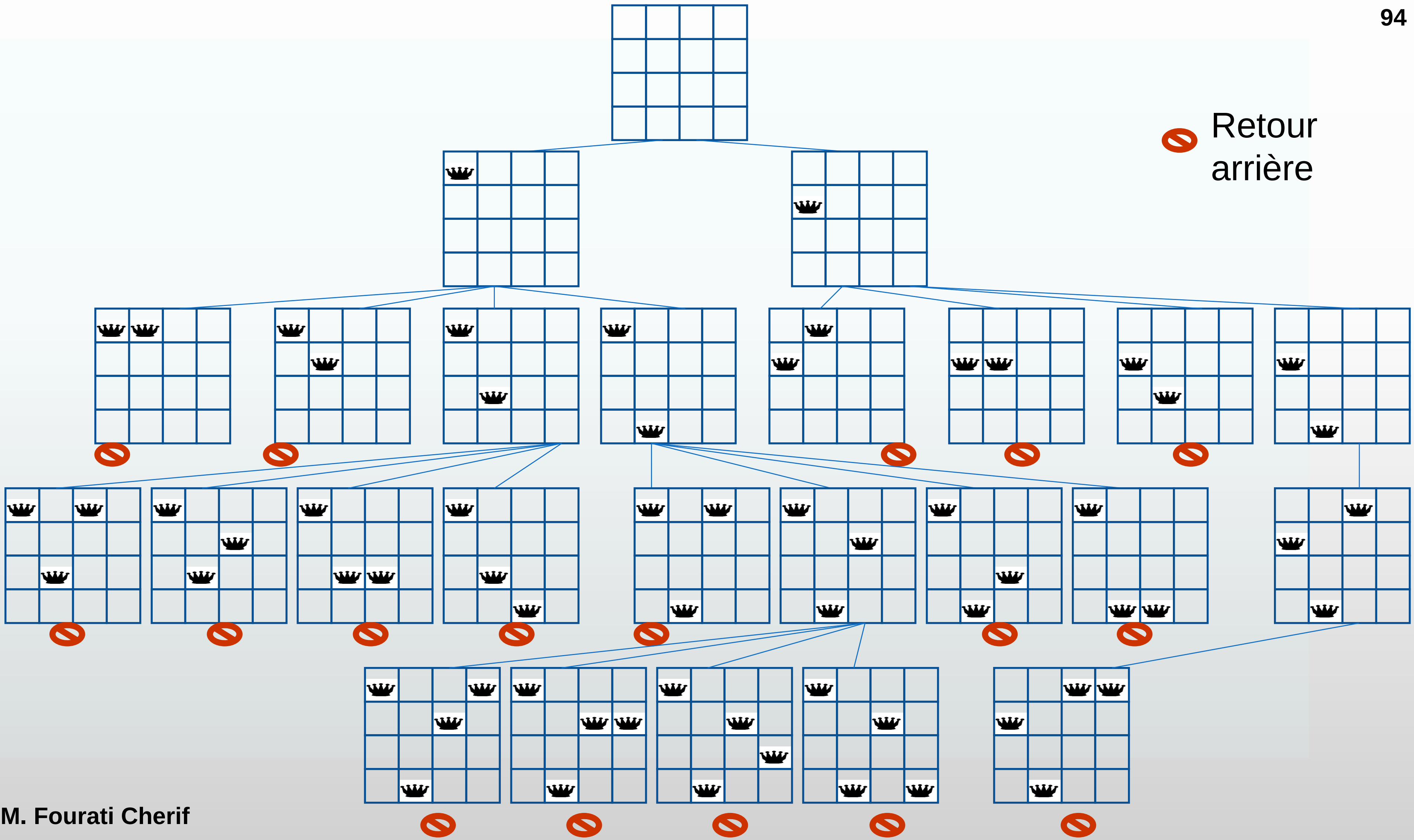
Exemple : résolution des 4 reines avec retour arrière



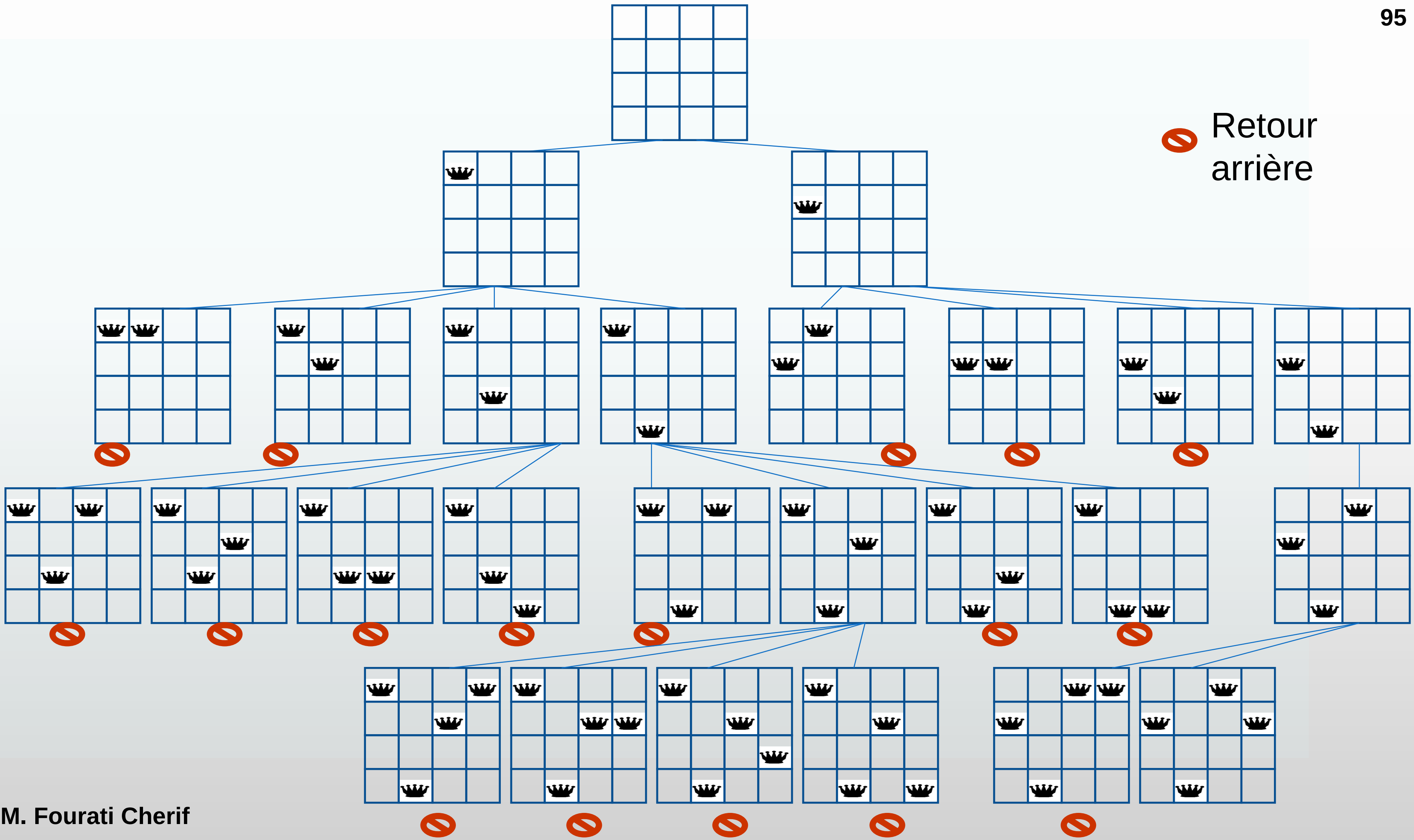
Exemple : résolution des 4 reines avec retour arrière



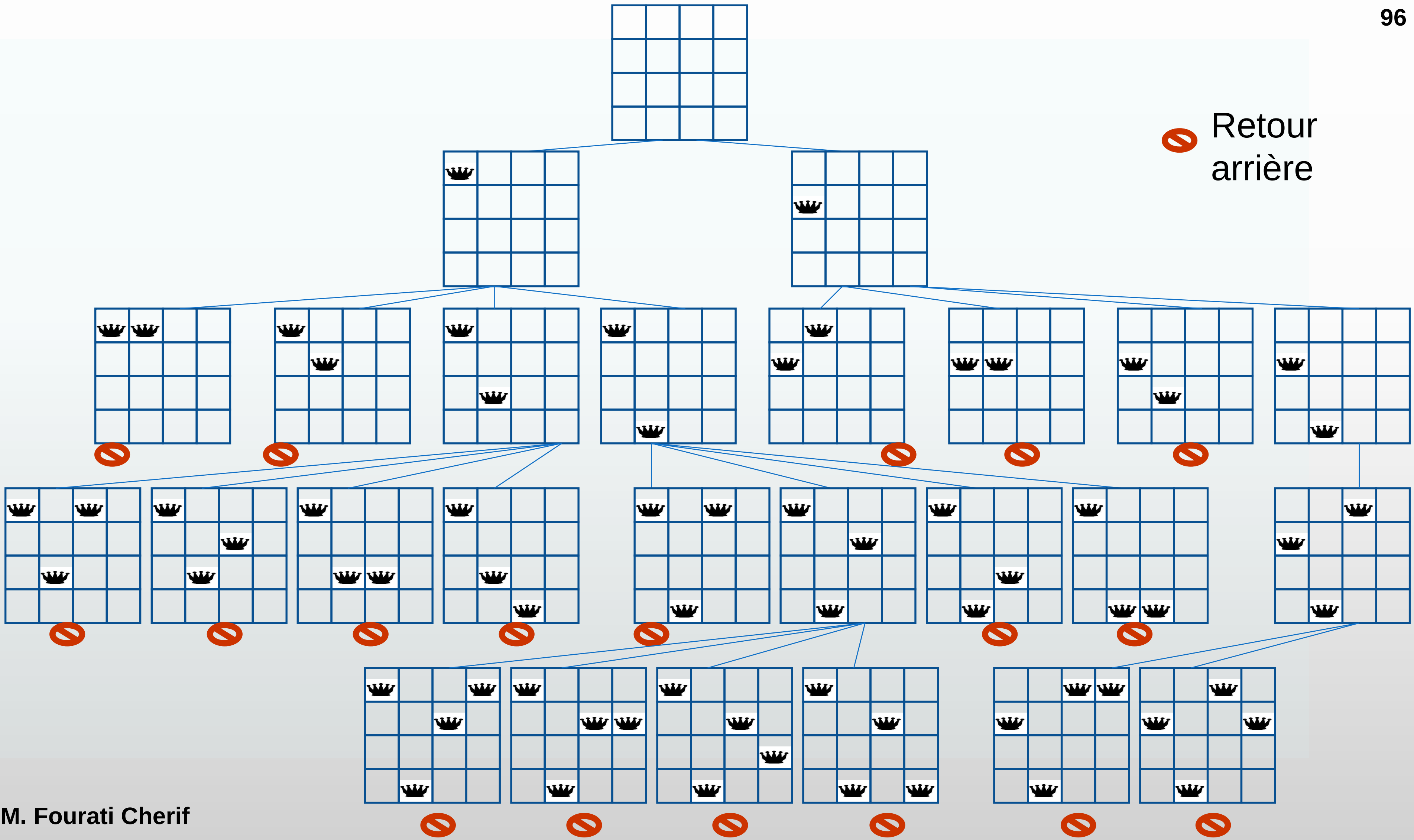
Exemple : résolution des 4 reines avec retour arrière



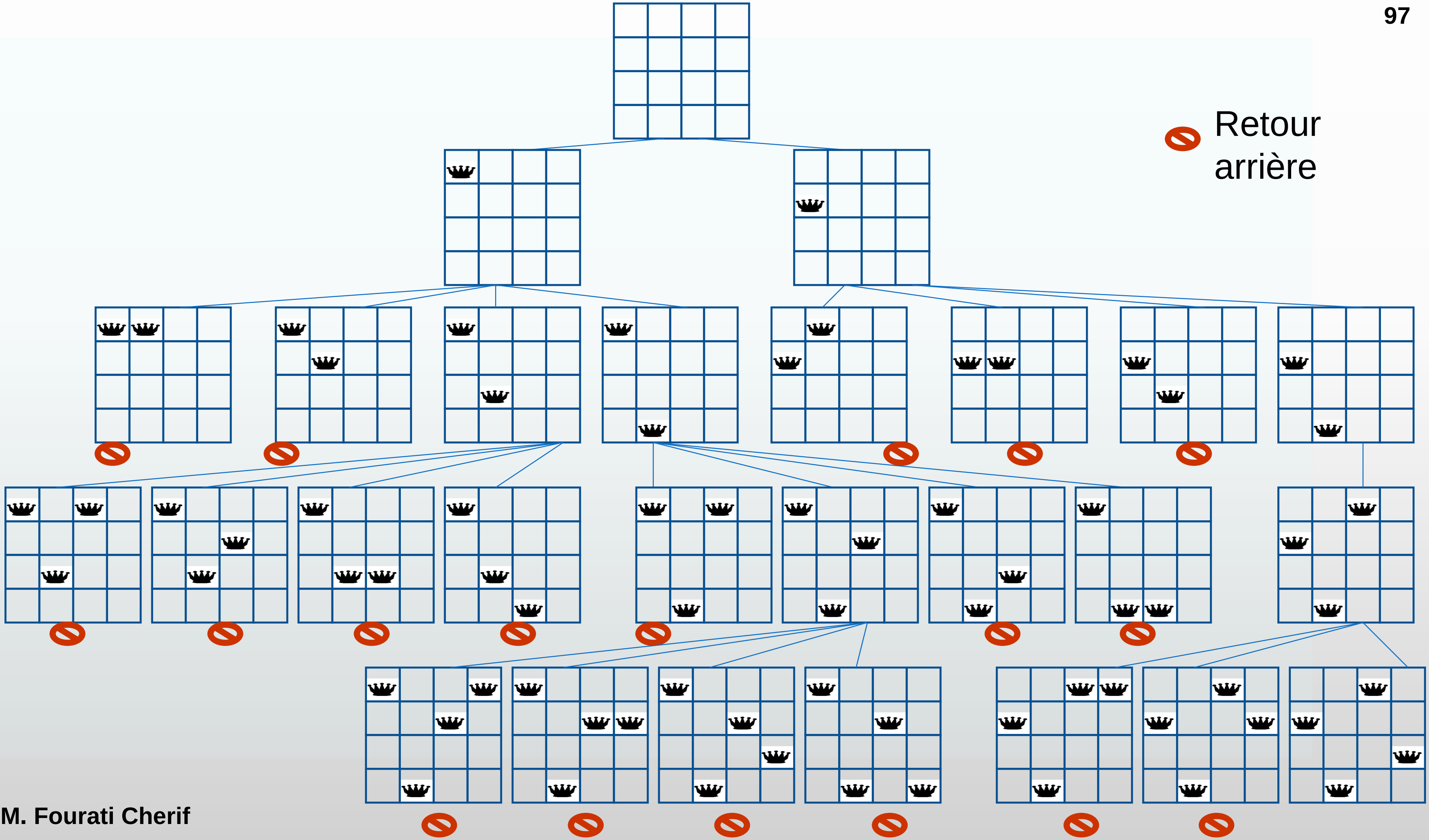
Exemple : résolution des 4 reines avec retour arrière



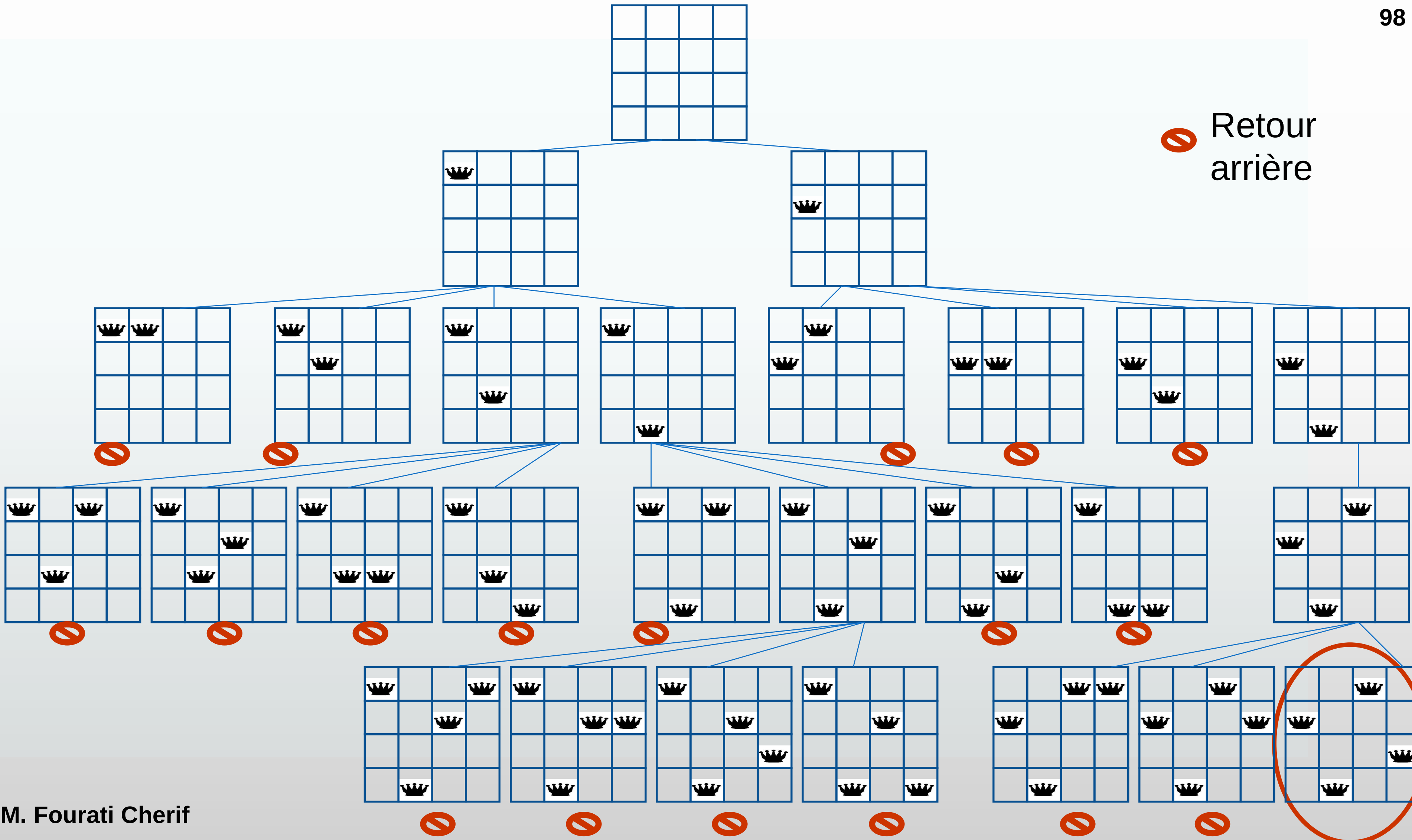
Exemple : résolution des 4 reines avec retour arrière



Exemple : résolution des 4 reines avec retour arrière



Exemple : résolution des 4 reines avec retour arrière



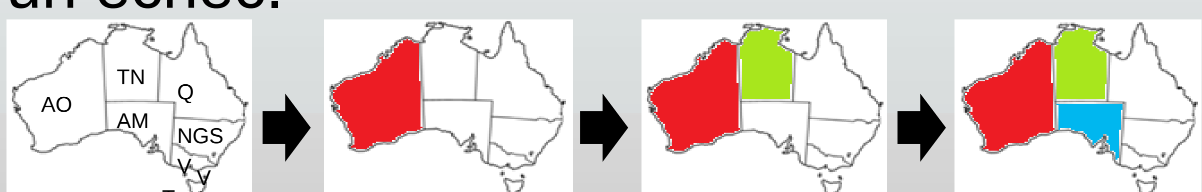
Problèmes pour la recherche avec retour arrière

- Q1 : Comment choisir la prochaine variable à assigner ?
- Q2 : Dans quel ordre les valeurs doivent être essayées ?
- Q3 : Quelles sont les répercussions de l'assignation d'une variable sur les autres variables non encore assignées (comment détecter les erreurs assez tôt et réduire l'espace d'états des assignations) ?
- Q4 : Lorsque l'exploration échoue à cause d'une violation de contrainte, l'exploration peut-elle éviter de répéter cet échec ?

Amélioration de l'efficacité de la recherche avec retour arrière

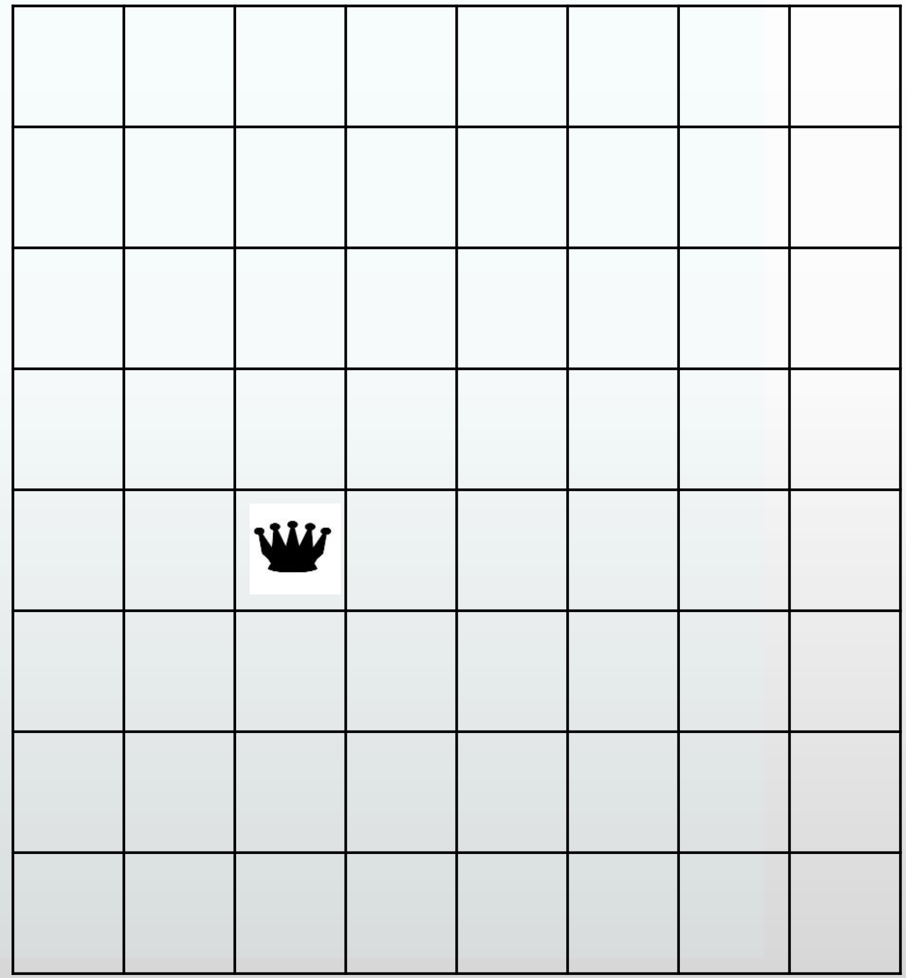
Heuristique du minimum valeurs restantes

- Q1 : Comment choisir la prochaine variable à assigner ?
- Ligne : *var* ← SÉLECTIONNER-VARIABLE-NON-ASSIGNÉE(*csp*)
- Sélectionner la variable non assignée dans l'ordre (X_1,X_2,…), mais généralement n'est pas la plus efficace.
- Application d'une heuristique du minimum valeurs restantes MRV(*minimum remaining values*) ou heuristique de la variable la plus contrainte : améliore l'efficacité puisqu'elle est la plus susceptible de provoquer un échec.



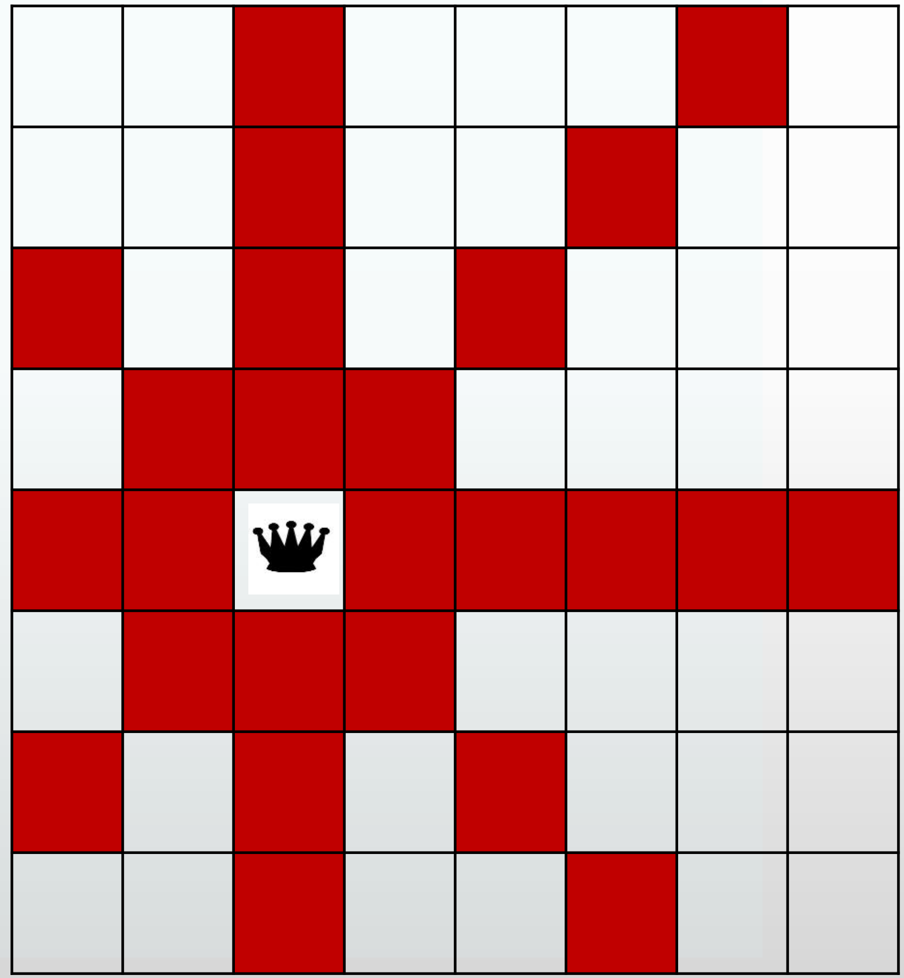
L’heuristique MRV sur les 8-Reines

- Placer aléatoirement une reine sur l’échiquier : pas de variables plus contraignante que les autres.



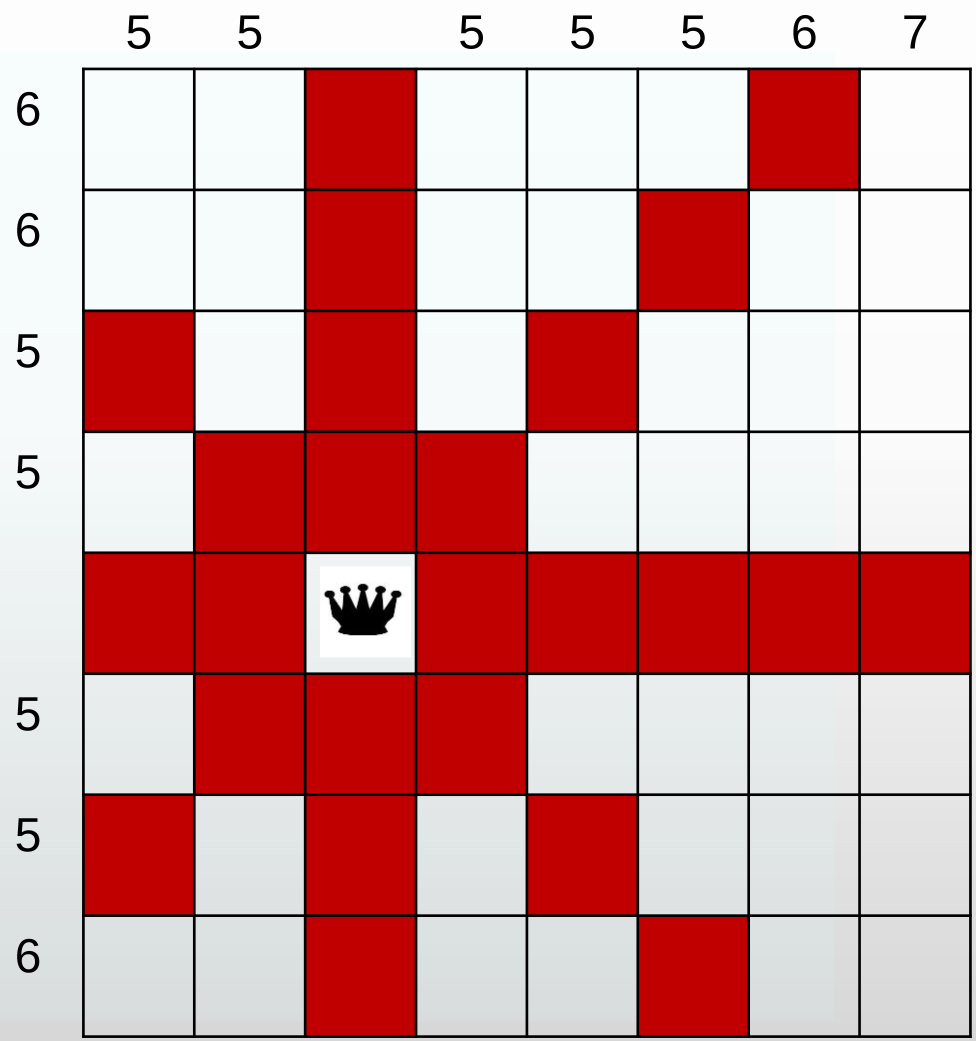
L'heuristique MRV sur les 8-Reines

- Placer une reine sur l'échiquier.
- Éliminer les carrés qui ne peuvent plus contenir de reines.



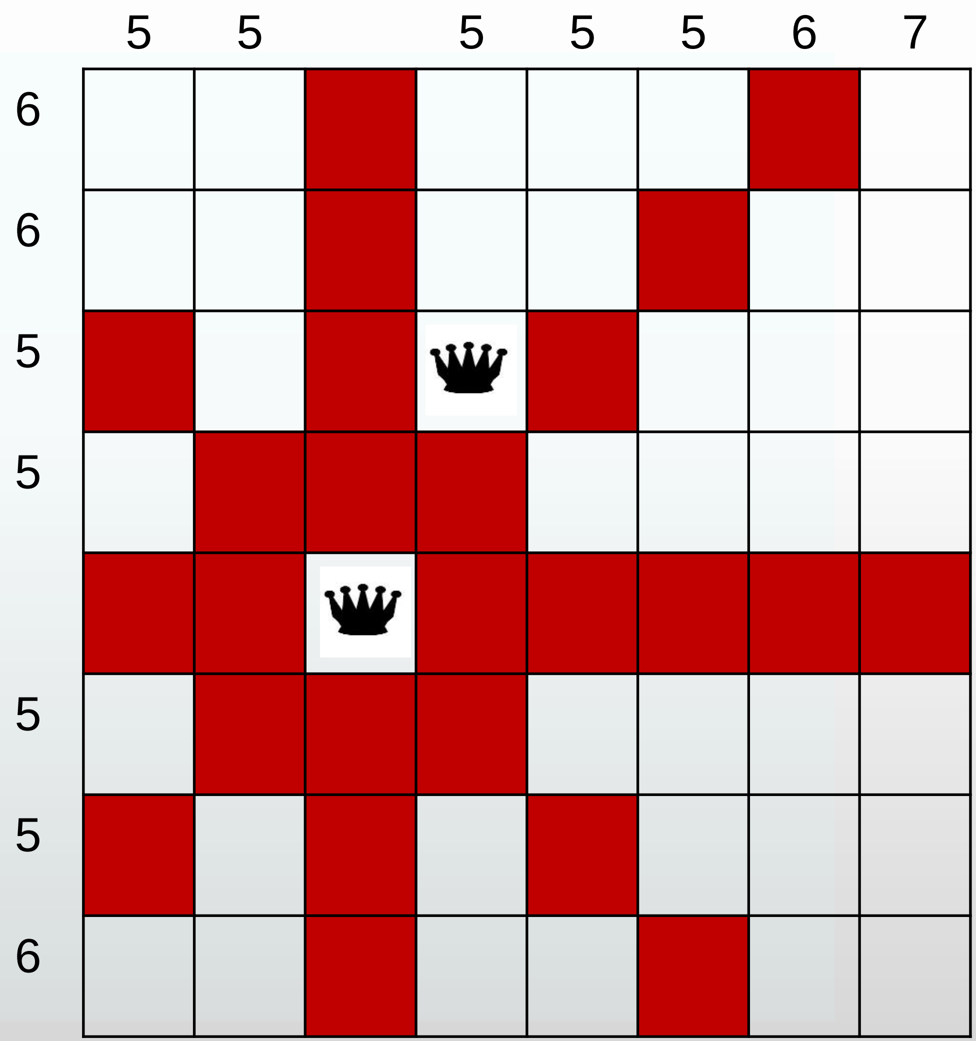
L’heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne.



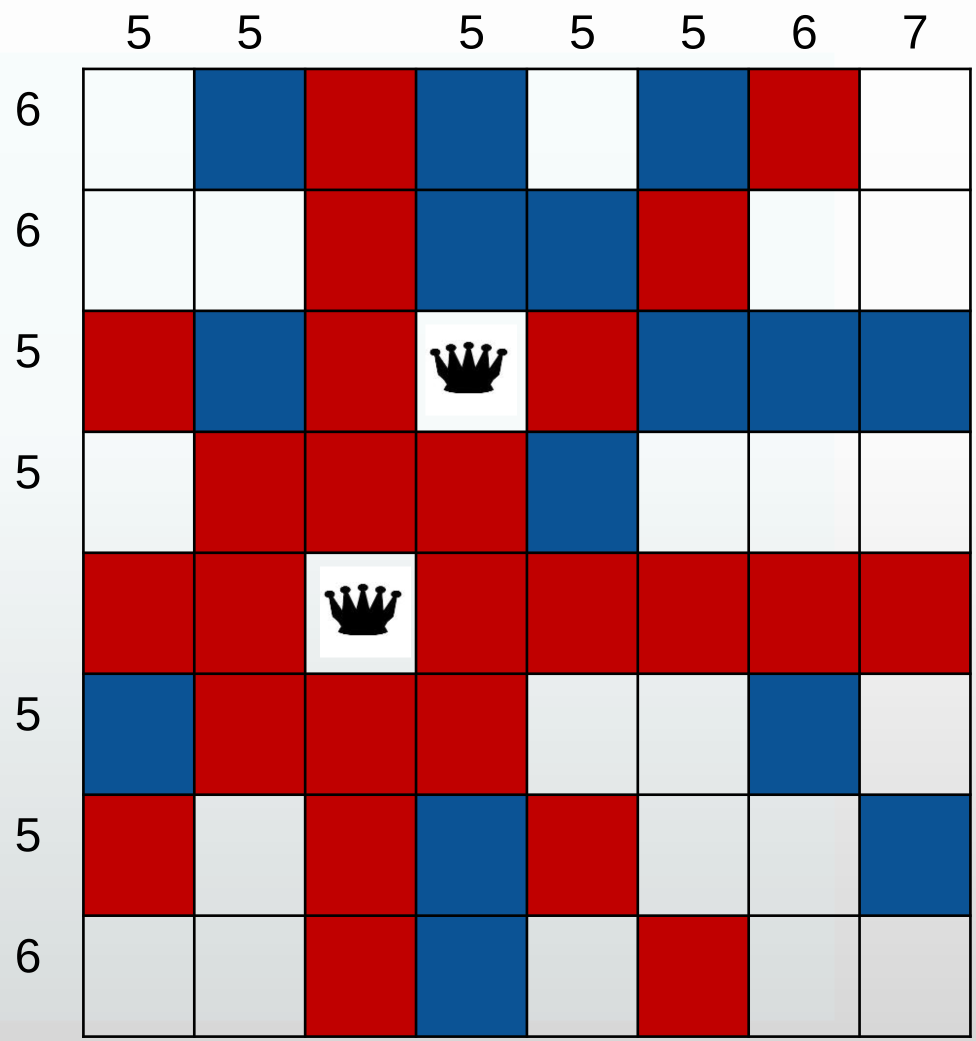
L’heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne
- Placer une reine dans une ligne ou colonne ayant le plus petit chiffre.



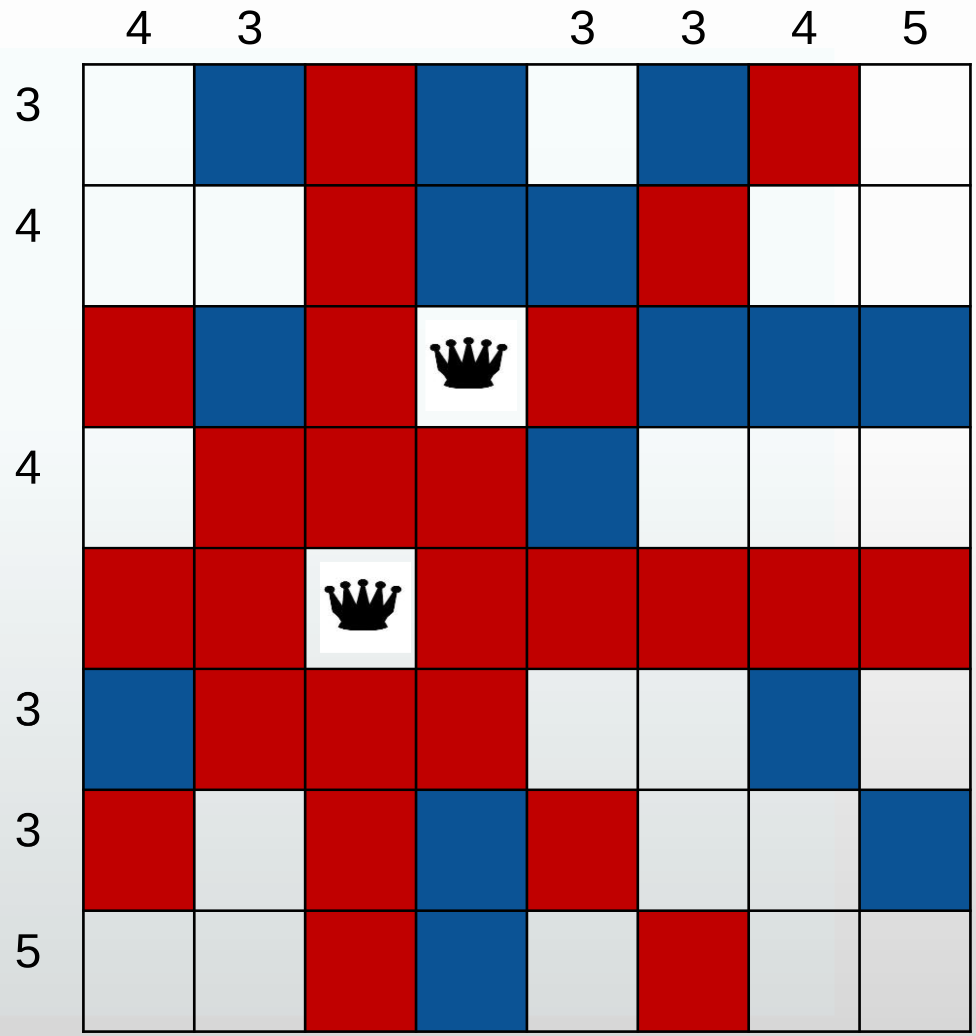
L’heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne
- Placer une reine dans une ligne ou colonne ayant le plus petit chiffre.
- Eliminer les carrés attaqués.



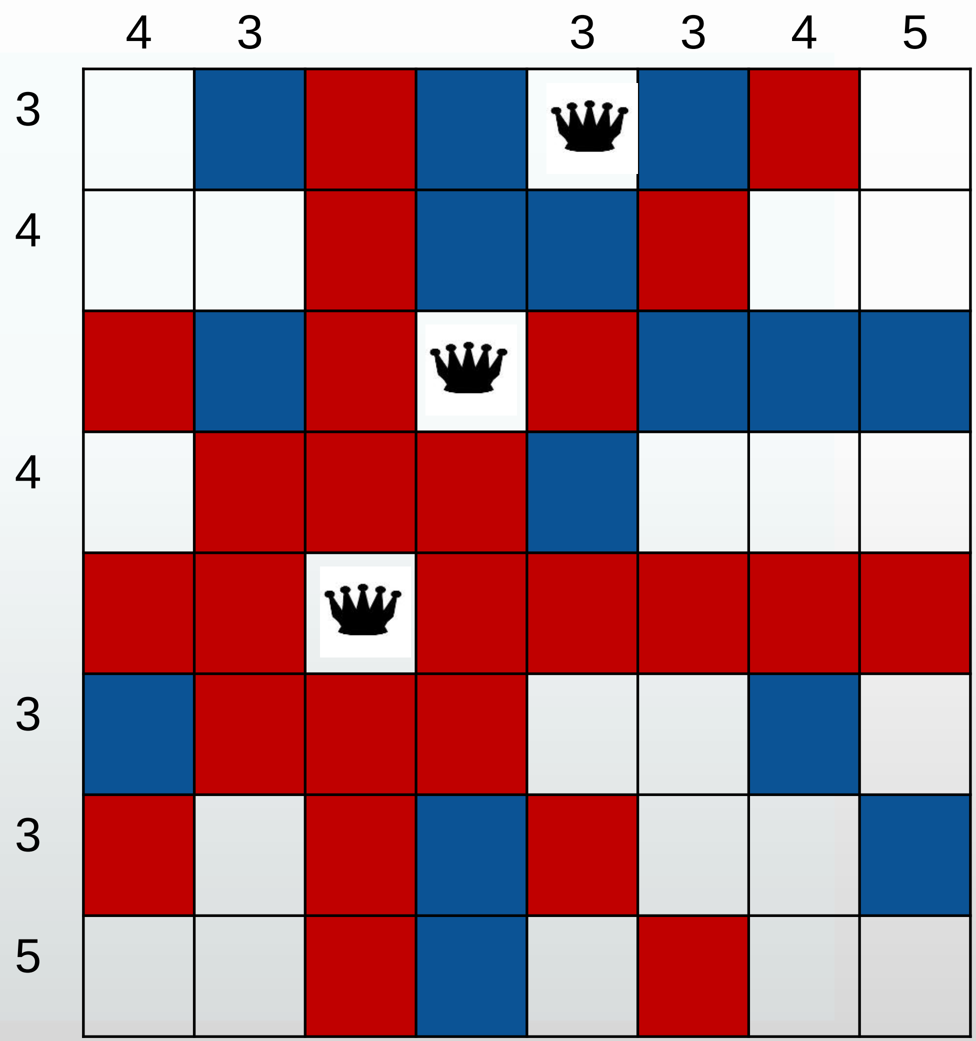
L’heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne.



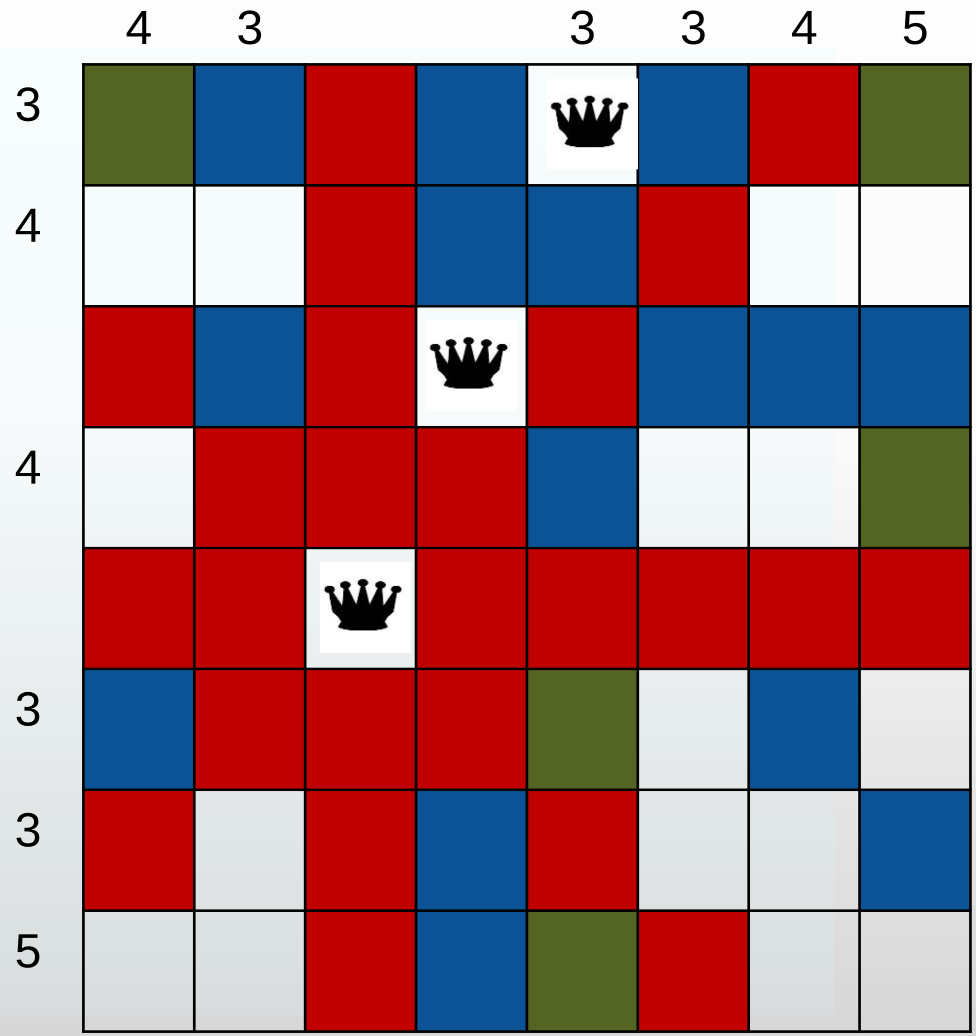
L’heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne
- Placer une reine dans une ligne ou colonne ayant le plus petit chiffre.



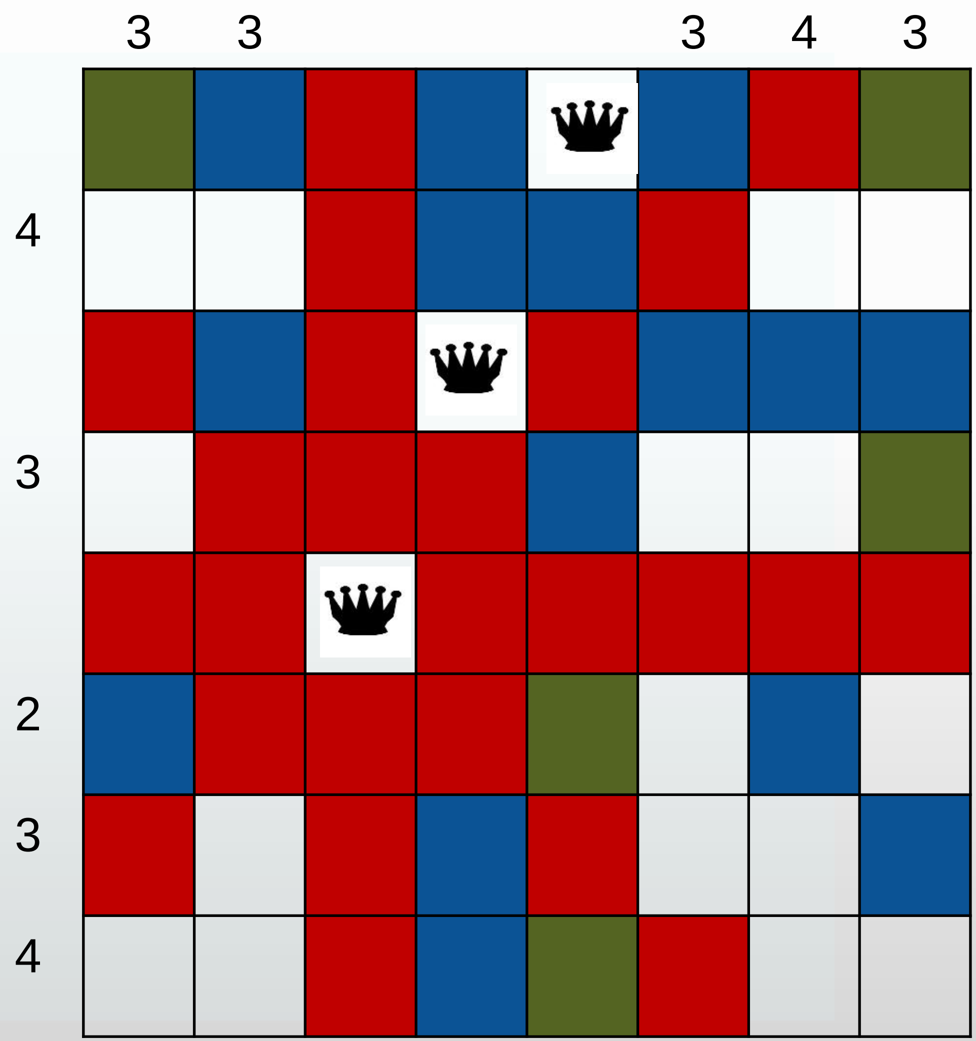
L'heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne
- Placer une reine dans une ligne ou colonne ayant le plus petit chiffre.
- Eliminer les carrés attaqués.



L'heuristique MRV sur les 8-Reines

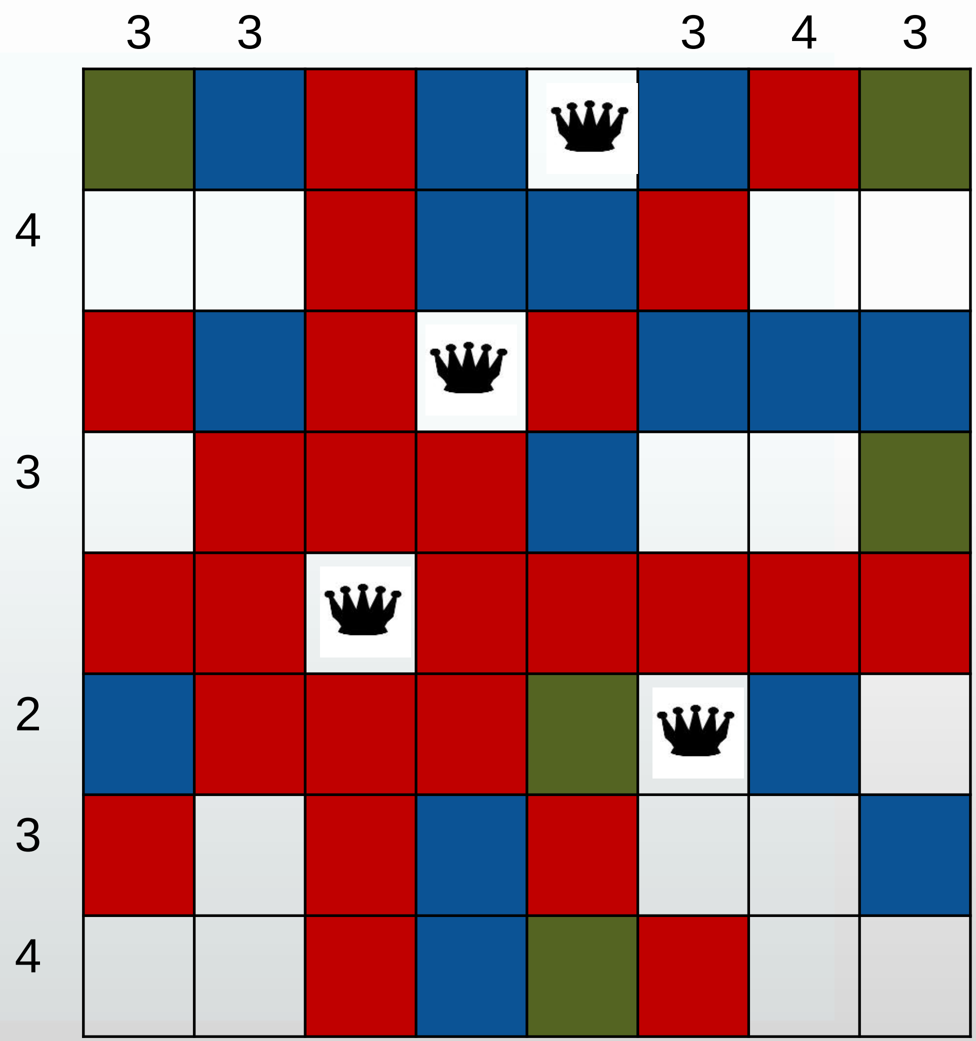
- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne



L’heuristique MRV sur les 8-Reines

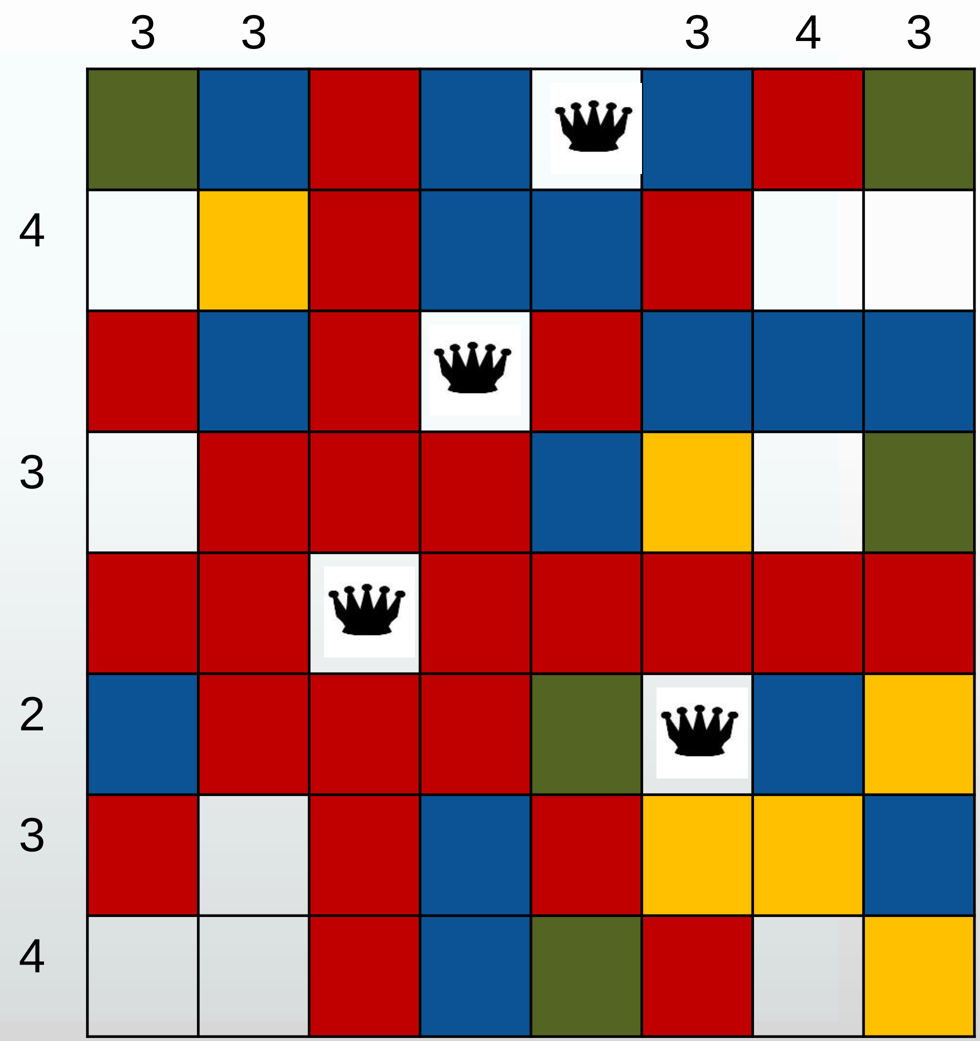
- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne

- Placer une reine dans une ligne ou colonne ayant le plus petit chiffre.



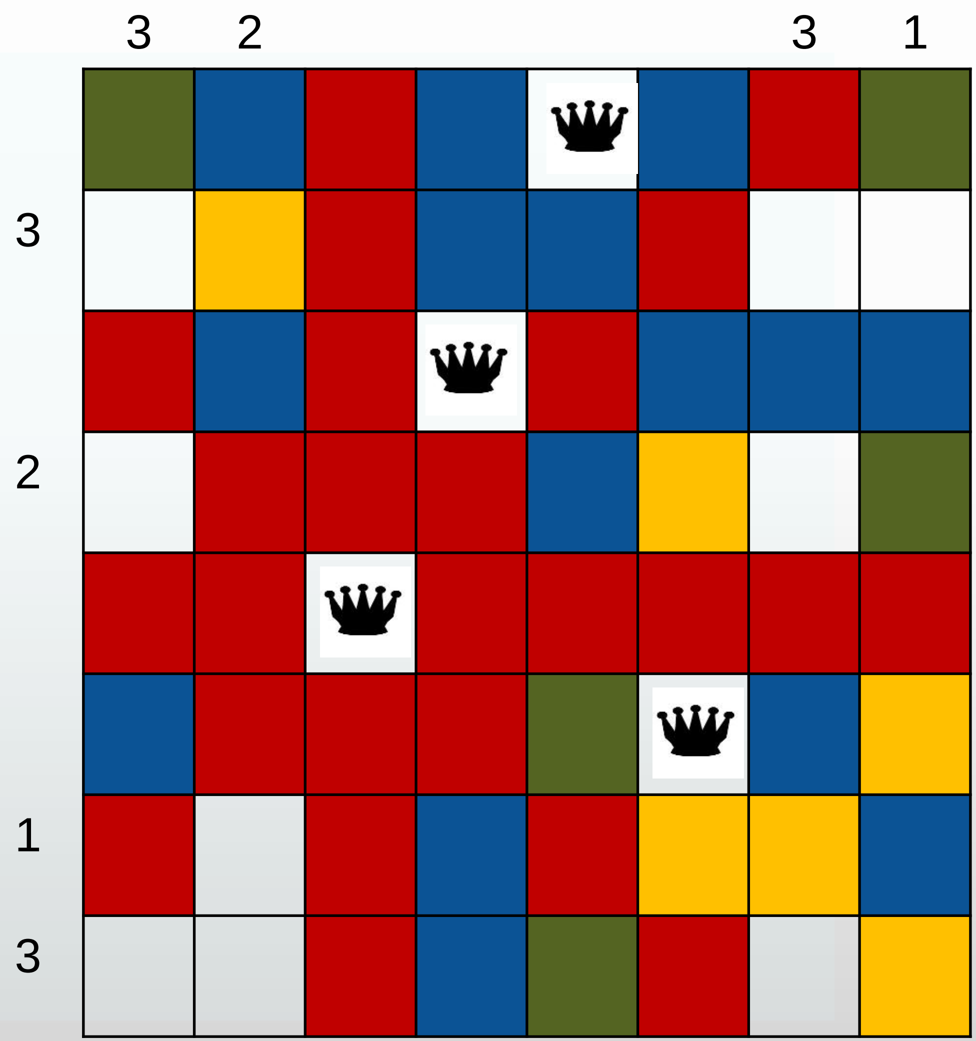
L’heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne
- Placer une reine dans une ligne ou colonne ayant le plus petit chiffre.
- Eliminer les carrés attaqués.



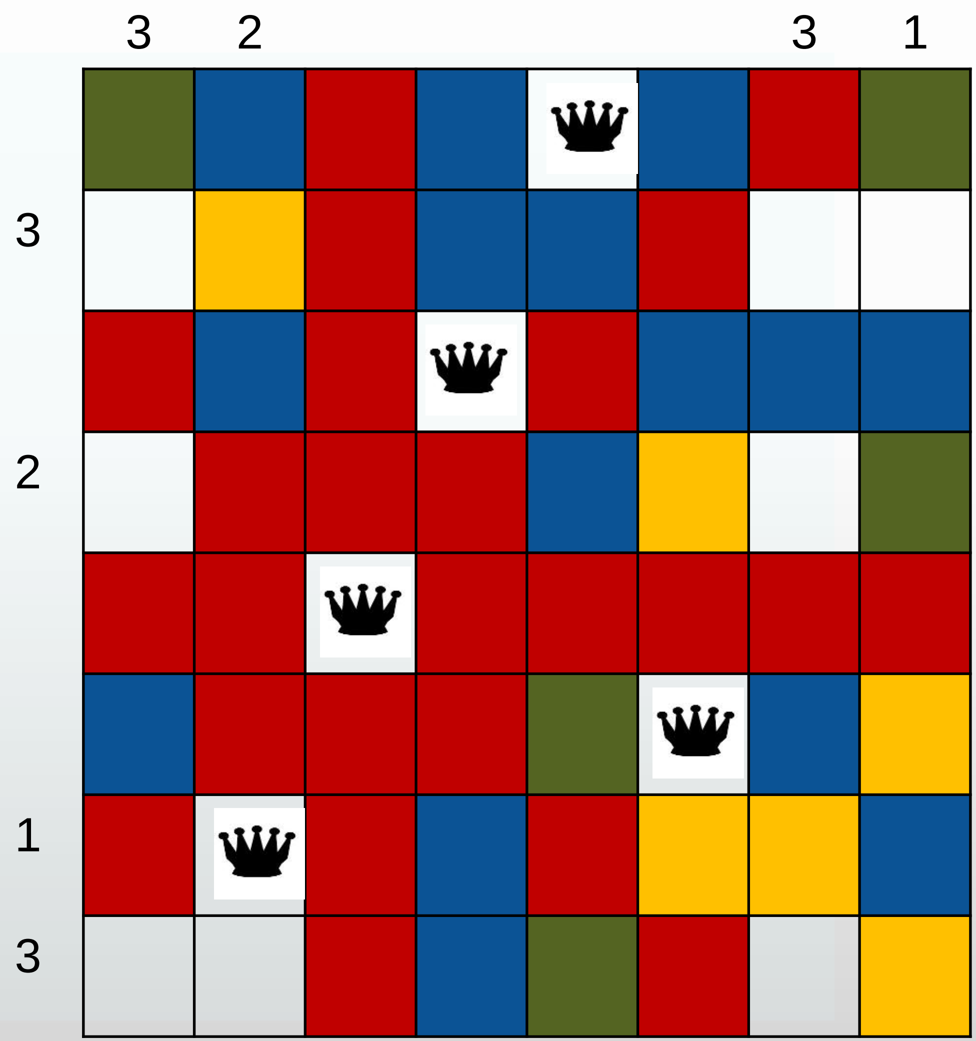
L’heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne.



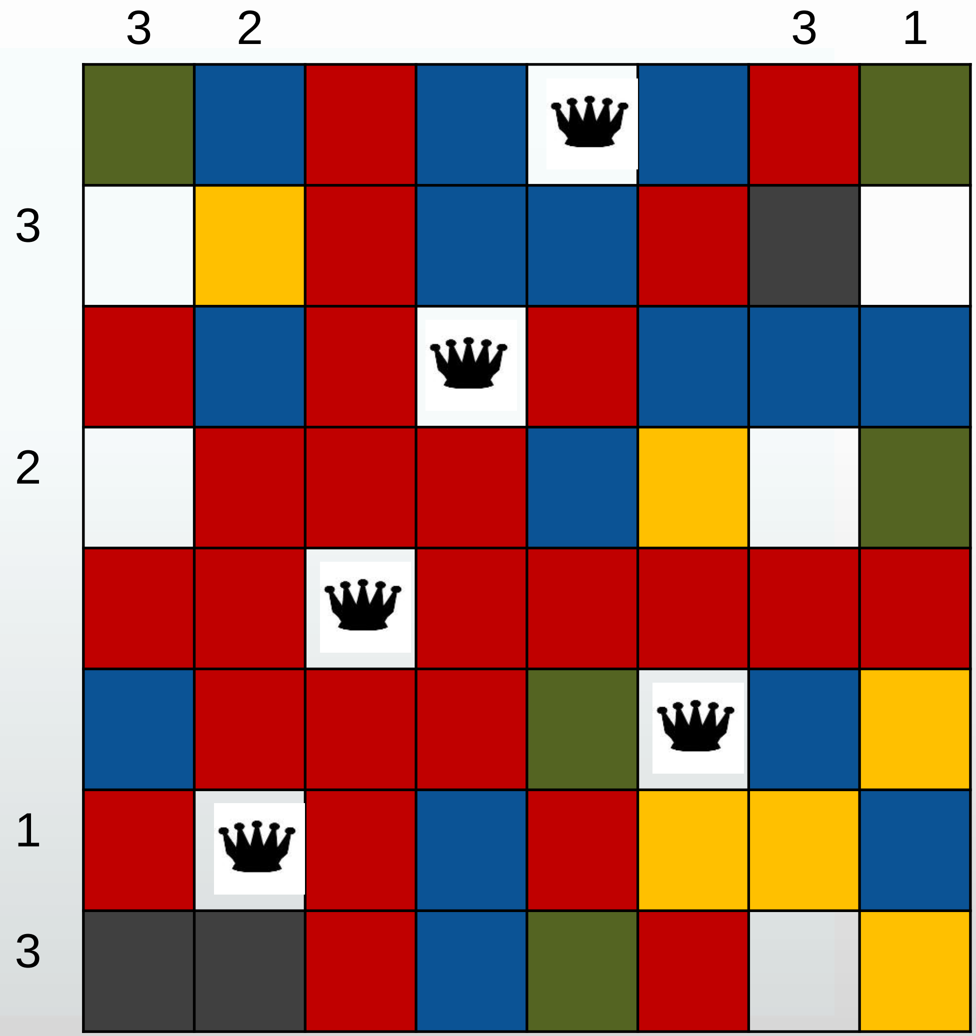
L’heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne
- Placer une reine dans une ligne ou colonne ayant le plus petit chiffre.



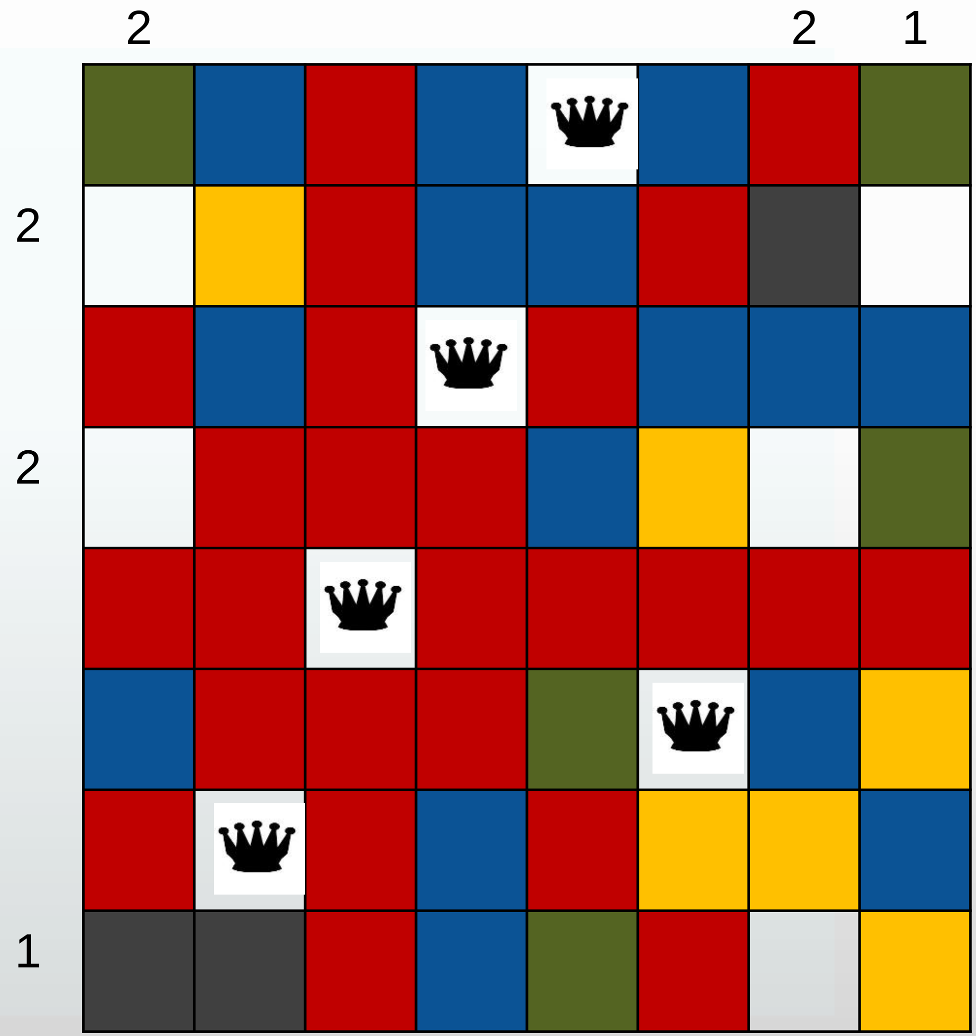
L'heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne
- Placer une reine dans une ligne ou colonne ayant le plus petit chiffre.
- Eliminer les carrés attaqués.



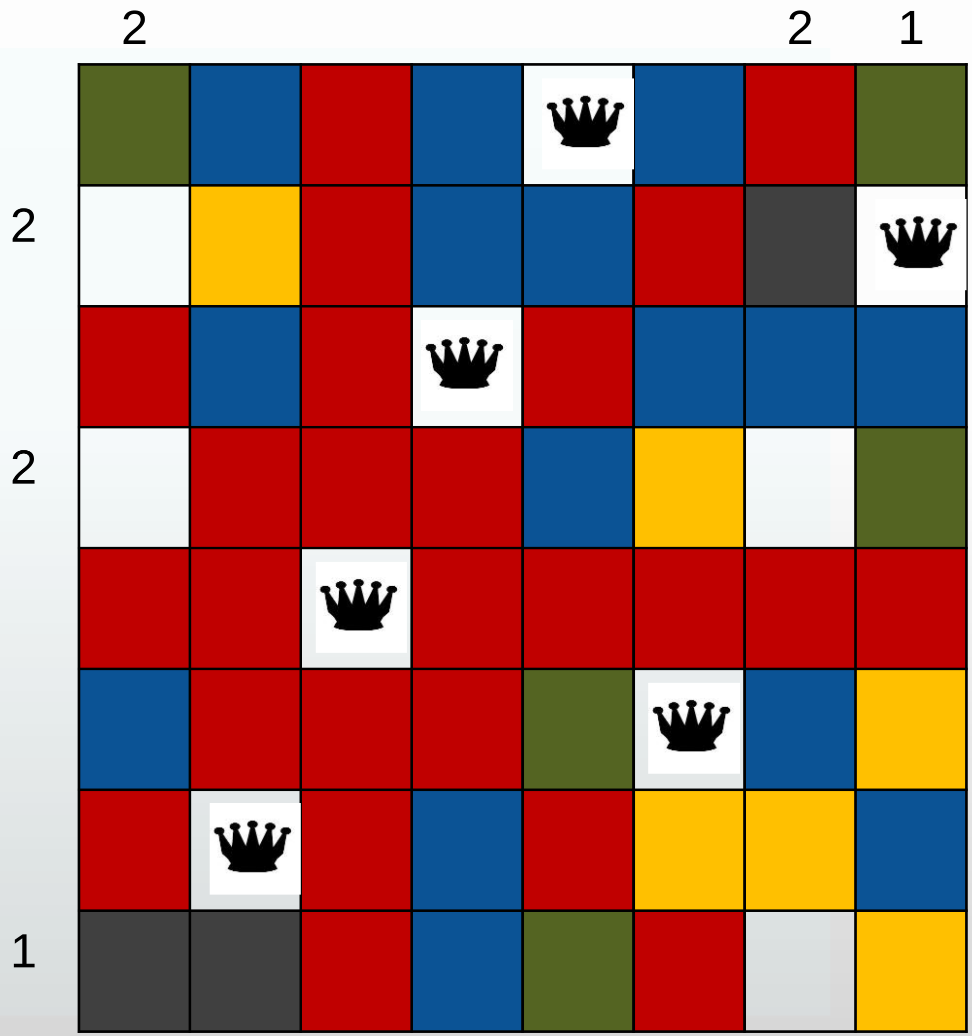
L’heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne



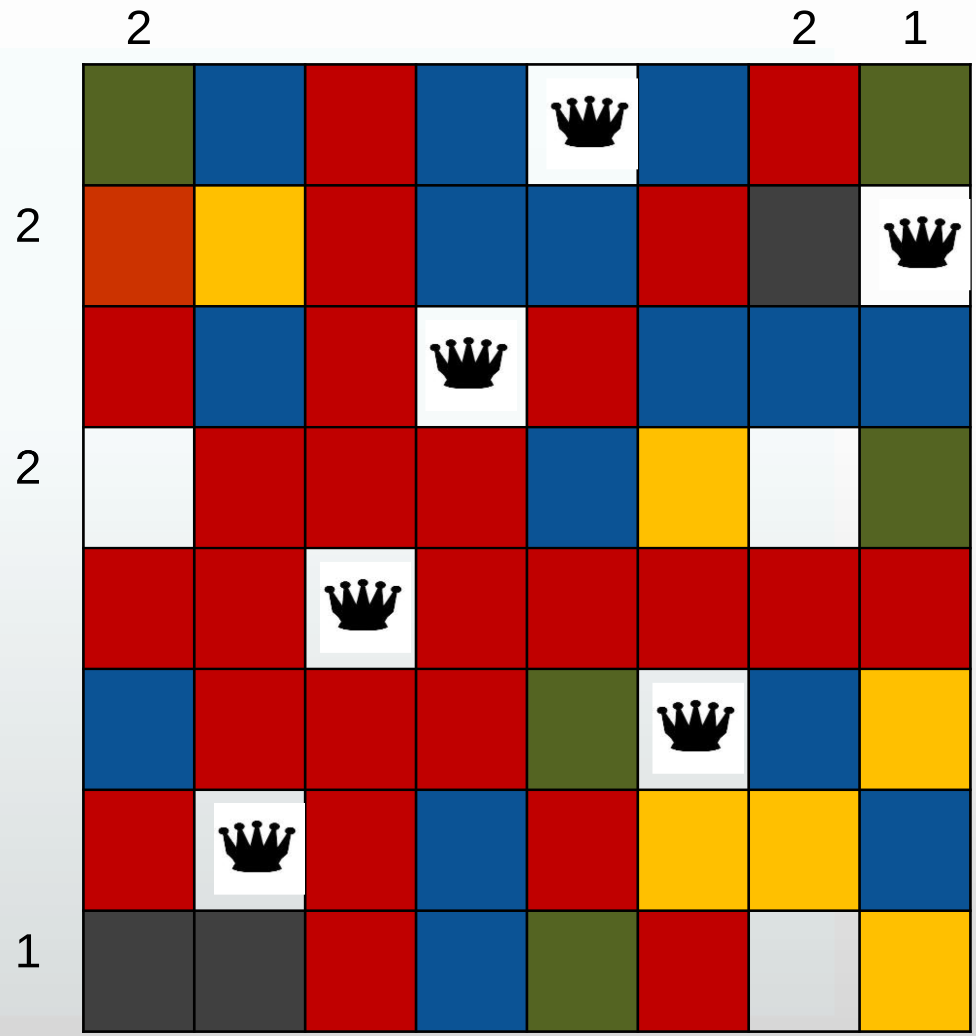
L'heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne
- Placer une reine dans une ligne ou colonne ayant le plus petit chiffre.



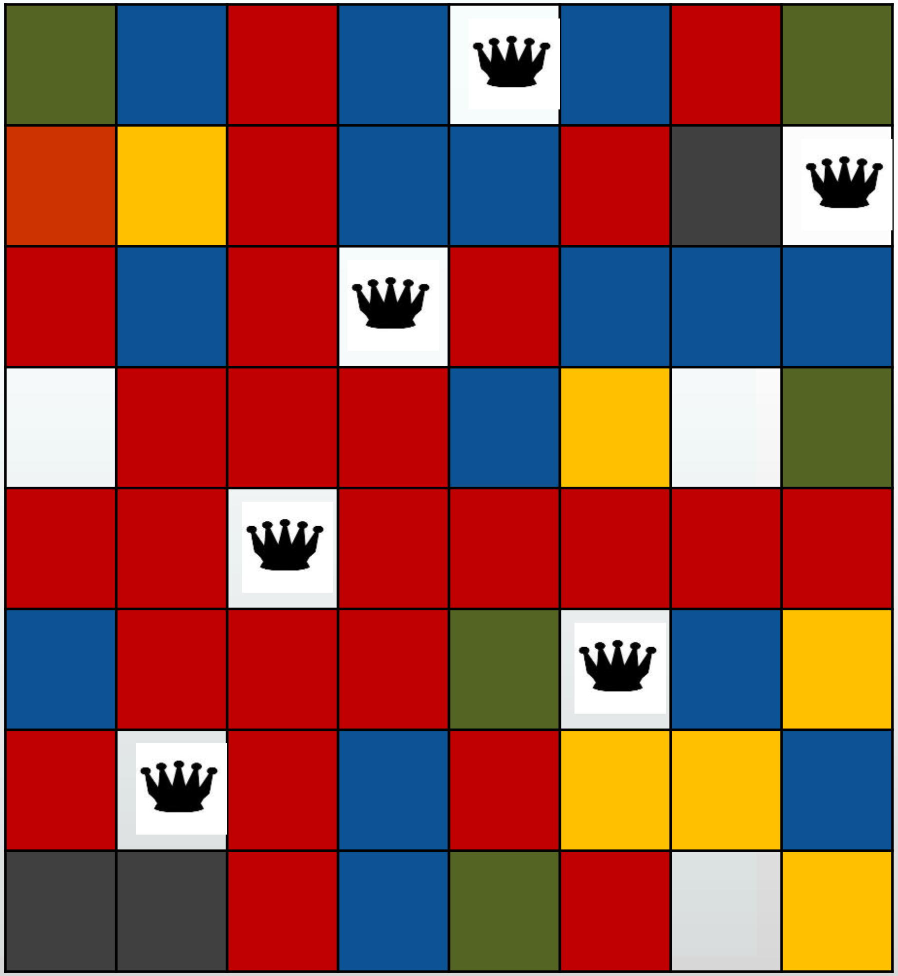
L’heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne
- Placer une reine dans une ligne ou colonne ayant le plus petit chiffre.
- Eliminer les carrés attaqués.



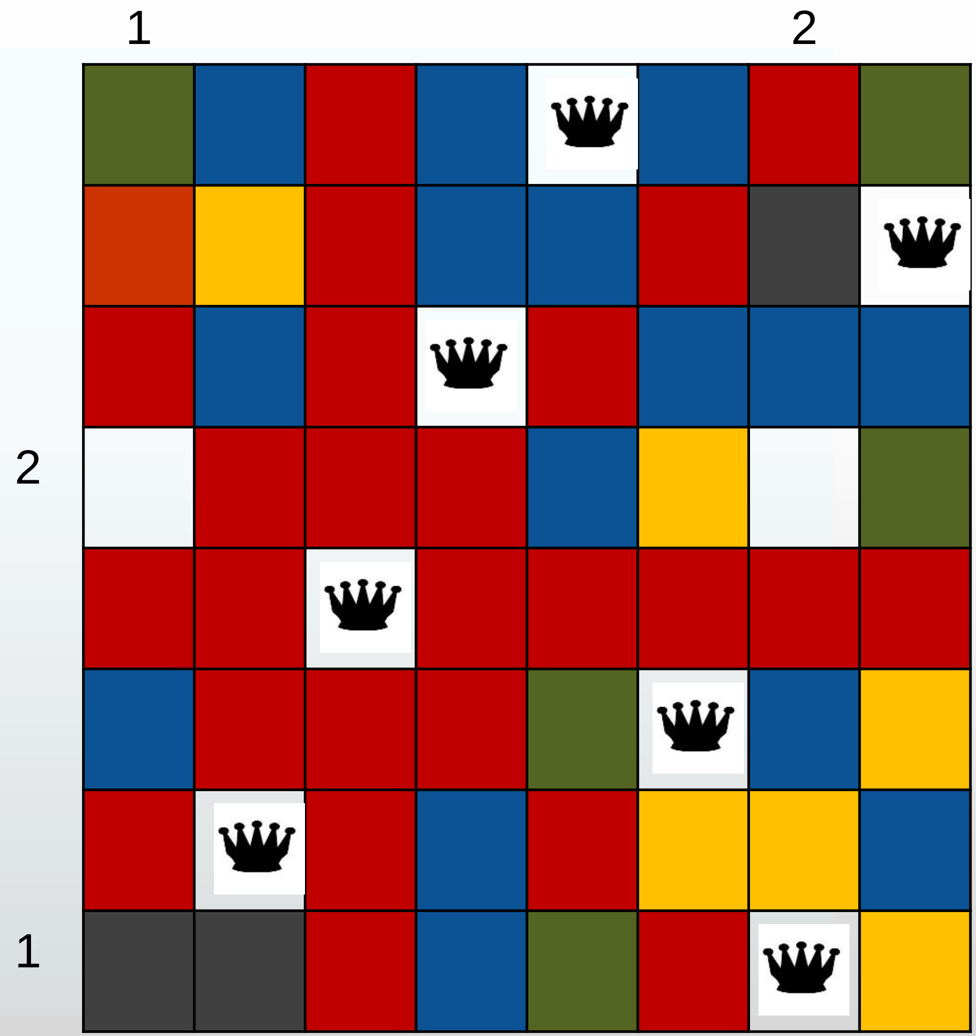
L’heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne.



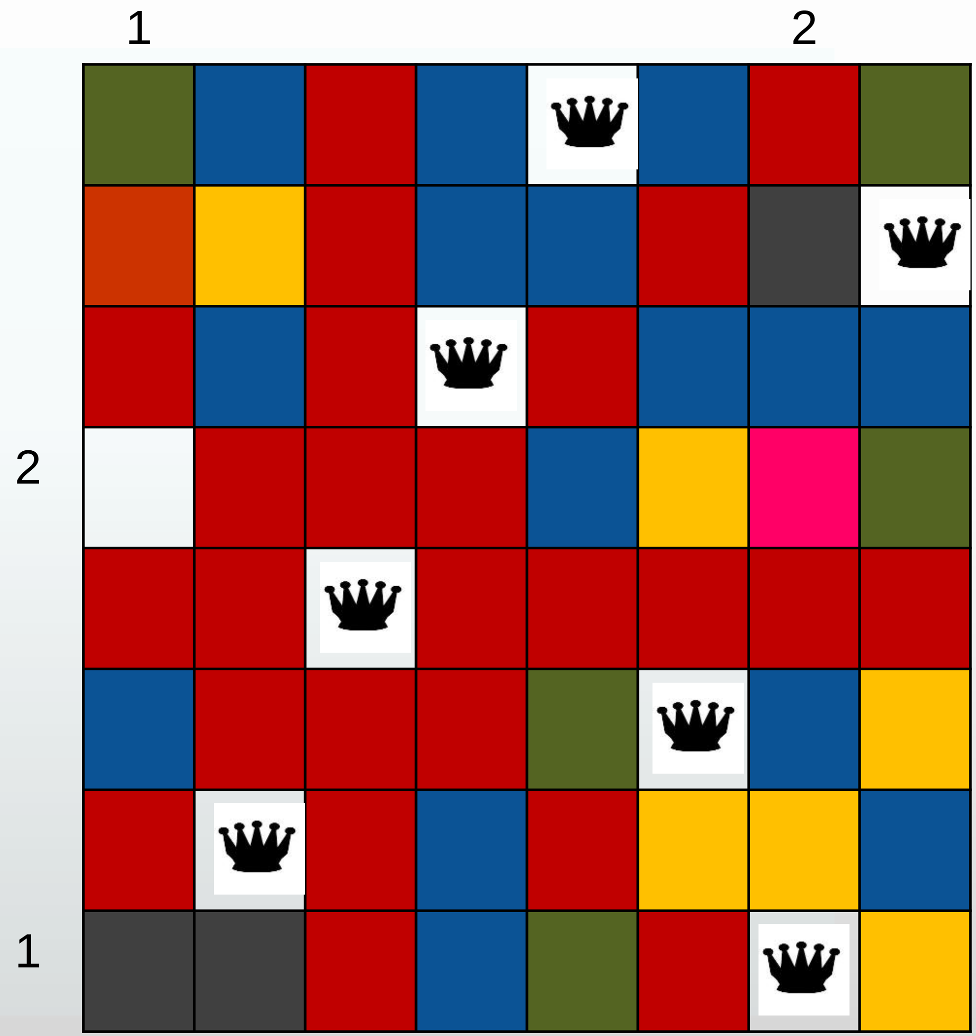
L'heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne.
- Placer une reine dans une ligne ou colonne ayant le plus petit chiffre.



L’heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne
- Placer une reine dans une ligne ou colonne ayant le plus petit chiffre.
- Eliminer les carrés attaqués.

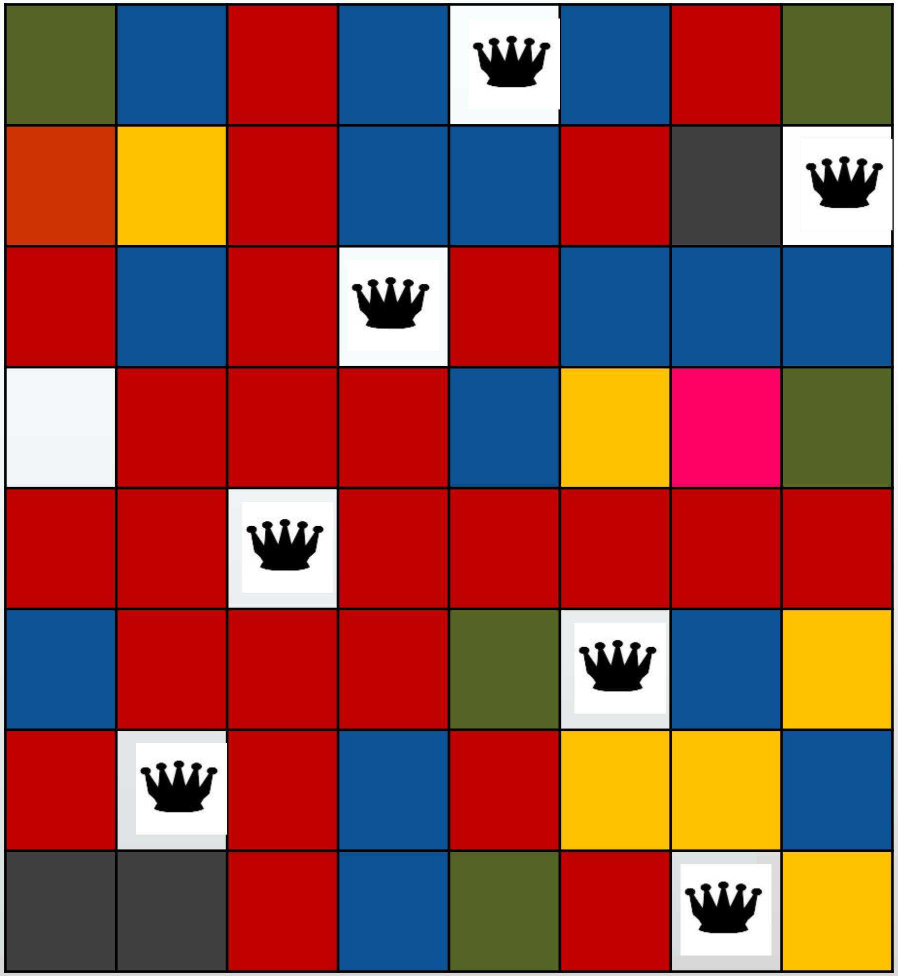


L’heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne.

1

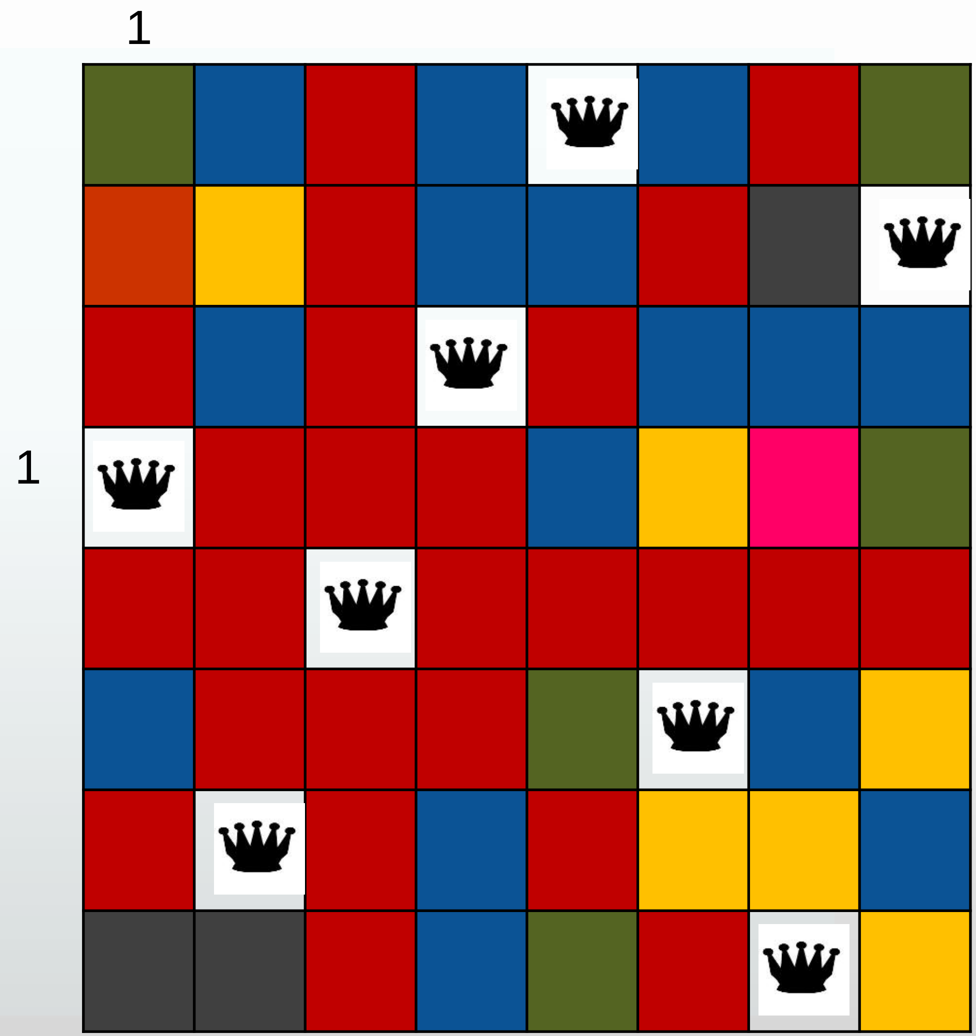
1



L’heuristique MRV sur les 8-Reines

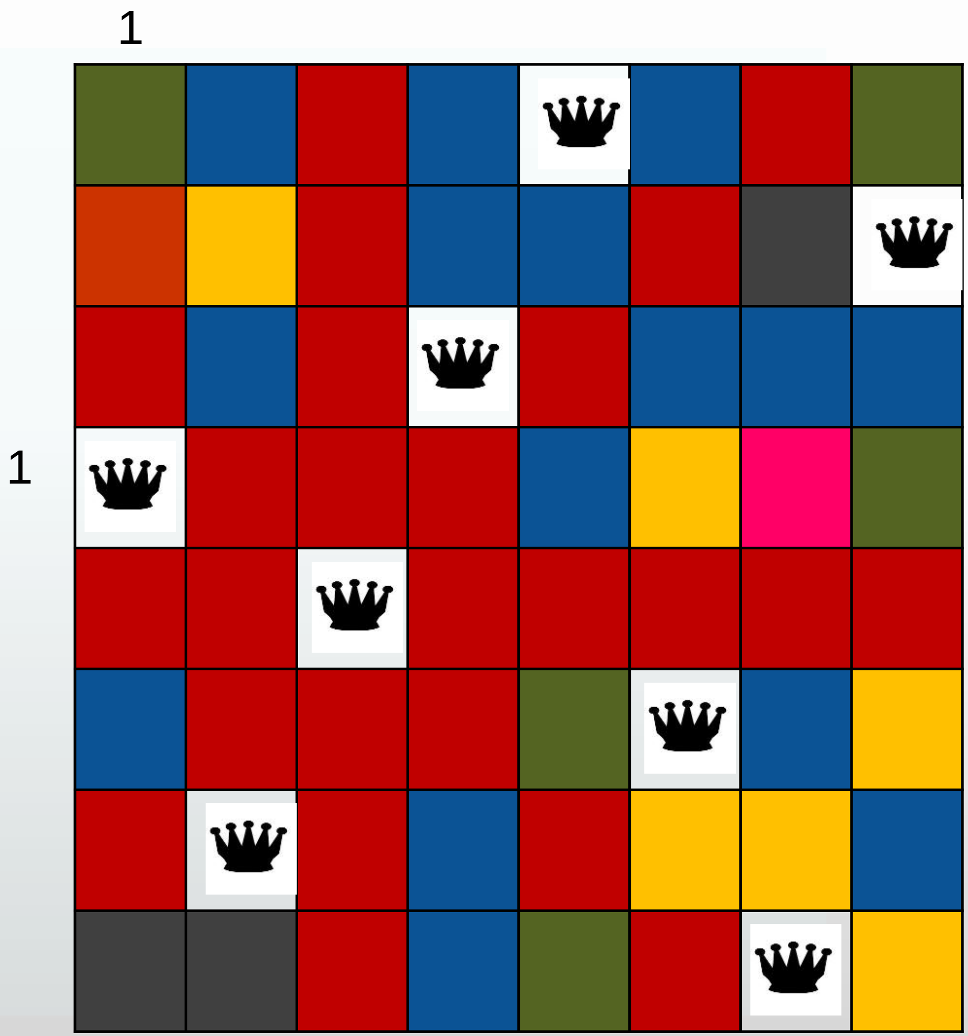
- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne

- Placer une reine dans une ligne ou colonne ayant le plus petit chiffre.



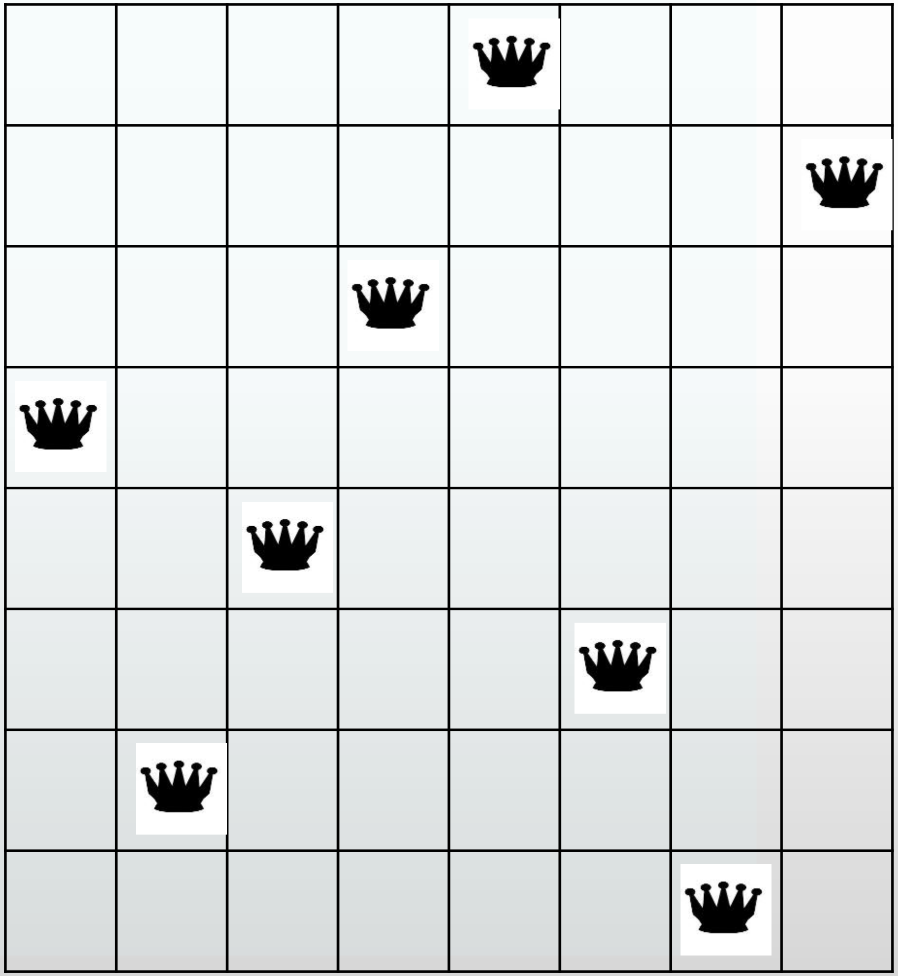
L’heuristique MRV sur les 8-Reines

- Compter le nombre de carrés non attaqués pour chaque ligne et chaque colonne
- Placer une reine dans une ligne ou colonne ayant le plus petit chiffre.
- Eliminer les carrés attaqués (rien à faire).



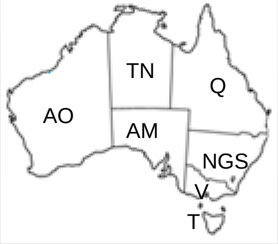
L’heuristique MRV sur les 8-Reines

- Solution (complète et cohérente).



Heuristique du degré : choix de la première variable

- Mais comment choisir la première variable à assigner ?
- Utiliser l'heuristique du degré (*degree heuristic*) : sélectionner la variable impliquée dans le plus grand nombre de contraintes.
- AM a un degré 5, c'est la plus contraignante.



Heuristique du degré : choix de la première variable

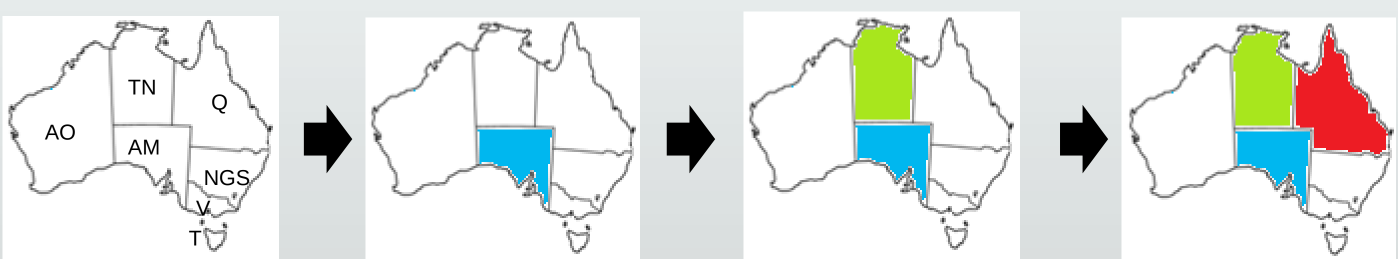
- Mais comment choisir la première variable à assigner ?

- Utiliser l'heuristique du degré (*degree heuristic*) : sélectionner la variable impliquée dans le plus grand nombre de contraintes.

- TN, Q et NGS sont de degré 3 : les plus contraignantes.

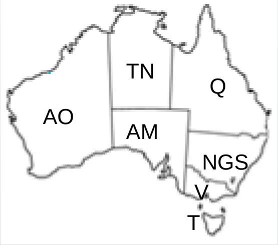
Heuristique du degré : choix de la première variable

- Mais comment choisir la première variable à assigner ?
- Utiliser l'heuristique du degré (*degree heuristic*) : sélectionner la variable impliquée dans le plus grand nombre de contraintes.
- Q a un degré 3.



Heuristique de la valeur la moins contraignante

- Q_2 : Une fois la variable choisie, dans quel ordre les valeurs doivent être essayées ?
- Elle privilégie la valeur qui exclut le moins de choix pour les variables voisines.
- L'heuristique essaie de laisser la plus grande liberté possible pour les assignations de variables restantes.

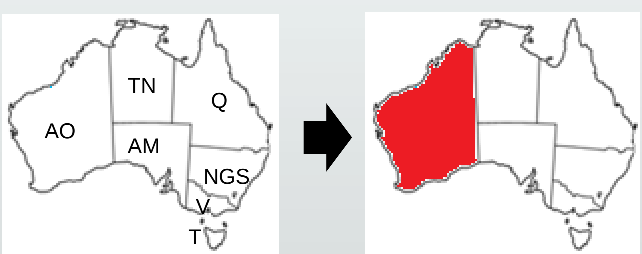


Heuristique de la valeur la moins contraignante

- Q$_2$: Une fois la variable choisie, dans quel ordre les valeurs doivent être essayées ?

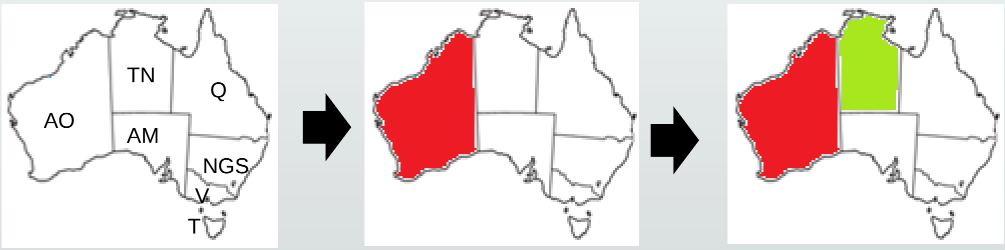
- Elle privilégie la valeur qui exclut le moins de choix pour les variables voisines.

- L'heuristique essaie de laisser la plus grande liberté possible pours les assignations de variables restantes.



Heuristique de la valeur la moins contraignante

- Q_2 : Une fois la variable choisie, dans quel ordre les valeurs doivent être essayées ?
- Elle privilégie la valeur qui exclut le moins de choix pour les variables voisines.
- L'heuristique essaie de laisser la plus grande liberté possible pours les assignations de variables restantes.

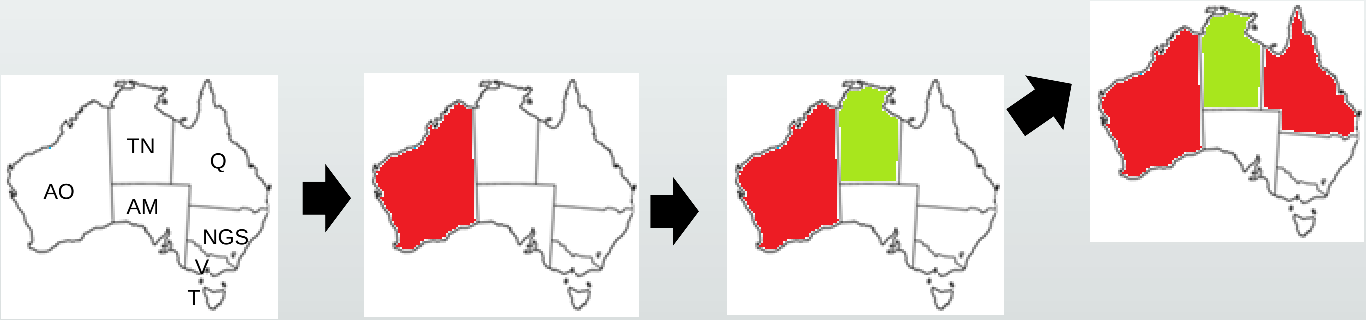


Heuristique de la valeur la moins contraignante

- Q_2 : Une fois la variable choisie, dans quel ordre les valeurs doivent être essayées ?

- Elle privilégie la valeur qui exclut le moins de choix pour les variables voisines.

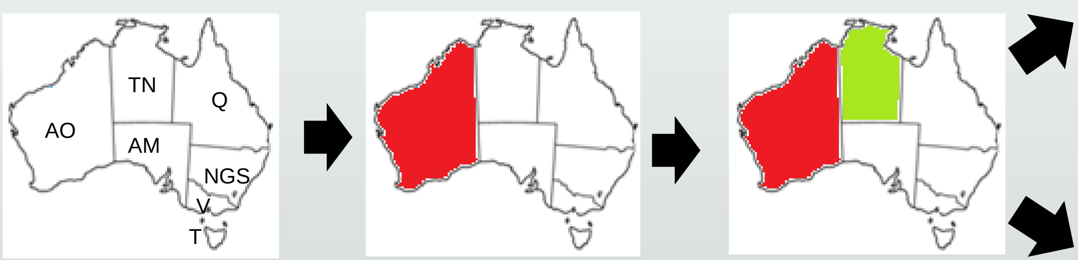
- L’heuristique essaie de laisser la plus grande liberté possible pours les assignations de variables restantes.

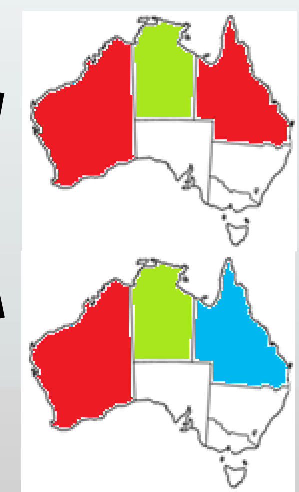


1 valeur restante pour AM

Heuristique de la valeur la moins contraignante

- Q_2 : Une fois la variable choisie, dans quel ordre les valeurs doivent être essayées ?
- Elle privilégie la valeur qui exclut le moins de choix pour les variables voisines.
- L'heuristique essaie de laisser la plus grande liberté possible pours les assignations de variables restantes.





1 valeur restante pour AM

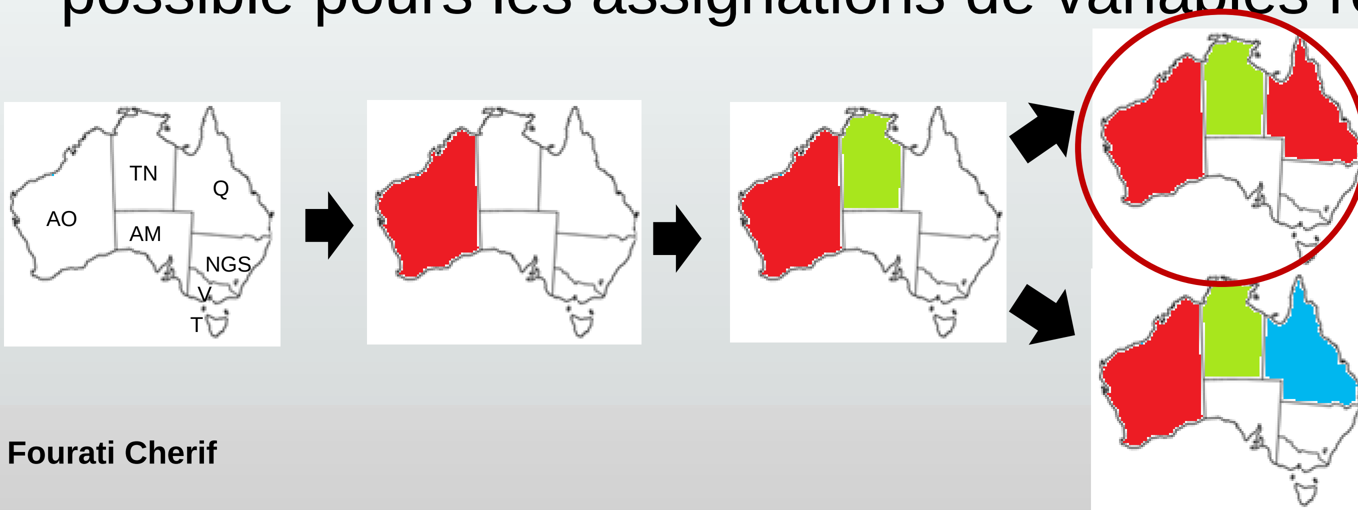
0 valeurs restantes pour AM

Heuristique de la valeur la moins contraignante

- Q$_2$: Une fois la variable choisie, dans quel ordre les valeurs doivent être essayées ?

- Elle privilégie la valeur qui exclut le moins de choix pour les variables voisines.

- L’heuristique essaie de laisser la plus grande liberté possible pours les assignations de variables restantes.



1 valeur restante pour AM

0 valeurs restantes pour AM

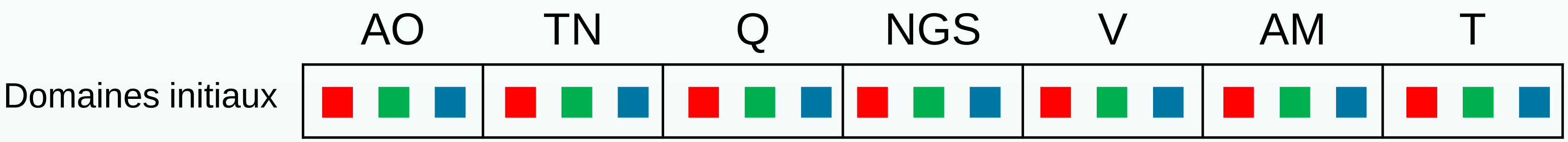
Association de l'exploration et de l'inférence

- Q3 : Comment réduire l'espace d'états des assignations ?
- Utiliser l'inférence au cours d'une exploration : chaque fois qu'une variable est assignée, de nouvelles réductions de domaines sur les variables voisines peuvent être inférées $\Rightarrow$ plus puissante que la cohérence locale qui est réalisée avant l'exploration.
- Algorithmes disponibles :
  - Vérification en avant (forward checking).
  - Propagation de contraintes : consistance d'arcs (Arc consistency).

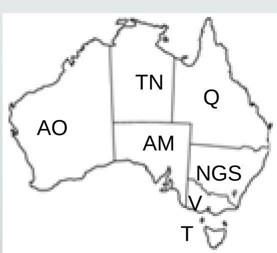
Exploration et inférence : Vérification en avant (*forward checking*)

- Chaque fois qu'une variable X est assignée, elle détermine sa cohérence d'arc :
  - Vérifier chaque variable non assignée Y connectée à X par une contrainte.
  - Supprimer du domaine de Y toutes les valeurs non cohérentes avec la valeur assignée à X.
- L'idée est de ne garder que les valeurs légales pour les variables non assignées et de mettre fin à le recherche lorsqu'une variable n'a plus de valeurs légales.

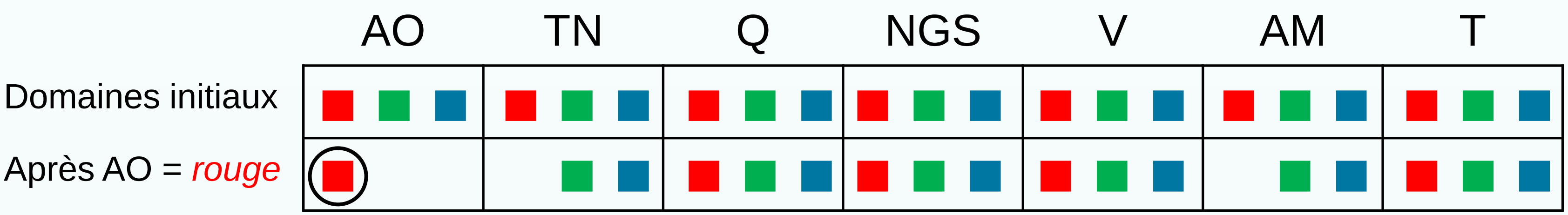
Exemple : vérification en avant pour le CSP coloration de la carte



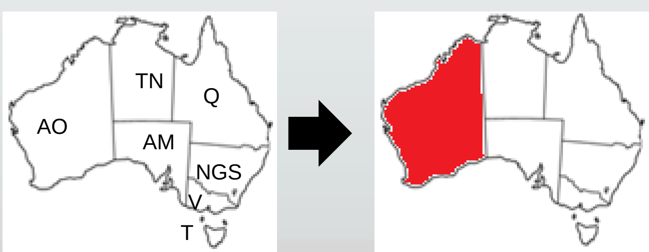
- Initialement la carte est vide et toutes les 3 couleurs sont possibles pour chacune des régions.



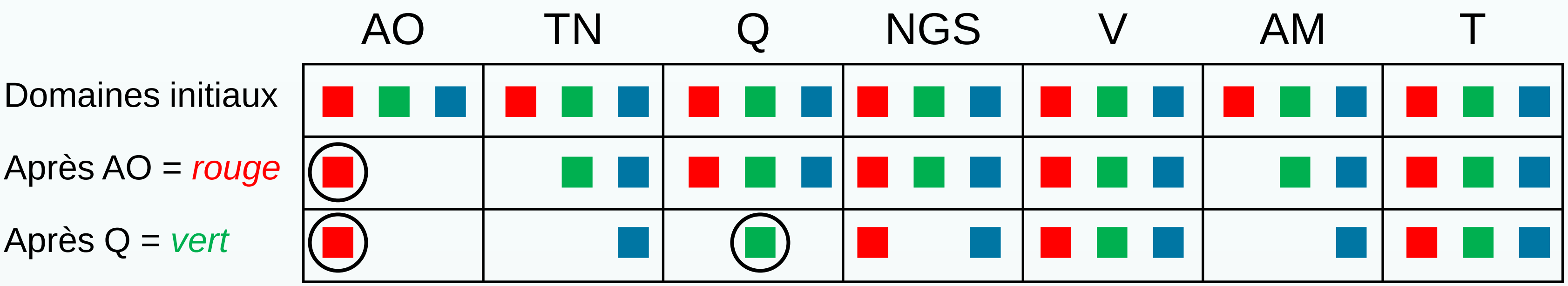
Exemple : vérification en avant pour le CSP coloration de la carte



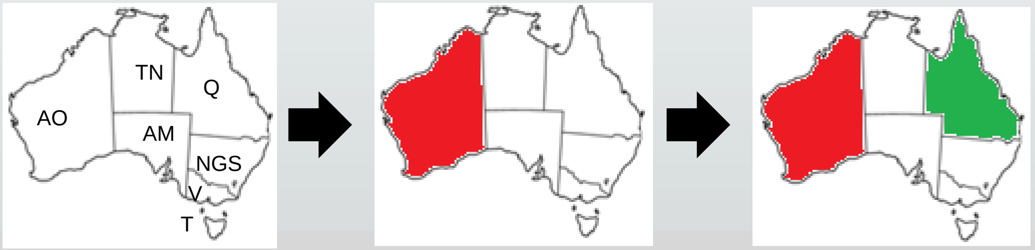
- Après l’assignation de AO = rouge, les domaines de NT et AM sont réduits à deux valeurs : vert et bleu.



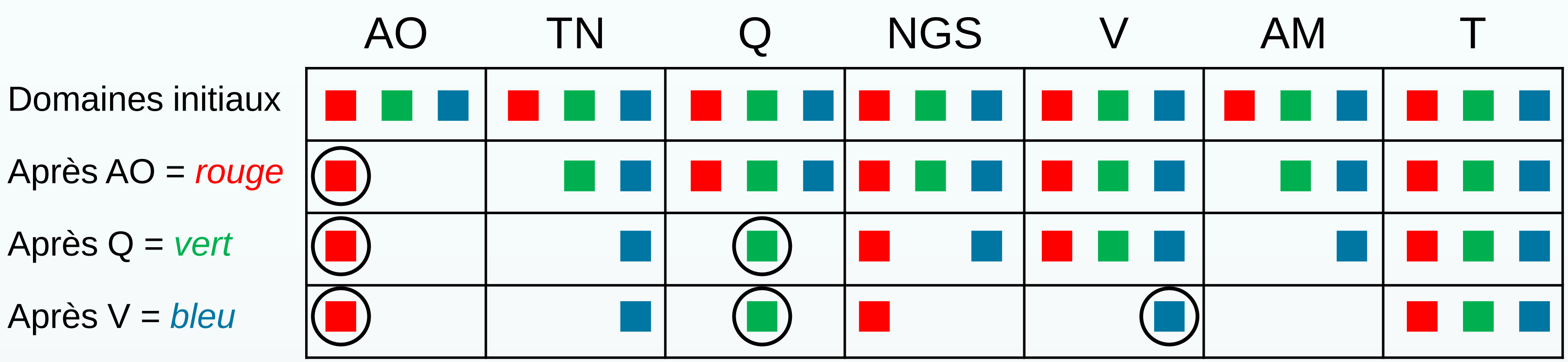
Exemple : vérification en avant pour le CSP coloration de la carte



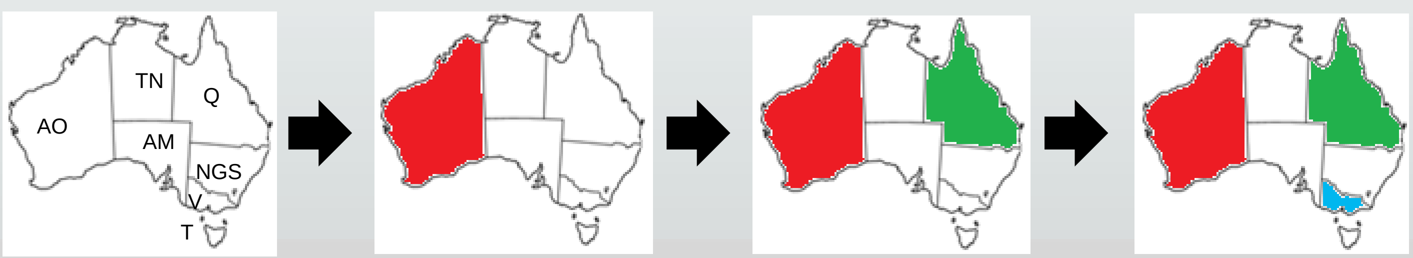
- Après l'assignation de AO et Q, les domaines de NT et AM sont réduits à une seule valeur : bleu.



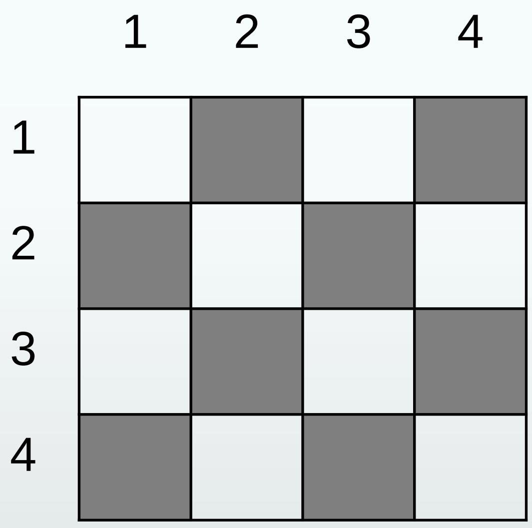
Exemple : vérification en avant pour le CSP coloration de la carte

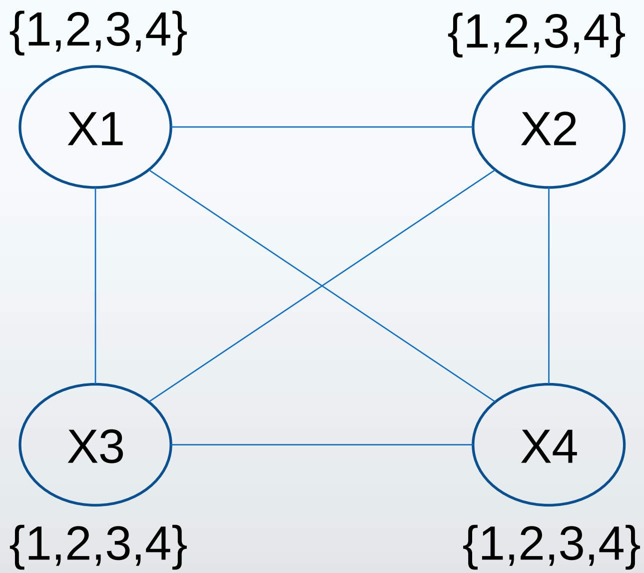


- Après l’assignation de AO, Q et V, le domaine de AM est devenu vide : détection d’une incohérence d’assignation, un retour arrière est alors appliqué.



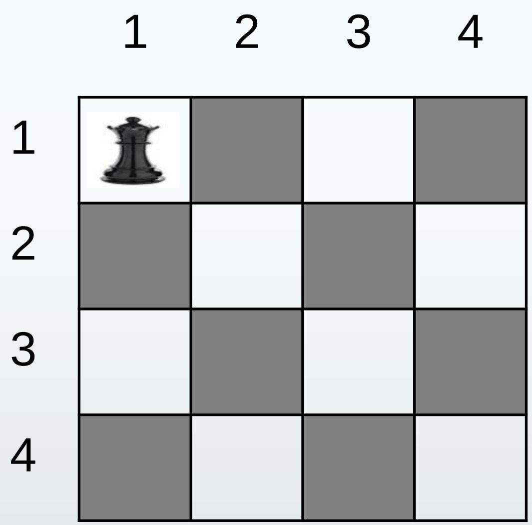
Vérification en avant : exemple des 4 reines

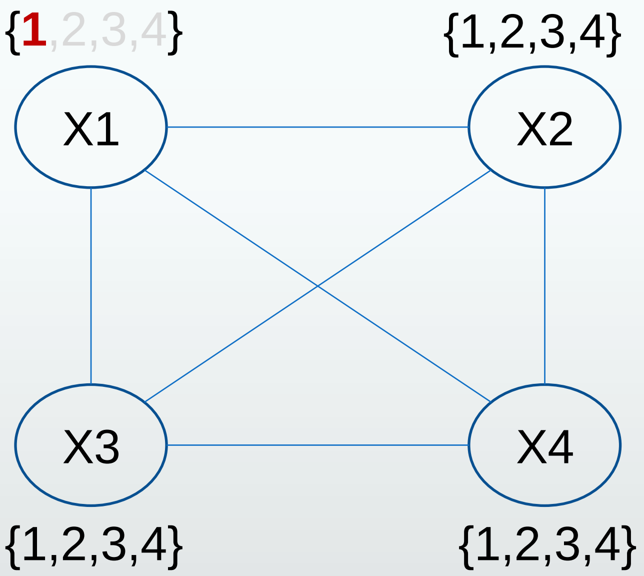




Vérification en avant : exemple des 4 reines

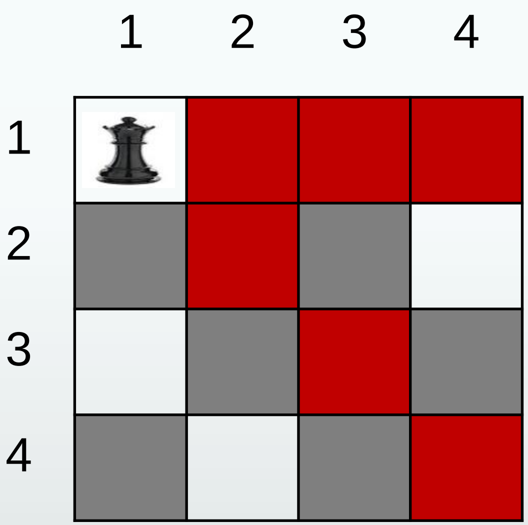
Placer une reine

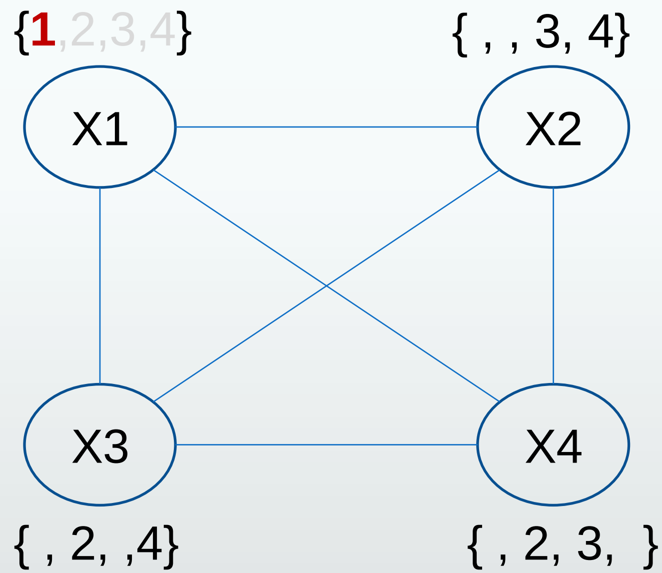




Vérification en avant : exemple des 4 reines

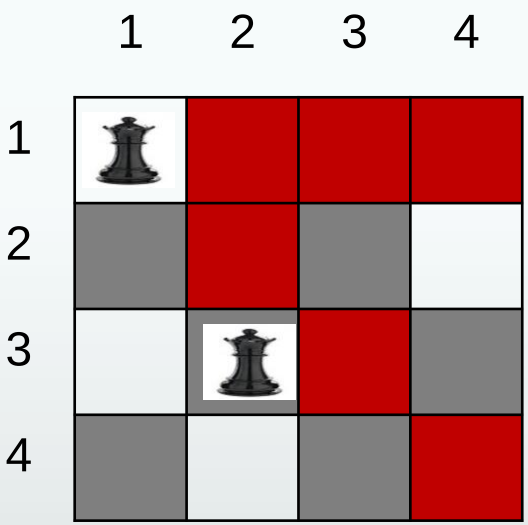
Retirer les carrés attaqués des domaines des reines restantes.

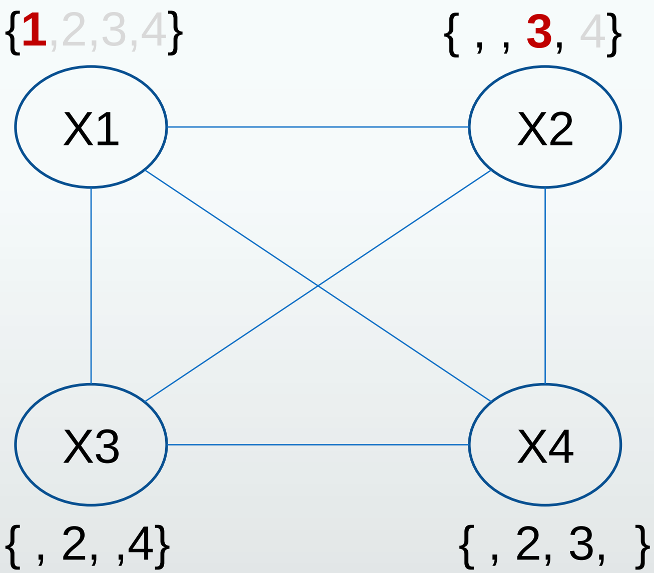




Vérification en avant : exemple des 4 reines

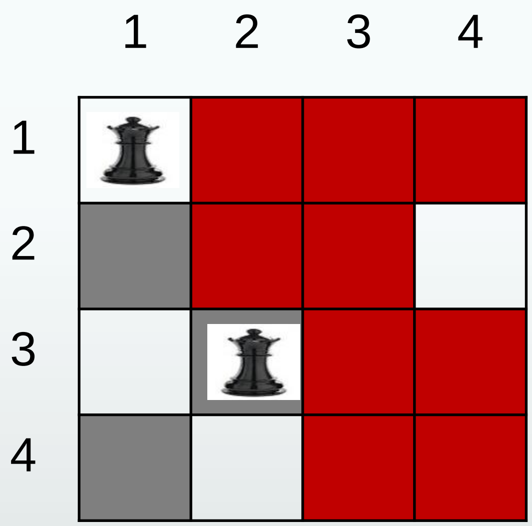
Placer une reine

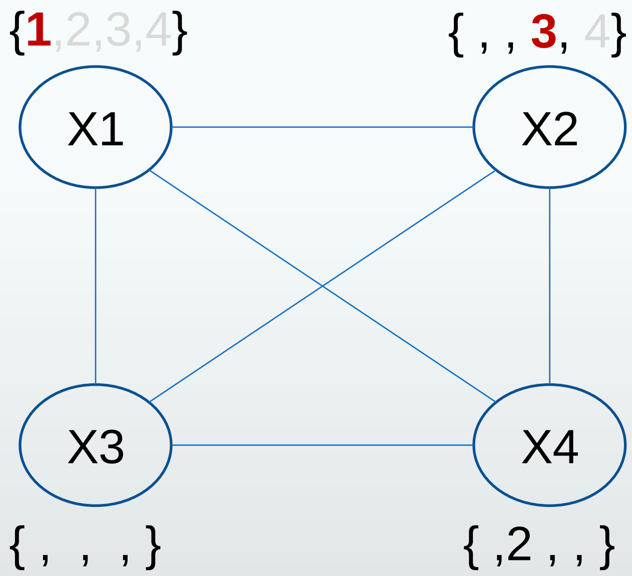




Vérification en avant : exemple des 4 reines

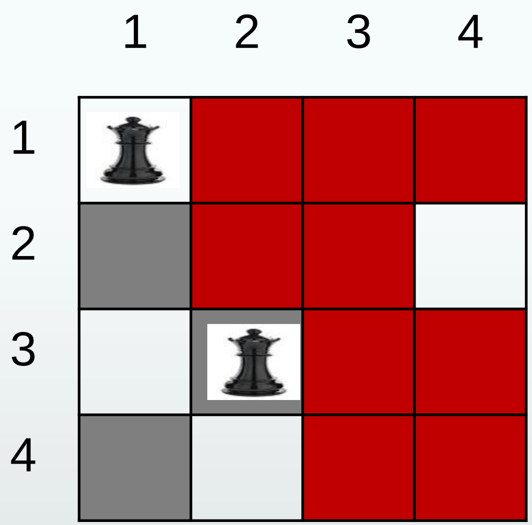
Retirer les carrés attaqués des domaines des reines restantes.

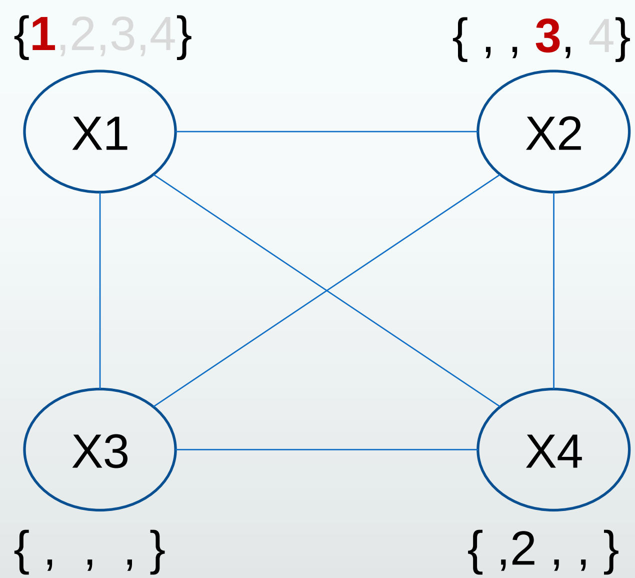




Vérification en avant : exemple des 4 reines

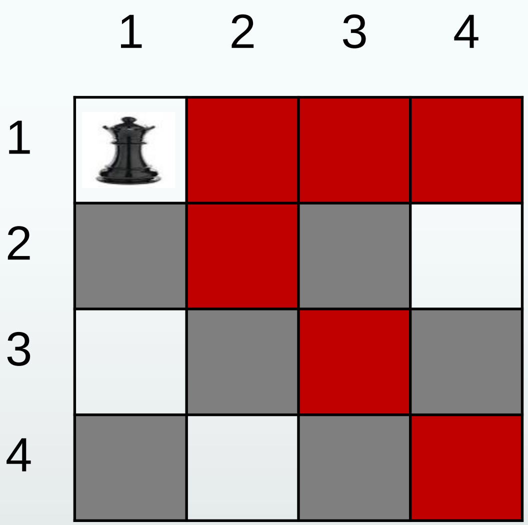
Domaine de X3 est vide

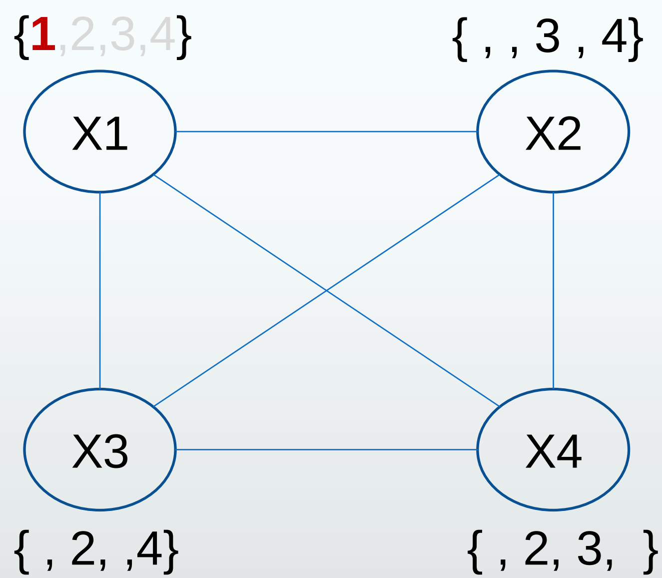




Vérification en avant : exemple des 4 reines

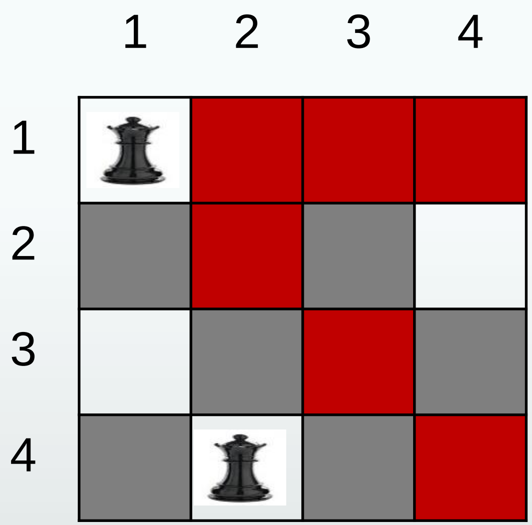
Retour arrière à la dernière variable assignée : X2

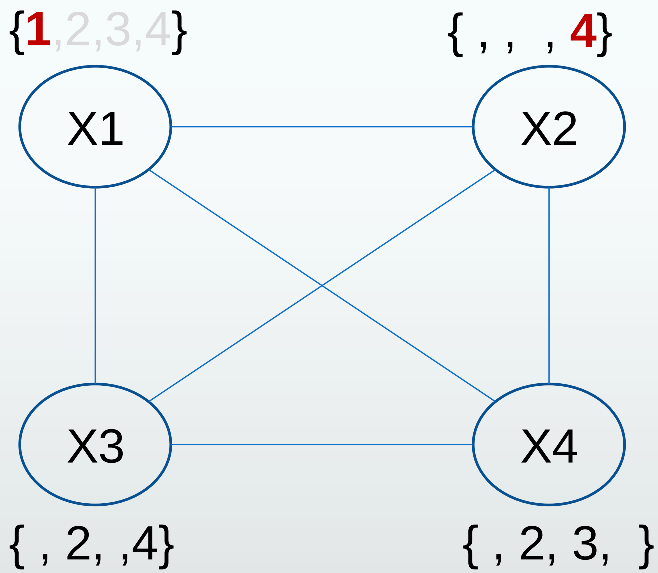




Vérification en avant : exemple des 4 reines

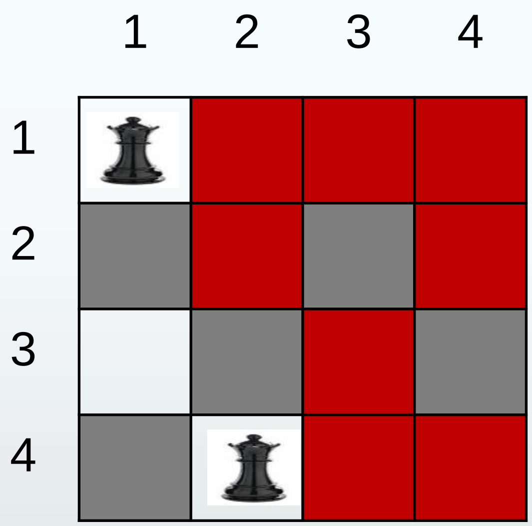
Placer une reine

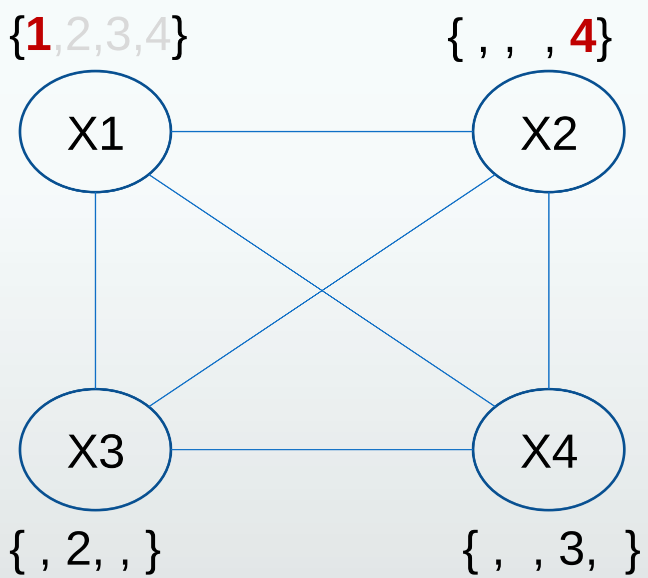




Vérification en avant : exemple des 4 reines

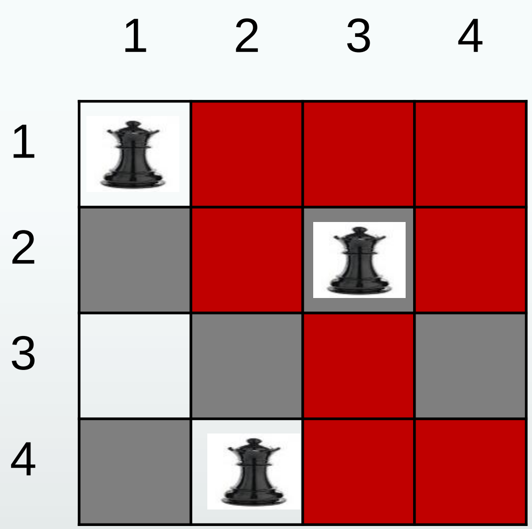
Retirer les carrés attaqués des domaines des reines restantes.

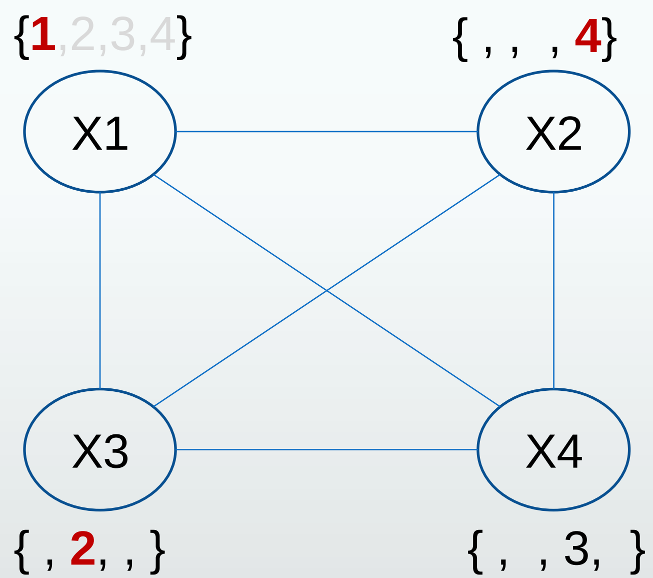




Vérification en avant : exemple des 4 reines

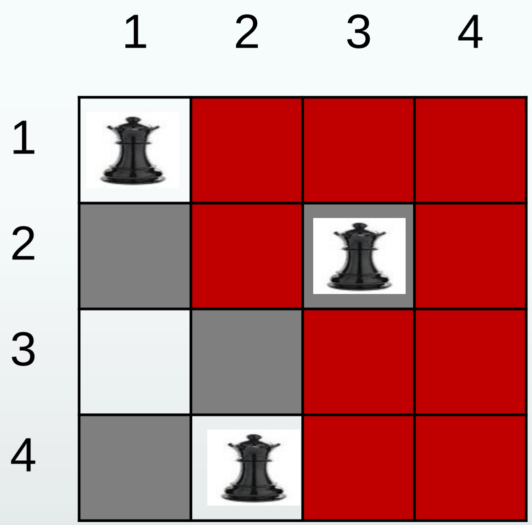
Placer une reine

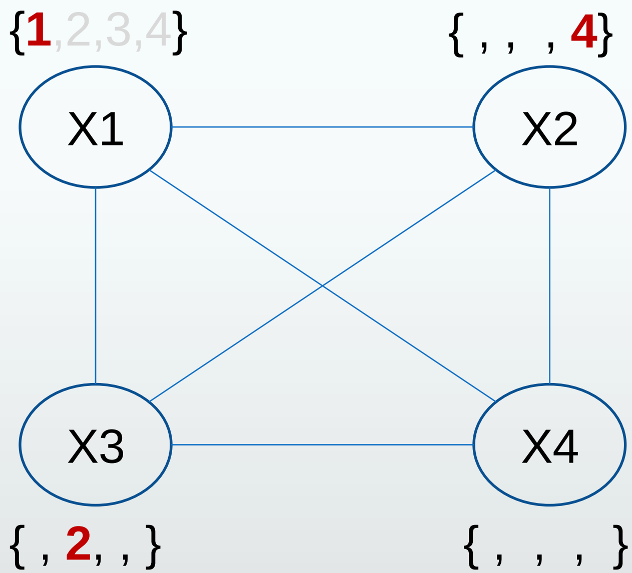




Vérification en avant : exemple des 4 reines

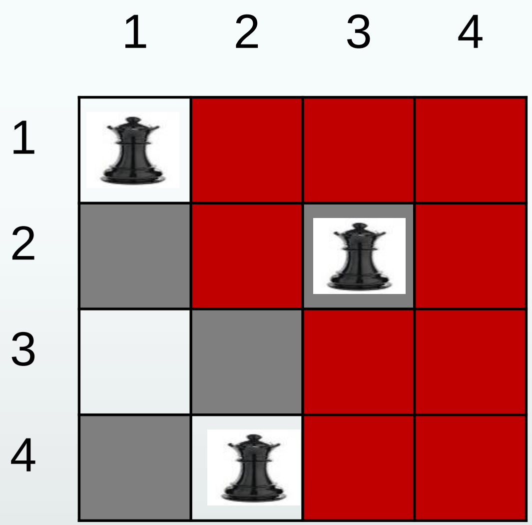
Retirer les carrés attaqués des domaines des reines restantes.

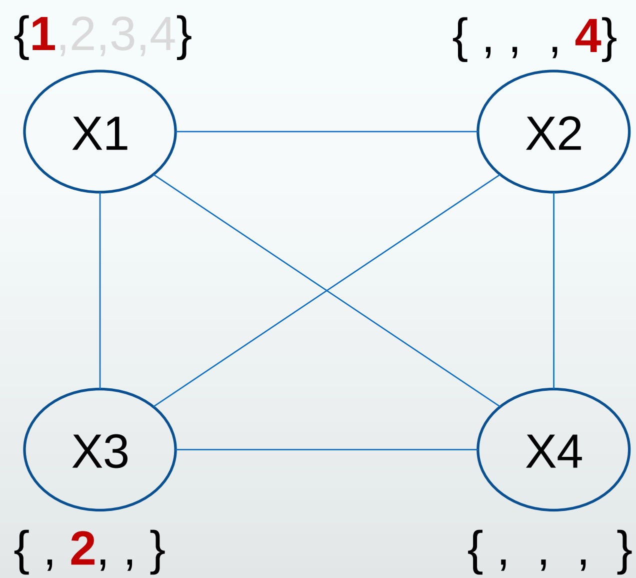




Vérification en avant : exemple des 4 reines

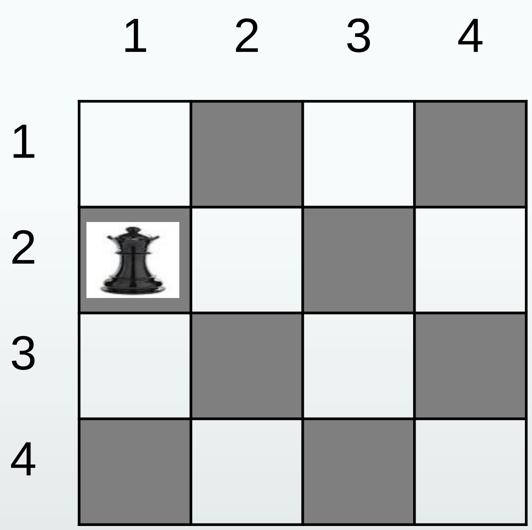
Domaine de X4 est vide ⇒ retour arrière à la dernière variable assignée : X3, mais aucune valeur disponible, alors retour à X2 : pas de valeurs donc X1.

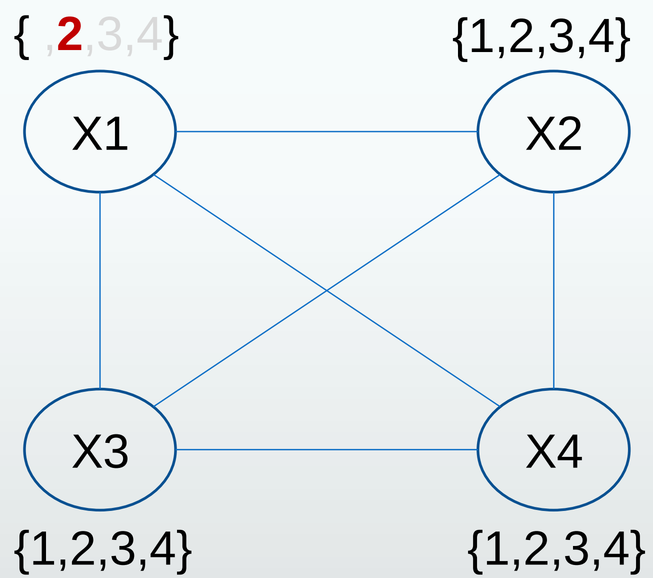




Vérification en avant : exemple des 4 reines

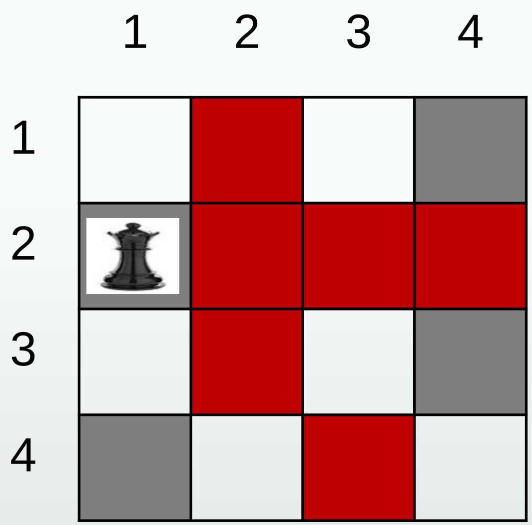
Placer une reine

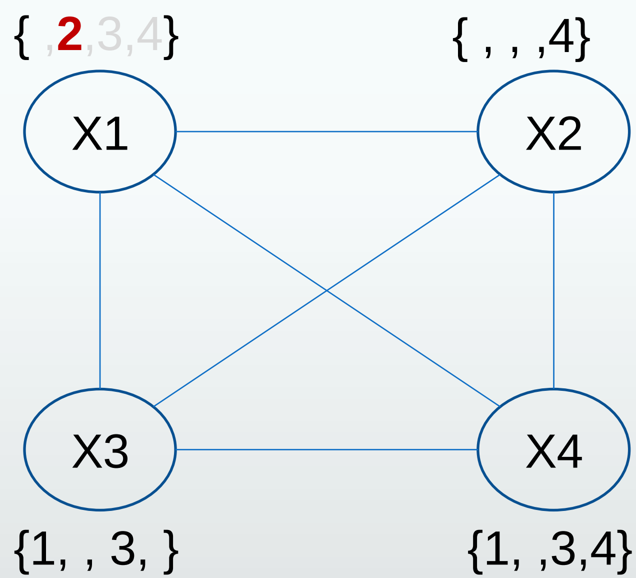




Vérification en avant : exemple des 4 reines

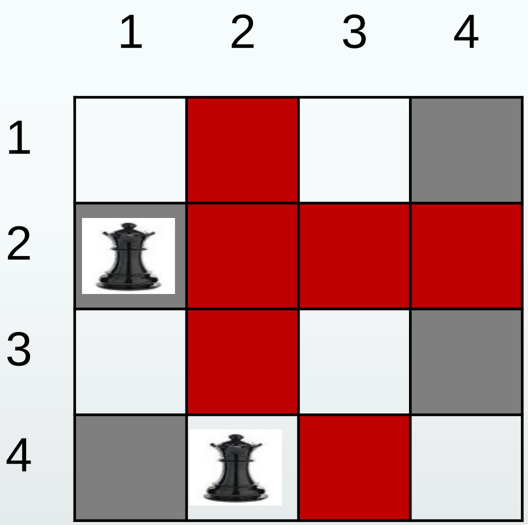
Retirer les carrés attaqués des domaines des reines restantes.

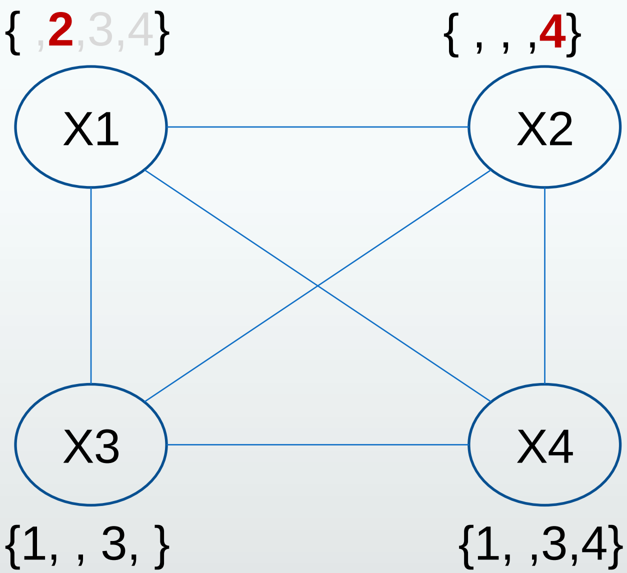




Vérification en avant : exemple des 4 reines

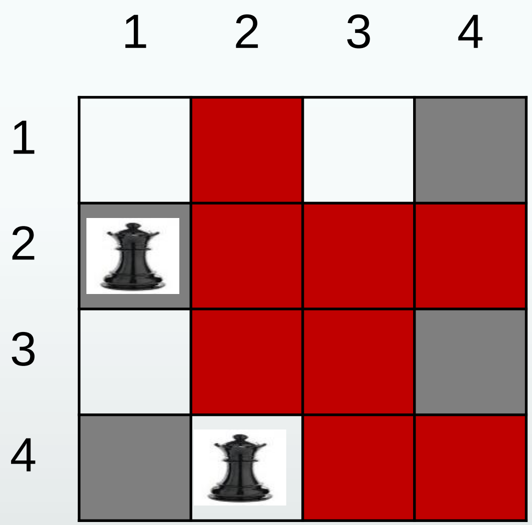
Placer une reine

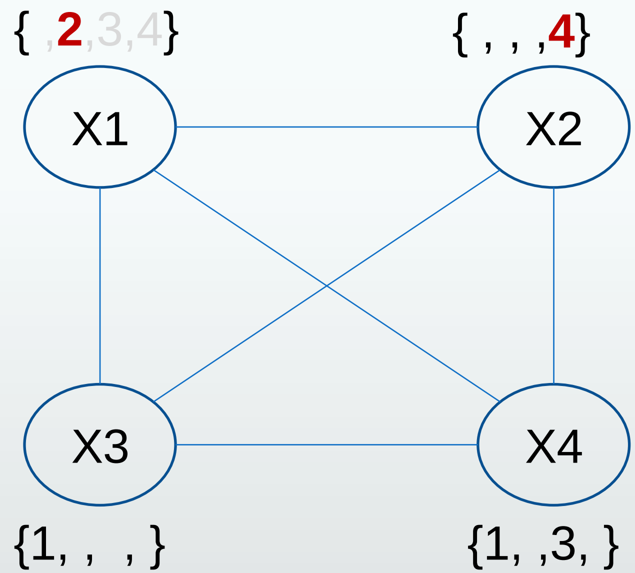




Vérification en avant : exemple des 4 reines

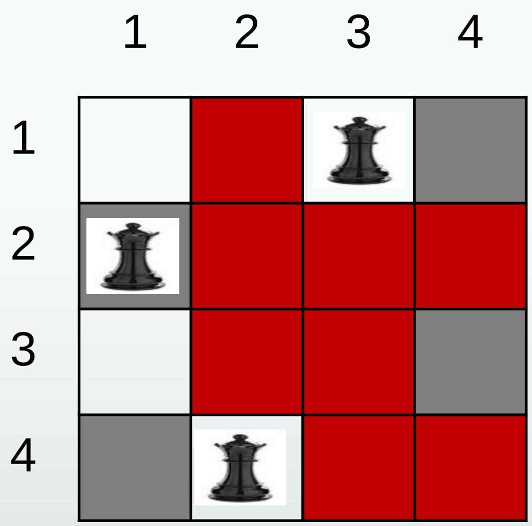
Retirer les carrés attaqués des domaines des reines restantes.

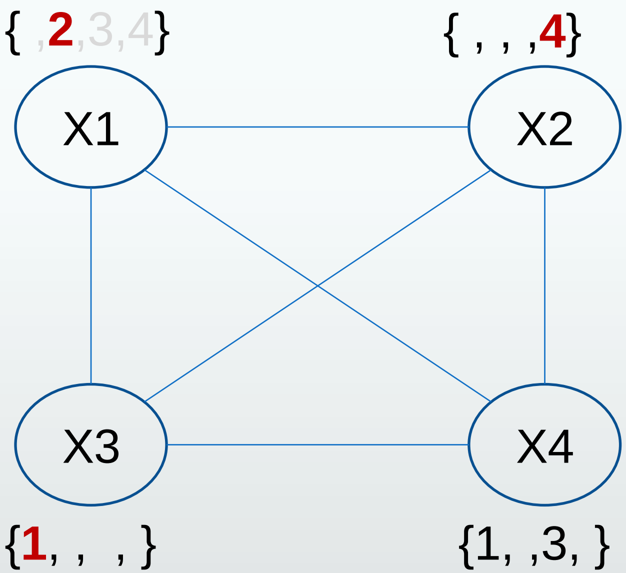




Vérification en avant : exemple des 4 reines

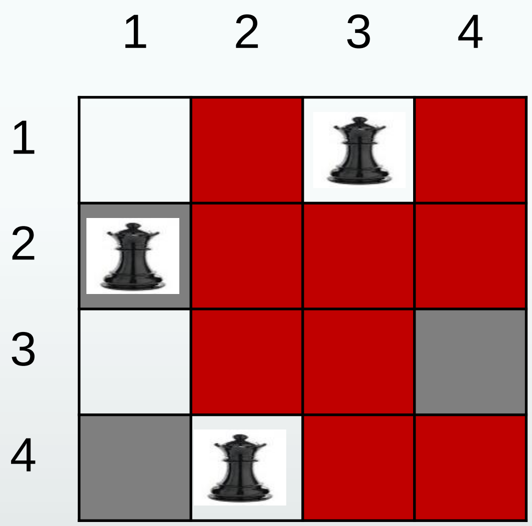
Placer une reine

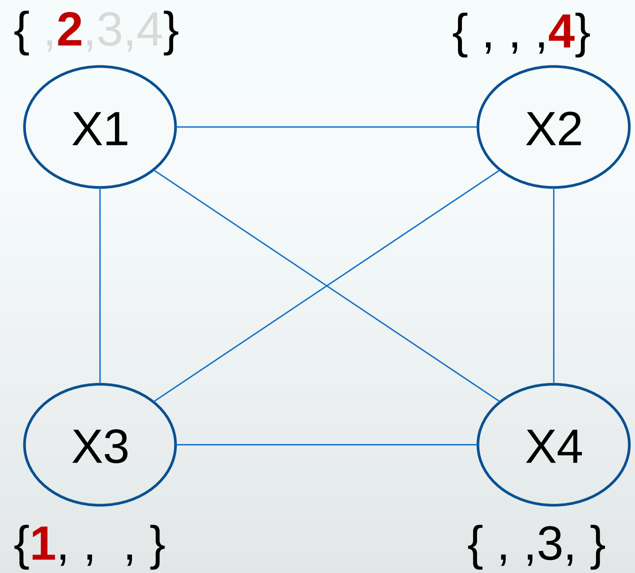




Vérification en avant : exemple des 4 reines

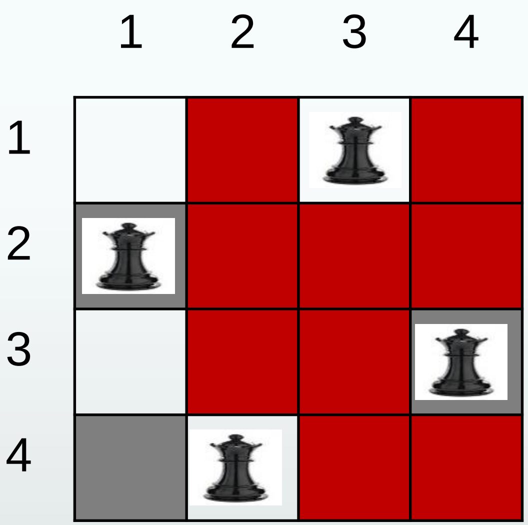
Retirer les carrés attaqués des domaines des reines restantes.

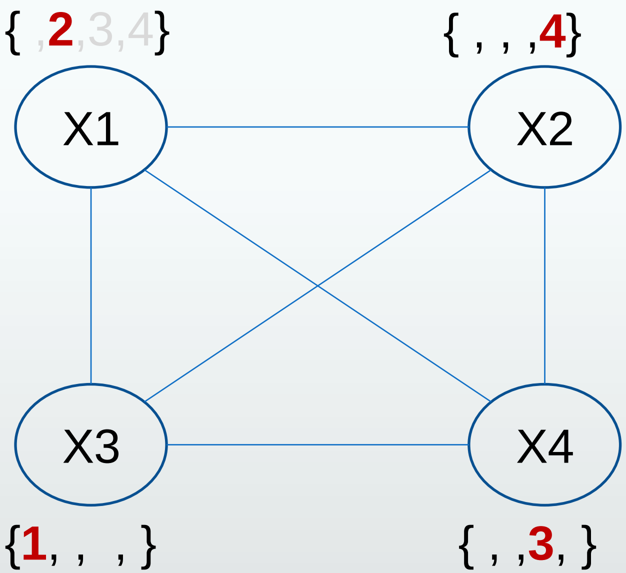




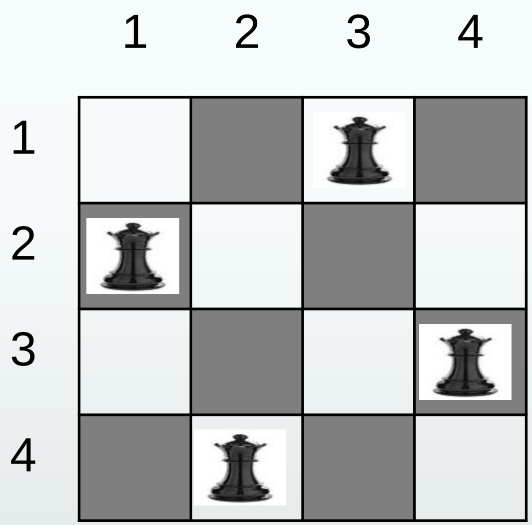
Vérification en avant : exemple des 4 reines

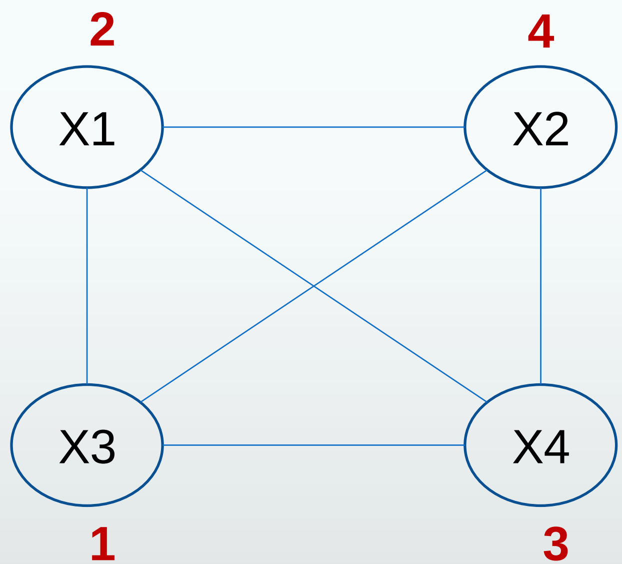
Placer une reine





Vérification en avant : exemple des 4 reines



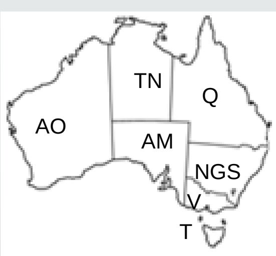


Exploration en avant et MRV

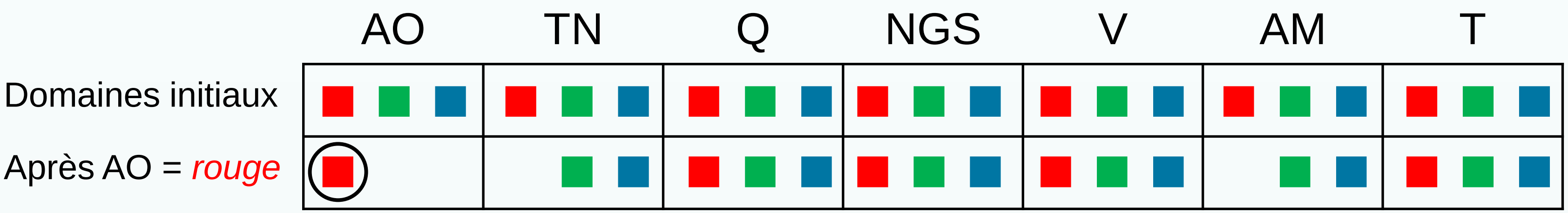
- Il est possible d’améliorer l’efficacité de la recherche en combinant l’heuristique MRV et l’exploration en avant.
- D’ailleurs, la vérification en avant peut être considérée comme une manière efficace de calculer incrémentalement l’information dont a besoin le MRV.

Exemple : vérification en avant & MRV pour le CSP coloration de la carte

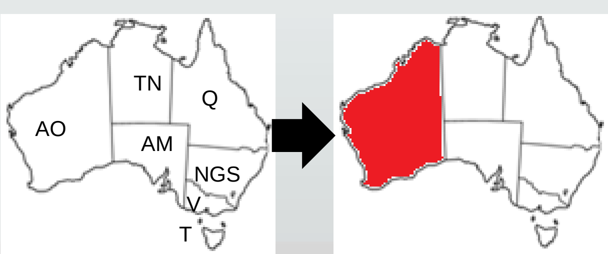
	AO	TN	Q	NGS	V	AM	T
Domaines initiaux							



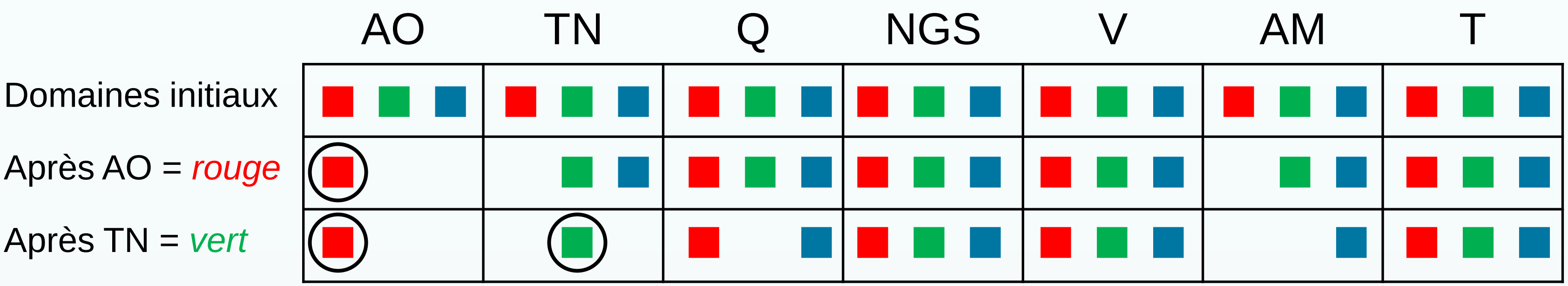
Exemple : vérification en avant & MRV pour le CSP coloration de la carte



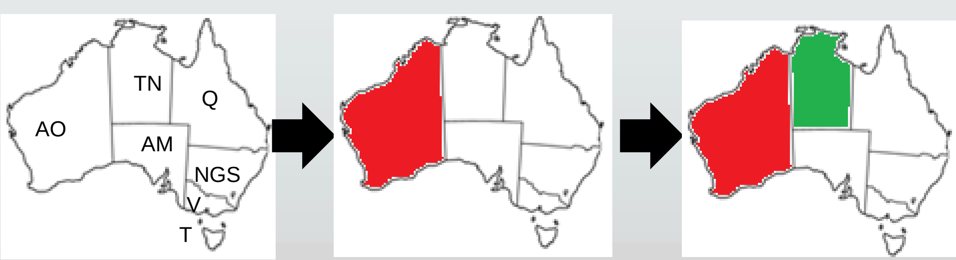
- Après le choix de AO = rouge, l'assignation contraigne TN et AM, on commence donc par traiter ces variables.



Exemple : vérification en avant & MRV pour le CSP coloration de la carte



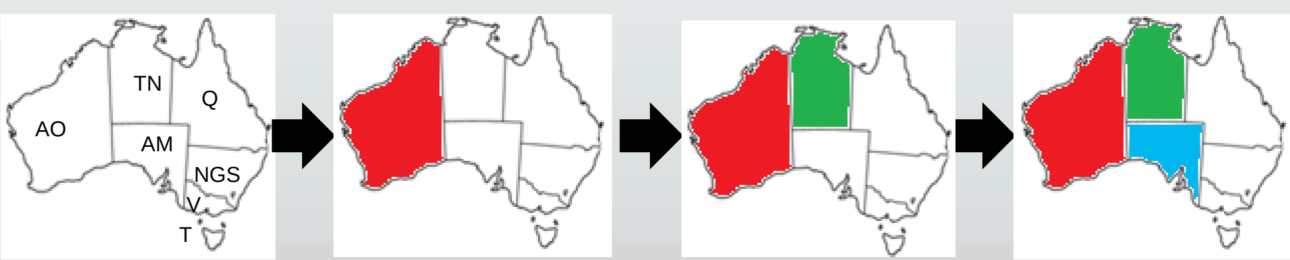
- On choisi TN = vert.



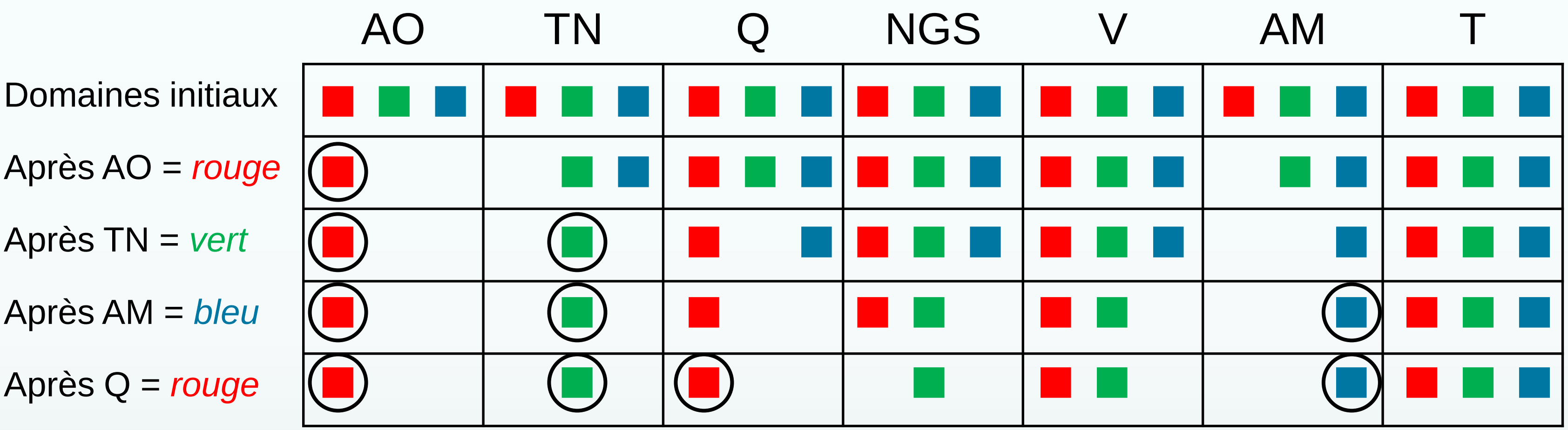
Exemple : vérification en avant & MRV pour le CSP coloration de la carte

	AO	TN	Q	NGS	V	AM	T
Domaines initiaux	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦
Après AO = *rouge*	🟥	🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟩 🟦	🟥 🟩 🟦
Après TN = *vert*	🟥	🟩	🟥 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟦	🟥 🟩 🟦
Après AM = *bleu*	🟥	🟩	🟥	🟥 🟩	🟥 🟩	🟦	🟥 🟩 🟦

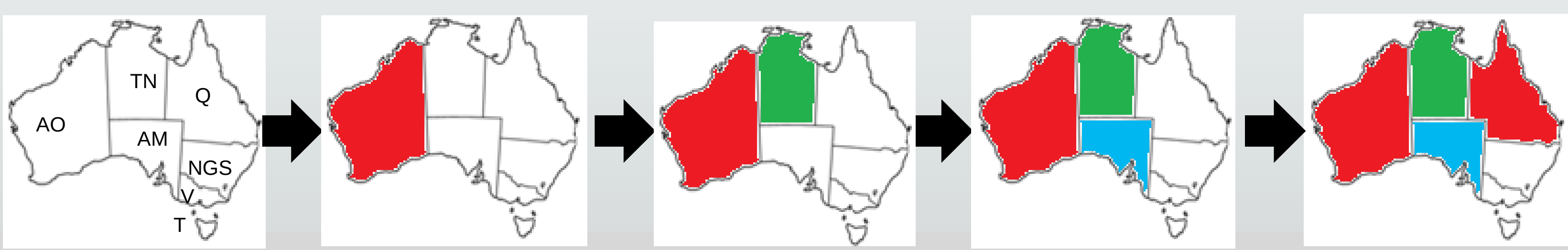
- On choisi AM=bleu.



Exemple : vérification en avant & MRV pour le CSP coloration de la carte



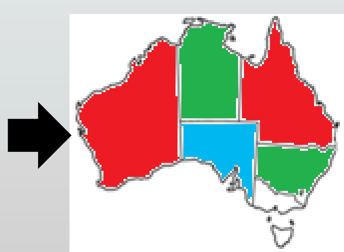
- On choisi Q = rouge : plus contraignante.



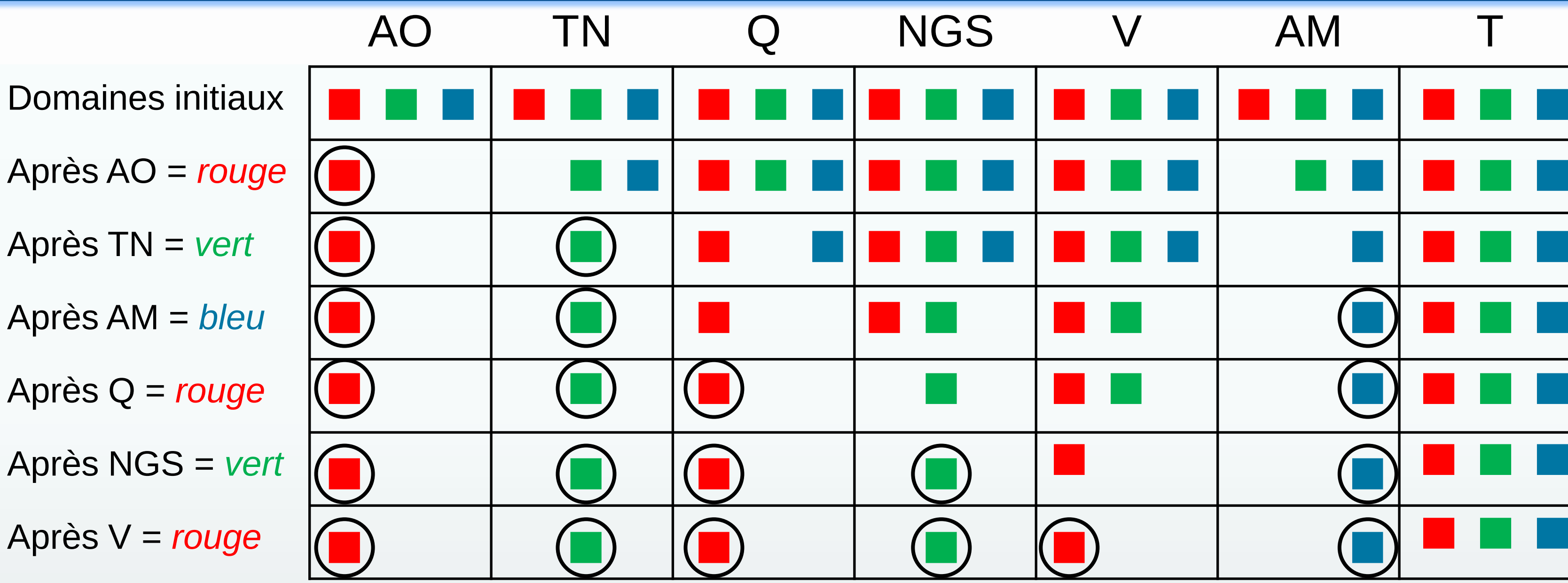
Exemple : vérification en avant & MRV pour le CSP coloration de la carte

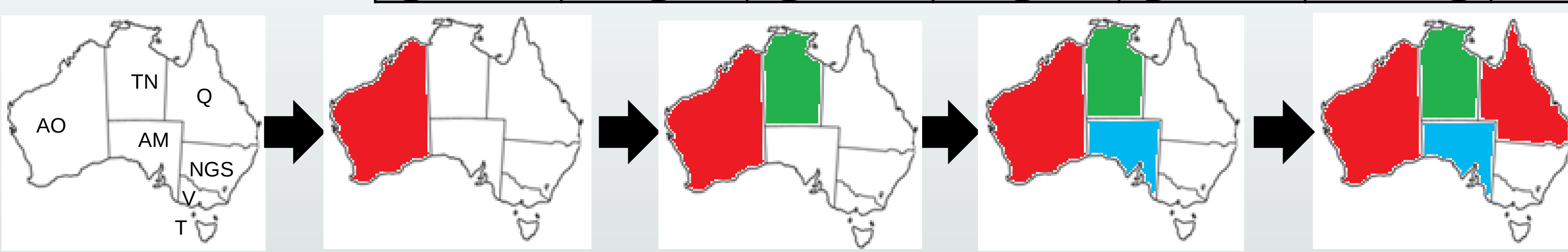
	AO	TN	Q	NGS	V	AM	T
Domaines initiaux	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦
Après AO = *rouge*	🟥	🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟩 🟦	🟥 🟩 🟦
Après TN = *vert*	🟥	🟩	🟥 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟦	🟥 🟩 🟦
Après AM = *bleu*	🟥	🟩	🟥	🟥 🟩	🟥 🟩	🟦	🟥 🟩 🟦
Après Q = *rouge*	🟥	🟩	🟥	🟩	🟥 🟩	🟦	🟥 🟩 🟦
Après NGS = *vert*	🟥	🟩	🟥	🟩	🟥	🟦	🟥 🟩 🟦

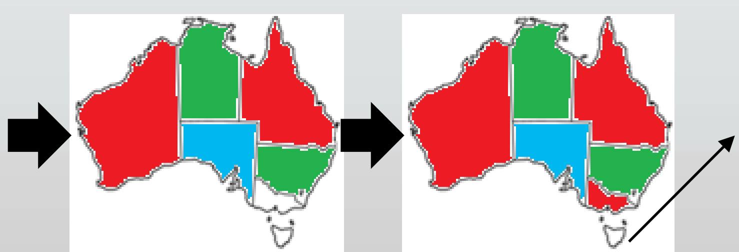
- On choisi NGS = vert.



Exemple : vérification en avant & MRV pour le CSP coloration de la carte







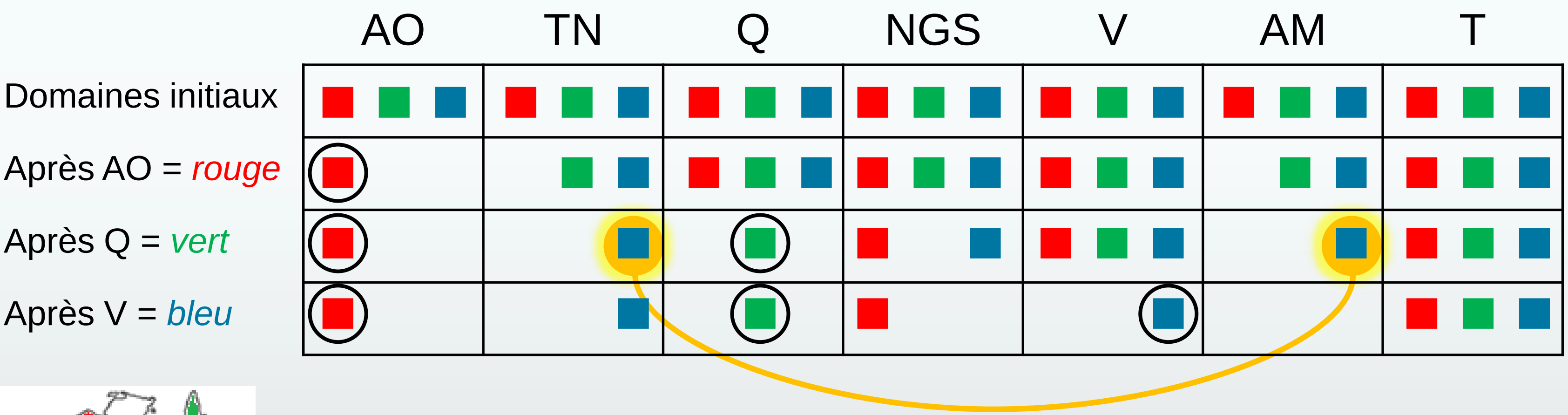
N'importe quelle valeur pour Tasmanie résout le CSP

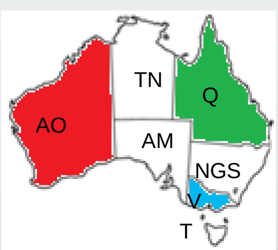
Limites de la vérification avant

- La vérification en avant ne permet pas de détecter des incohérences inévitables :

Limites de la vérification avant

- La vérification en avant ne permet pas de détecter des incohérences inévitables :

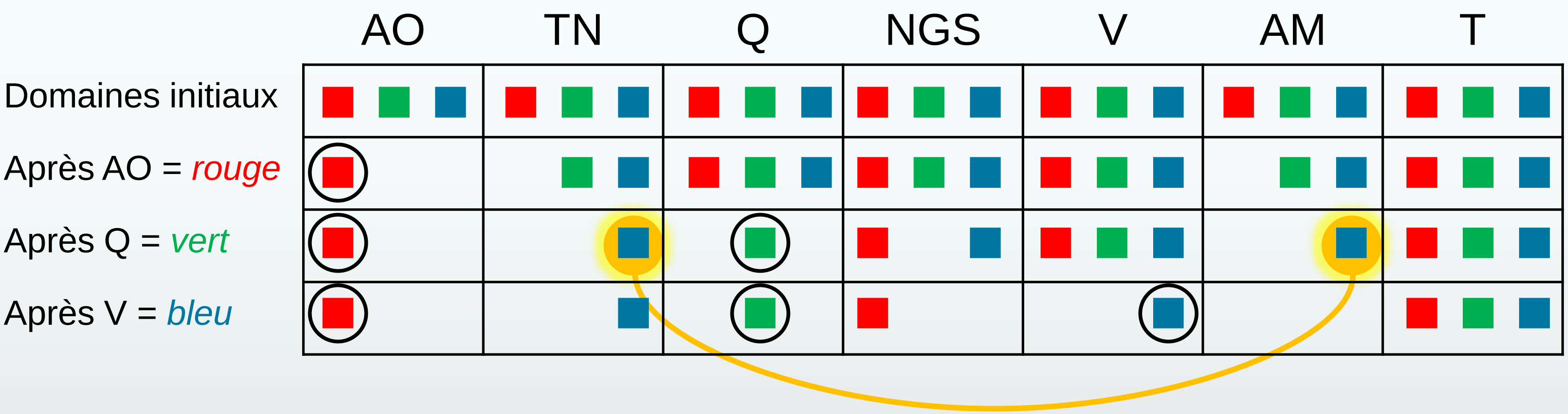




A partir d'ici aucune assignation cohérente n'est possible.

Limites de la vérification avant

- La vérification en avant ne permet pas de détecter des incohérences inévitables :



A partir d'ici aucune assignation cohérente n'est possible.

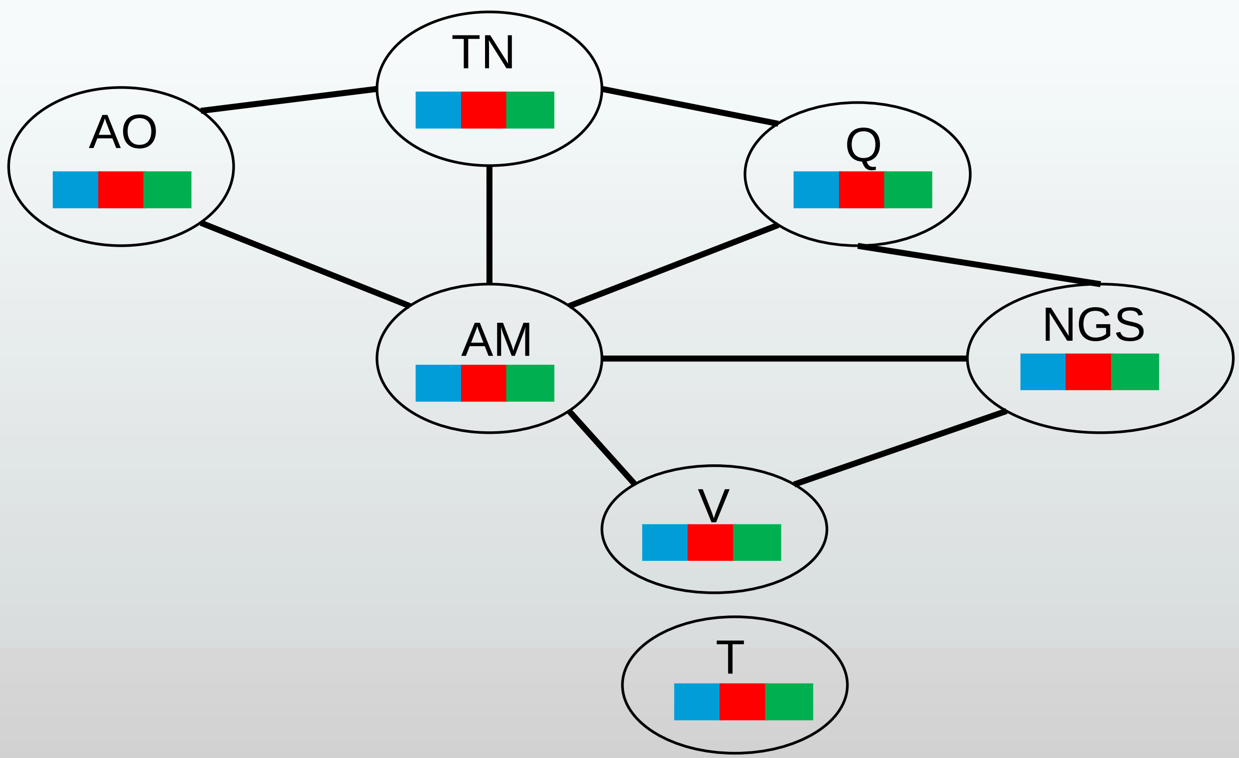
- Elle rend la variable courante arc-cohérente, mais pas toutes les autres !

Propagation de contraintes : consistance d'arc

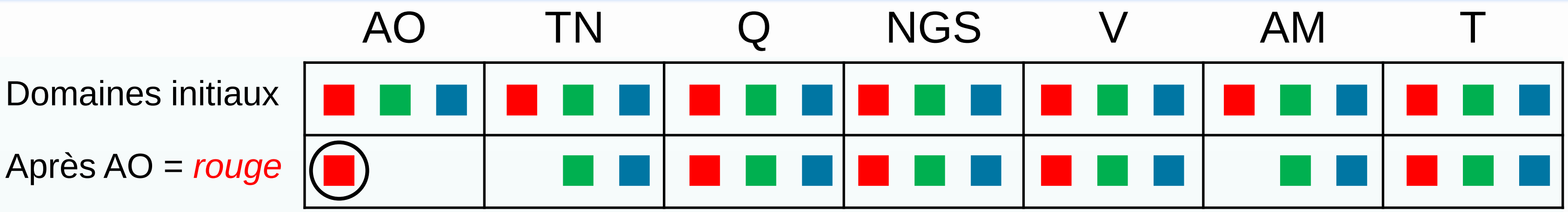
- Prendre en considération les implications d'une contrainte afin de restreindre la recherche.
- Rappel : un arc de *X* vers *Y* est cohérent $X \rightarrow Y$ si pour toute valeur de X, il existe au moins une valeur cohérente (permise) pour *Y*.
- Le maintien de la cohérence d'arc (MAC) appelle l'algorithme AC-3 chaque fois que X_i est assignée :
  - La file contient uniquement les arcs (X_j,X_i) pour toute variable non assignée X_j avec une contrainte (X_j,X_i).
  - A partir de là, AC-3 effectue une propagation de contraintes habituelle.
  - AC-3 échoue si une incohérence est détectée : retour arrière.

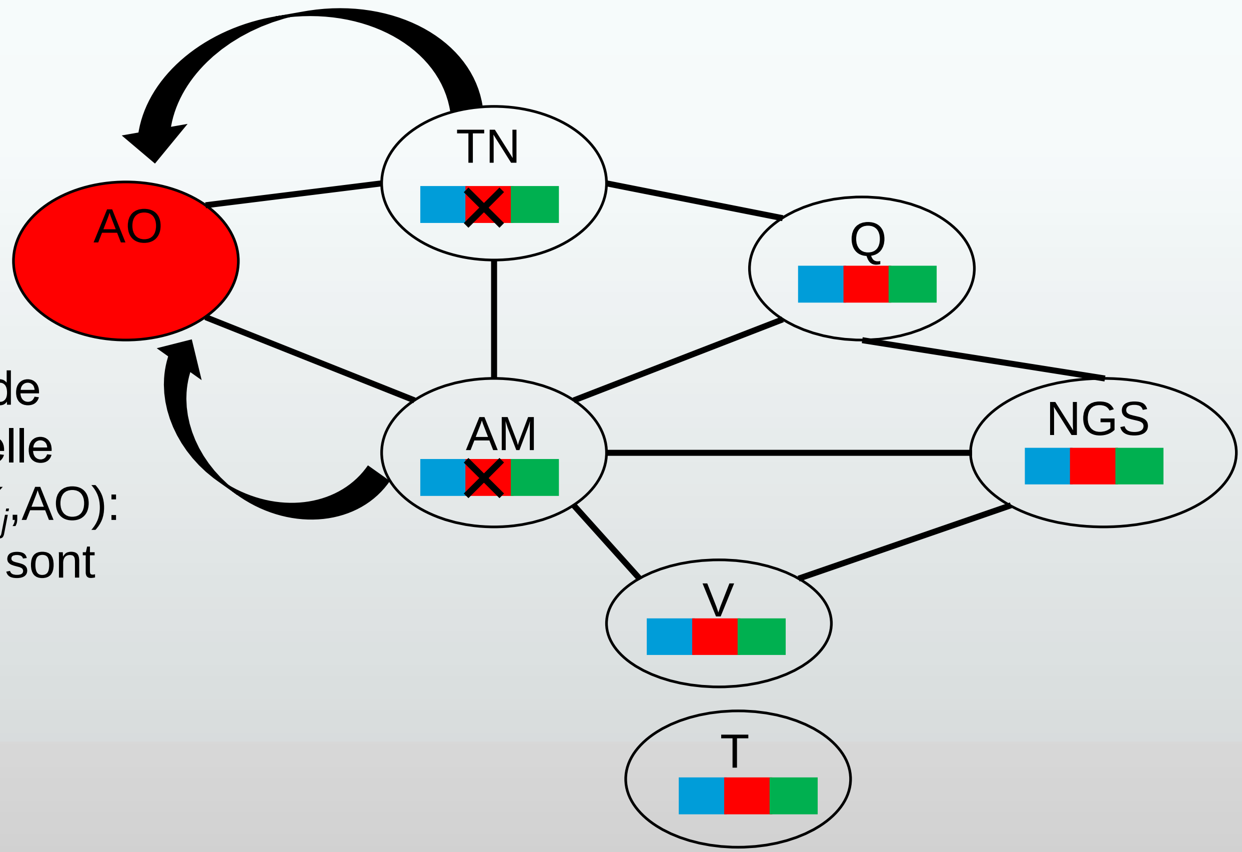
MAC sur la carte de l’Australie

Domaines initiaux



MAC sur la carte de l'Australie



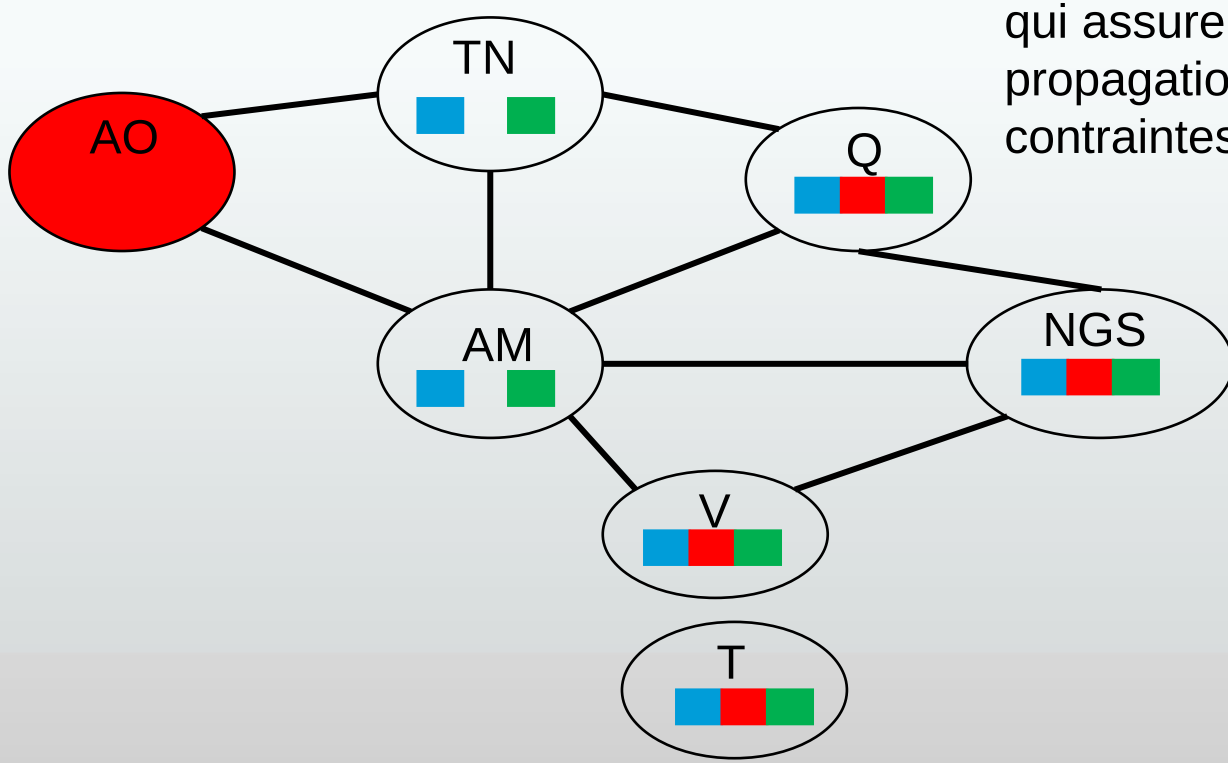


Après l'assignation de AO, l'inférence appelle AC-3 sur les arcs (X_j,AO): TN $\rightarrow$AO, AM $\rightarrow$AO sont donc cohérents.

MAC sur la carte de l’Australie

	AO	TN	Q	NGS	V	AM	T
Domaines initiaux							
Après AO = *rouge*							

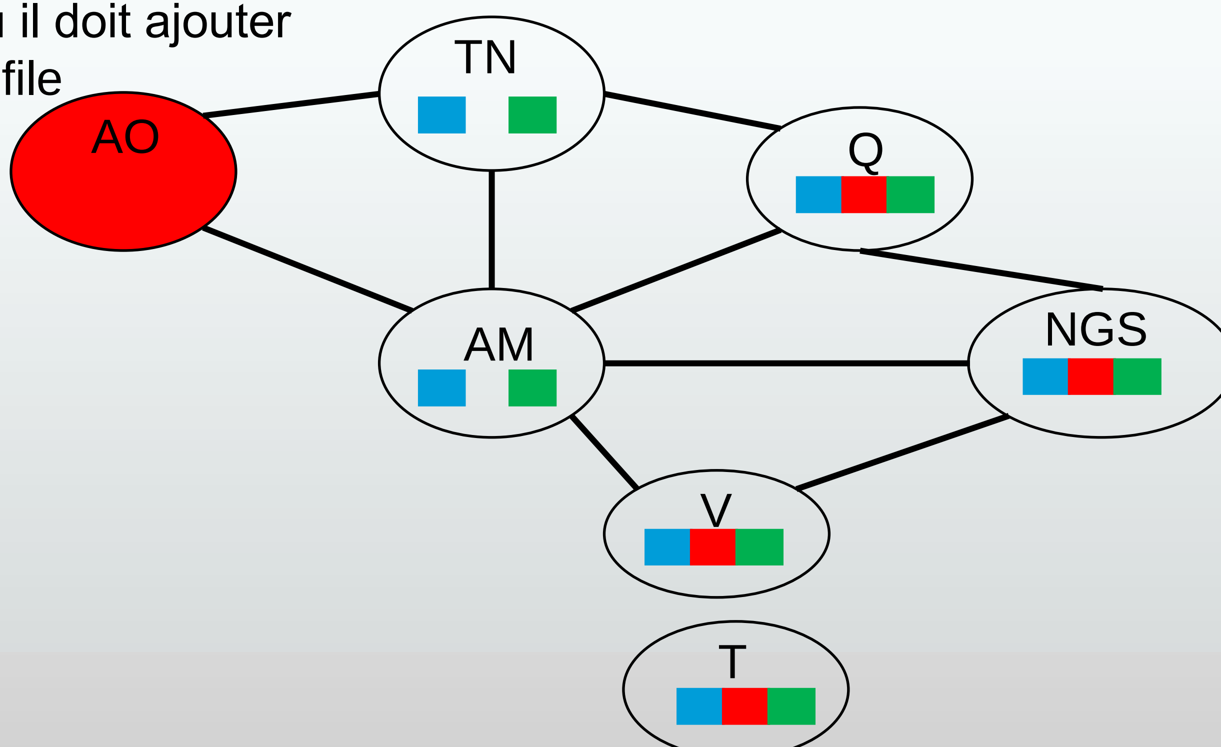
A partir de là c’est l’exécution de AC-3 qui assure la propagation de contraintes



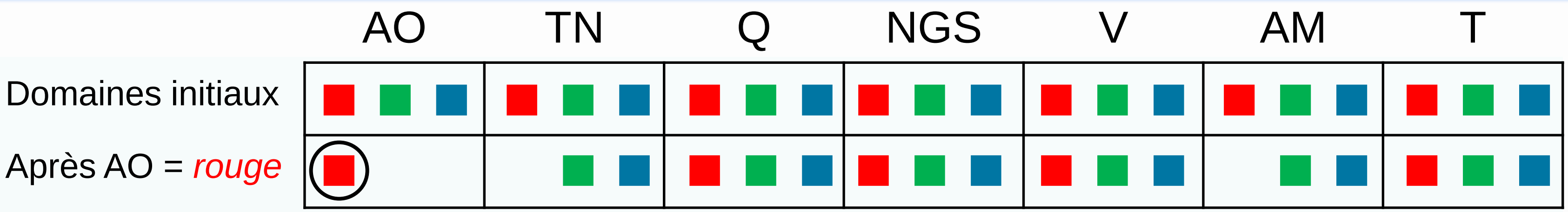
MAC sur la carte de l’Australie

	AO	TN	Q	NGS	V	AM	T
Domaines initiaux							
Après AO = *rouge*							

L’application de AC-3 sur TN→AO a impliqué la modification du domaine de TN, d’où il doit ajouter les arcs (X_k,TN) à la file pour traitement sauf (AO,TN):
{Q → TN,AM → TN}

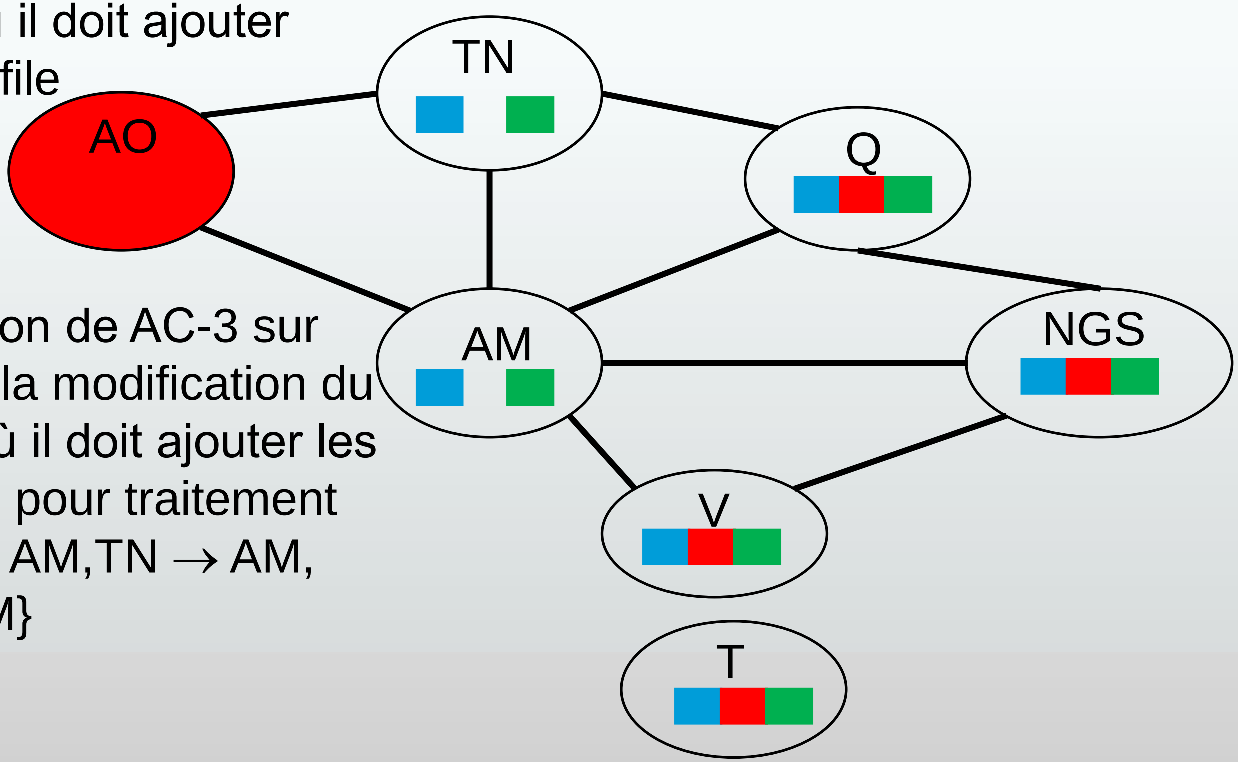


MAC sur la carte de l’Australie

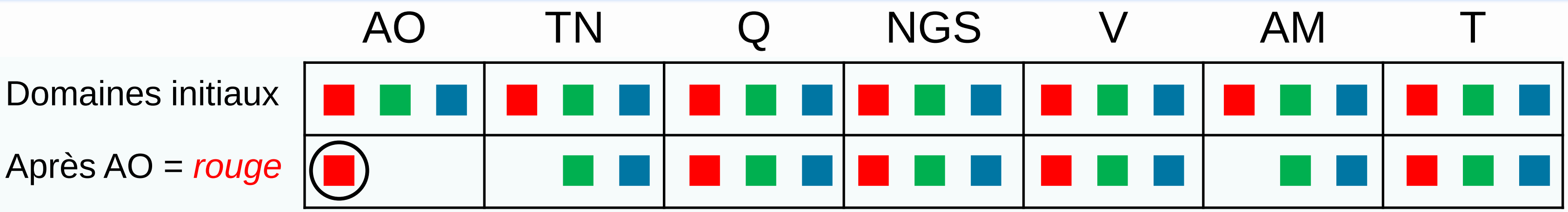


L’application de AC-3 sur TN→AO a impliqué la modification du domaine de TN, d’où il doit ajouter les arcs (X_k,TN) à la file pour traitement sauf (AO,TN): {Q → TN,AM → TN}

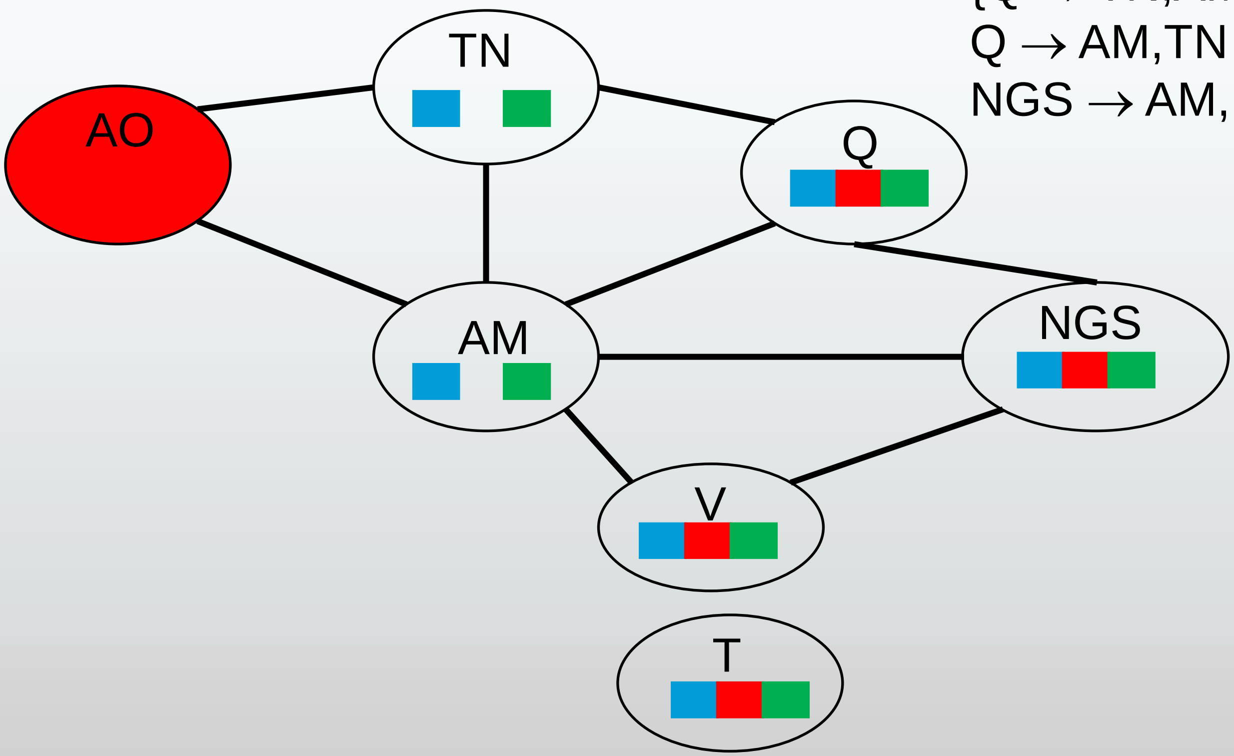
De même, l’application de AC-3 sur AM→AO a impliqué la modification du domaine de AM, d’où il doit ajouter les arcs (X_k,AM) à la file pour traitement sauf (AO,AM): {Q → AM,TN → AM, NGS → AM, V → AM}



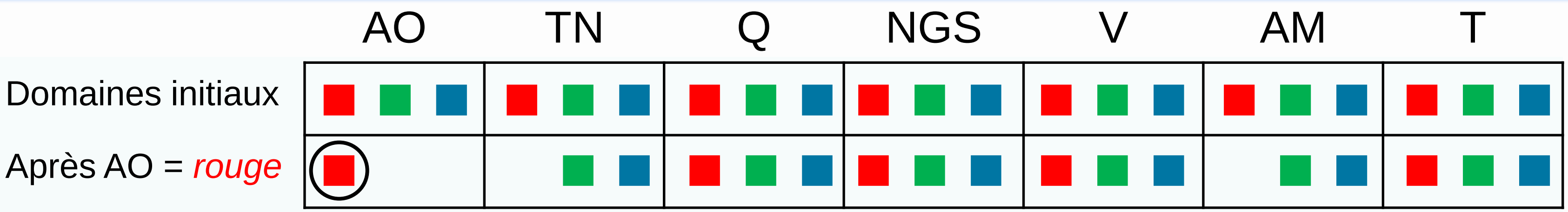
MAC sur la carte de l'Australie



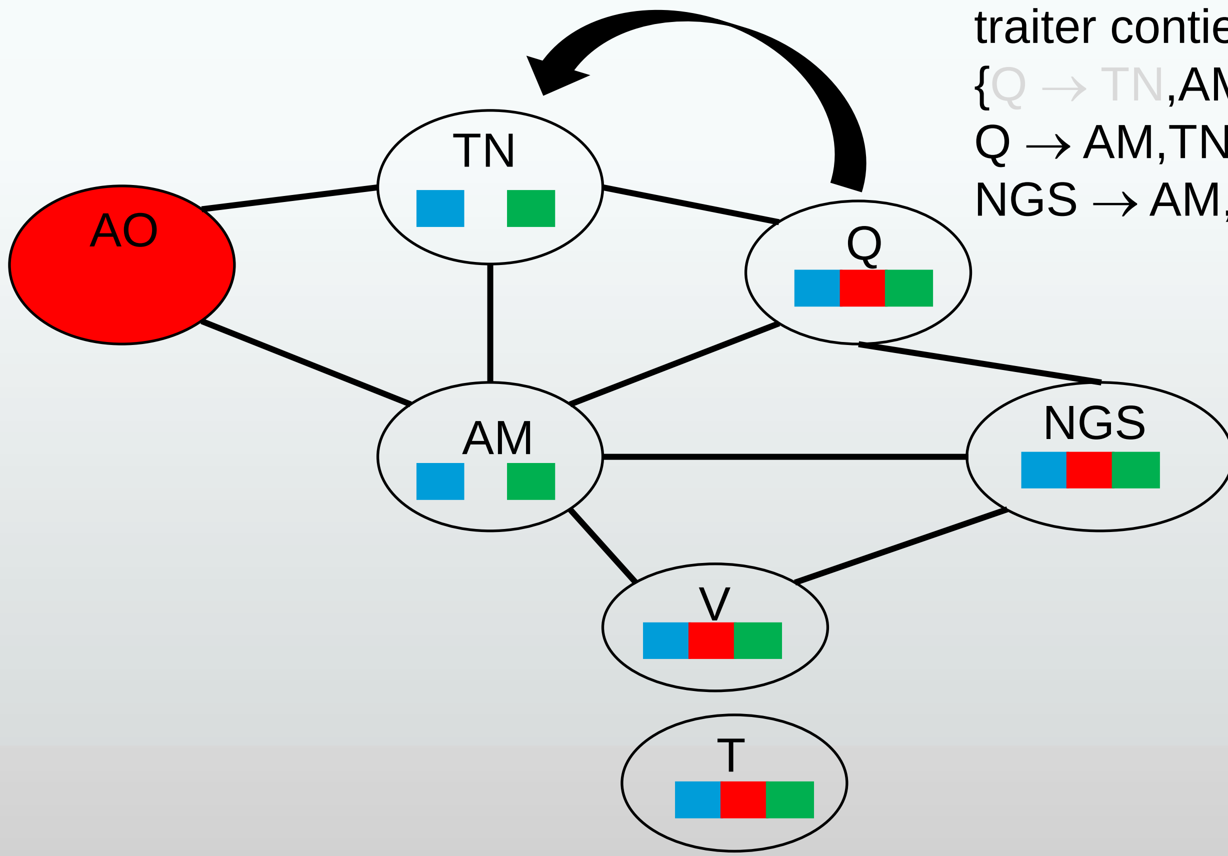
La file des arcs à traiter contient donc: $\{Q \rightarrow TN, AM \rightarrow TN, Q \rightarrow AM, TN \rightarrow AM, NGS \rightarrow AM, V \rightarrow AM\}$



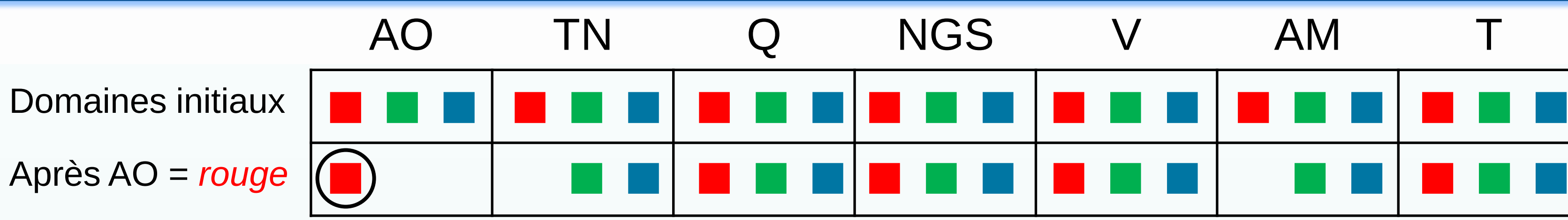
MAC sur la carte de l'Australie



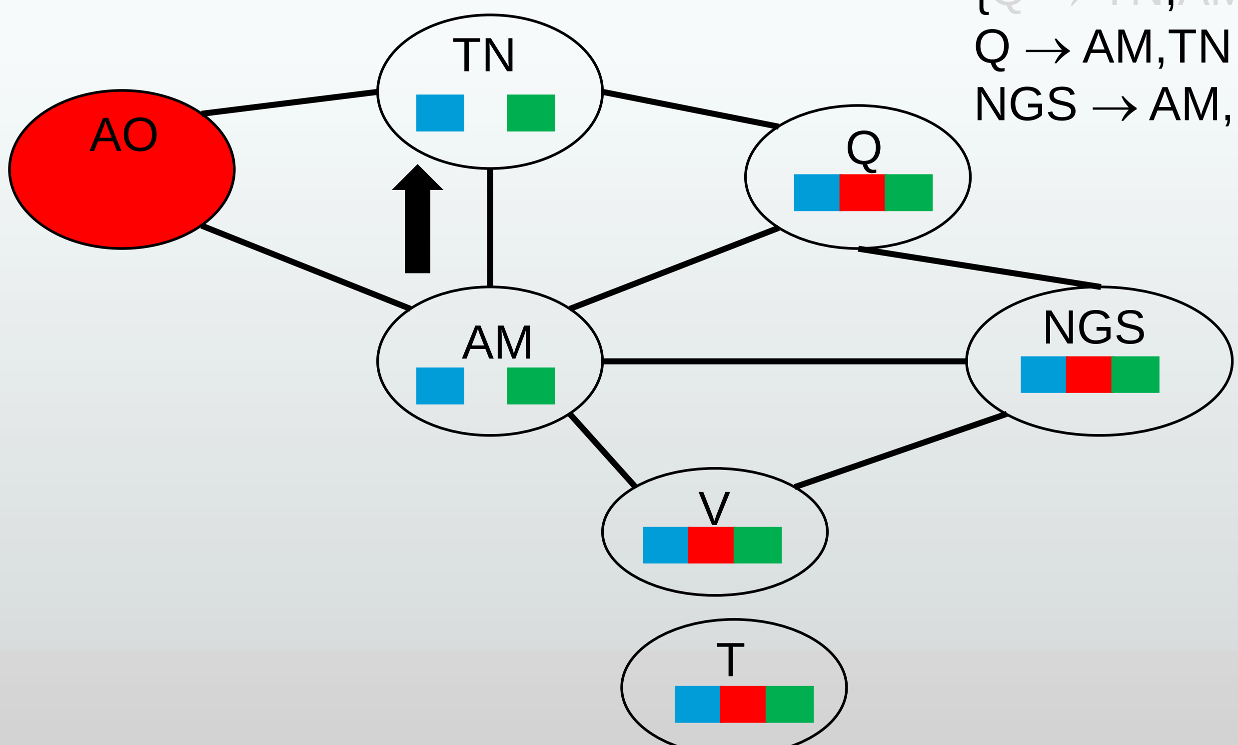
La file des arcs à traiter contient donc: {$Q \rightarrow TN$, $AM \rightarrow TN$, $Q \rightarrow AM$, $TN \rightarrow AM$, $NGS \rightarrow AM$, $V \rightarrow AM$}



MAC sur la carte de l'Australie



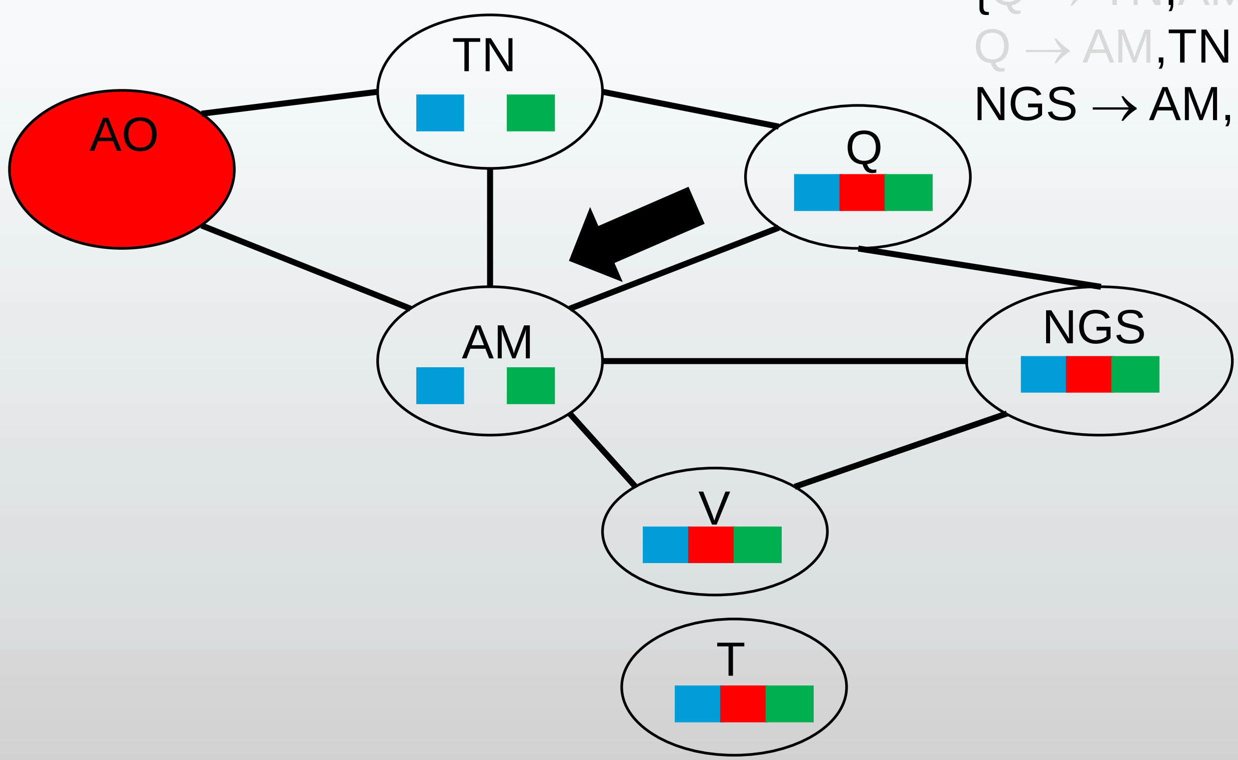
La file des arcs à traiter contient donc:
$\{Q \rightarrow TN, AM \rightarrow TN,$
$Q \rightarrow AM, TN \rightarrow AM,$
$NGS \rightarrow AM,\ V \rightarrow AM\}$



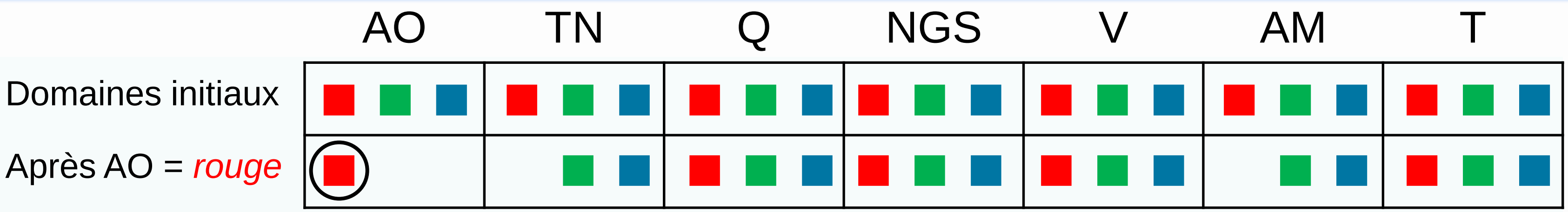
MAC sur la carte de l’Australie

	AO	TN	Q	NGS	V	AM	T
Domaines initiaux							
Après AO = *rouge*							

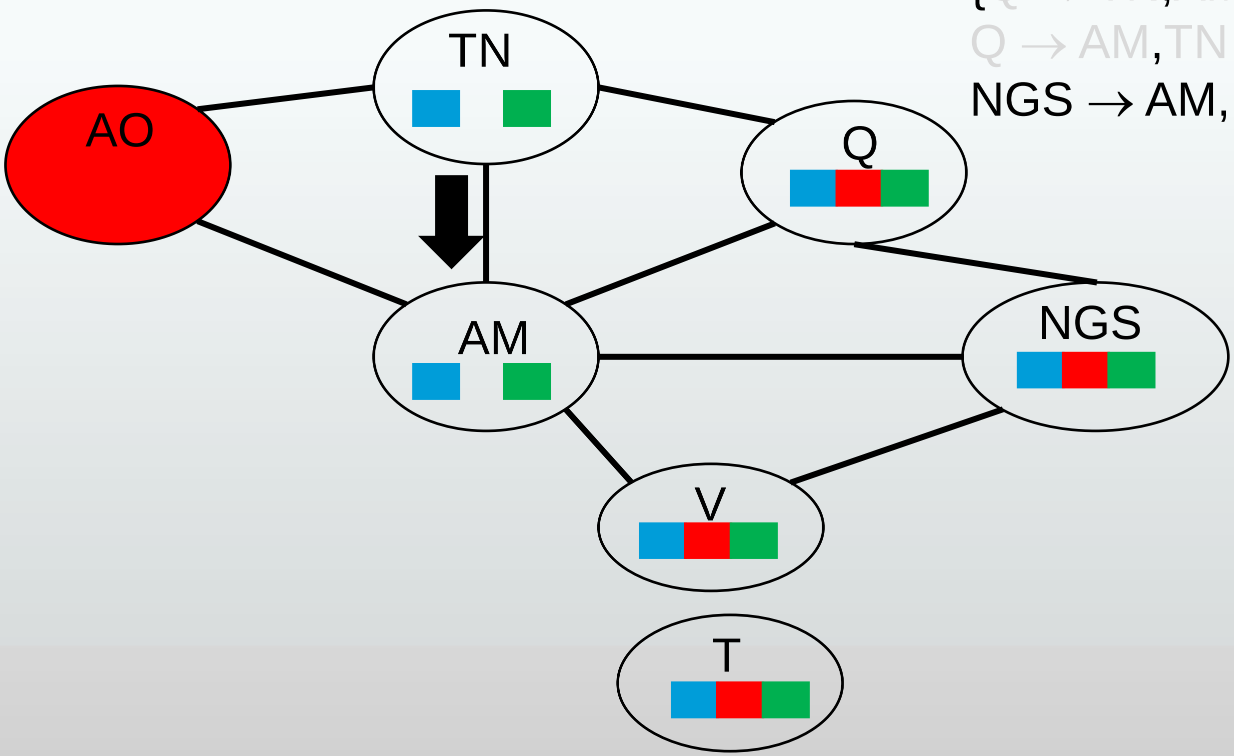
La file des arcs à traiter contient donc: {Q → TN, AM → TN, Q → AM, TN → AM, NGS → AM, V → AM}



MAC sur la carte de l'Australie



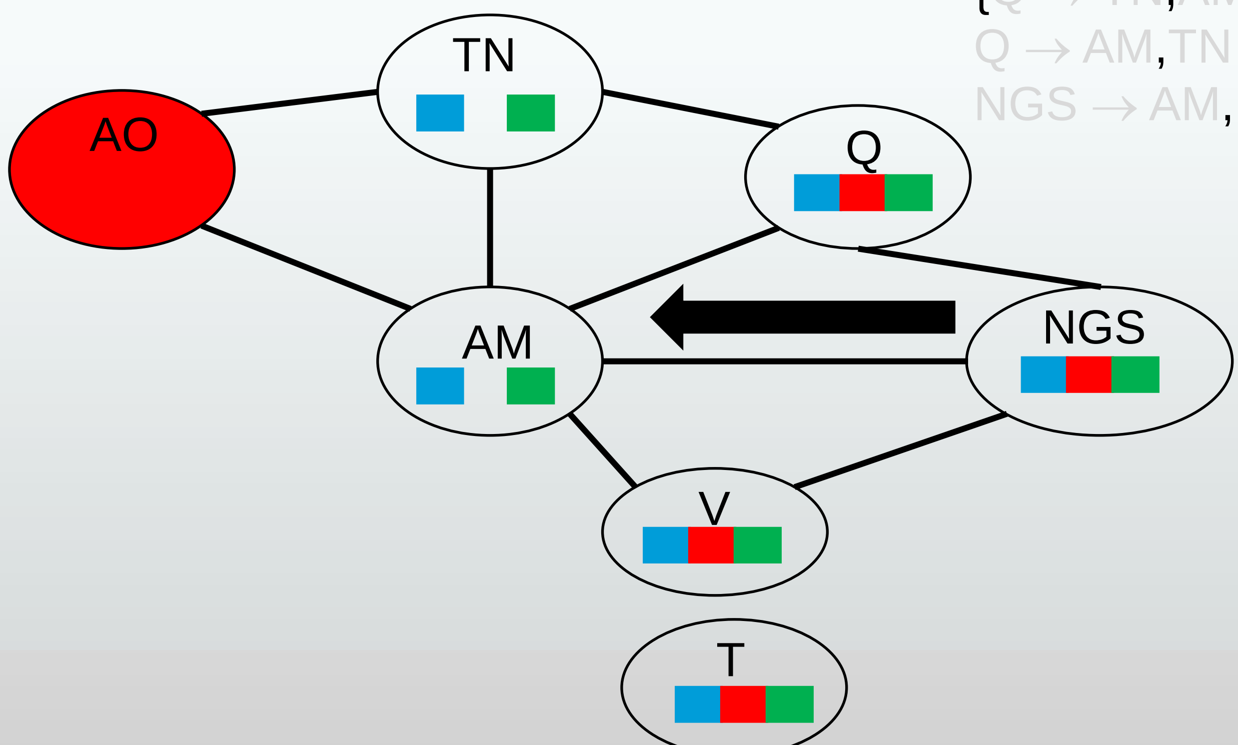
La file des arcs à traiter contient donc:
{$Q \rightarrow TN, AM \rightarrow TN,$
$Q \rightarrow AM, TN \rightarrow AM,$
$NGS \rightarrow AM, V \rightarrow AM$}



MAC sur la carte de l’Australie

	AO	TN	Q	NGS	V	AM	T
Domaines initiaux							
Après AO = *rouge*							

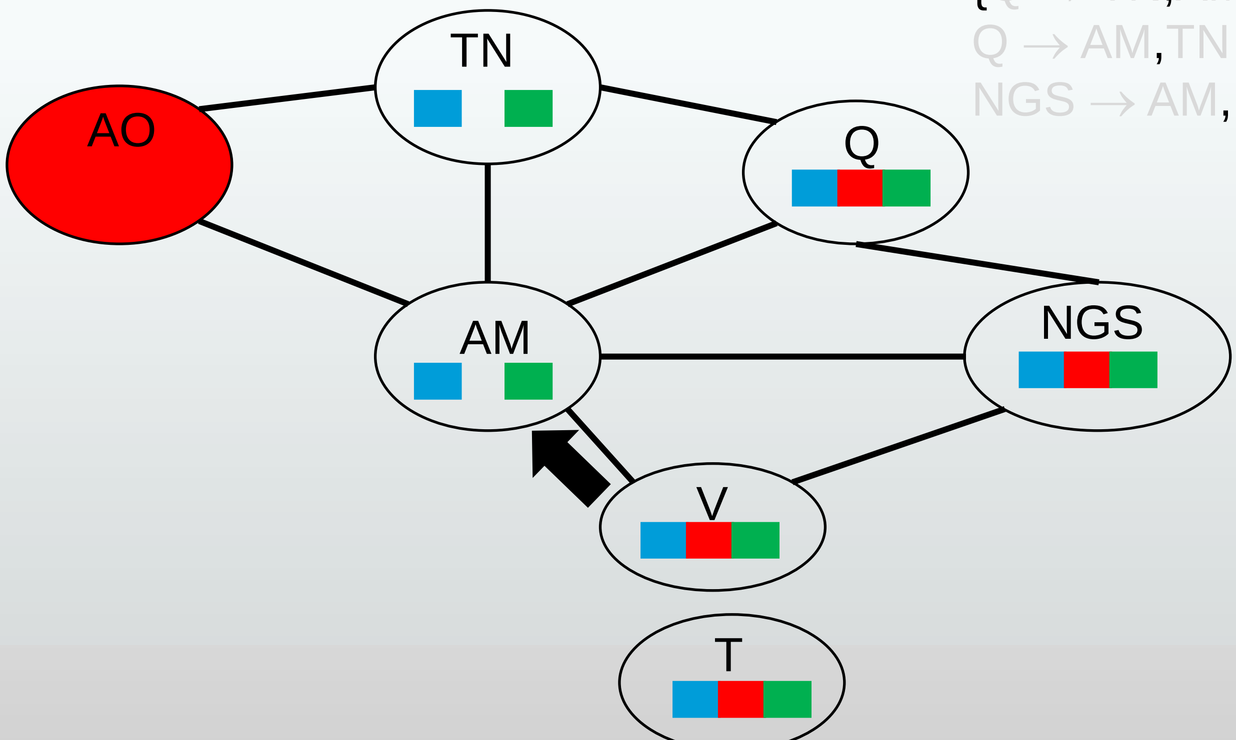
La file des arcs à traiter contient donc: $\{Q \rightarrow TN, AM \rightarrow TN, Q \rightarrow AM, TN \rightarrow AM, NGS \rightarrow AM, V \rightarrow AM\}$



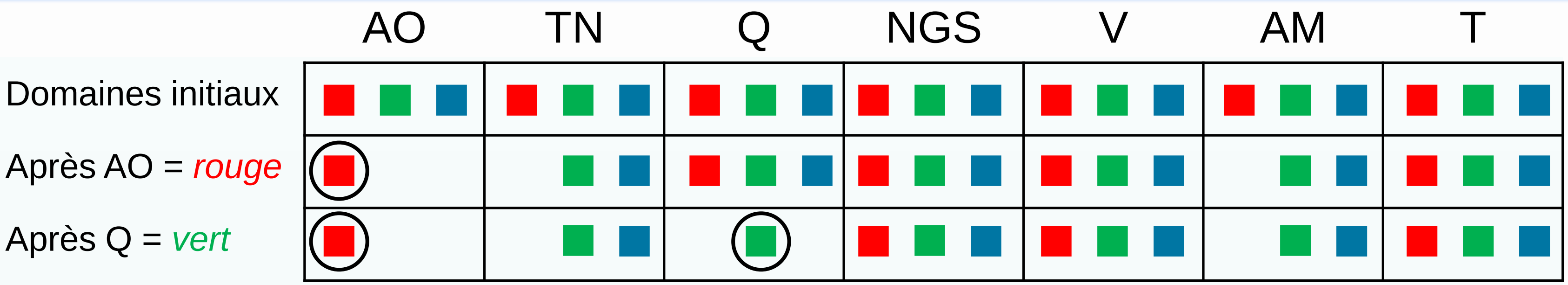
MAC sur la carte de l'Australie

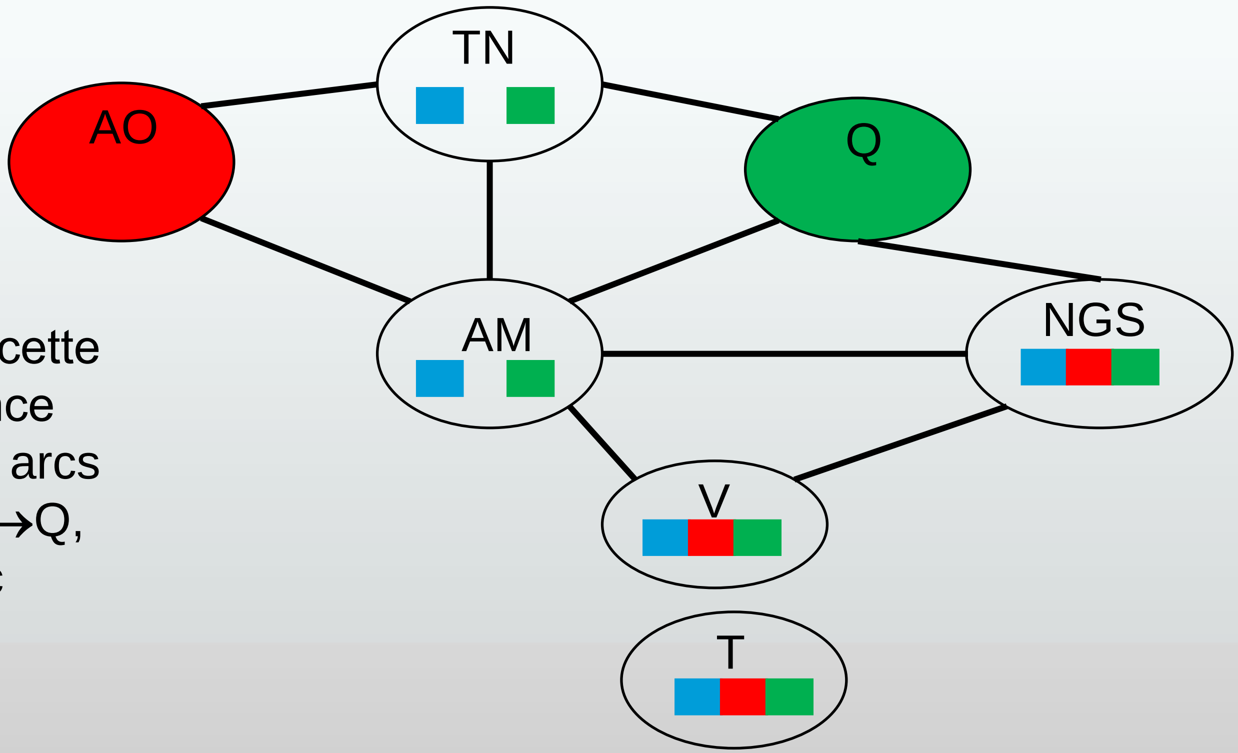
	AO	TN	Q	NGS	V	AM	T
Domaines initiaux							
Après AO = *rouge*							

La file des arcs à traiter contient donc:
$\{Q \rightarrow TN, AM \rightarrow TN, Q \rightarrow AM, TN \rightarrow AM, NGS \rightarrow AM, V \rightarrow AM\}$



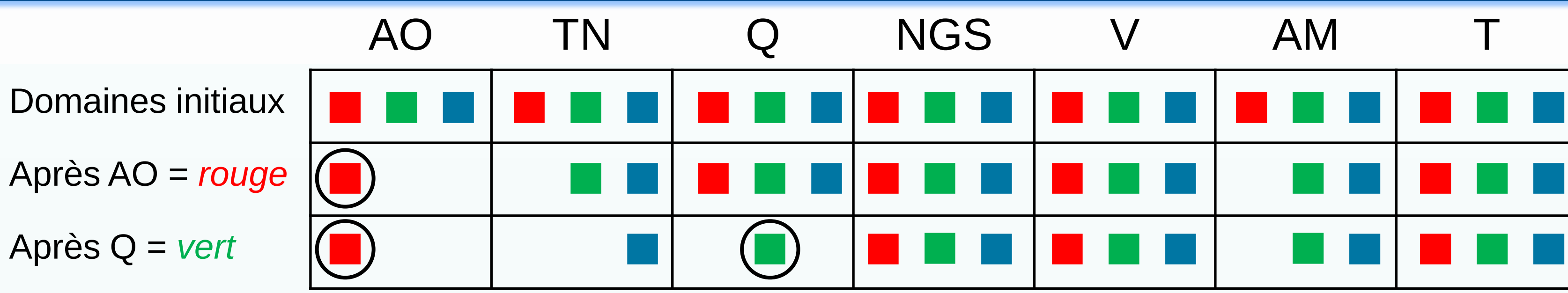
MAC sur la carte de l'Australie

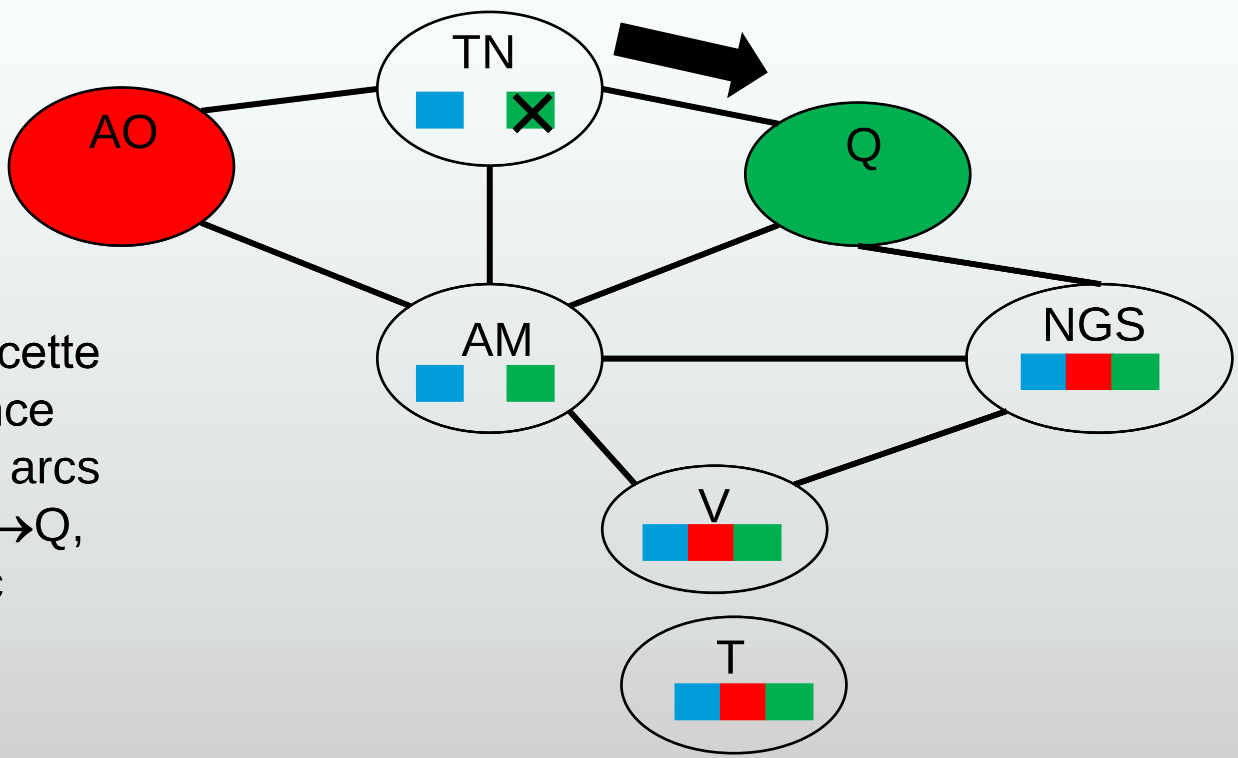




L'algorithme choisit d'assigner Q, après cette assignation, l'inférence appelle AC-3 sur les arcs (X_j,Q): TN →Q, AM →Q, NGS →Q sont donc cohérents.

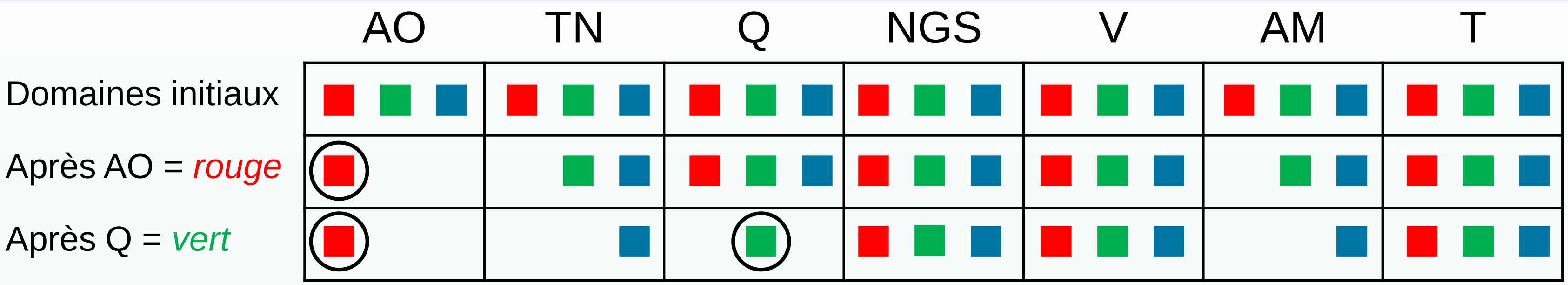
MAC sur la carte de l'Australie

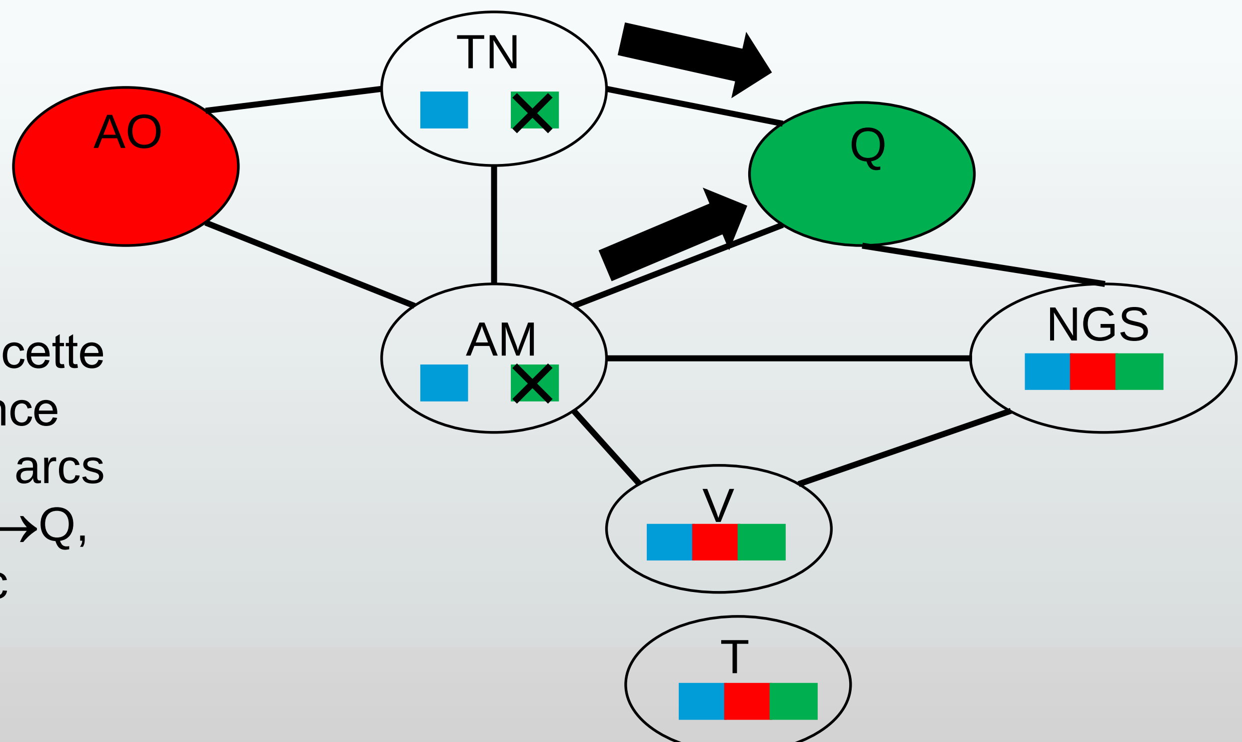




L'algorithme choisit d'assigner Q, après cette assignation, l'inférence appelle AC-3 sur les arcs (X_j,Q): TN →Q, AM →Q, NGS →Q sont donc cohérents.

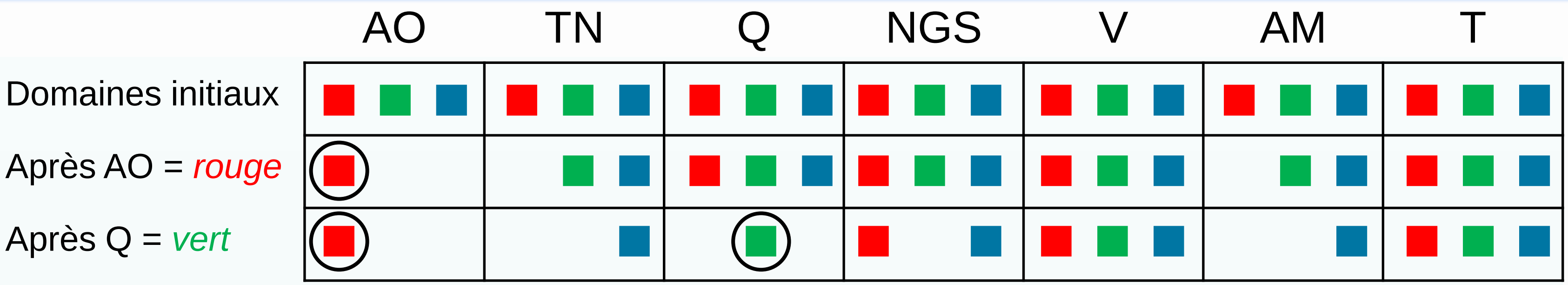
MAC sur la carte de l’Australie

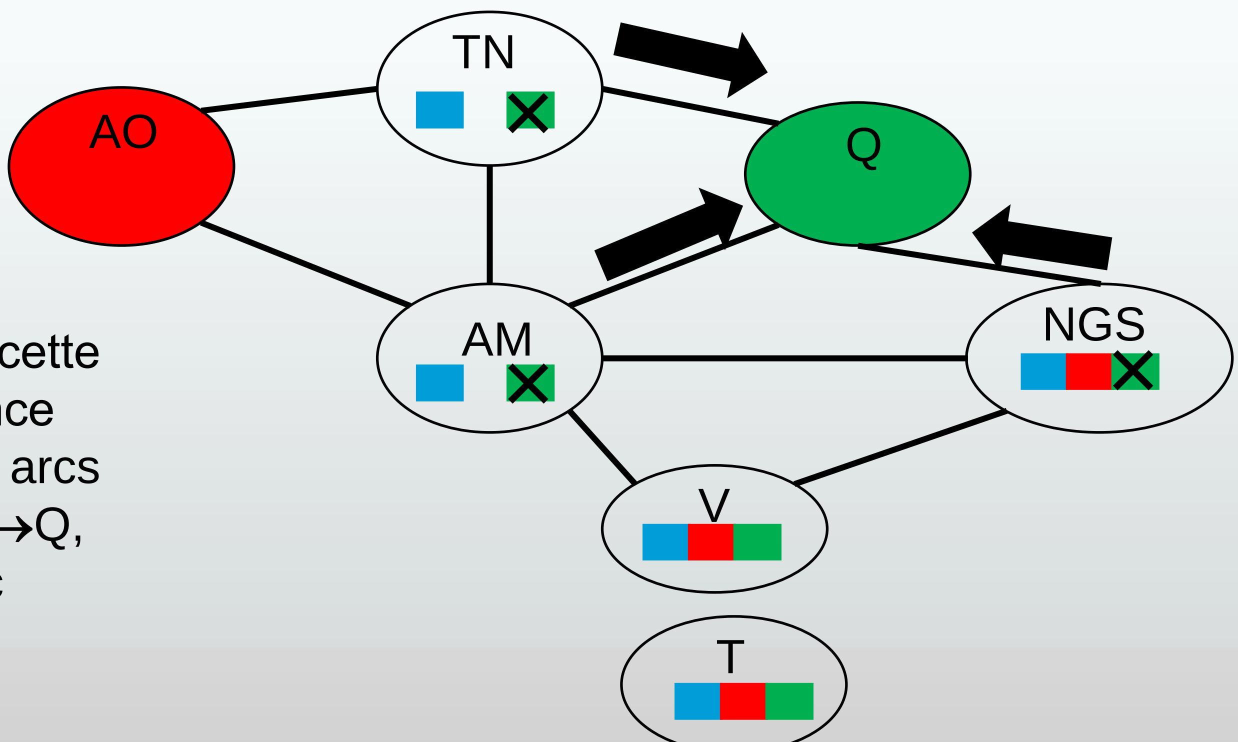




L’algorithme choisit d’assigner Q, après cette assignation, l’inférence appelle AC-3 sur les arcs (X_j,Q): TN →Q, AM →Q, NGS →Q sont donc cohérents.

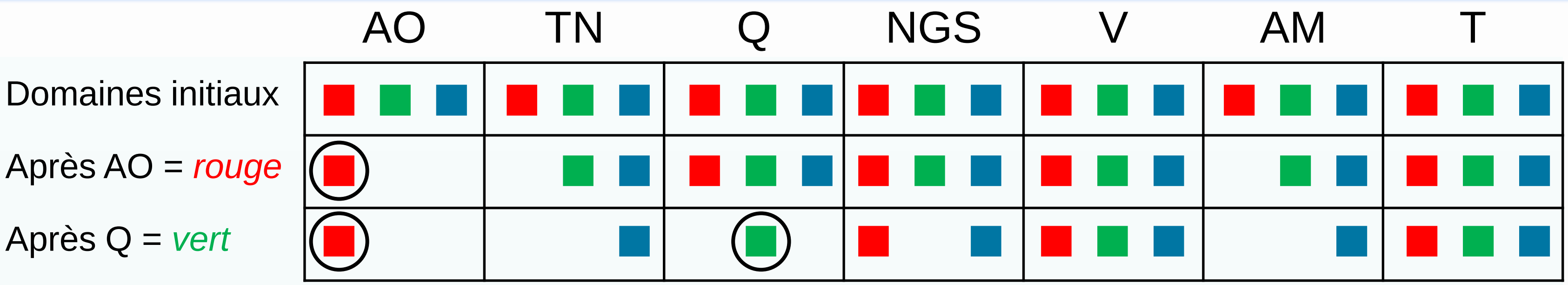
MAC sur la carte de l'Australie



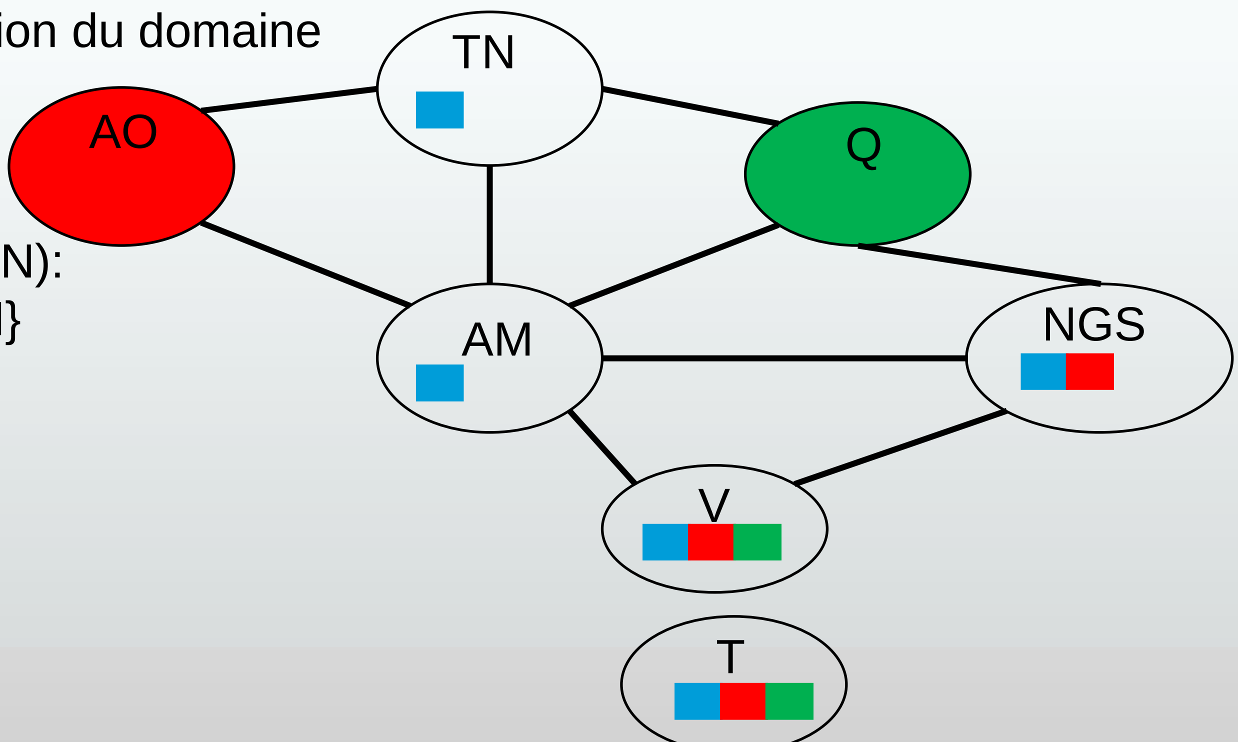


L'algorithme choisit d'assigner Q, après cette assignation, l'inférence appelle AC-3 sur les arcs (X_j,Q): TN →Q, AM →Q, NGS →Q sont donc cohérents.

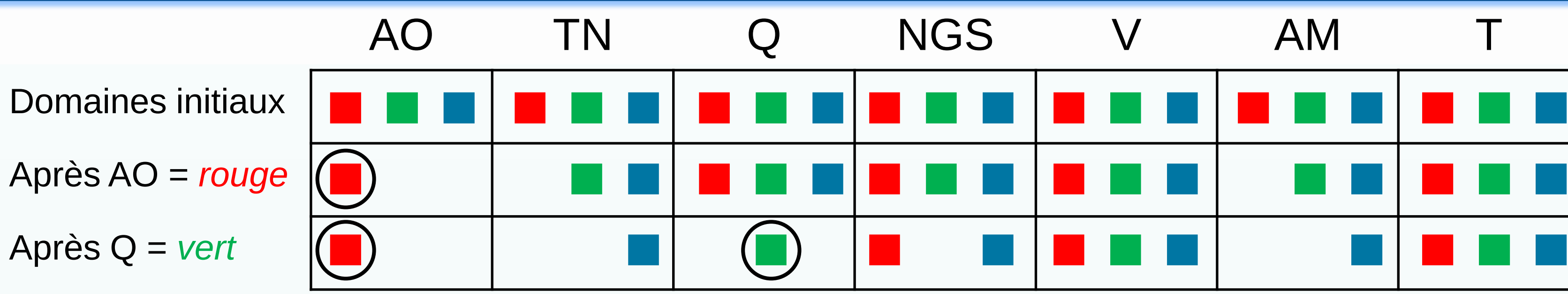
MAC sur la carte de l'Australie



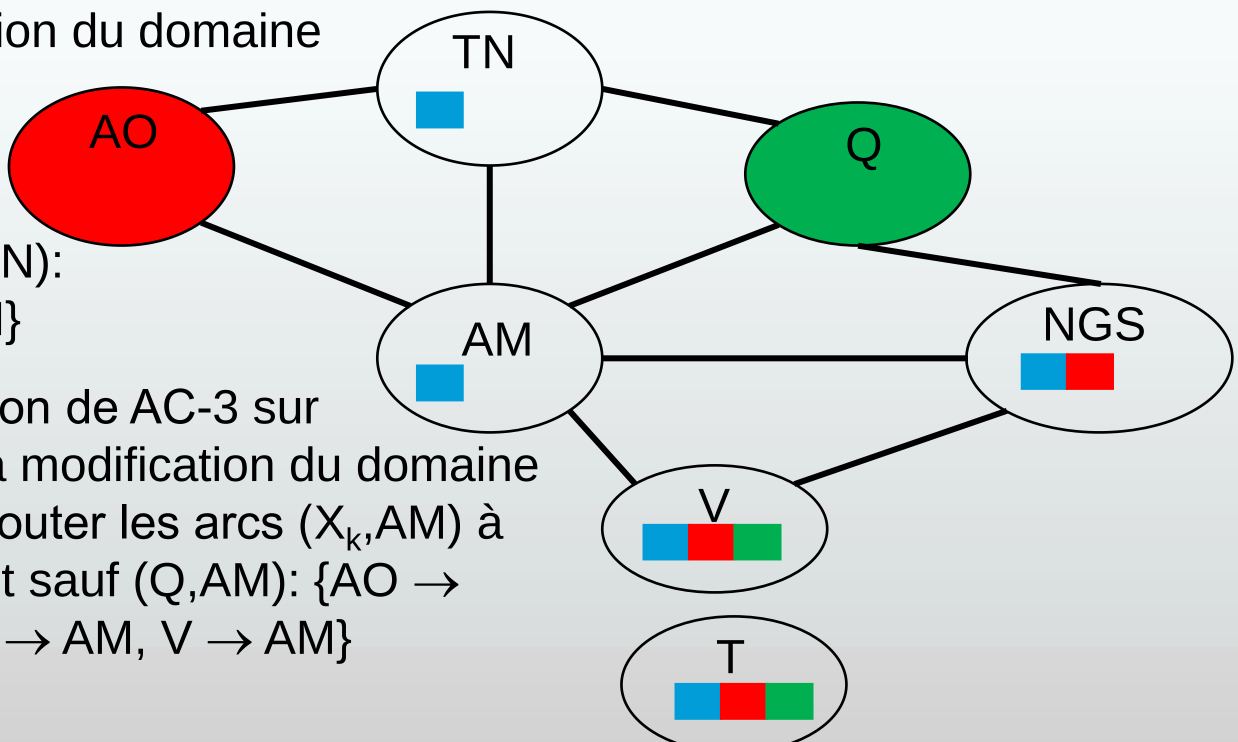
L'application de AC-3 sur TN→Q a impliqué la modification du domaine de TN, d'où il doit ajouter les arcs (X_k, TN) à la file pour traitement sauf (Q,TN): {AO → TN,AM → TN}



MAC sur la carte de l'Australie



L'application de AC-3 sur TN→Q a impliqué la modification du domaine de TN, d'où il doit ajouter les arcs (X_k,TN) à la file pour traitement sauf (Q,TN): {AO → TN,AM → TN}

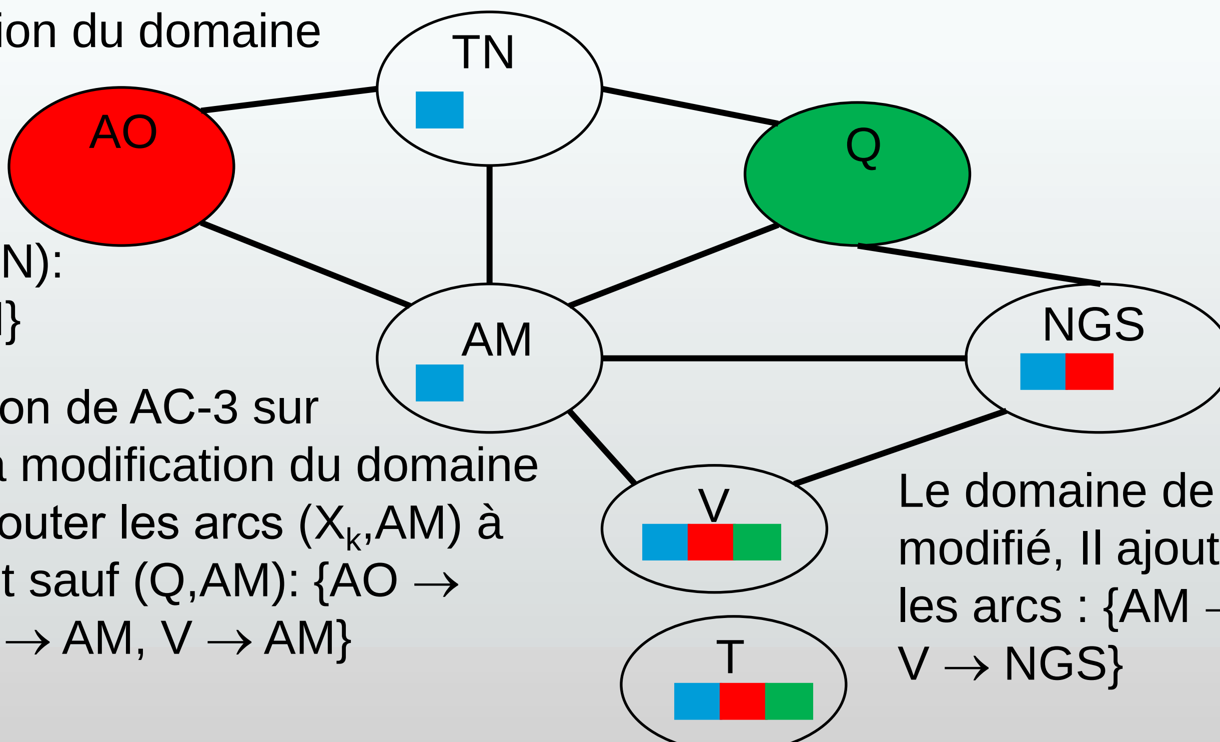


De même, l'application de AC-3 sur AM→Q a impliqué la modification du domaine de AM, d'où il doit ajouter les arcs (X_k,AM) à la file pour traitement sauf (Q,AM): {AO → AM,TN → AM, NGS → AM, V → AM}

MAC sur la carte de l'Australie

	AO	TN	Q	NGS	V	AM	T
Domaines initiaux	rouge vert bleu	rouge vert bleu	rouge vert bleu	rouge vert bleu	rouge vert bleu	rouge vert bleu	rouge vert bleu
Après AO = *rouge*	rouge	vert bleu	rouge vert bleu	rouge vert bleu	rouge vert bleu	vert bleu	rouge vert bleu
Après Q = *vert*	rouge	bleu	vert	rouge bleu	rouge vert bleu	bleu	rouge vert bleu

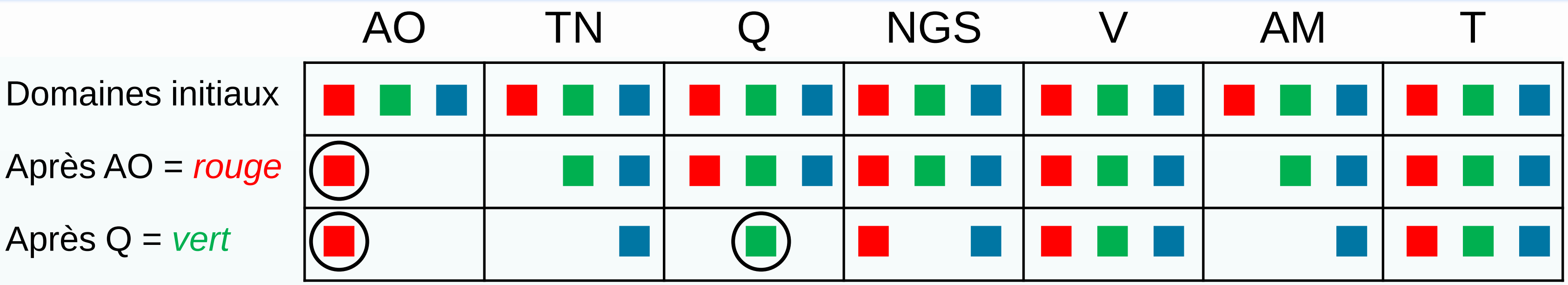
L'application de AC-3 sur TN→Q a impliqué la modification du domaine de TN, d'où il doit ajouter les arcs (X_k,TN) à la file pour traitement sauf (Q,TN): {AO → TN,AM → TN}



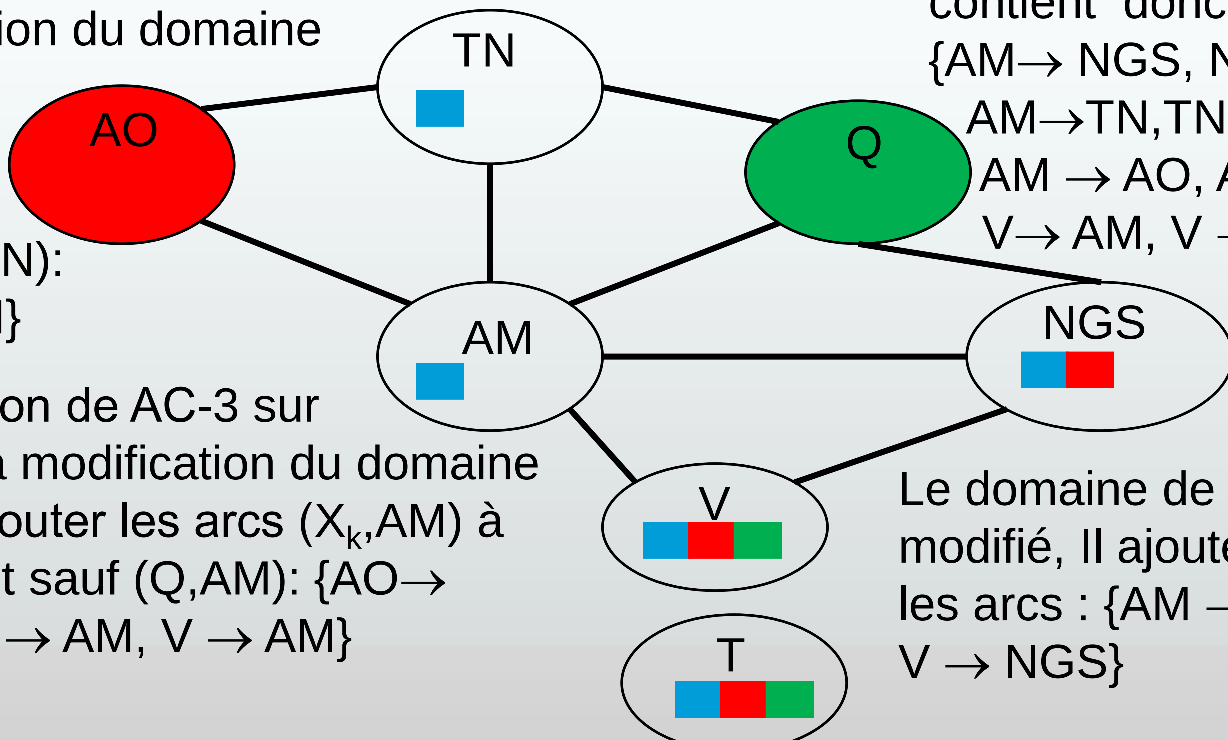
De même, l'application de AC-3 sur AM→Q a impliqué la modification du domaine de AM, d'où il doit ajouter les arcs (X_k,AM) à la file pour traitement sauf (Q,AM): {AO → AM,TN → AM, NGS → AM, V → AM}

Le domaine de NGS est modifié, Il ajoute aussi les arcs : {AM → NGS, V → NGS}

MAC sur la carte de l'Australie



L'application de AC-3 sur TN→Q a impliqué la modification du domaine de TN, d'où il doit ajouter les arcs (X_k,TN) à la file pour traitement sauf (Q,TN): {AO → TN,AM → TN}

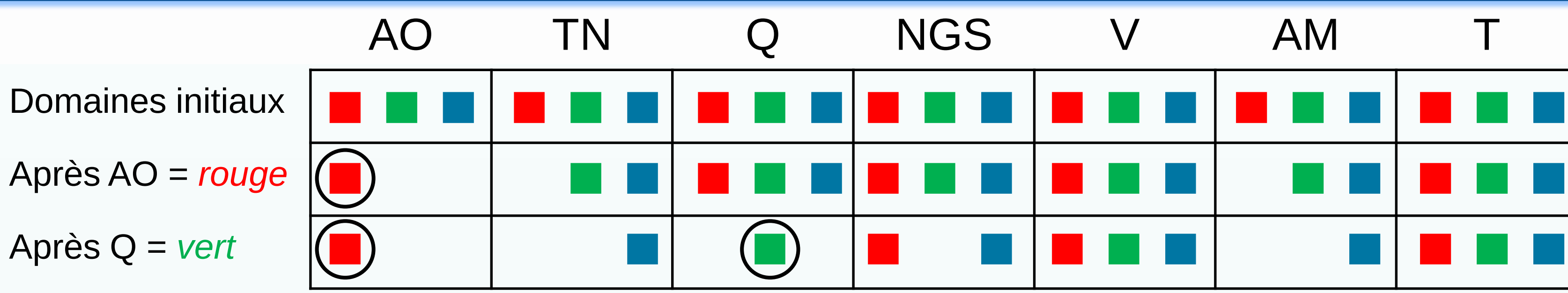


La file des arcs à traiter contient donc: {AM→ NGS, NGS→AM, AM→TN,TN → AM, AM → AO, AO→ AM, V→ AM, V → NGS}

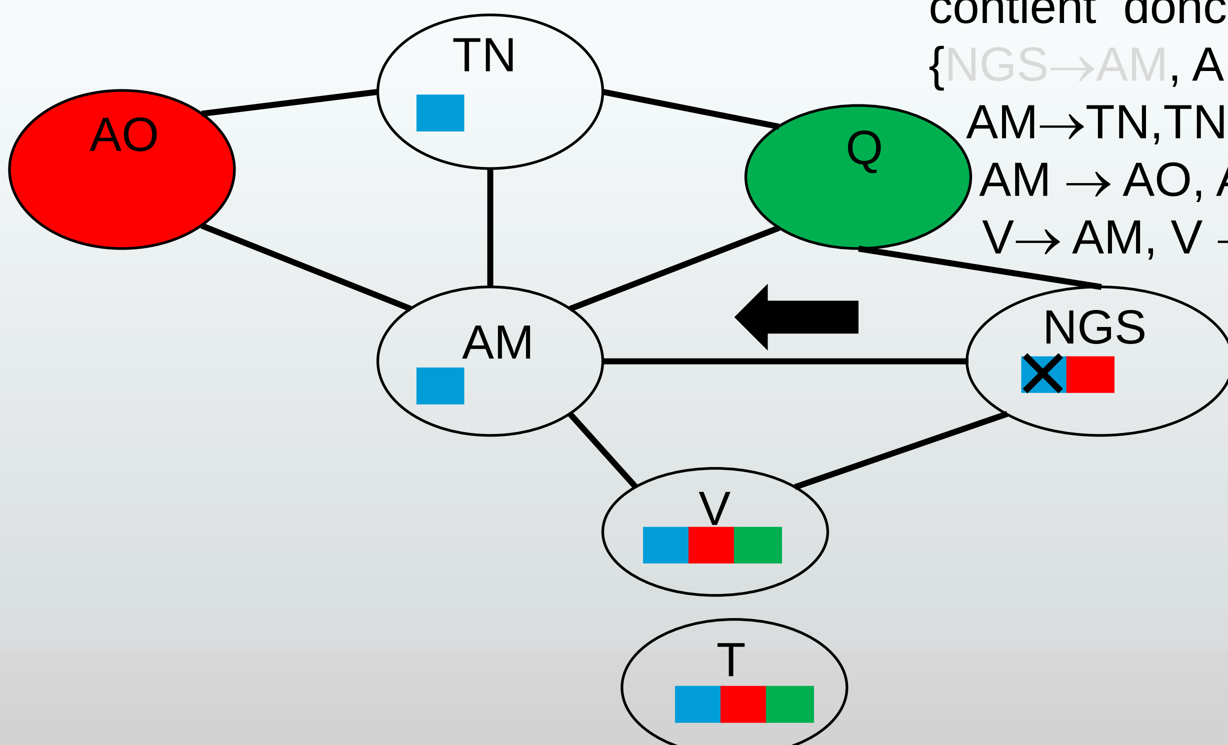
De même, l'application de AC-3 sur AM→Q a impliqué la modification du domaine de AM, d'où il doit ajouter les arcs (X_k,AM) à la file pour traitement sauf (Q,AM): {AO→ AM,TN → AM, NGS → AM, V → AM}

Le domaine de NGS est modifié, Il ajoute aussi les arcs : {AM → NGS, V → NGS}

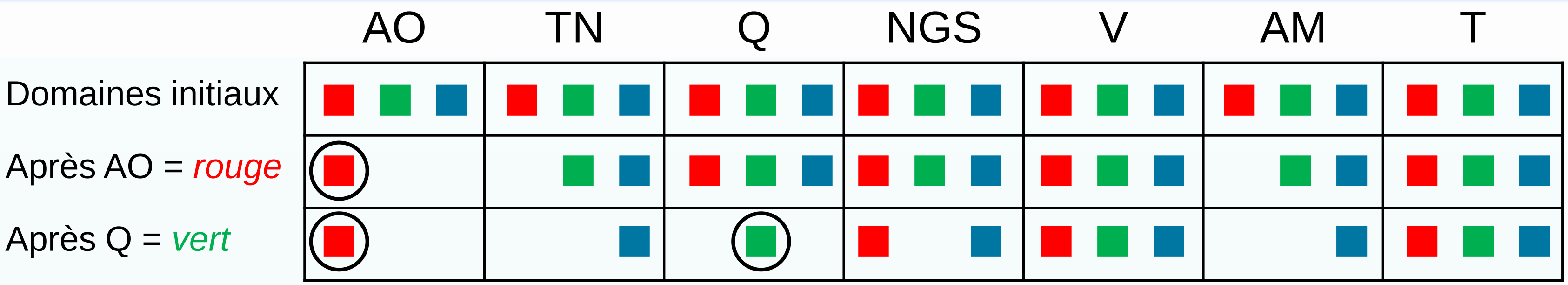
MAC sur la carte de l'Australie



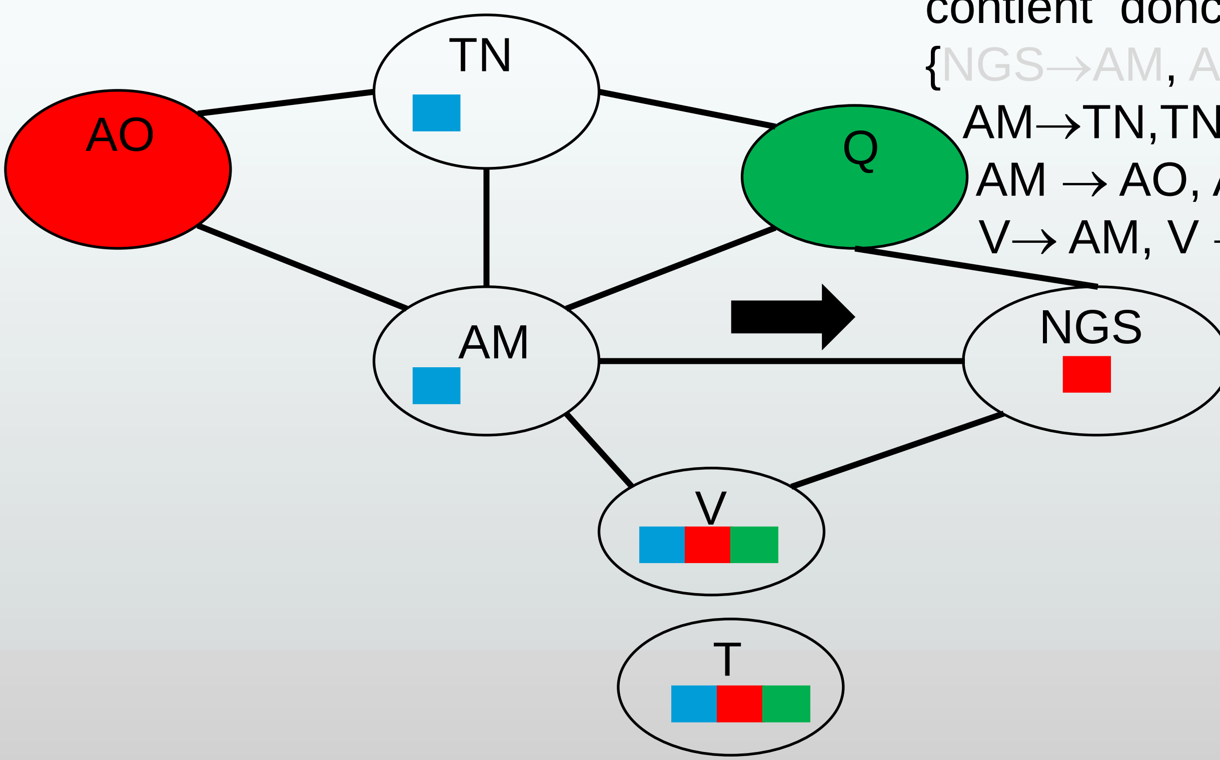
La file des arcs à traiter contient donc:
{NGS→AM, AM→ NGS, AM→TN,TN → AM, AM → AO, AO→ AM, V→ AM, V → NGS}



MAC sur la carte de l'Australie



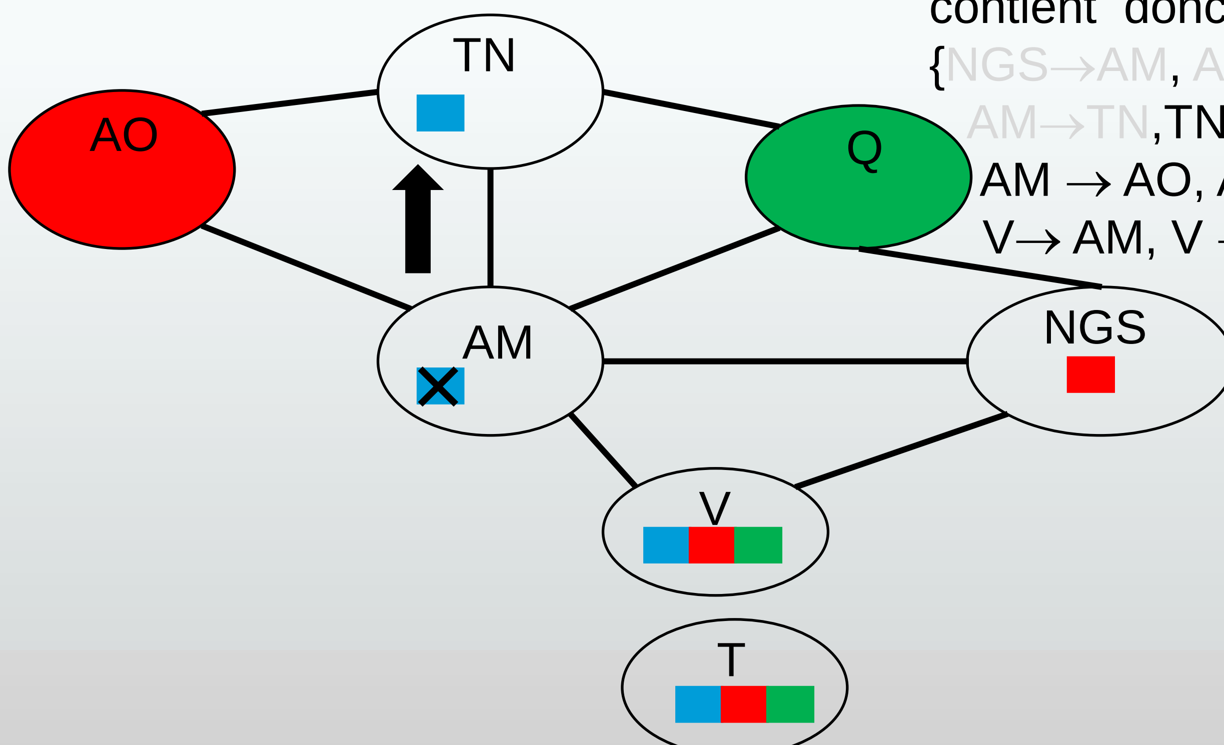
La file des arcs à traiter contient donc:
{NGS→AM, AM→ NGS, AM→TN,TN → AM, AM → AO, AO→ AM, V→ AM, V → NGS}



MAC sur la carte de l'Australie

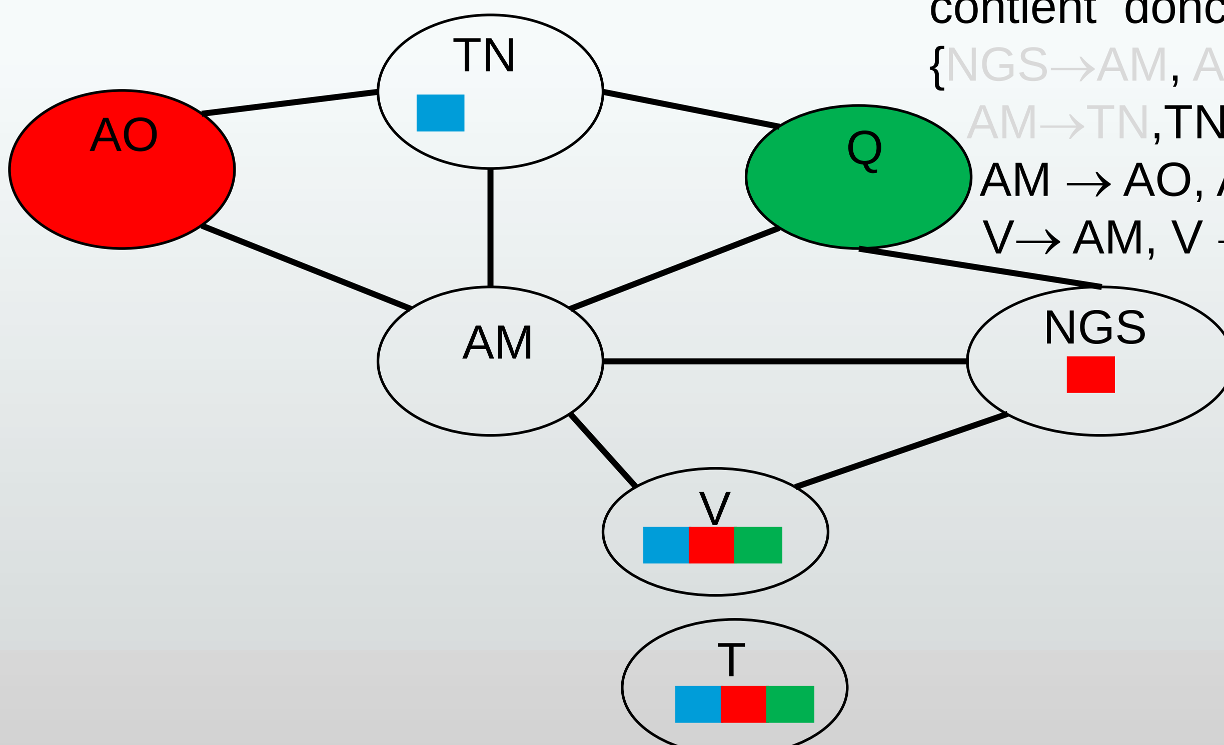
	AO	TN	Q	NGS	V	AM	T
Domaines initiaux	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
Après AO = *rouge*	🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
Après Q = *vert*	🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦

La file des arcs à traiter contient donc:
{NGS→AM, AM→ NGS, AM→TN,TN → AM, AM → AO, AO→ AM, V→ AM, V → NGS}



MAC sur la carte de l'Australie

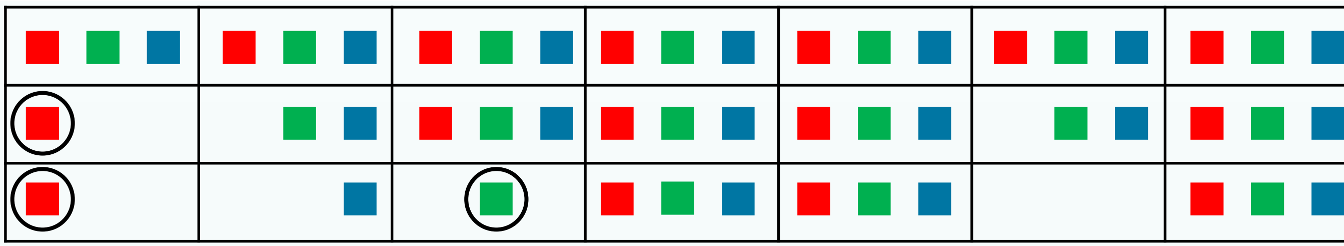
	AO	TN	Q	NGS	V	AM	T
Domaines initiaux	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■
Après AO = *rouge*	■	■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■	■ ■ ■
Après Q = *vert*	■	■	■	■ ■ ■	■ ■ ■		■ ■ ■



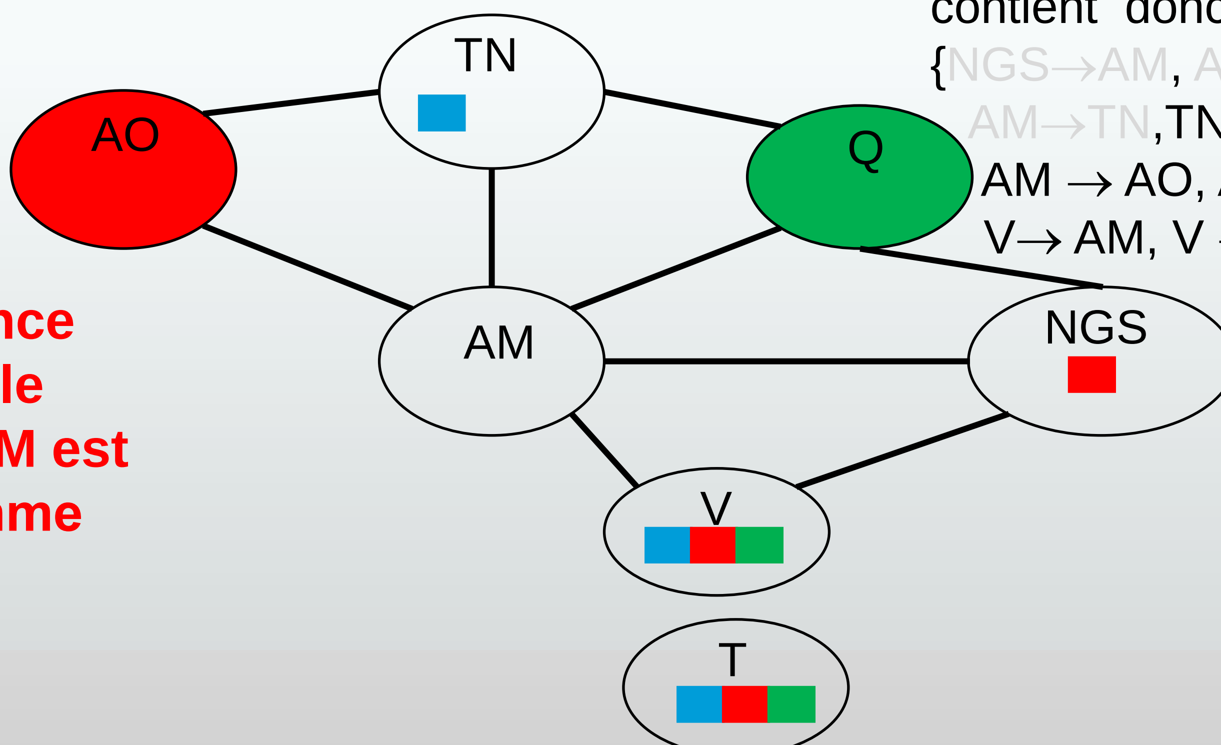
La file des arcs à traiter contient donc: {NGS→AM, AM→ NGS, AM→TN,TN → AM, AM → AO, AO→ AM, V→ AM, V → NGS}

MAC sur la carte de l'Australie

	AO	TN	Q	NGS	V	AM	T
Domaines initiaux							
Après AO = *rouge*							
Après Q = *vert*							



La file des arcs à traiter contient donc:
{NGS→AM, AM→ NGS, AM→TN,TN → AM, AM → AO, AO→ AM, V→ AM, V → NGS}



Une incohérence est détectée : le domaine de AM est vide, l'algorithme s'arrête !

MAC VS Vérification avant

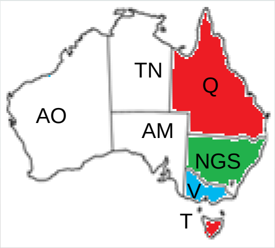
- L'algorithme MAC est strictement plus puissant que la vérification en avant.

- La vérification en avant vérifie la cohérence d'arcs mais uniquement sur les arcs initiaux de la file d'attente du MAC et ne propage pas récursivement les contraintes suite à la modification des domaines des variables.

Retour arrière intelligent : examen en amont

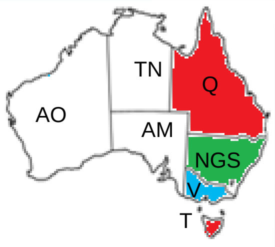
- L'algorithme de *backtracking* : en cas d'échec d'une branche, il revient en arrière à la dernière variable et essaye une autre valeur $\Rightarrow$ c'est le *backtracking chronologique*.

- C'est une stratégie souvent inefficace :
  - Soit l'ordre d'assignation suivant : Q, NGS, V, T, AM, AO, TN et que l'assignation partielle courante est {Q=rouge, NGS=vert, V=bleu, T=rouge}.
  - L'assignation suivante de AM n'est pas possible, et le backtracking remonte à T pour essayer une nouvelle valeur : ceci ne résout pas le conflit avec AM.



Retour arrière intelligent : examen en amont

- L'idée est de retourner à la variable qui a causé l'échec.
  - Ensemble de conflit : Garder une trace d'un ensemble d'assignation en conflit avec la valeur de AM ({Q=rouge, NGS=vert, V=bleu}).



  - Backjumping (saut en arrière) : remonte à l'assignation la plus récente dans l'ensemble de conflit. Ici, il sauterait la Tasmanie et essaierait une nouvelle valeur pour V.

Retour arrière intelligent : examen en amont

- La mise en œuvre est simple :
  - On modifie la recherche avec backtracking afin qu’elle stocke l’ensemble de conflit (le forward checking calcule déjà implicitement cet ensemble) lorsqu’il cherche une valeur légale à assigner :
    - Lorsqu’une assignation X=x cause l’élimination d’une valeur du domaine de Y, il ajoute X=x à l’ensemble de conflit de Y.
    - Si la dernière valeur est supprimée du domaine de Y, les assignations de l’ensemble de conflit de Y sont ajoutées à l’ensemble de conflit de X.
  - S’il ne trouve pas de valeur légale, l’algorithme retourne à l’élément le plus récent de l’ensemble de conflit.

Illustration du backjumping

- Les assignations se feront dans l'ordre : Q, NSW, V, T, AM, AO, TN.

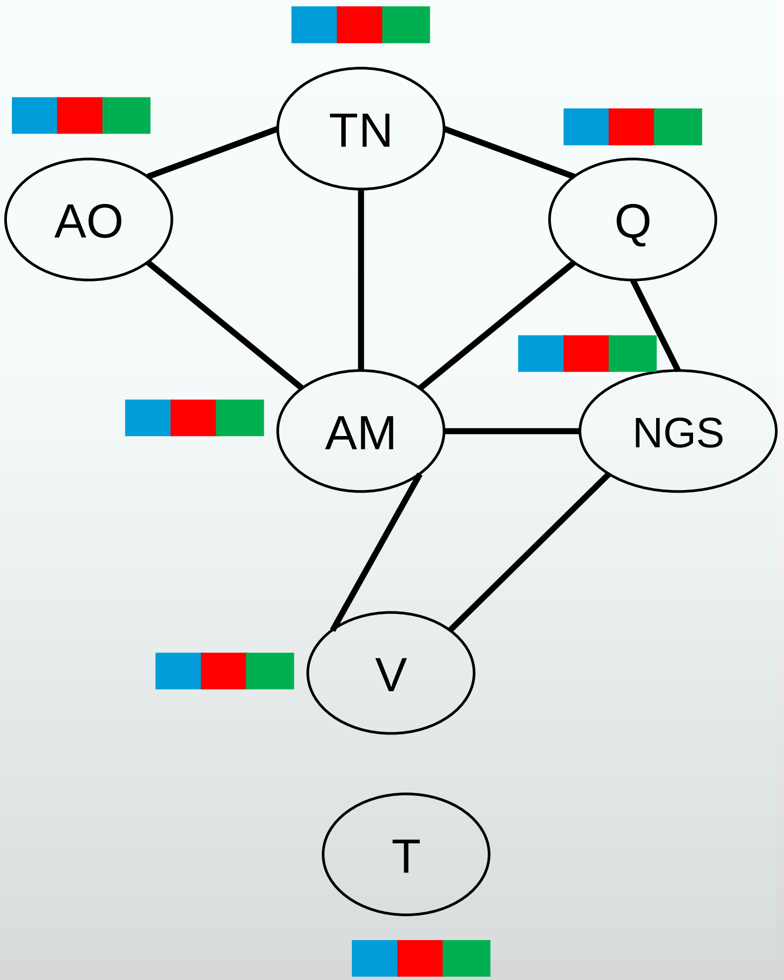


Illustration du backjumping

- Assignation : {Q=rouge}.

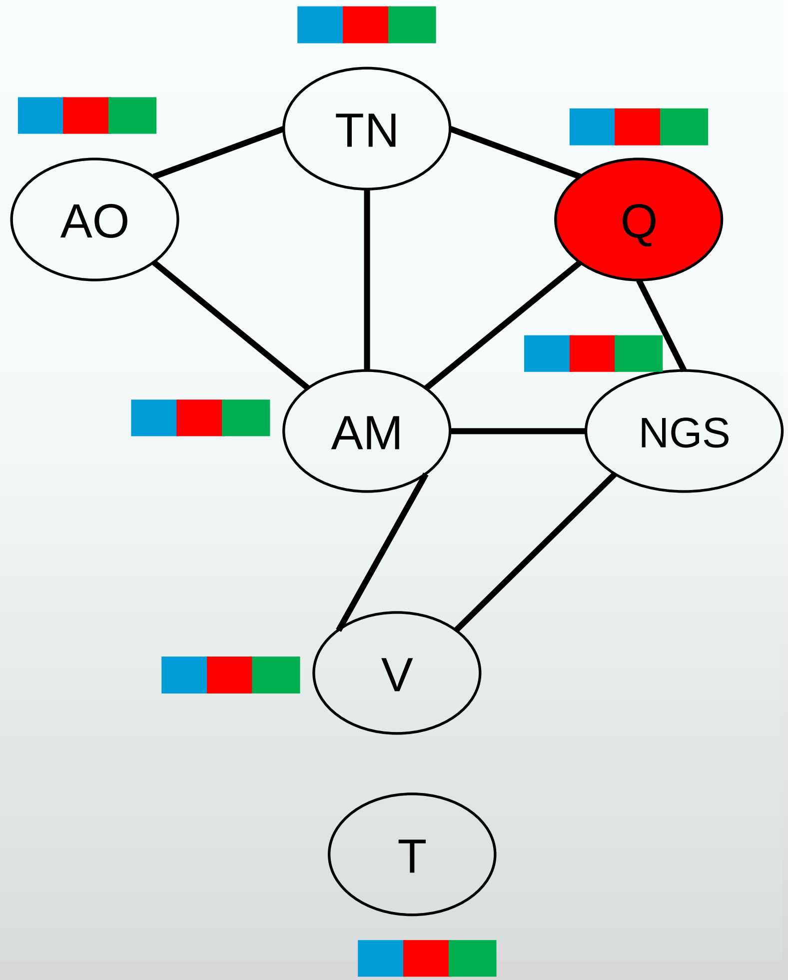


Illustration du backjumping

- Assignation : {Q=rouge}.
- Calcul des l'ensembles de conflit.

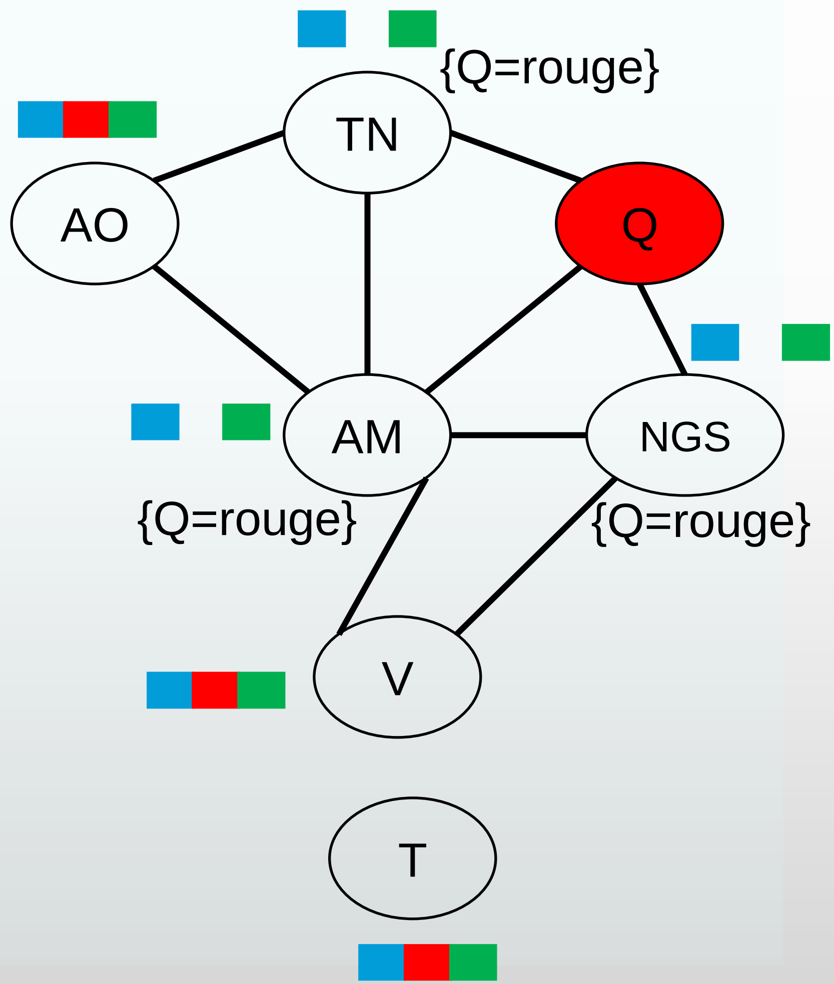


Illustration du backjumping

- Assignations : {Q=rouge, NGS=vert}.

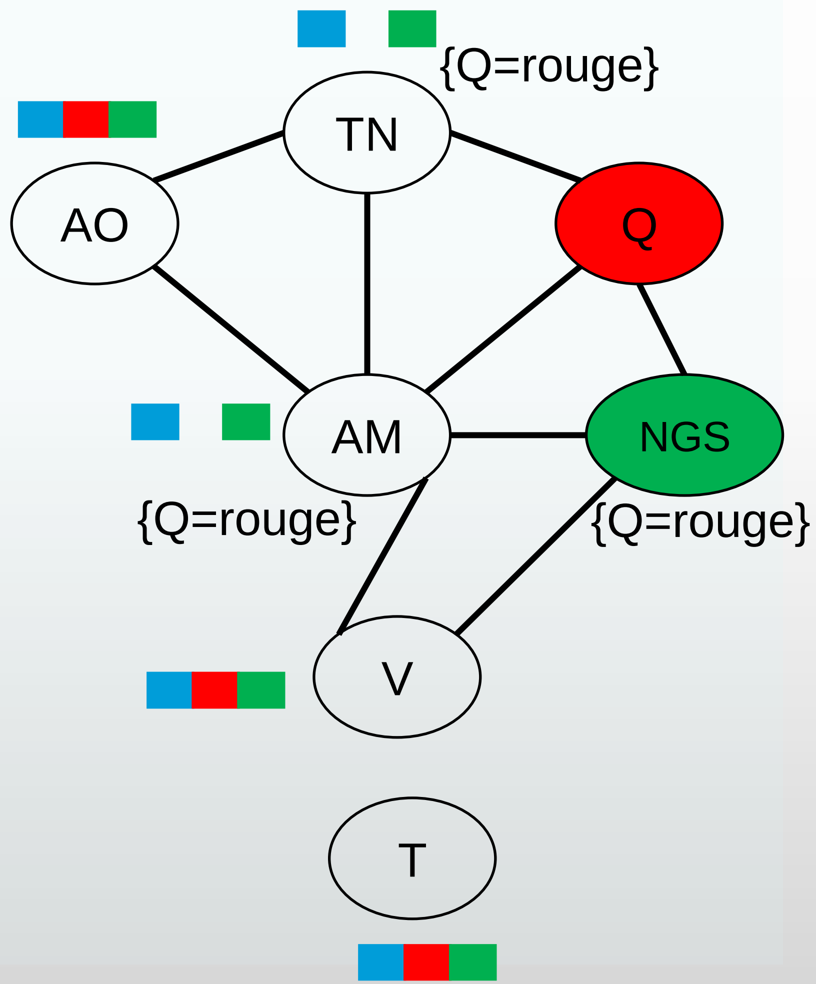


Illustration du backjumping

- Assignations : {Q=rouge, NGS=vert}.
- Calcul des l'ensembles de conflit.

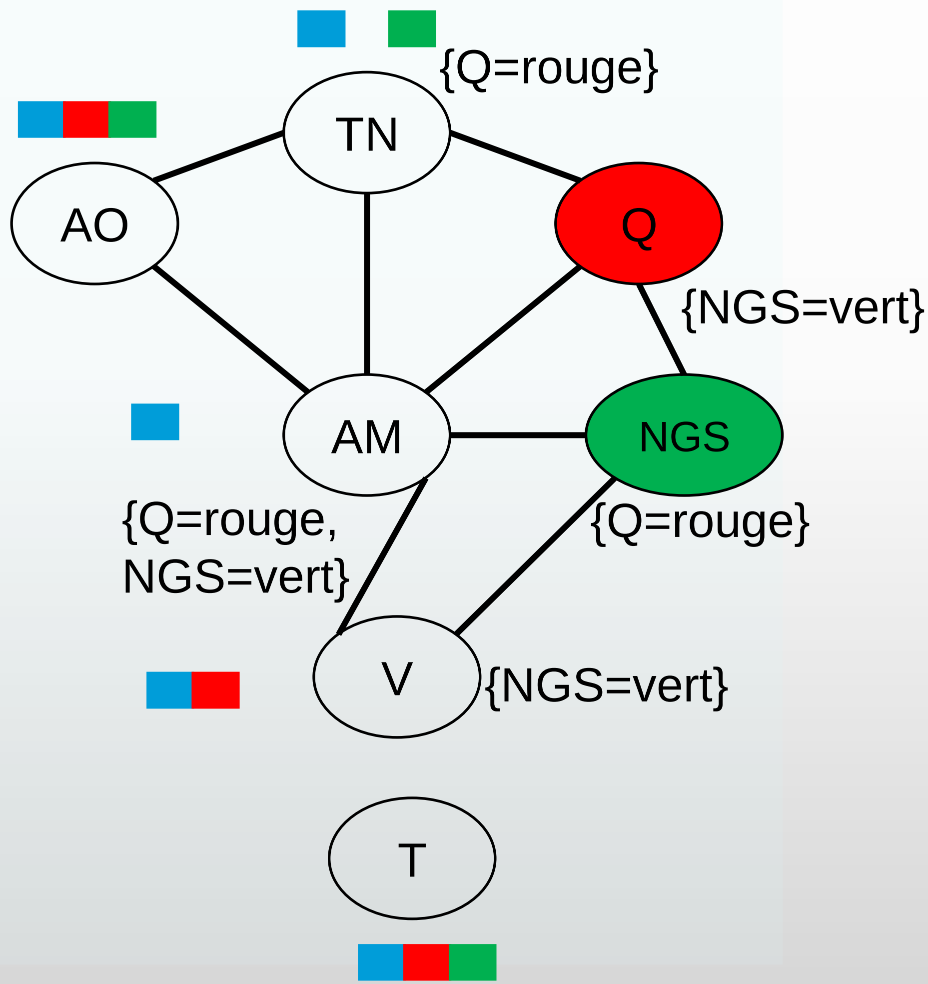


Illustration du backjumping

- Assignations : {Q=rouge, NGS=vert, V=bleu}.

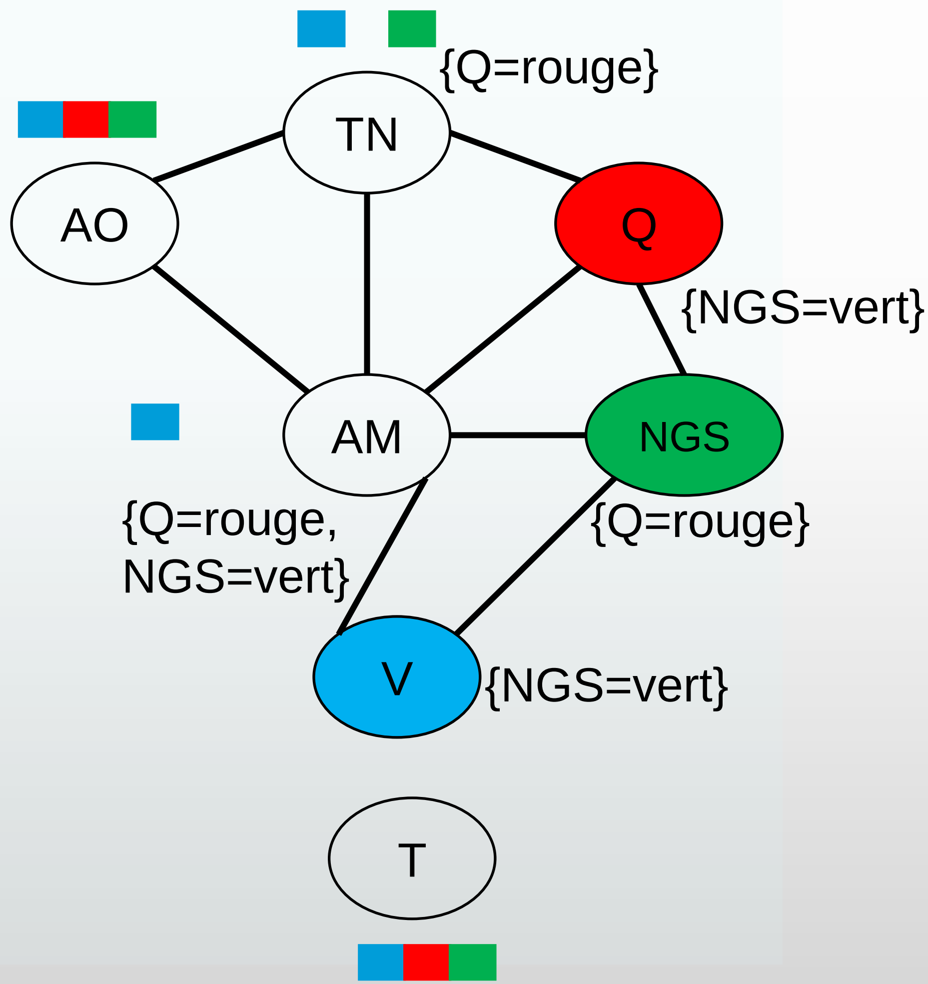


Illustration du backjumping

- Assignations : {Q=rouge, NGS=vert, V=bleu}.
- Calcul des l'ensembles de conflit.

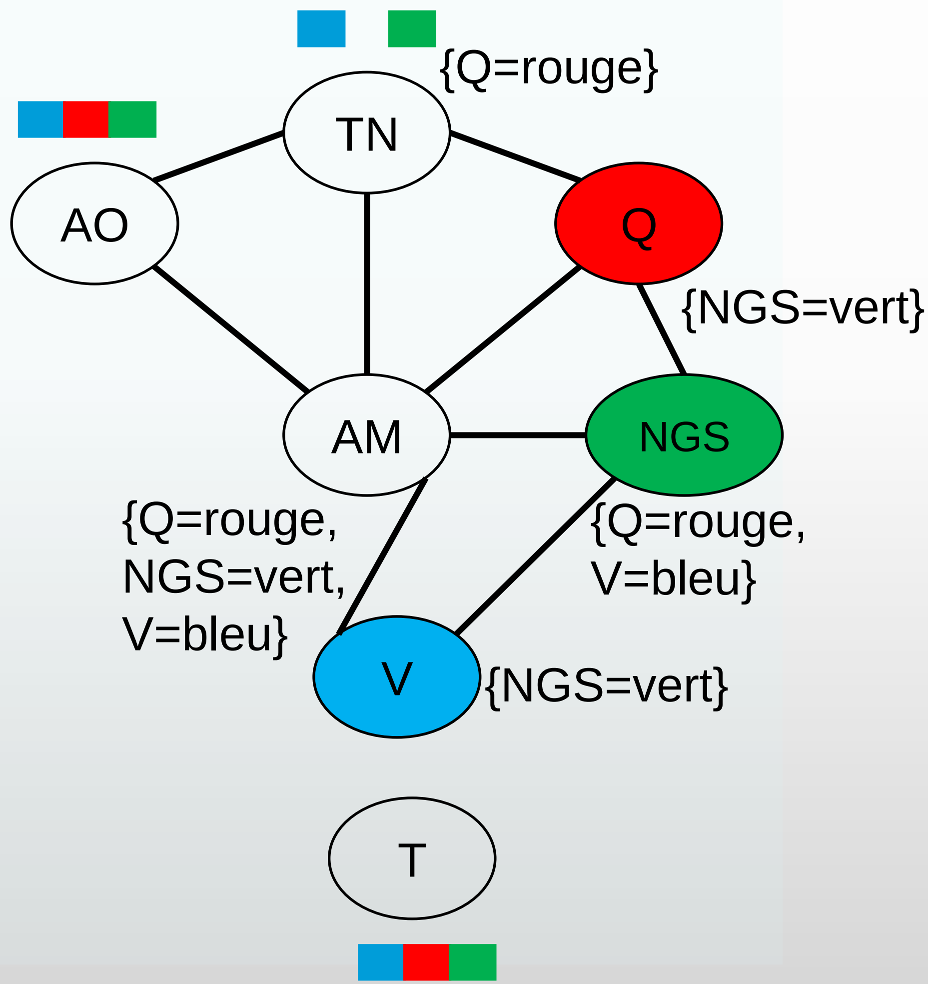


Illustration du backjumping

- Assignations : {Q=rouge, NGS=vert, V=bleu}.
- Calcul des l'ensembles de conflit.
- Domaine de AM est vide ⇒ ajouter les conflits de AM à conflits de V.

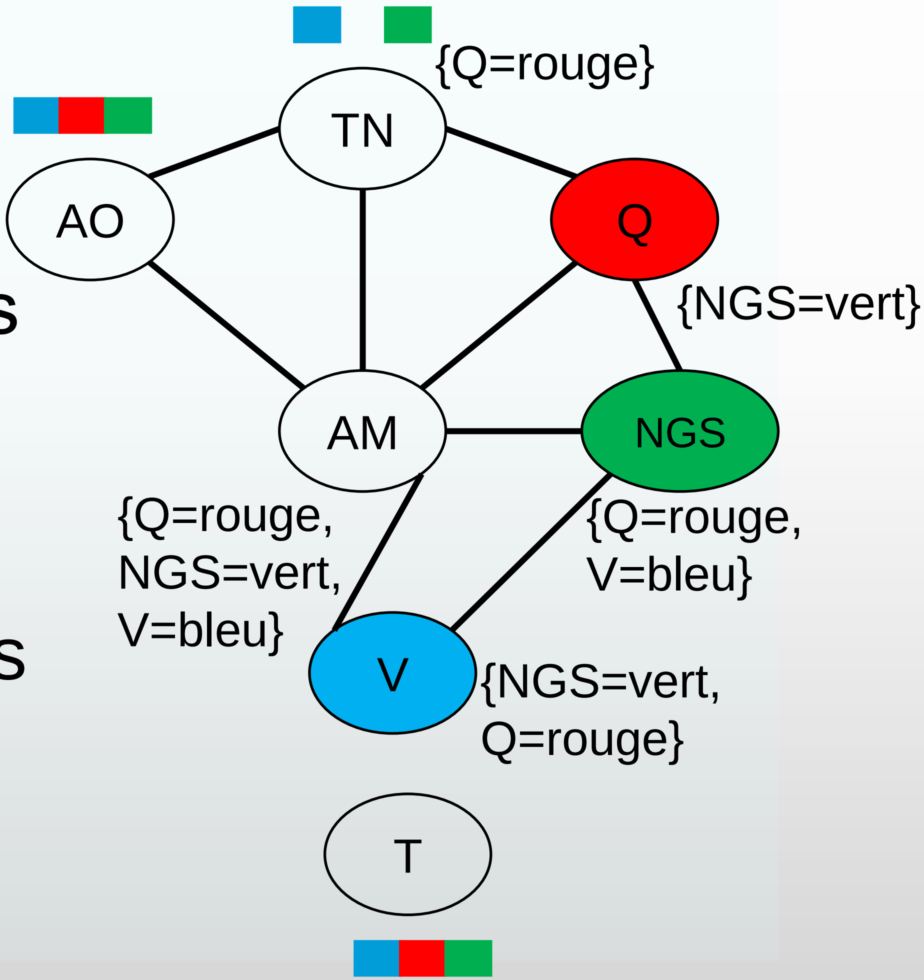


Illustration du backjumping

- Assignations : {Q=rouge, NGS=vert, V=bleu, T=rouge}.

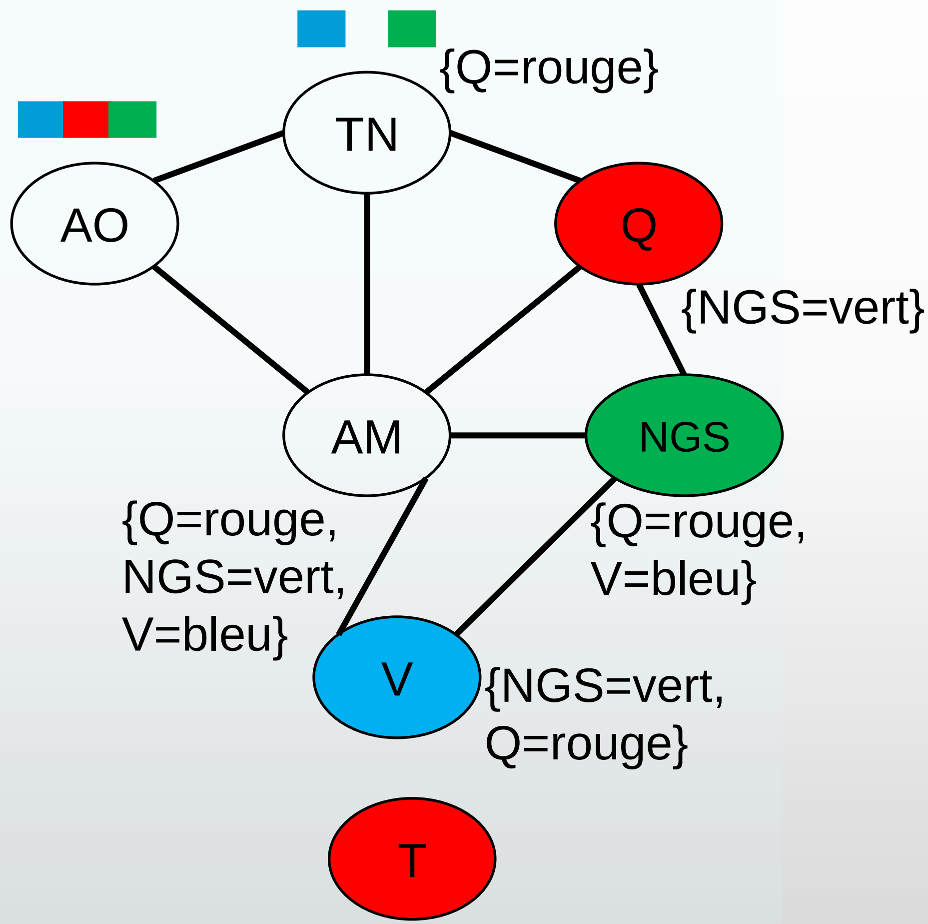


Illustration du backjumping

- Assignations : {Q=rouge, NGS=vert, V=bleu, T=rouge}.
- Calcul des l'ensembles de conflit : rien à faire.

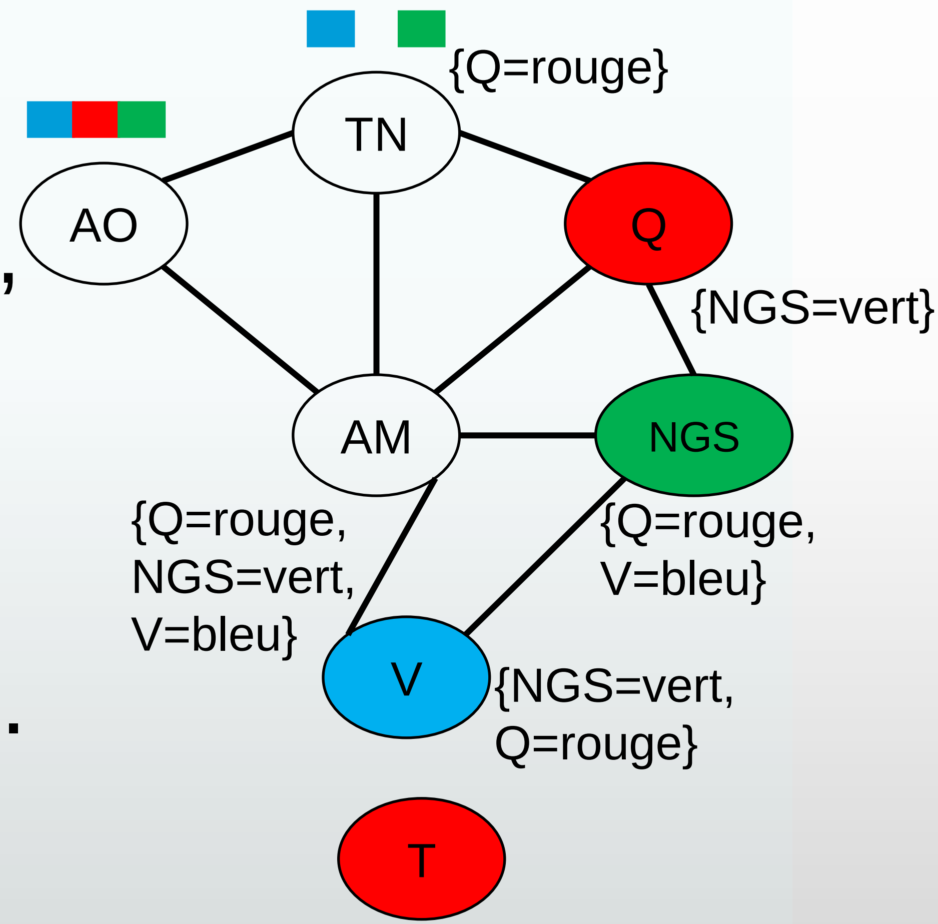


Illustration du backjumping

- Assignations : {Q=rouge, NGS=vert, V=bleu, T=rouge}.
- AM n'a plus de valeurs légales.

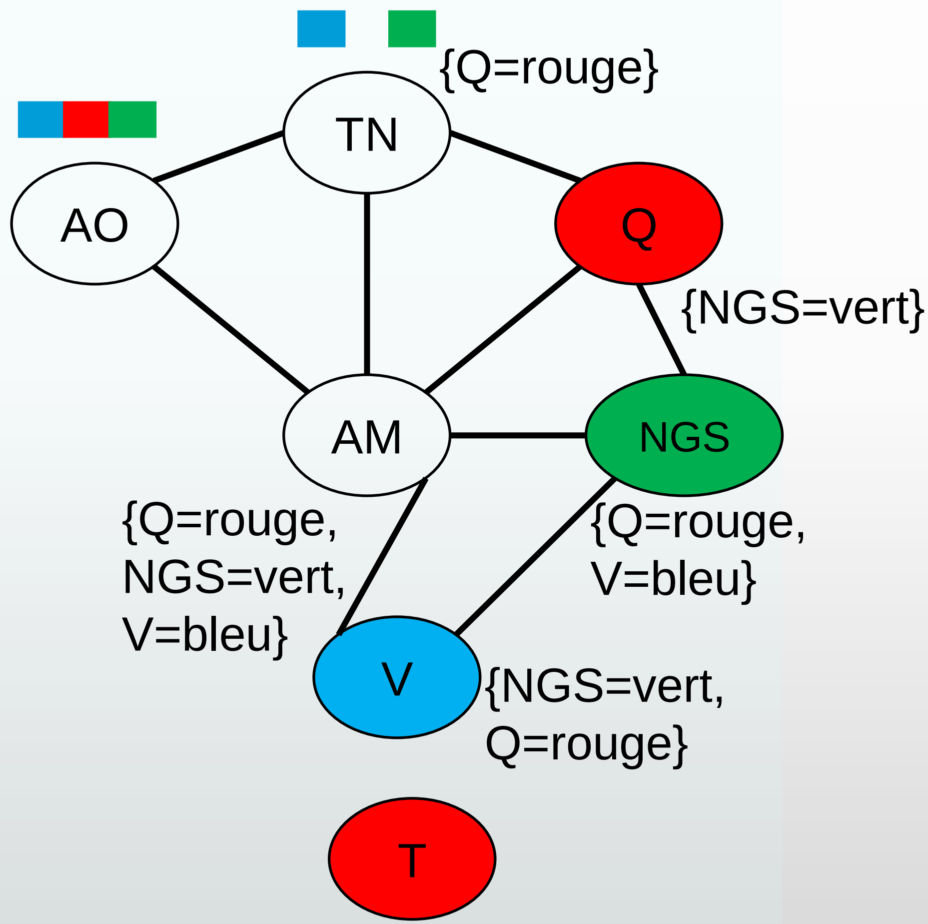


Illustration du backjumping

- Assignations : {Q=rouge, NGS=vert, V=bleu, T=rouge}.
- AM n'a plus de valeurs légales.
- Le backtracking reviendra à T=rouge : ne résout pas le problème.

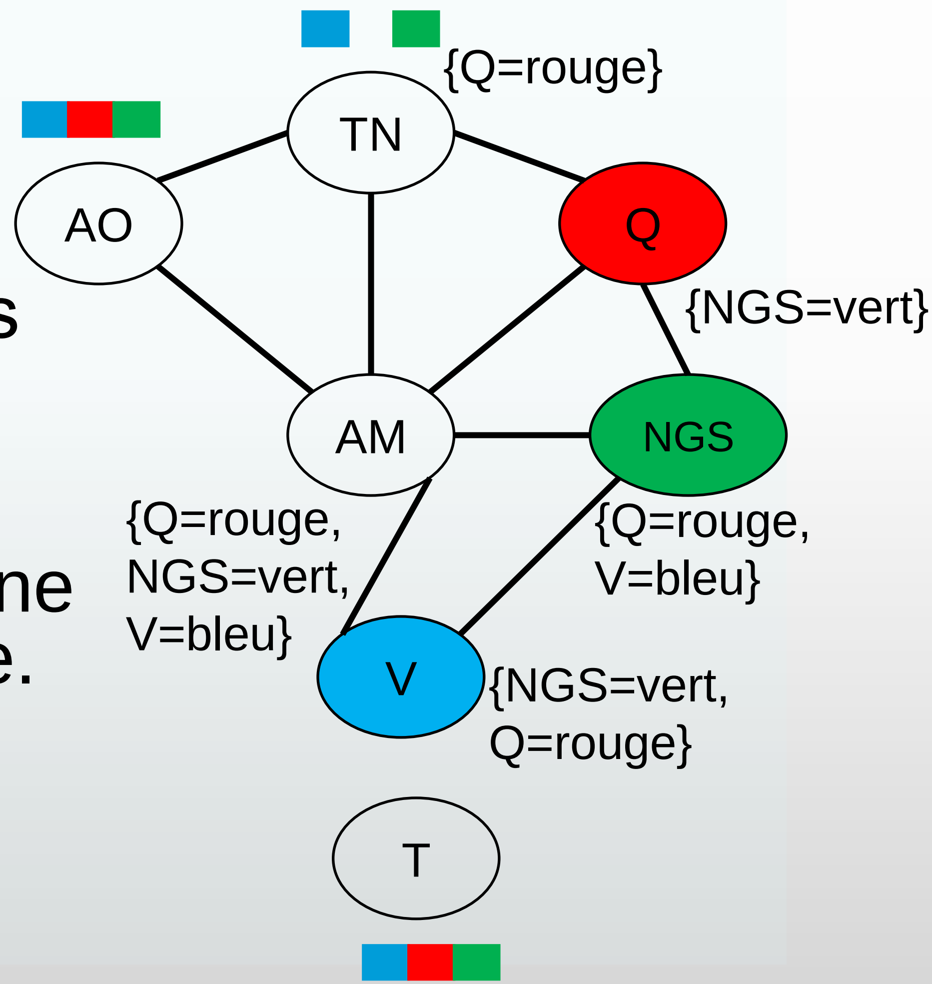
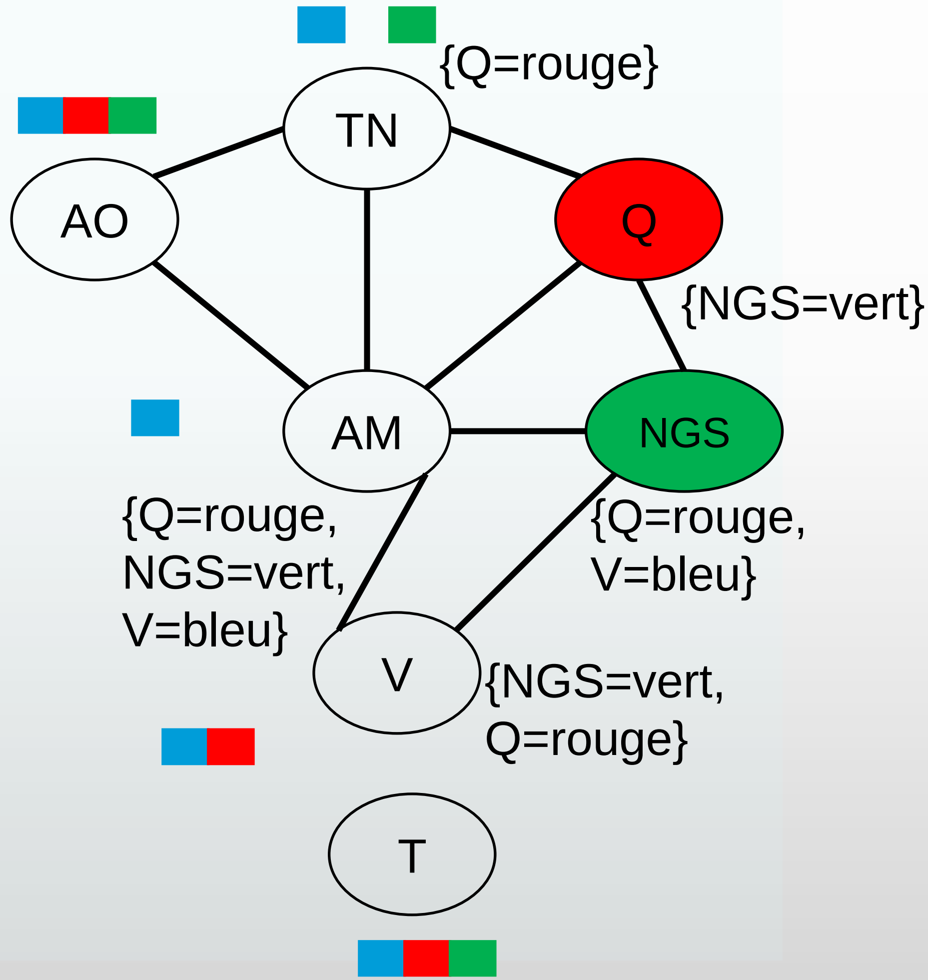


Illustration du backjumping

- Assignations : {Q=rouge, NGS=vert, V=bleu, T=rouge}.
- AM n'a plus de valeurs légales.
- Les backtracking reviendra à T=rouge : ne résout pas le problème.
- Le backjumping reviendra à l'élément le plus récent dans l'ensemble de conflit de AM : V=bleu.

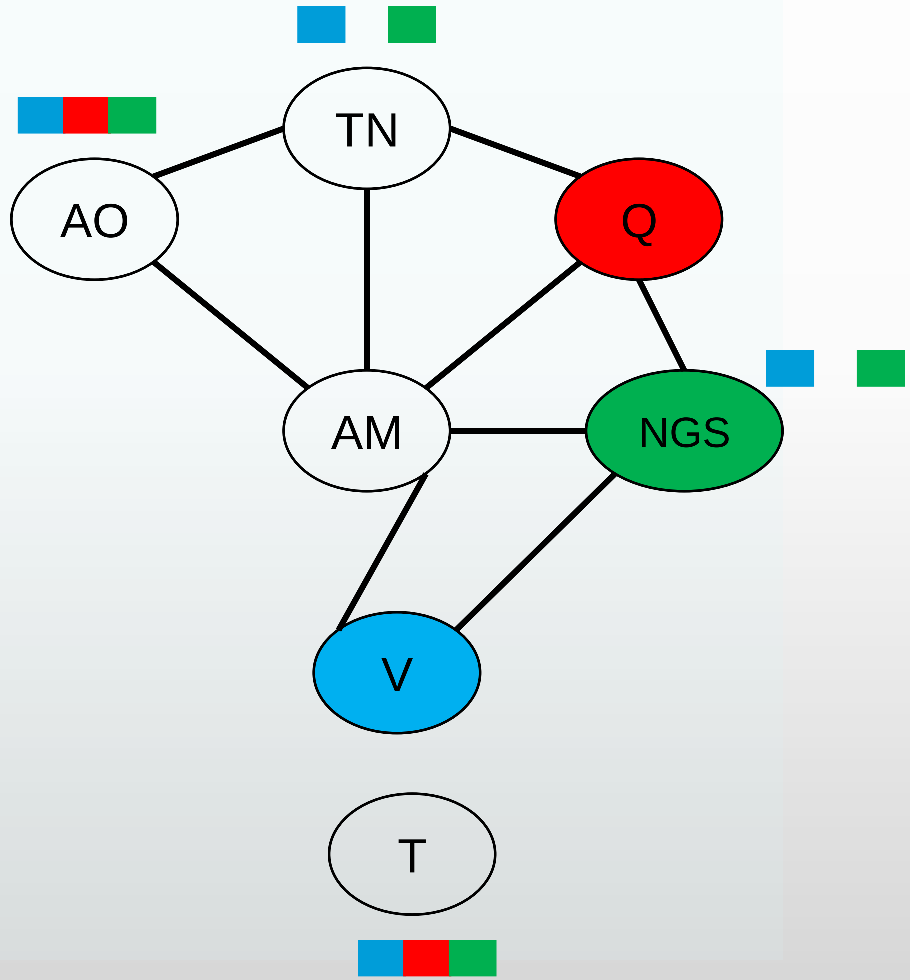


Retour arrière intelligent : examen en amont

- Cependant le backjumping se produit lorsque toutes les valeurs d'un domaine sont en conflit avec l'assignation courante.
- Mais le forward checking détecte déjà ceci et empêche l'exploration d'atteindre ce nœud : toute branche enlevée par backjumping dans l'arbre de recherche est aussi enlevée par forward checking.
- Pour que le backjumping soit utile il faut réviser la notion de conflit.

Rappel : exécution du forward checking sur l'exemple

- Après l'assignation V=bleu, le domaine de AM devient vide et le retour arrière est fait.

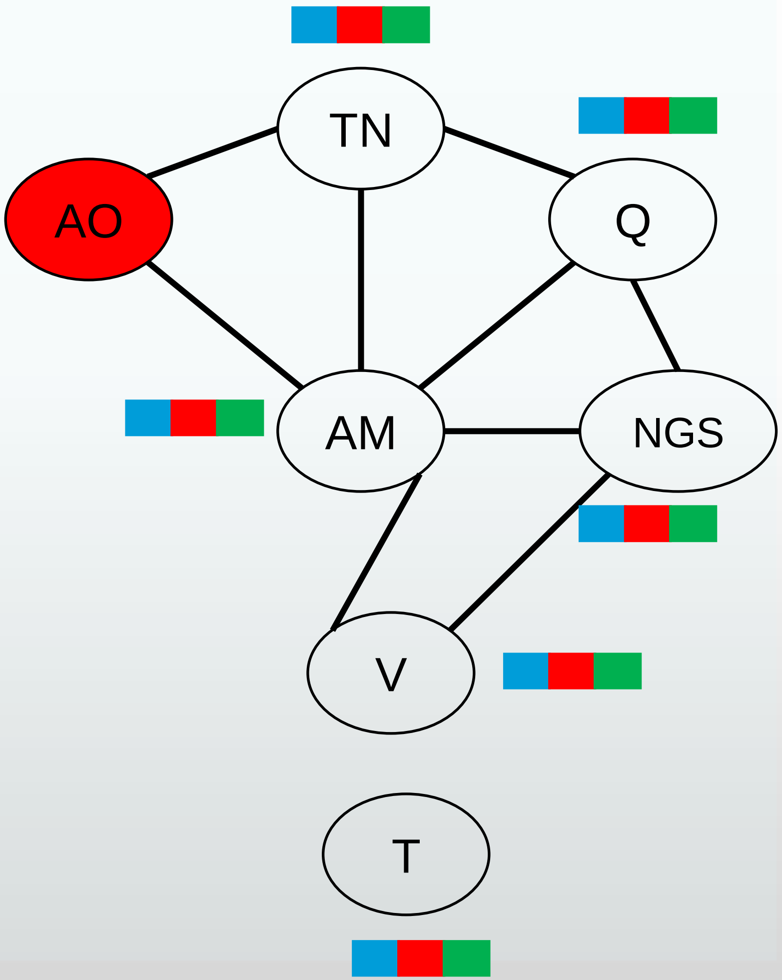


Backjumping orienté conflit

- Lors d'une recherche, si une variable *X* a un domaine vide :
  - Effectuer un backjumping sur la variable la plus récente de l'ensemble de conflit de *X*, soit *Y*.
  - L'ensemble de conflit de *Y* est mis à jour :
    - $conf(Y) \leftarrow conf(Y) \cup conf(X) - \{Y\}$

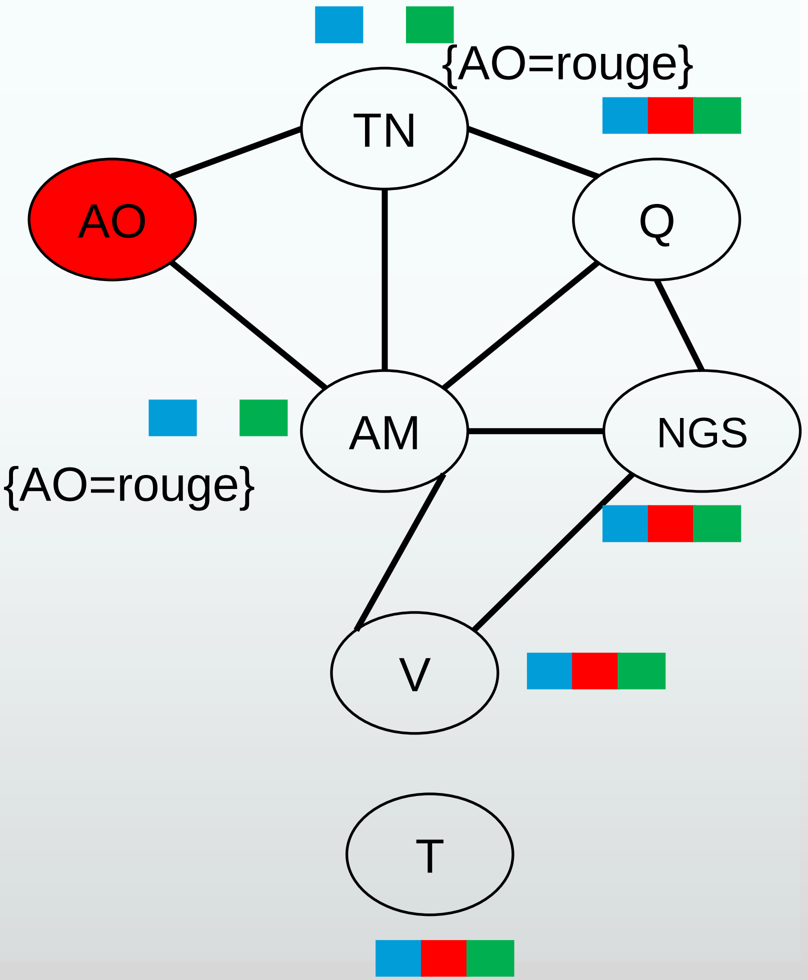
Backjumping orienté conflit : exemple

- Ordre des assignations : AO, NGS, T, TN, Q, V; AM.



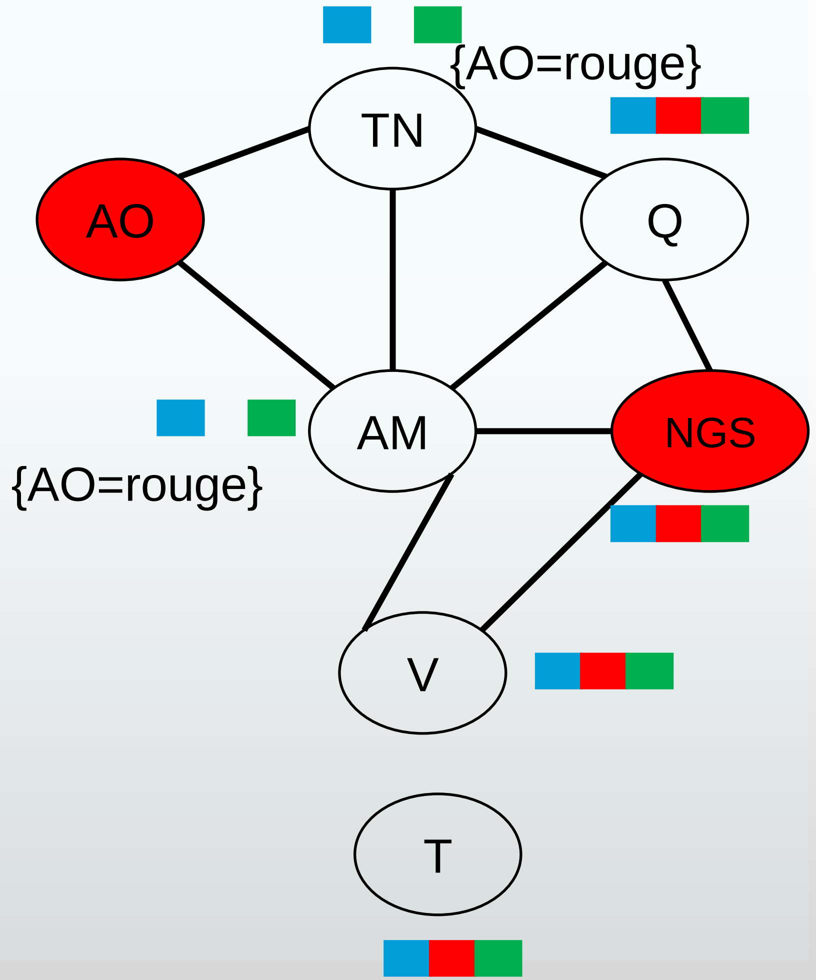
Backjumping orienté conflit

- Ordre des assignations : AO, NGS, T, TN, Q, V; AM.



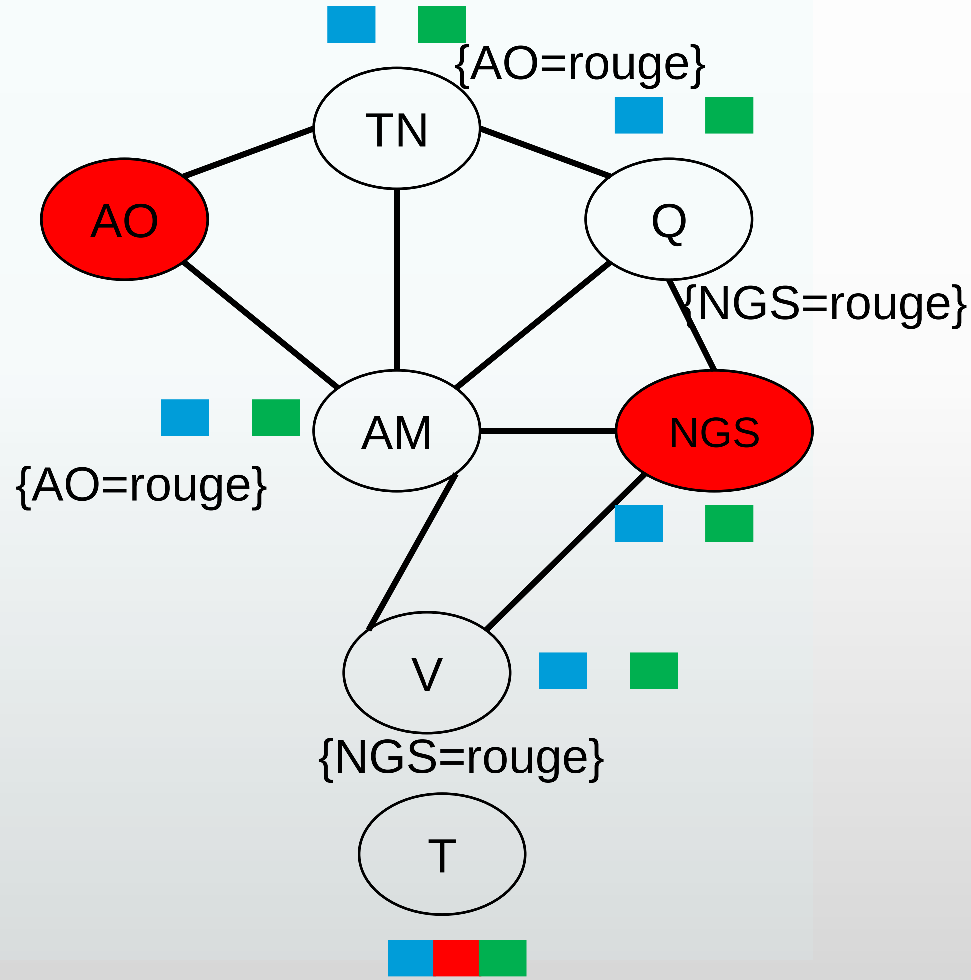
Backjumping orienté conflit

- Ordre des assignations : AO, NGS, T, TN, Q, V; AM.



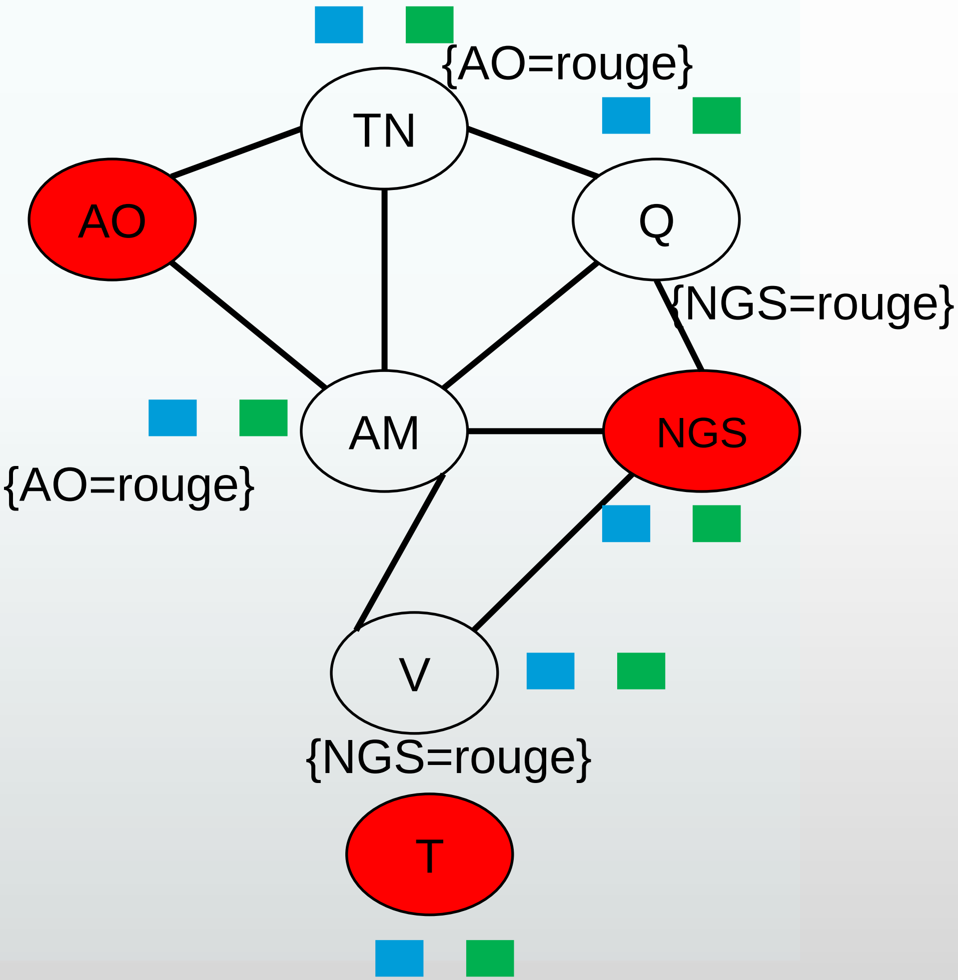
Backjumping orienté conflit

- Ordre des assignations : AO, NGS, T, TN, Q, V; AM.



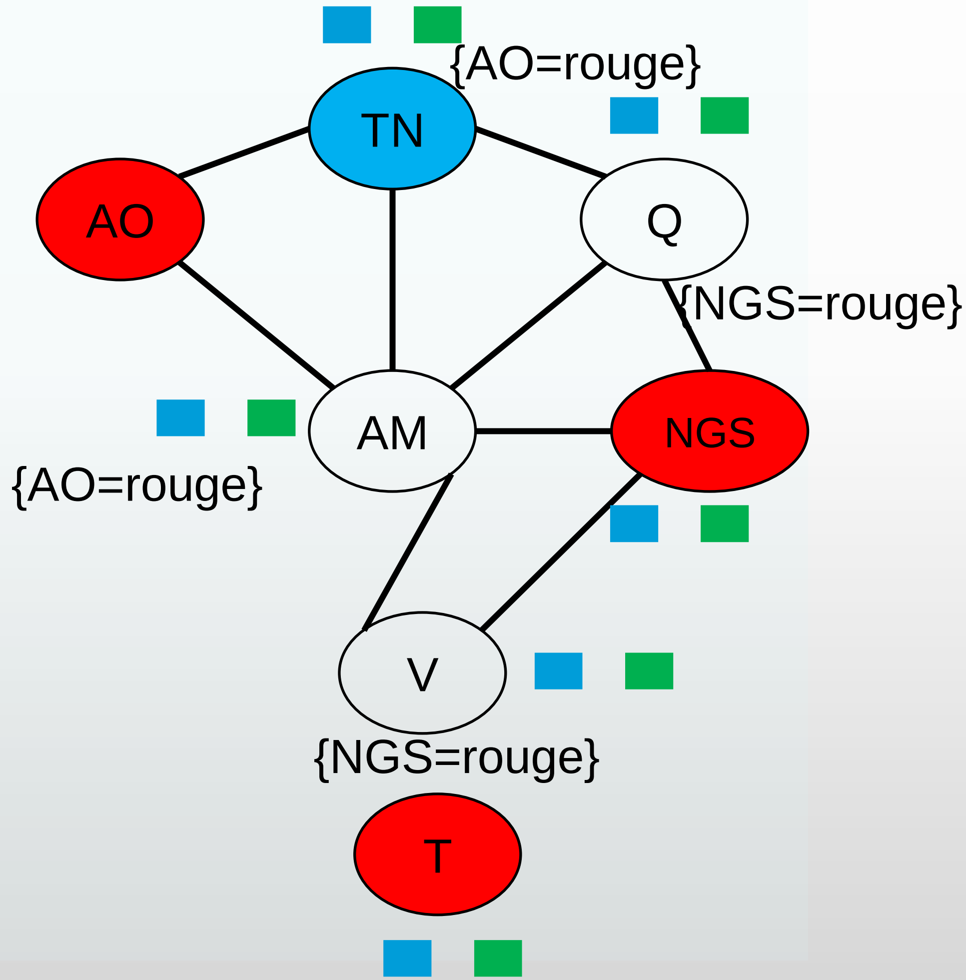
Backjumping orienté conflit

- Ordre des assignations : AO, NGS, T, TN, Q, V; AM.



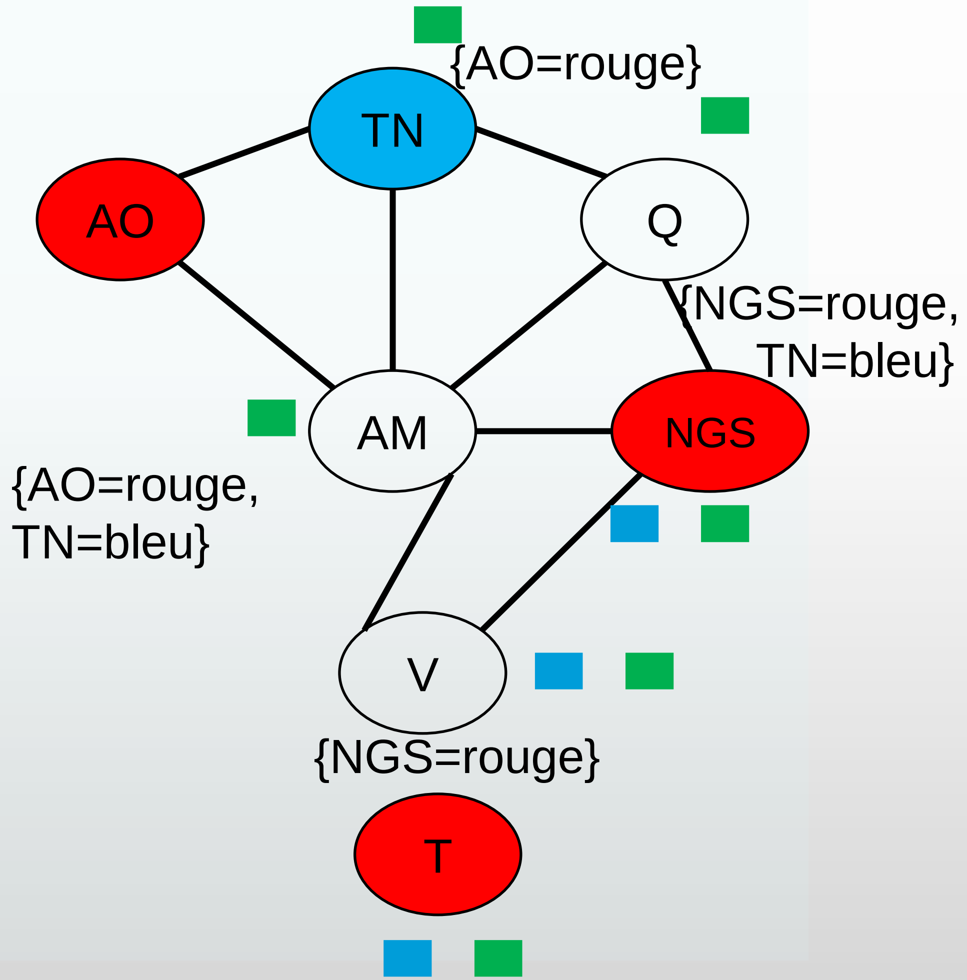
Backjumping orienté conflit

- Ordre des assignations : AO, NGS, T, TN, Q, V; AM.



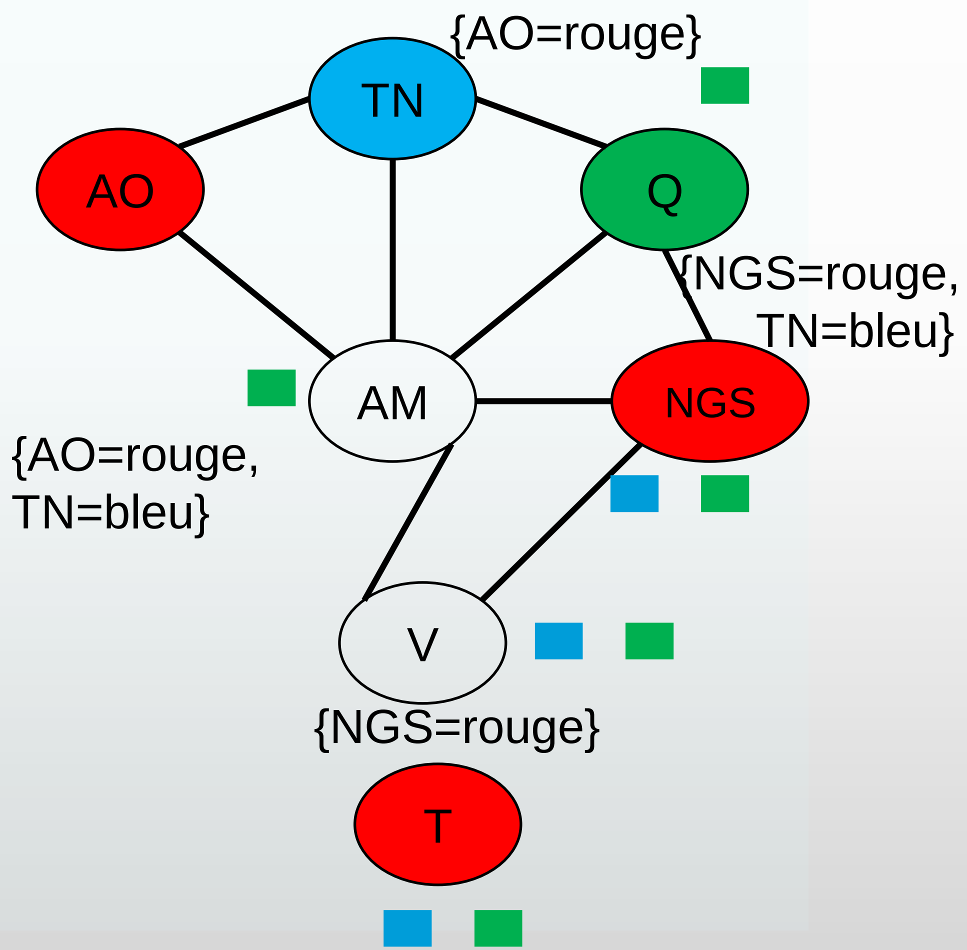
Backjumping orienté conflit

- Ordre des assignations : AO, NGS, T, TN, Q, V; AM.



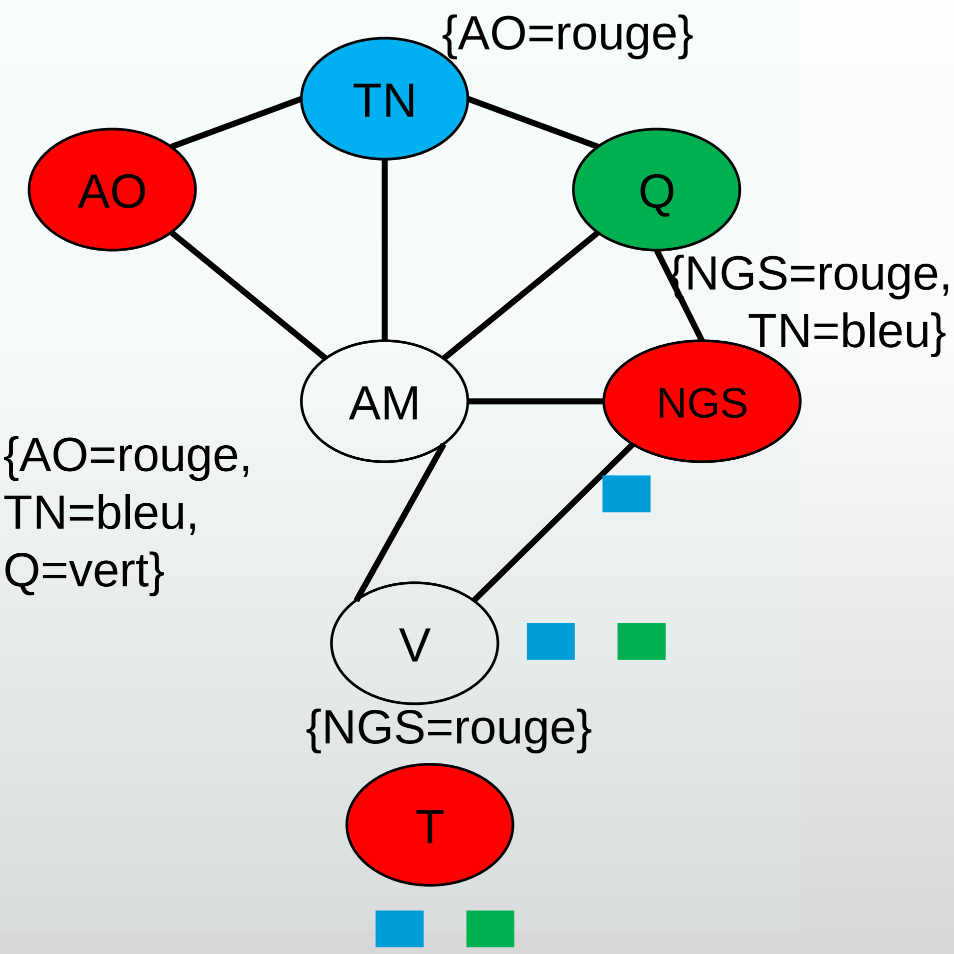
Backjumping orienté conflit

- Ordre des assignations : AO, NGS, T, TN, Q, V; AM.



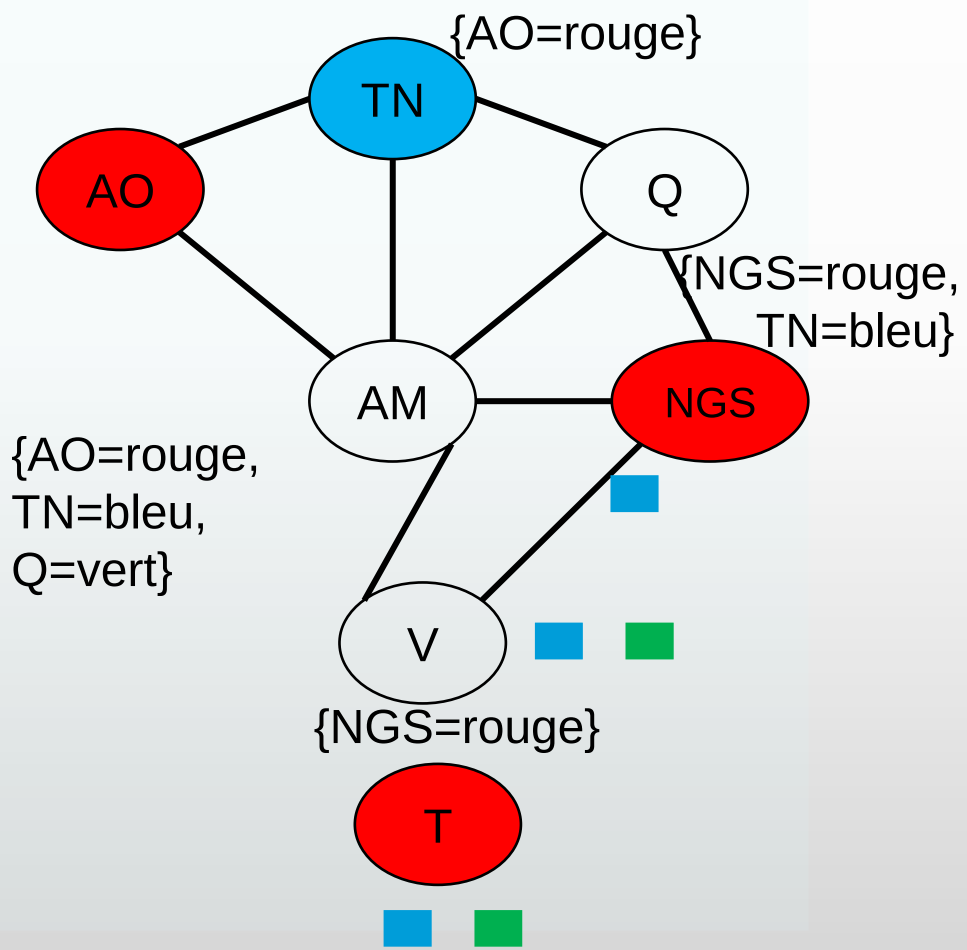
Backjumping orienté conflit

- Ordre des assignations : AO, NGS, T, TN, Q, V; AM.



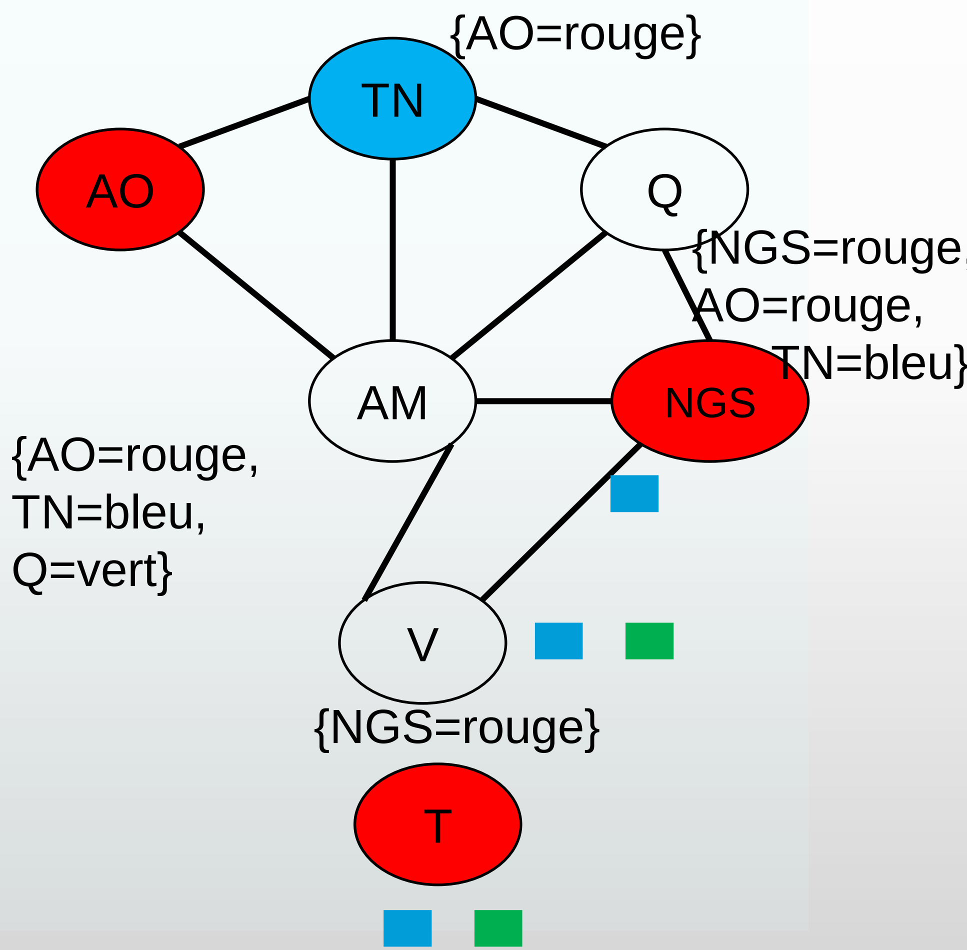
Backjumping orienté conflit

- Ordre des assignations : AO, NGS, T, TN, Q, V; AM.
- Le domaine de AM est vide ⇒ remonter à Q.



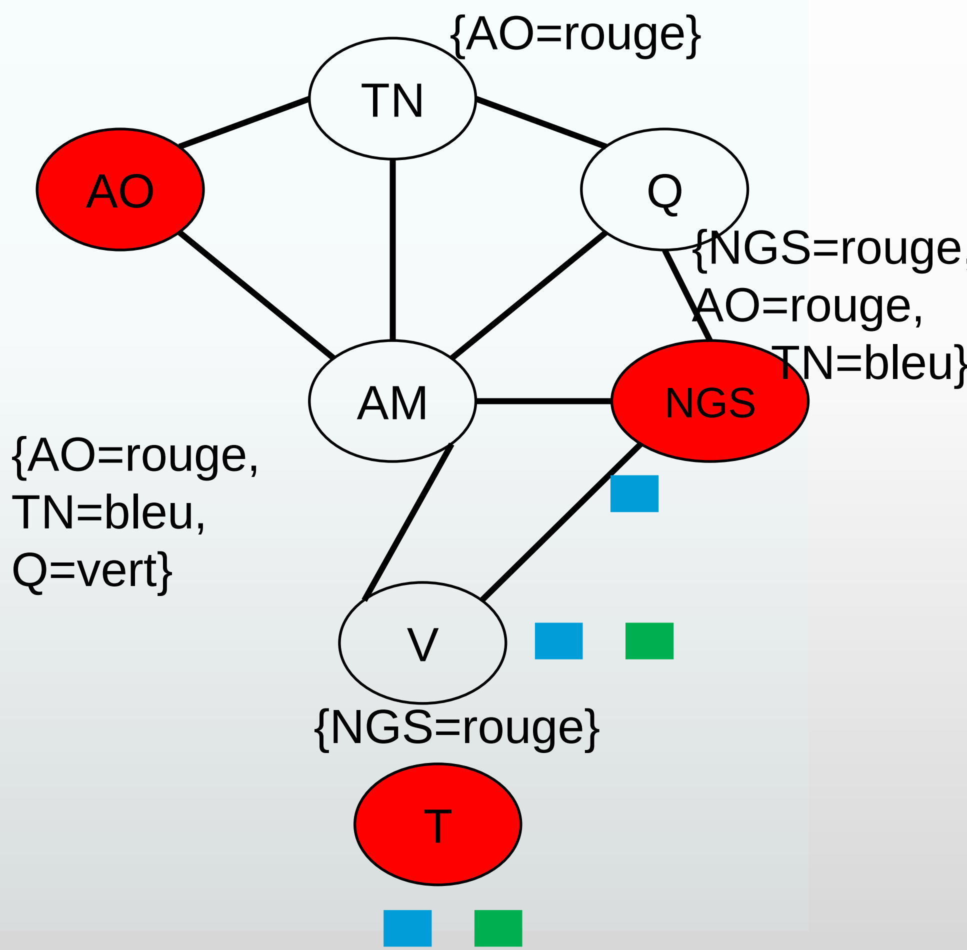
Backjumping orienté conflit

- Ordre des assignations : AO, NGS, T, TN, Q, V; AM.
- Echec pour Q, remonter à TN.



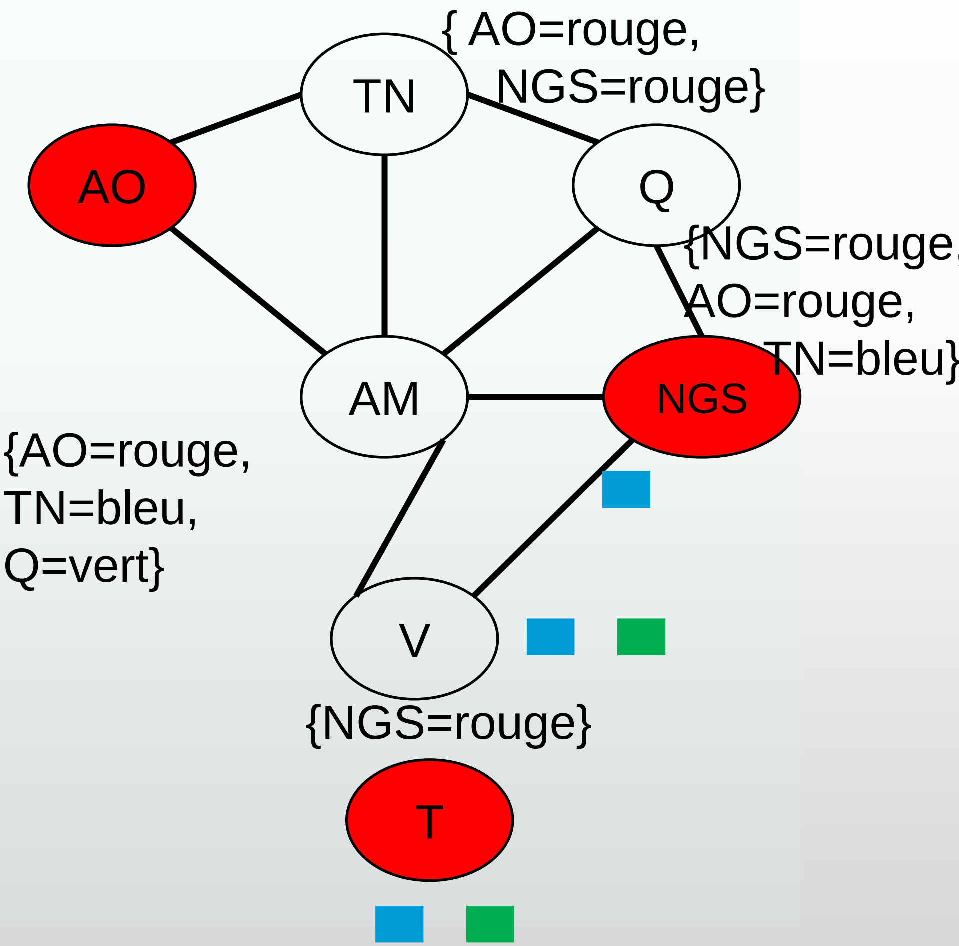
Backjumping orienté conflit

- Ordre des assignations : AO, NGS, T, TN, Q, V; AM.
- TN absorbe l'ensemble de conflit de Q sauf TN.



Backjumping orienté conflit

- Ordre des assignations : AO, NGS, T, TN, Q, V; AM.
- TN absorbe l'ensemble de conflit de Q sauf TN.
- Et remontre ainsi vers NGS.



Recherche locale pour les CSPs

Recherche locale pour les CSP

- Les algorithmes d'exploration locale peuvent être très efficaces pour la résolution de CSP:
  - Ils utilisent une formulation par états complets : l'état initial assigne une valeur à chaque variable (habituellement non cohérente) et la fonction successeur procède en modifiant une seule variable à la fois.
  - Pour choisir une nouvelle valeur pour une variable, l'heuristique la plus évidente, min-conflicts, sélectionne la valeur associée au nombre minimal de conflits avec les autres variables (minimiser h=nombre de conflits).

Recherche locale pour les CSP : algorithme MIN-CONFLICTS

```
fonction MIN-CONFLITS(csp, max_étapes) retourne une solution ou échec
  entrées: csp, un problème à satisfaction de contraintes
           max_étapes, le nombre d'étapes permises avant de renoncer

  courante ← une assignation initiale complète pour csp
  pour i = 1 to max_étapes faire
     si courante est une solution pour csp alors retourner courante
     var ← une variable conflictuelle choisie au hasard dans csp.VARIABLES
     valeur ← la valeur v pour var qui minimise CONFLITS(var, v, courante, csp)
     assigner var=valeur dans courante
  retourner échec
```

Min-Conflict

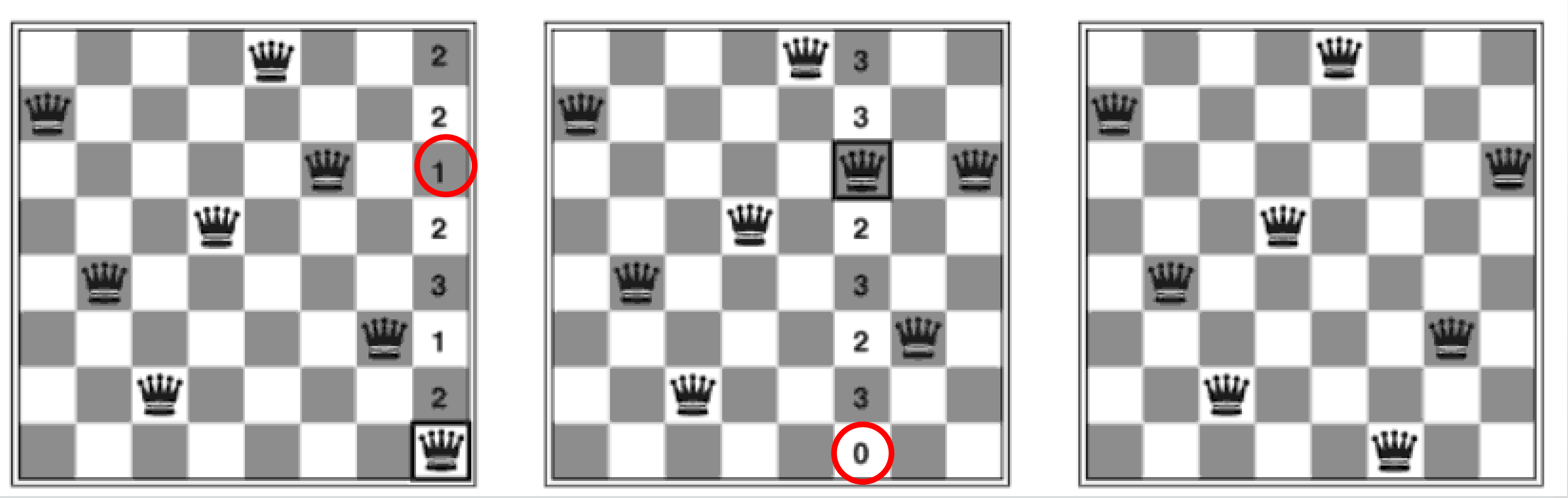
- L’algorithme Min-conflict est très efficace.
- Pour les N-Reines, la durée d’exécution de l’algorithme est indépendant de la taille du problème : pour 1 million de reines, le problème est résolu en 50 étapes en moyenne.
- L’algorithme a également été utilisé pour synchroniser les observations du télescope Hubble : le temps nécessaire pour planifier une semaine d’observations est passé de 3 semaines à 10 minutes.

Recherche locale pour les CSP : éviter les plateaux

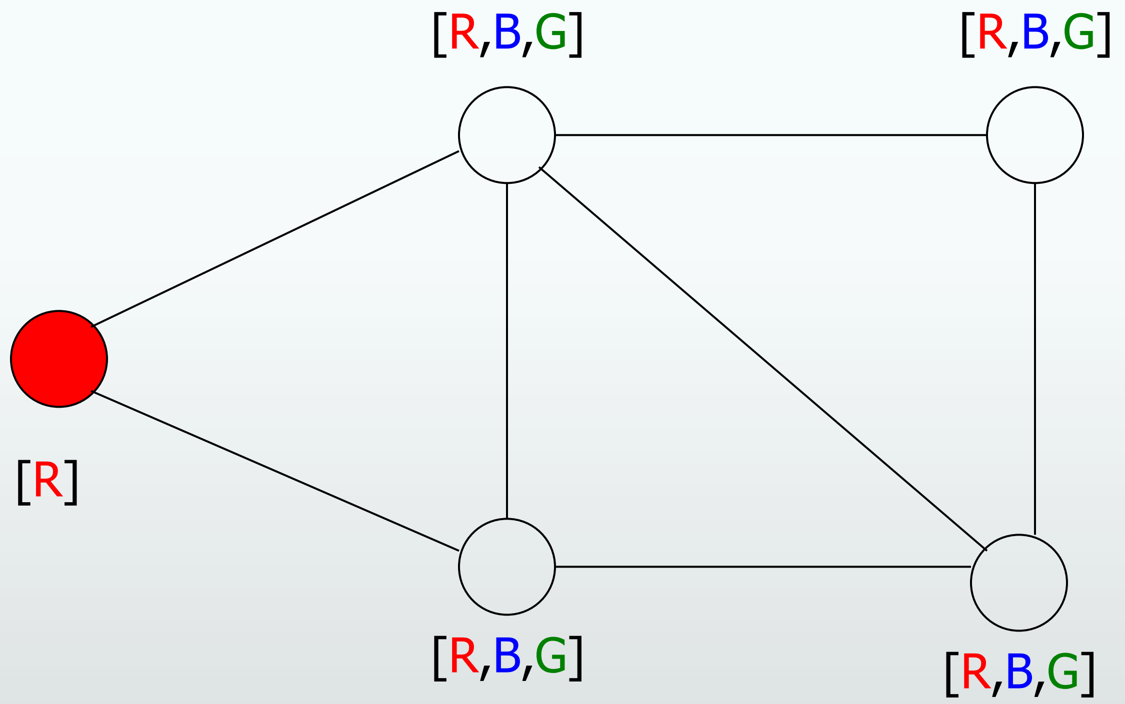
- L'exploration de plateau : permet de se déplacer latéralement à un autre état avec le même score.
- Exploration taboue : maintien une liste des états récemment visités et en interdisant à l'algorithme d'y retourner.
- Le recuit simulé peut aussi être utilisé.
- Pondération de contraintes : focalise la recherche sur les contraintes importantes en les pondérant itérativement :
  - Toutes les contraintes reçoivent initialement une pondération de 1 : Wi = 1.
  - A chaque étape, l'algorithme choisit une paire de variable/valeur à modifier pour obtenir le plus faible poids total sur toutes les contraintes violées.
  - Les poids sont ajustés en incrémentant de 1 les contraintes violées par l'assignation actuelle.

Recherche locale pour les CSP : exemple des n-reines

- Utilisation de l'algorithme min-conflits pour les 8-reines.
- Cherche à minimiser la fonction *objectif*.



Exercice



Exercice

- On s’intérresse 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 ?