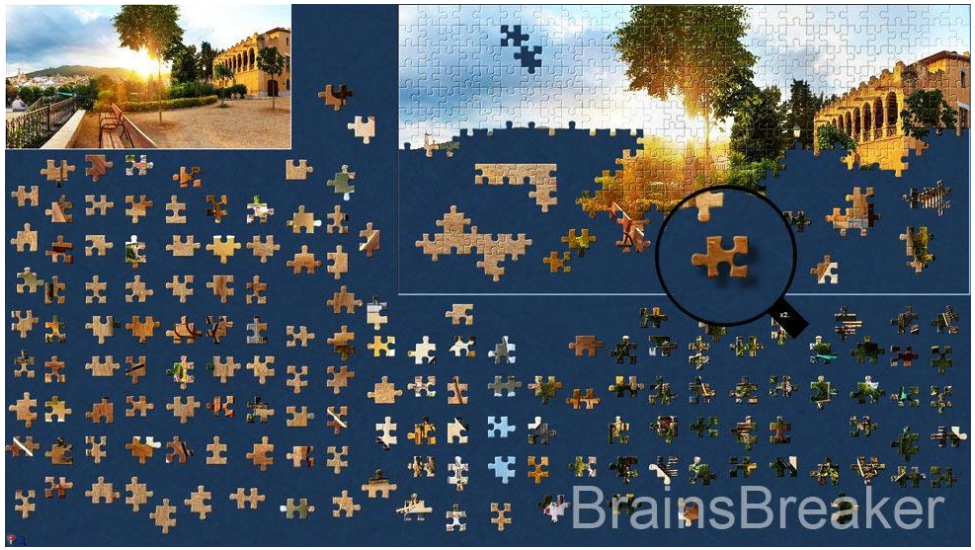
Universitat Autònoma de Barcelona



CERCA PER A SATISFACCCIÓ DE RESTRICCIONS

Resolució de problemes de presa de decisió
per exploració d'alternatives

Coneixement, Raonament i Incertesa.

Exemple: Assignar valors a les següents lletres, de manera que es verifiqui la següent suma:

$$\begin{array}{cccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

s'han de trobar valors del conjunt: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

que assignats a les variables: {D, E, Y, N, R, O, S, M}

Verifiquin la suma donada

Solució: {D=4, E=5, Y=9, N=5, R=0, O=0, S=9, M=1}

Problema de satisfacció de restriccions: és un problema en el que a partir d'un conjunt de variables:

$$\text{Variables} = \{X_1, X_2, \cdots, X_n\}$$

i a partir d'un domini de valors que poden prendre aquestes variables:

$$\text{Domini} = \{v_1, v_2, \cdots, v_k\}$$

es vol trobar un conjunt d'assignacions de totes les variables amb valors del domini que verifiquin un conjunt de

Restriccions del problema

Arbre de satisfacció de restriccions: és una representació de coneixement que és un arbre semàntic amb les següents particularitats:

Lèxic: restriccions, conjunt de variables, variables assignades i no-assignades, domini de les variables,

Estructural:

b=#Domini de les variables (factor d'expansió)

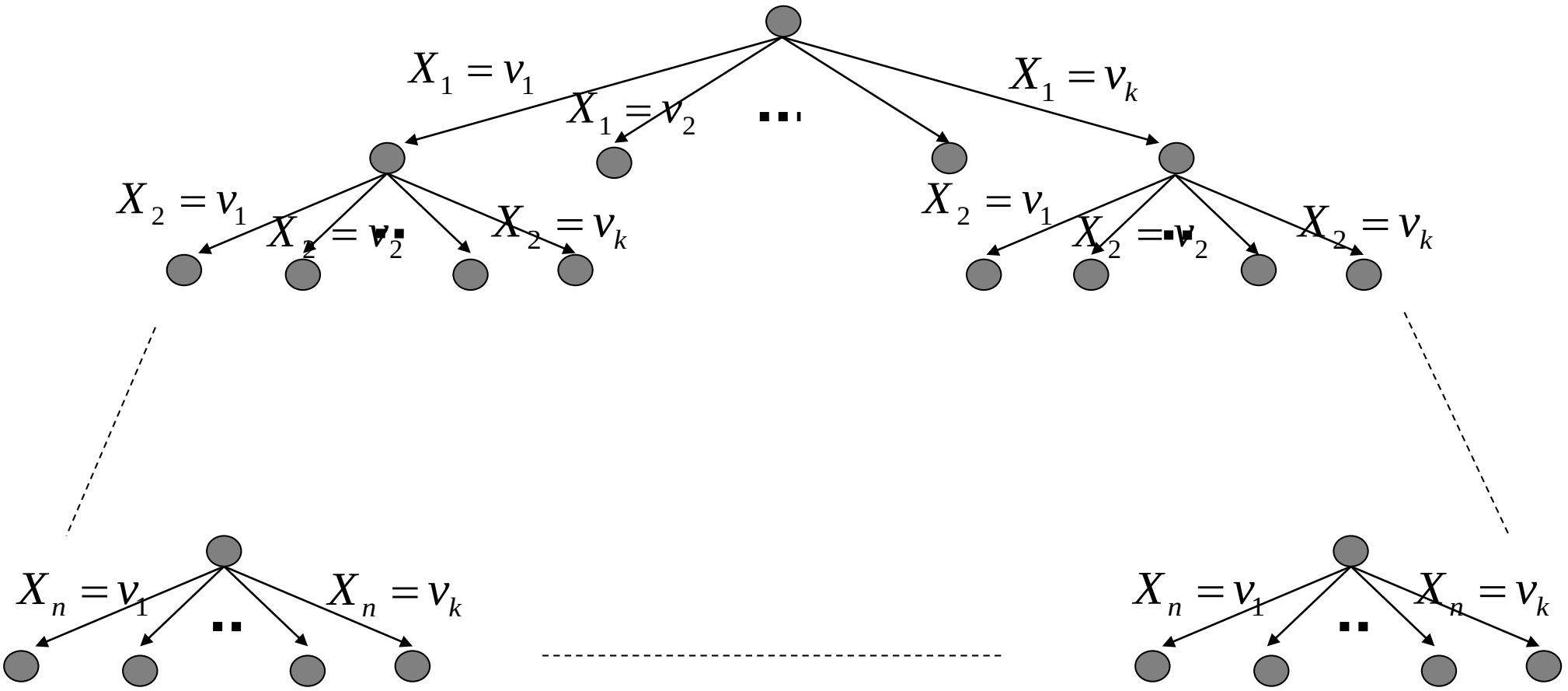
d=#Conjunt de variables (profunditat de l'arbre)

Semàntica: Node= Comprovació de restriccions sobre les variables assignades.
Branca= Assignació d'un valor del domini a una variable.

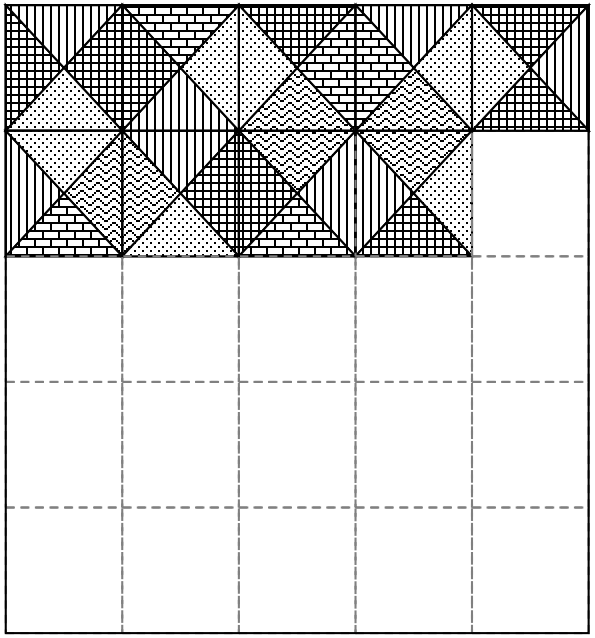
Procedimental:

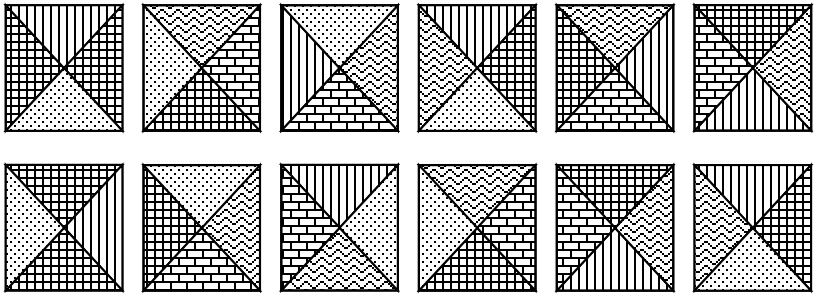
- Proc. Backtracking.
- Proc. Backtracking amb forward checking.
- Proc. comproven les restriccions sobre un conjunt de variables assignades.

Arbre de satisfacció de restriccions: b=k, prof.=n



Exercici: Descriviu el problema de resoldre un puzzle com un problema de satisfacció de restriccions.





$$\text{Variables} = \{X_1, X_2, \cdots, X_n\} = ?$$

$$\text{Domini} = \{v_1, v_2, \cdots, v_k\} = ?$$

Restriccio ns ?

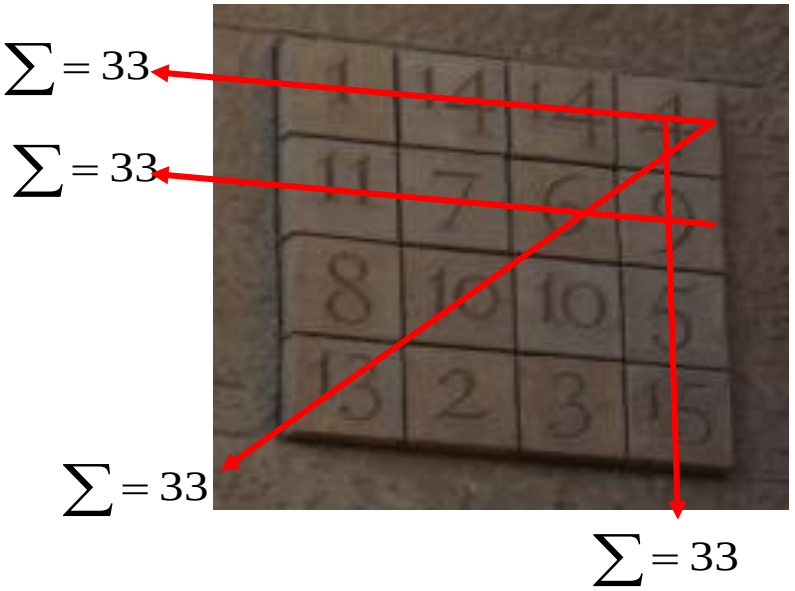
Exercici: Descriviu el problema de la construcció d'un quadrat màgic, com un problema de satisfacció de restriccions

$$\text{Variables} = \{X_1, X_2, \cdots, X_n\} = ?$$

$$\text{Domini} = \{v_1, v_2, \cdots, v_k\} = ?$$

$$\text{Restriccio ns} ?$$

6	1	8	$\sum = 15$
7	5	3	$\sum = 15$
2	9	4	$\sum = 15$
			$\sum = 15$



```
Funcio Backtracking(LVA,LVNA,R,D)
    Si (LVNA és buida) llavors Retornar(LVA) fSi
    Var=Cap(LVNA);
    Per a cada (valor del Domini(Var, D) que podem assignar a Var) fer
        Si (SatisfaRestriccions([Var valor],LVA,R)) llavors
            Res=Backtracking(Insertar([Var, valor],LVA),Cua(LVNA),R,D);
            Si (Res és una solució completa) llavors
                Retornar(Res);
            Fsi
        Fsi
    Fper
    Retornar(Falla)
FFuncio
```

SatisfaRestriccions(A,LVA,R): Retorna cert si l'assignació A afegida la llista d'assignacions LVA satisfan les restriccions R.

Domini(V,D): Retorna un valor del domini D per a la variable V.

Insertar(e,L):Retorna la llista resultant d'afegir e al principi de L.

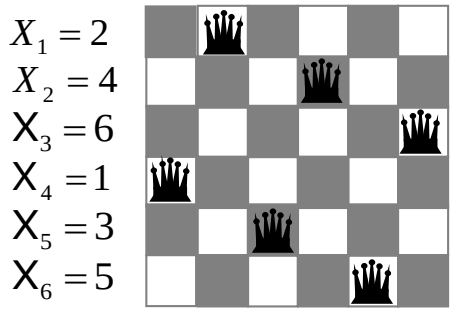
Cua(L) i Cap(L): Retornen la cua i el cap de L, respectivament.

Tipus de restriccions

- Restriccions unaries: una única variable involucrada.
  - e.g. $X \neq$ *verd*
- Restriccions binaries: dues variables involucrades.
  - e.g. $X \neq Y$
- Restriccions ordre superior: 3 o més variables involucrades.
  - e.g. restriccions criptoaritmetiques.
- Preferencies (soft constraints) e.g. *vermell* és millor que el *verd*. Sovint es representa per una funció de cost d'assignació per a cada variable $\rightarrow$ problemes de d'optimització restringida.

Exemple: Descripció del problema de les N reines sobre una taulell de NxN com un problema de satisfacció de restriccions.

Exemple del cas N=6



$$Variables = \{X_1, X_2, \cdots, X_N\} = V$$

X_i : Representa la reina de la fila i

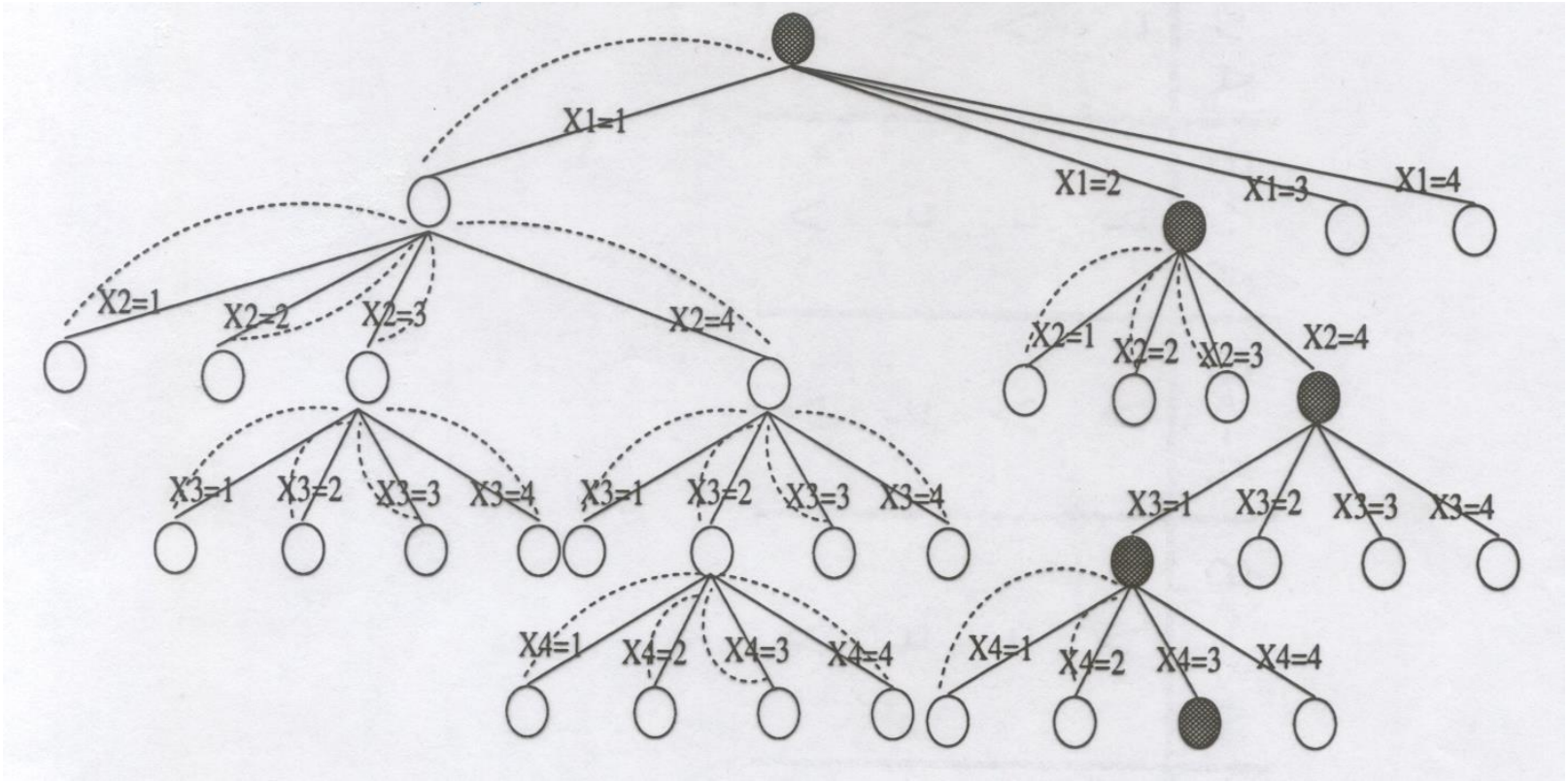
$$Domini = \{1,...,N\} = D$$

k : Columna on es posa la reina

Restriccions:

$$\forall X_i, X_j \in V : j > i \Rightarrow \begin{cases} X_i \neq X_j \\ X_j \neq X_i + (j-i) \\ X_j \neq X_i - (j-i) \end{cases}$$

L'arbre que generarà el *backtracking* pel problema de les 4 reines



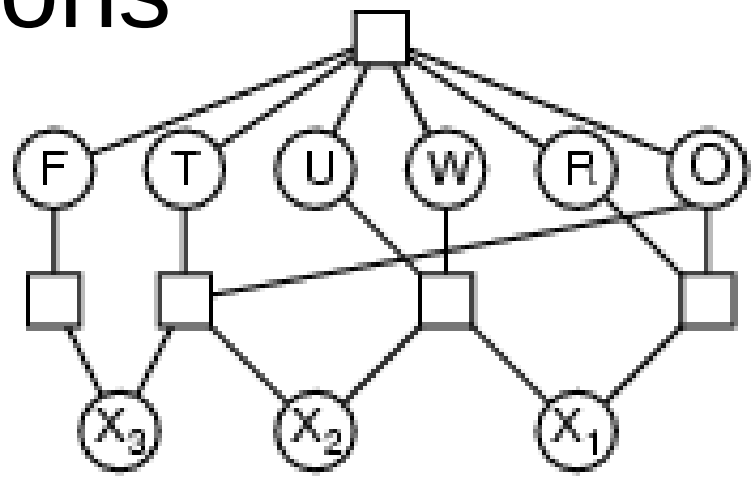
Sobre l'eficiència de la representació del coneixement

- **Formulació 1**:
  Força bruta $\binom{64}{8} = 4.426.165.368$ 92 solucions
- **Formulació 2**: Donat que no pot haver-hi més d'una reina per fila, podem replantejar el problema com:
  "*col·locar una reina en cada fila del taulell de forma que no s'amenacin*".
  En aquest cas, per a veure si dues reines s'amenaçen només cal veure si comparteixen columna o diagonal.
  Per tant, tota solució del problema pot representar-se amb una 8-tupla $(x_1,\ldots,x_8)$ en la que x_i és la columna en la que es col·loca la reina que està en la fila i del taulell.
  L'espai de solucions consta de 8^8 8-tuples (**16.777.216** 8-tuples)
  - restriccions explícites:
    $C_i = \{1,2,3,4,5,6,7,8\}\ 1 \leq i \leq 8$
  - restriccions implícites:
    No pot haver-hi dues reines en la mateixa columna o en la mateixa diagonal
- **Formulació 3:** Donat que no pot haver-hi més d'una reina per columna, només fa falta que considerem les 8-tuples $(x_1,\ldots,x_8)$ que siguin permutacions de (1,2,...,8)
  L'espai de solucions consta de 8! 8-tuples (**40.320** 8-tuples)

Representació del problema

Graf de restriccions

$$\begin{array}{cccc} & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$

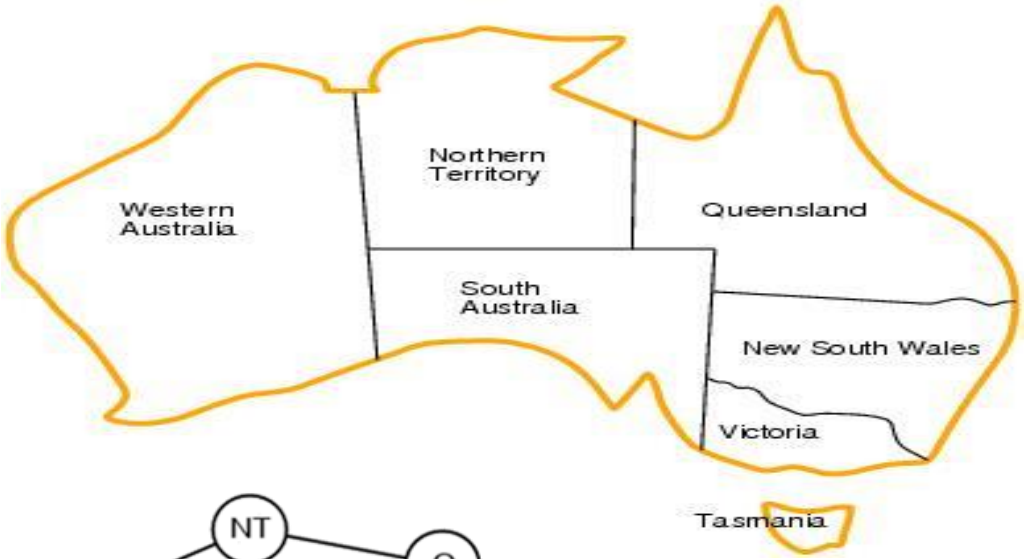


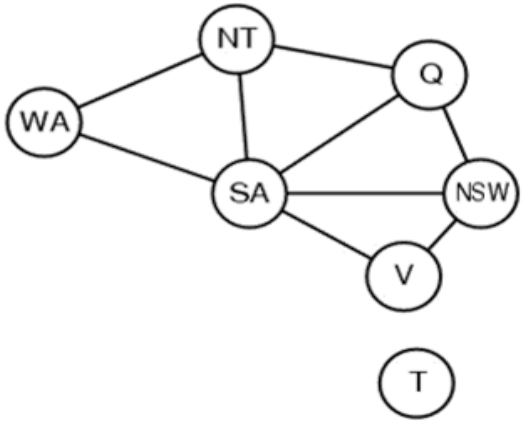
- Variables: $F\ T\ U\ W\ R\ O$ $\quad X_1\ X_2\ X_3$
- Dominis: $\{0,1,2,3,4,5,6,7,8,9\}$ $\quad \{0,1\}$
- Restriccions: *Alldiff (F,T,U,W,R,O)*
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F,\ T \neq 0,\ F \neq 0$

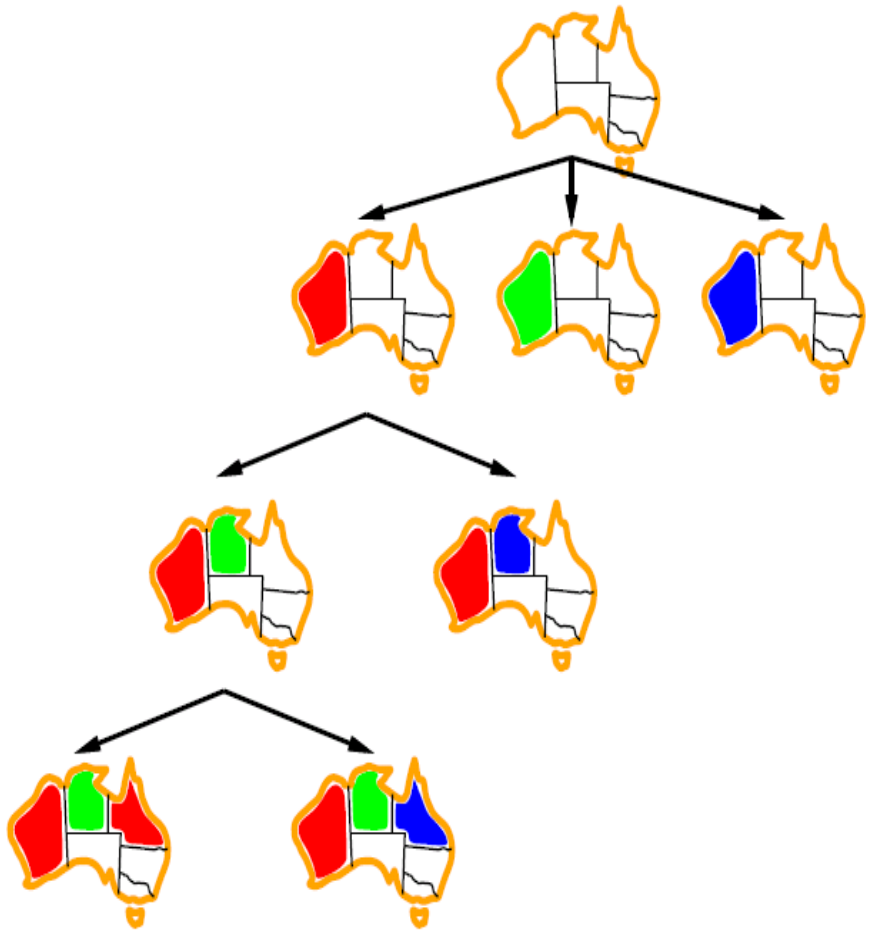
Variables

Restriccions

Anem a pintar







Millora de l'eficiència de CSP (constraint satisfaction problem)

- Introducció d'heurísitques

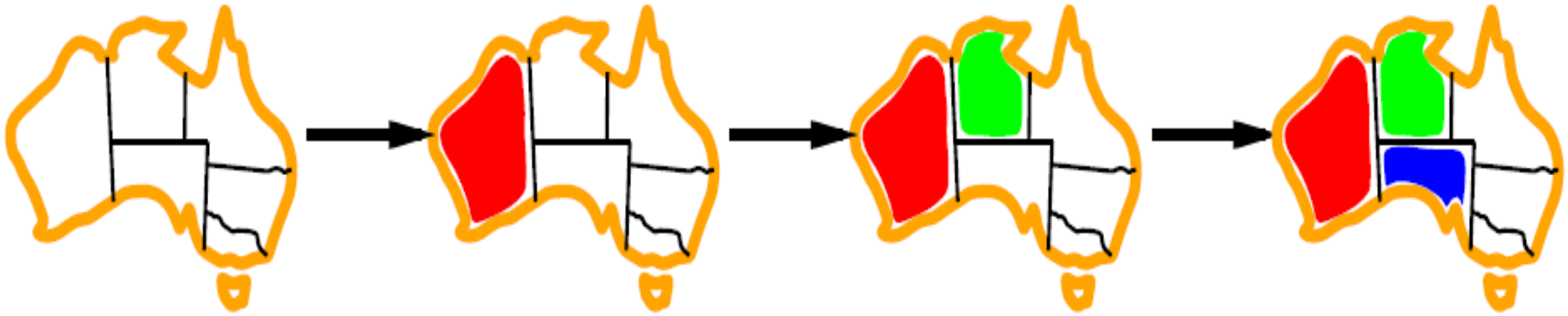
- Per a CSPs alguns metodes generals poden augmentar l'eficiencia en CPU.,
  - Quina variable és la següent a ser assignada?
  - En quin ordre hem de triar els valors que provem?
  - Podem detectar fallades abans?
  - Podem aprofitar l'estructura del problema?

Podem ordenar LVNA de forma diferent?

```
Funcio Backtracking(LVA,LVNA,R,D)
    Si (LVNA és buida) llavors Retornar(LVA) fSi
    Var=Cap(LVNA);
    Per a cada (valor del Domini(Var, D) que podem assignar a Var) fer
        Si (SatisfaRestriccions([Var valor],LVA,R)) llavors
            Res=Backtracking(Insertar([Var, valor],LVA),Cua(LVNA),R,D);
            Si (Res és una solució completa) llavors
                Retornar(Res);
            Fsi
        Fsi
    Fper
    Retornar(Falla)
FFuncio
```

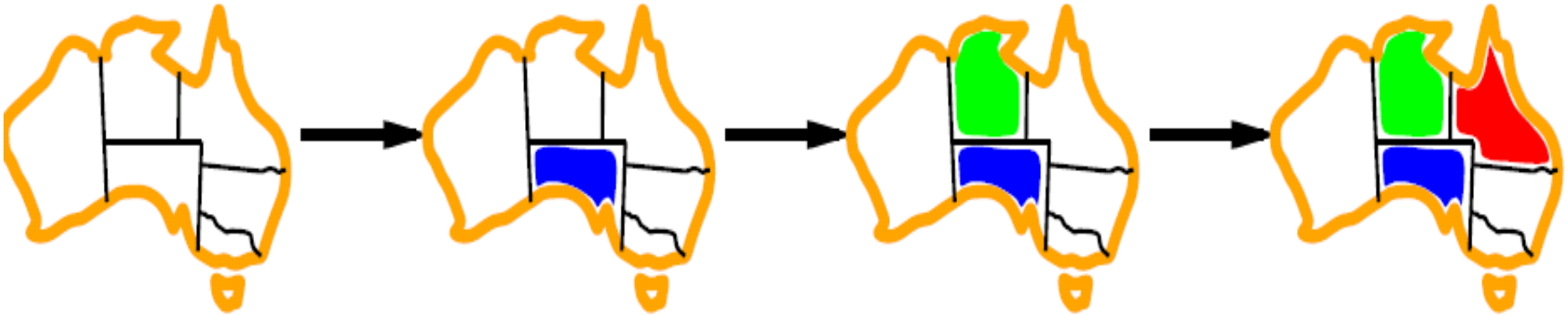
SatisfaRestriccions(A,LVA,R): Retorna cert si l'assignació A afegida la llista d'assignacions LVA satisfan les restriccions R.
Domini(V,D): Retorna un valor del domini D per a la variable V.
Insertar(e,L):Retorna la llista resultant d'afegir e al principi de L.
Cua(L) i Cap(L): Retornen la cua i el cap de L, respectivament.

Minimum remaining values (MRV)



- És a dir: heruística de la variable més 'restringida'
- *Regla Heuristica*: triar la variable amb menys moviments legals
  - Detectarà fallades de forma inmediata si X no té moviments possibles sense explorar primer les altres variables

Heurística de grau de variable



- *Regla Heurística*: seleccionar la variable que està involucrada en el major número de restriccions sobre altres variables no assignades.

Però en quin ordre hem de provar els valors de les variables?

Podem ordenar els valors de forma diferent?

```
Funcio Backtracking(LVA,LVNA,R,D)
    Si (LVNA és buida) llavors Retornar(LVA) fSi
    Var=Cap(LVNA);
    Per a cada (valor del Domini(Var, D) que podem assignar a Var) fer
        Si (SatisfaRestriccions([Var valor],LVA,R)) llavors
            Res=Backtracking(Insertar([Var, valor],LVA),Cua(LVNA),R,D);
            Si (Res és una solució completa) llavors
                Retornar(Res);
            Fsi
        Fsi
    Fper
    Retornar(Falla)
FFuncio
```

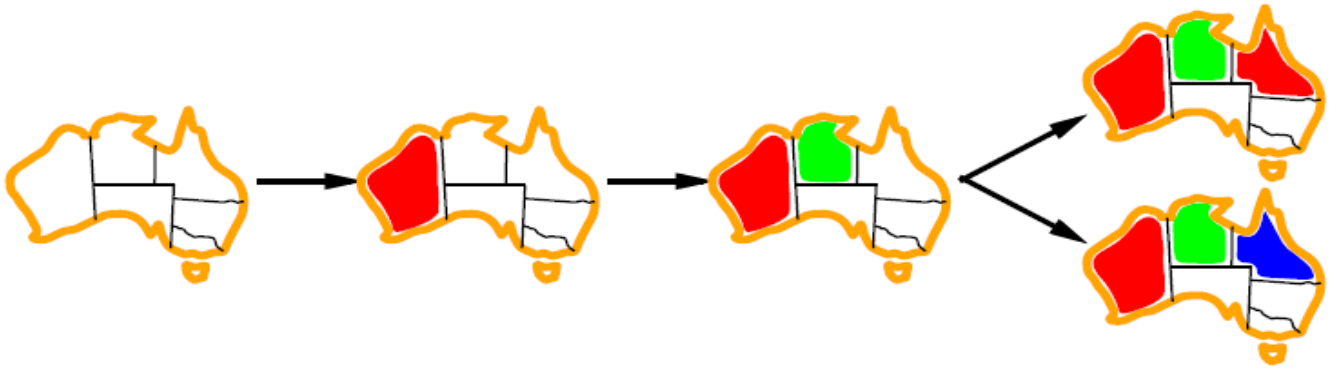
SatisfaRestriccions(A,LVA,R): Retorna cert si l'assignació A afegida la llista d'assignacions LVA satisfan les restriccions R.

Domini(V,D): Retorna un valor del domini D per a la variable V.

Insertar(e,L):Retorna la llista resultant d'afegir e al principi de L.

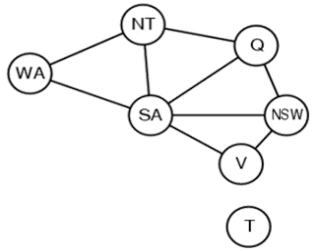
Cua(L) i Cap(L): Retornen la cua i el cap de L, respectivament.

Ordre de valor per menys restrictiu



Permet 1 valor per a SA

Permet 0 valors per a SA



- Heurística de valor menys restrictiu

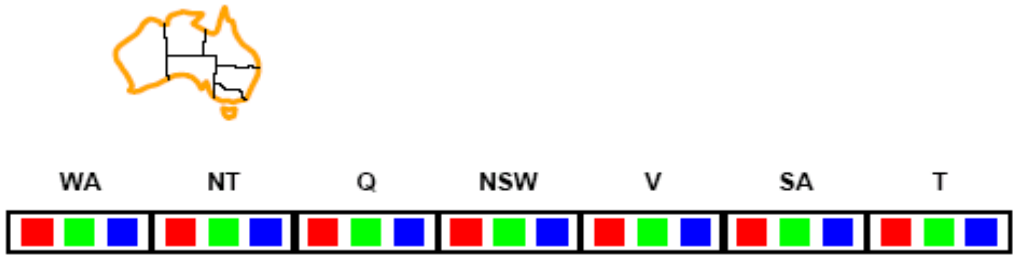
- Regla Heurística: donada una variable triem el valor que menys restingeix les altres variables
  - deixa la màxima flexibilitat per a assignacions següents

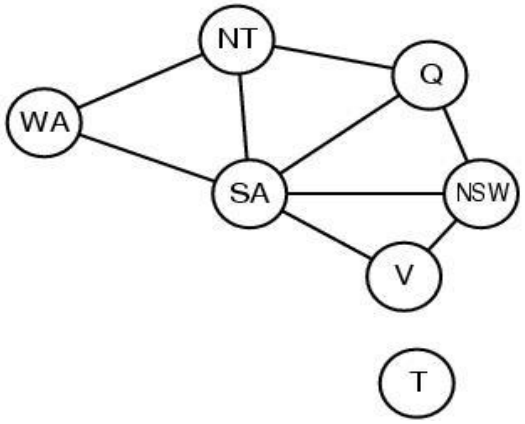
Millora de l'eficiència de CSP (constraint satisfaction problem)

- Introducció d'heurísitques

- Per a CSPs alguns metodes generals poden augmentar l'eficiencia en CPU.,
  - Quina variable és la següent a ser assignada? ✓
  - En quin ordre hem de triar els valors que provem? ✓
  - Podem detectar fallades abans?
  - Podem aprofitar l'estructura del problema?

Forward checking

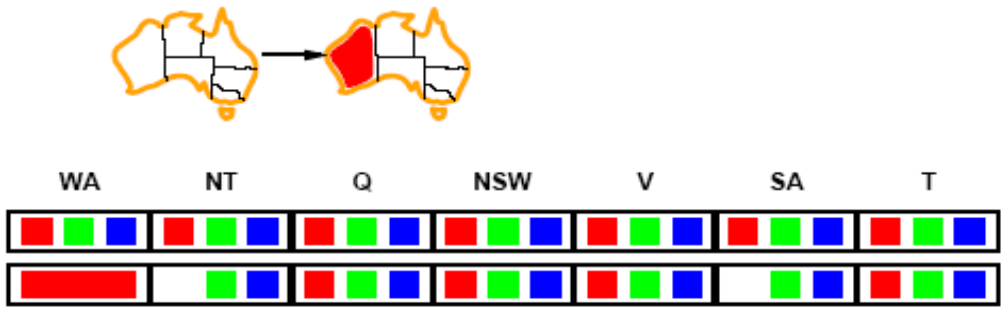


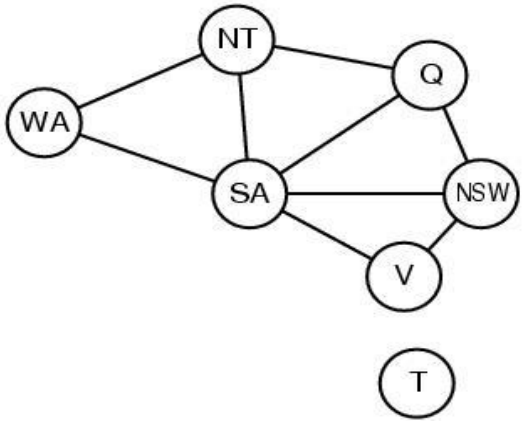


- Podem detectar una fallada inevitable abans?
- Idea del *Forward checking:* mantenim la traça del valors legals que queden per a cada variable no assignada.

  Acabem la cerca (de la branca) quan una variable dins LVNA es queda sense valors.

Forward checking

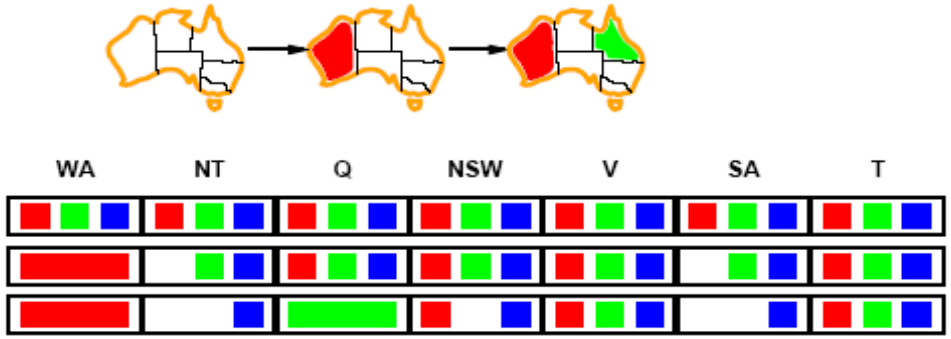


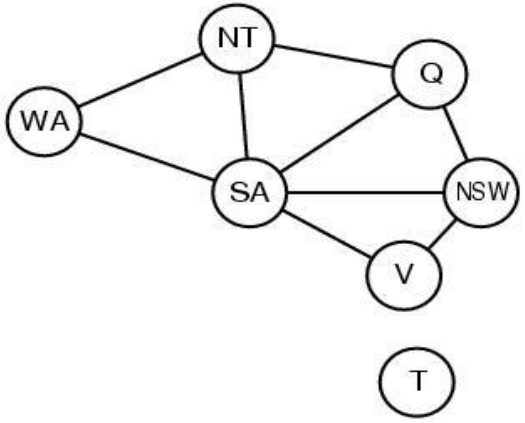


- Assignar *{WA=vermell}*

- Efectes sobre altres variables connectades per restriccions a WA
  - *NT no pot tenir vermell*
  - *SA no pot tenir vermell*

Forward checking

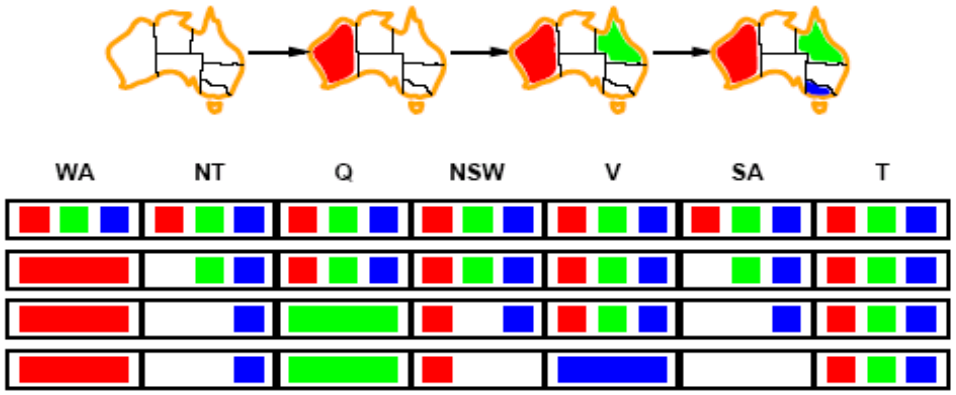


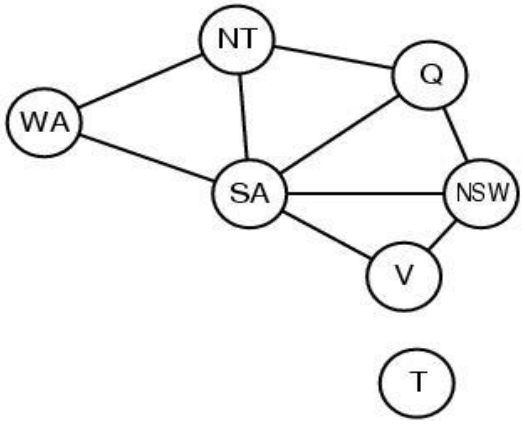


- Assignar *{Q=verd}*

- Efectes sobre altres variables connectades per restriccions a Q
  - *NT no pot ser verd*
  - *NSW no pot ser verd*
  - *SA no pot ser verd*

Forward checking





- Assignar *{V=blau}*

- Efectes sobre altres variables connectades per restriccions a V
  - *NSW no pot ser blau*
  - *SA és buida*
- FC ha detectat que les assignacions parcials són inconsistents amb les restriccions i és produeix bactracking.

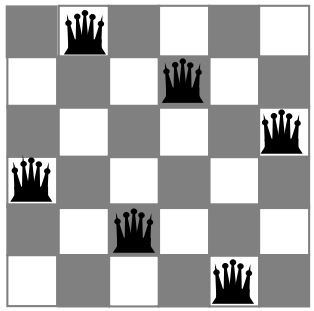
```
Funcio BackForwardChecking(LVA,LVNA,R,DA)
    Si (LVNA és buida) llavors Retornar(LVA) fSi
    Var=Cap(LVNA);
    Per a cada (valor del Domini(Var, DA) que podem assignar a Var) fer
        Si (SatisfaRestriccions(‘(Var valor),LVA,R)) i
          (DA=ActualitzarDominis((’(Var valor), LVNA,R)<>fals)  llavors
            R= BackForwardChecking (Insertar(‘(Var valor),LVA),Cua(LVNA),R,DA);
            Si (R és una solució completa) llavors
                Retornar(Resultat);
            Fsi
        Fsi
    Fper
    Retornar(Falla)
FFuncio
```

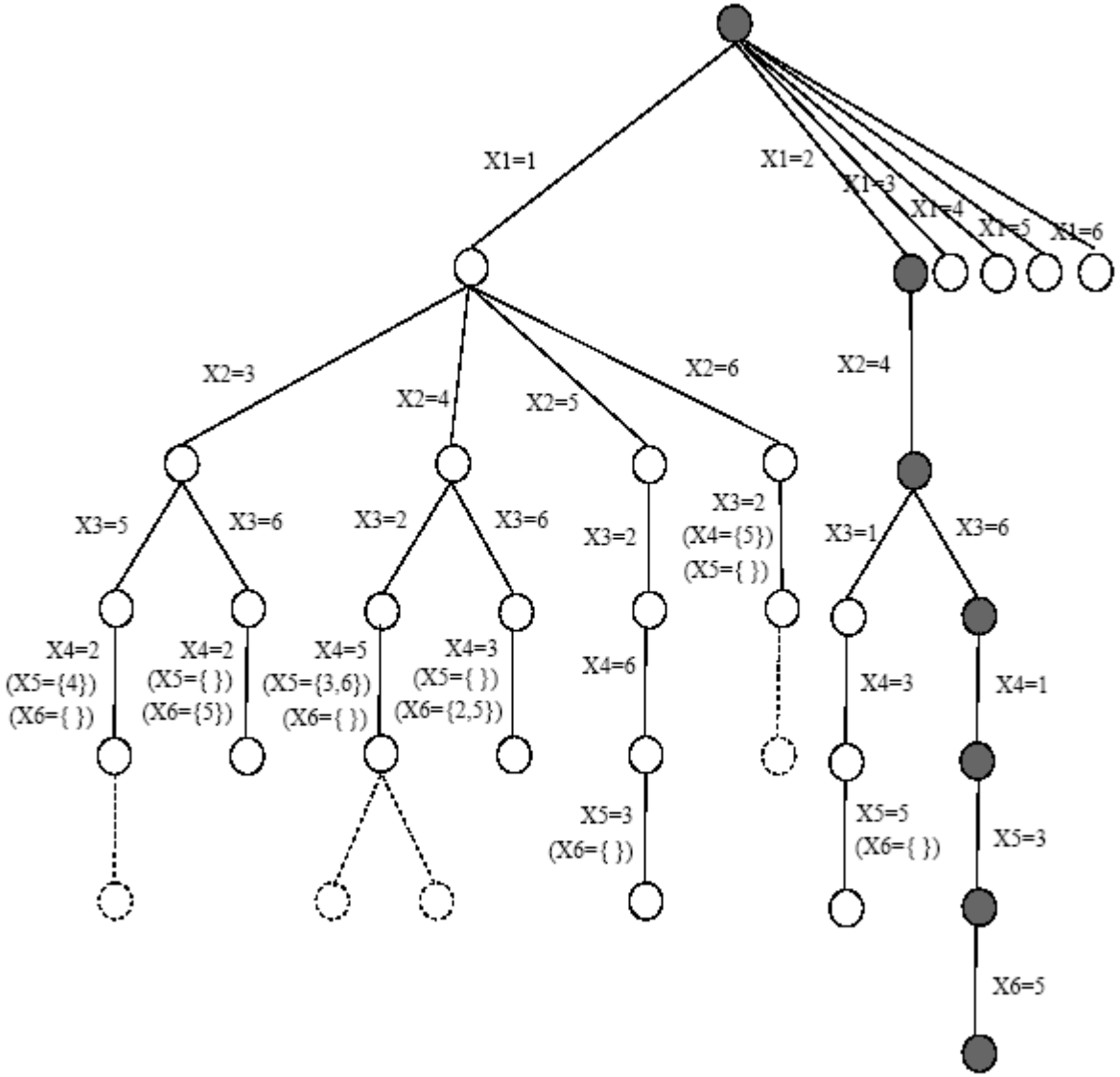
ActualitzarDominis(‘(X v),L,R): Retorna la llista dels dominis per a les variables no assignades de L considerant les restriccions de R despres d’assignar X amb v, retorna fals si algun domini actualitzat és buit.

Exemple: Aplicació del Backtracking amb *forward-checking* al problema de les 6 reines.

Exemple del cas N=6

$X_1 = 2$
$X_2 = 4$
$X_3 = 6$
$X_4 = 1$
$X_5 = 3$
$X_6 = 5$



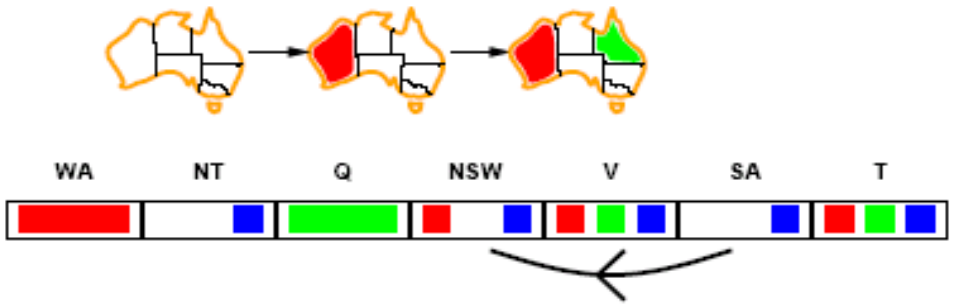


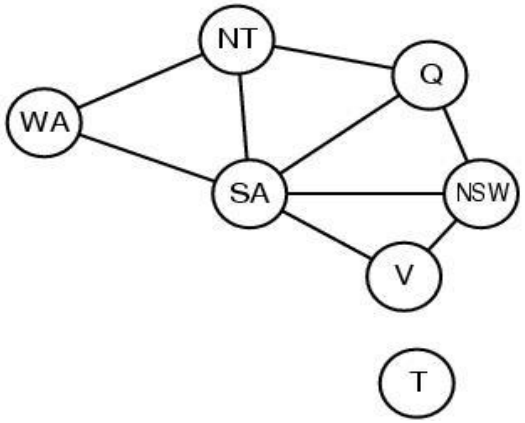
Propagació de Restriccions (CP)

- CP va més enllà del FC forçant repetidament restriccions localment
  - Necessita ser més ràpid que fer la cerca per a ser efectiva

- La consistencia d'Arcs (Arc-consistency: AC) és un procediment sistemàtic per a la propagació de restriccions.

Arc consistency





- *Un Arc* $X \rightarrow Y$ és consistent si

  per a cada valor x de X hi ha algun valor y de Y consistent amb x

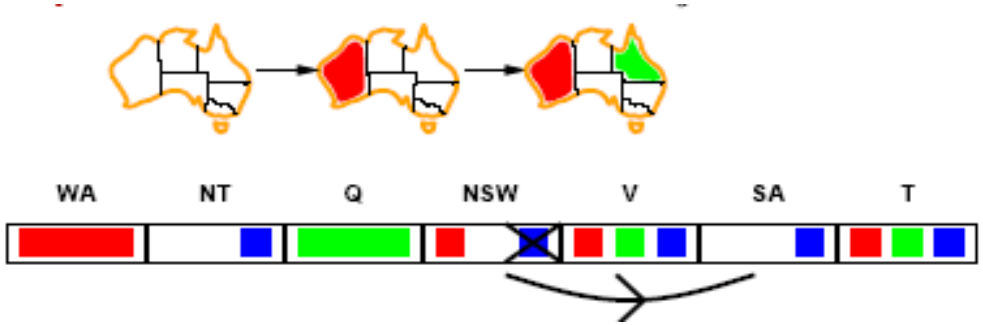
  (notar que és una propietat dirigida)

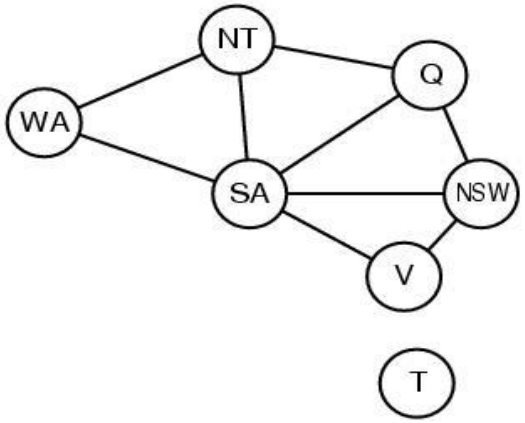
- Considerem l'estat de cerca després d'assignar WA i Q:

  $SA \rightarrow NSW$ és consistent si

  SA=blau i *NSW=vermell*

Arc consistency

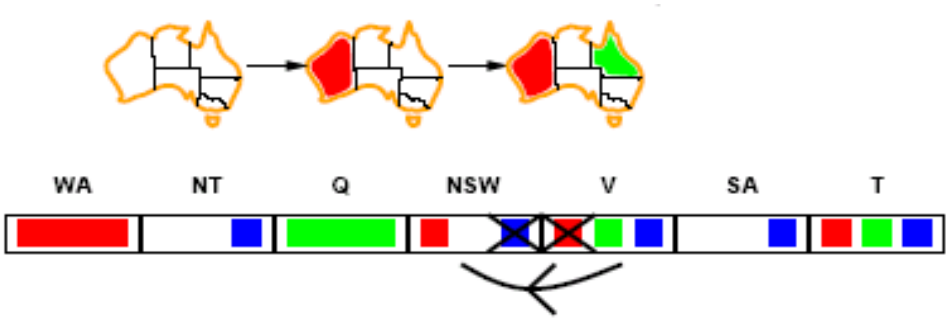


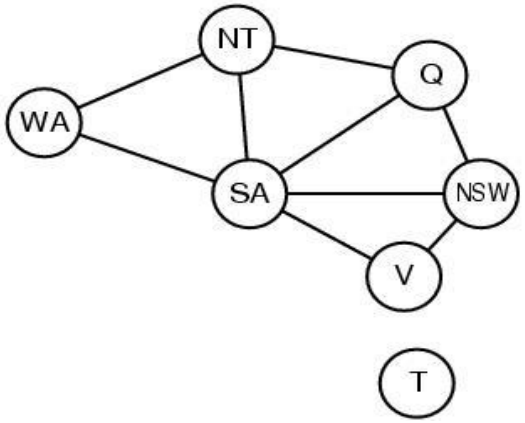


- *Un Arc X → Y* és consistent si
  per a cada valor x de *X* hi ha algun valor y de Y consistent amb x

- *NSW → SA* és consistent si
  NSW=vermell i *SA=blau*
  NSW=blau i SA=???

Arc consistency





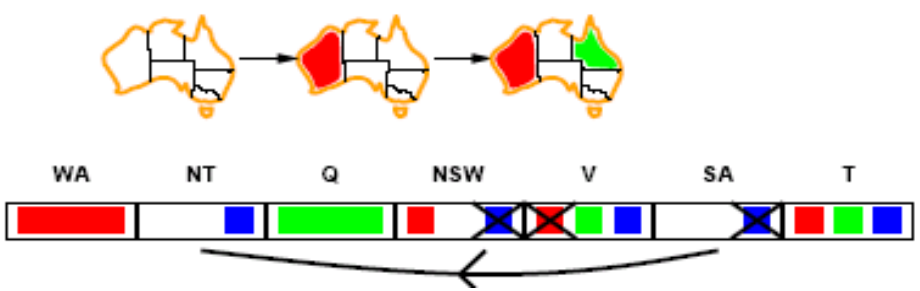
- Podem forçar la consitència:

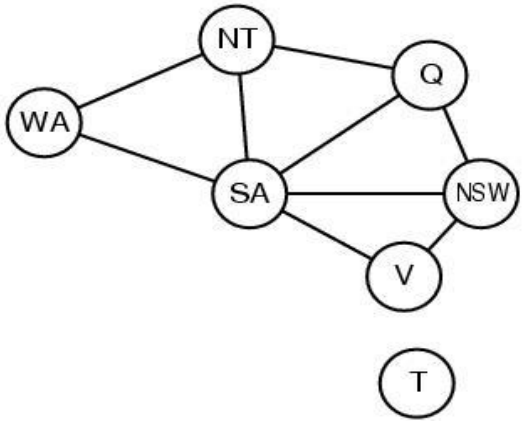
  Un Arc es pot fer consitent eliminant les x que no tenen consitencia amb y

  En el cas: eliminant blau de *NSW*

- Continuem propagant restriccions….
  - Comprovar $V \rightarrow NSW$
  - No consistencia per V = vermell
  - Eliminem vermell de *V*

Arc consistency





Continuem propagant restriccions….

- $SA \rightarrow NT$ no és consistent
  - I no es pot fer consistent sense deixar el domini vuit!!!

- L'Arc consistency detecta fallada abans que FC

Comprovació d'Arc consistency

- S'executa com a preprocess o després de cada assignació
- AC s'ha de correr repetidament fins que no hi ha inconsistencies
- Compromís
  - Requereix sobrecarrega però generalment és més efectiu que la cerca directa
  - Pot eliminar grans parts (inconsistents) de l'espai d'estats més efectivament que ho fa la cerca
- Necessitem un mètode sistemàtic per la comprovació d'arcs
  - Si X perd un valor, els veïns de X han de ser recomprobats:

    i.e. els arcs entrants podern convertirse en inconsistents de nou (els sortints es mantenen consistents).

Algorisme d'Arc consistency (AC-3)

funcio AC-3(*LVNA,R,D*) **retorna** D

variables locals: *A,* inicialment la cua dels arcs del problema

mentre A no es buida **fer**

(X, Y) ← CAP(A)

A = cua(A)

si ELIMINAR-VALORS-INCONSISTENTS (X, Y, *ref D*)

per tot Z **en** VEINS[X]-Y **fer**

A = [A | (Z, X)]

retorna D

funcio ELIMINAR-VALORS-INCONSISTENTS(X, Y, D) **retorna** veritat sii eliminen algun valor

eliminat← *fals*

per tot x **in** D[X] **do**

si cap valor y de D[Y] *permet* (x,y) satisfer la restricció entre X and Y

llavors eliminar x de D[Y]; *eliminat*← *cert*

retorna *eliminat*

(Mackworth, 1977)

Cerca Local per a CSPs

- Usa una representació de "estat complet"
  - Estat inicial = totes les variables tenen assignades un valor
  - Estat successor = canviar 1 (o més) valors

- Per a CSPs
  - Permetre estats amb restricions no completes (al contrari de backtracking)
  - Les operacions (a cada pas) **reassignem** valors de variables
  - Ex: hill-climbing en n-reines

- Selecció de Variable: selecció aletatoria entre les variables conflictives (no compleixen alguna restricció)

- Seleccíó de valor: heuristica *min-conflicts*
  - Seleccionar el nou valor que resulti en un número de conflictes minim amb les alters variables

Cerca local per a CSPs

funcio MIN-CONFLICTS(*csp, max_steps*) **return** solucio o fallada

inputs: *csp*, un problema de satisfacció de restriccions [VARIABLES, DOMINI, RESTRICCIONS]

max_steps, numero maxim de passos abans d'abandonar

actual ← una inicialització completa per al csp incorrecta

for *i* = 1 to *max_steps* **do**

if *actual* es solució per a *csp* **then** return *actual* *#compleix totes les RESTRICCIONS*

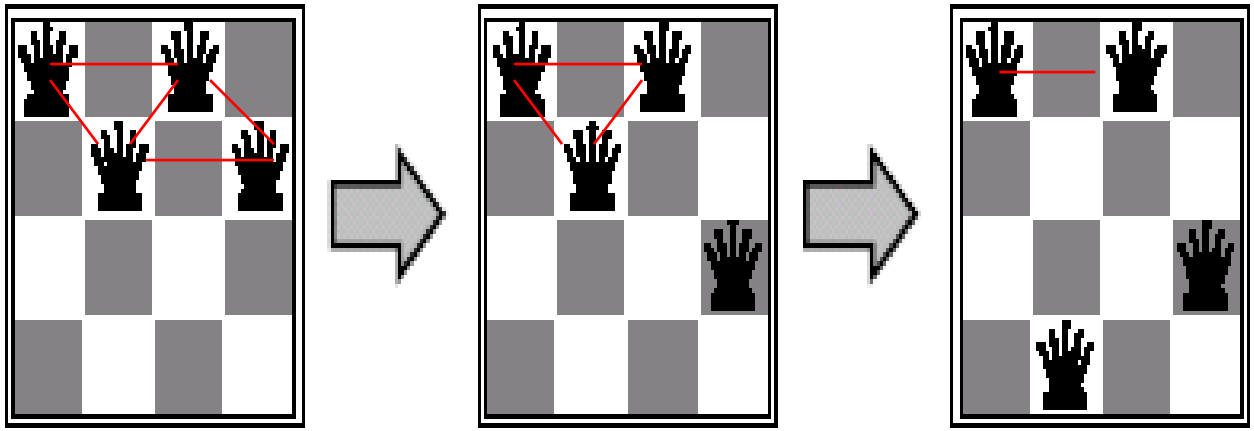
var ← una variable escollida aleatoriament amb conflicte

valor ← el valor v per a var que minimitza CONFLICTS(*var,v,actual,csp*)

set *var = valor* en *actual*

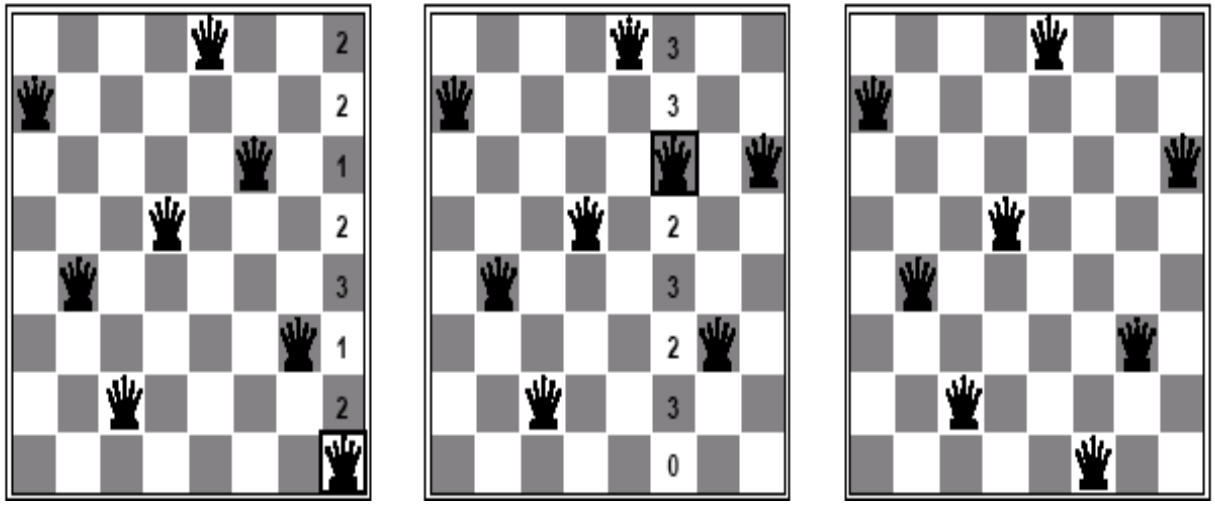
return *fallida*

exemple Min-conflicts



Usem la heurística min-conflicts amb hill-climbing.

exemple 2 Min-conflicts



- Una solució de "dos passos" per al problema de les 8-reines amb heurística min-conflicts
- A cada pas es tria una reina per a la reassignació en la seva columna
- L'algorisme mou la reina en la casella de min-conflict. Els empats es resolen aleatoriament.

Comparativa d'algorismes CSP en diferents problemes

Problem	Backtracking	BT+MRV	Forward Checking	FC+MRV	Min-Conflicts
USA	(> 1,000K)	(> 1,000K)	2K	60	64
n-Queens	(> 40,000K)	13,500K	(> 40,000K)	817K	4K
Zebra	3,859K	1K	35K	0.5K	2K

USA: 4 colorejat
n-queens: n = 2 to 50
Zebra: https://en.wikipedia.org/wiki/Zebra_Puzzle

Exemples de Problemes de satisfacció de restriccions:

1. Problema de la criptoaritmètica.
2. Problema de les N reines.
3. Problema de la construcció d'un quadrat màgic.
4. Problema de la resolució d'un puzzle.
5. Problema de la correspondència de grafs.
6. Problema de la interpretació 3D d'imatges 2D.
7. Problemes d'assignació: p.ex. Qui explica que
8. Problemes d'horaris:p .ex: quan s'ofereix una assignatura i on
9. Gestió de transport
10. Gestió de producció

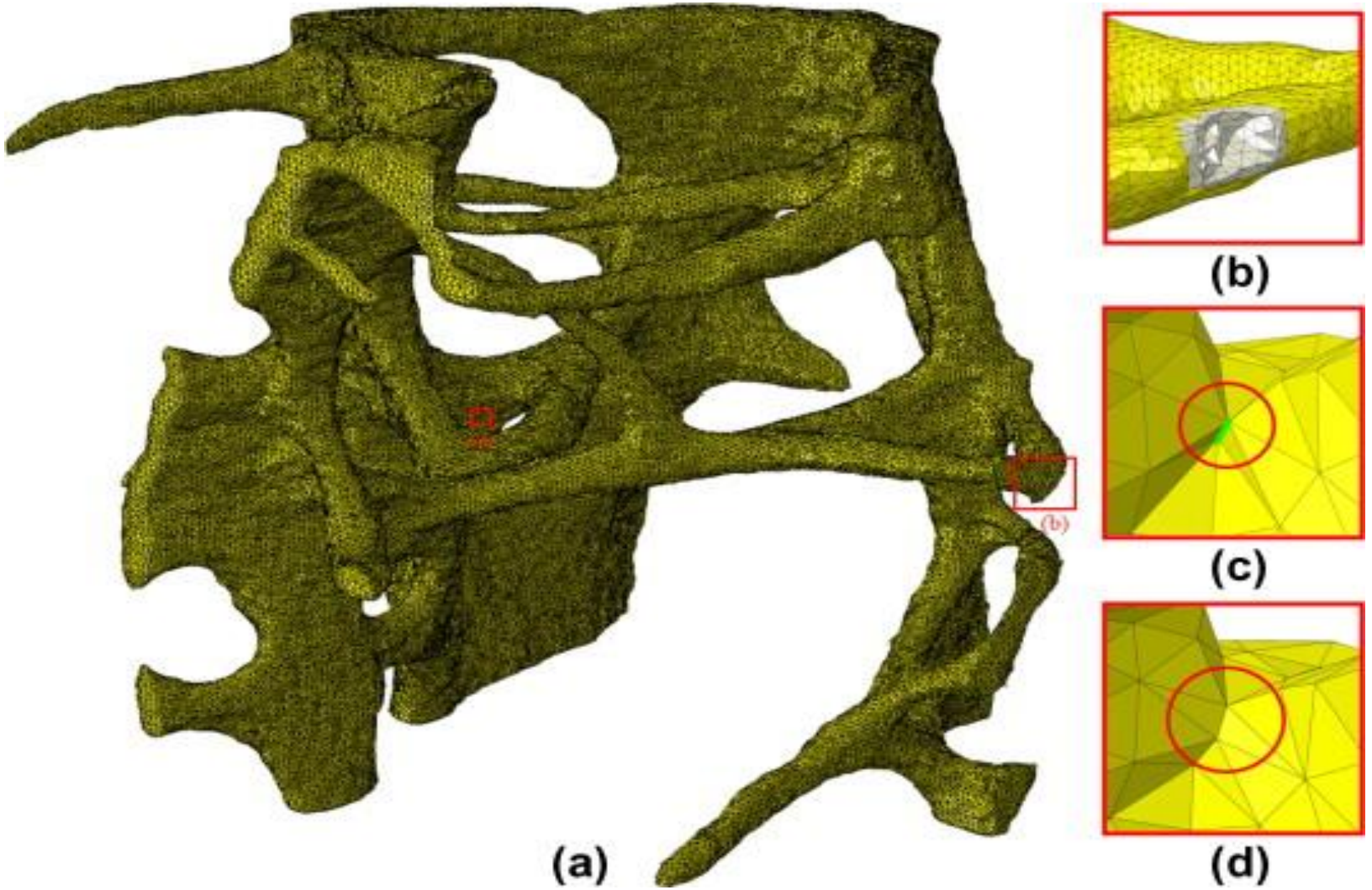


Fig. 8 (a) Triangular and tetrahedral meshes of another trabecular bone structure. (b) One local region with some tetrahedral elements removed. (c) and (d) show zoom-in details of a local region with topology ambiguity (the red circle). The green...

Yongjie Zhang , Jin Qian
Dual Contouring for domains with topology ambiguity
Computer Methods in Applied Mechanics and Engineering Volumes 217?220 2012 34 - 45
http://dx.doi.org/10.1016/j.cma.2012.01.004

Constraint Programming

“Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.”

Eugene C. Freuder, Constraints, April 1997