CSP come ricerca nello spazio degli stati

- CSP:
  - state e' definito da variabili X_i con valori presi dai domini D_i
  - goal test e' un insieme di vincoli che specificano le combinazioni di valori permesse (in modo intensionale).
  - Operatori sono assegnammenti di valori a variabili
- Linguaggio di formalizzazione generale
- Algoritmi general-purpose per la soluzione.

CSP come ricerca

- Stato iniziale: assegnamento vuoto { }
- Funzione Successore: assegna un valore ad una variabile non ancora legata in modo che sia legale con gli assegnamenti gia' fatti.
  → fallisci se non esiste
- Goal test: l'assegnamento e' completo (tutte le variabili sono legate).

1. Schema identico per tutti i CSPs
2. Profondita limitata n se n sono le variabili.
   → usa depth-first search
3. La strada e' irrilevante.
4. Problema commutativo con $\cdot d^n$ foglie.

Standard Backtracking

- Depth-first search per CSPs con singolo assegnamento di variabili e chiamata standard backtracking
- Standard Backtracking search e' l'algoritmo di ricerca non-informata basilare per CSP.

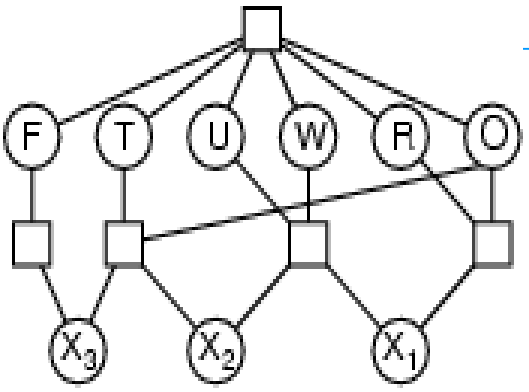
```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return RECURSIVE-BACKTRACKING({}, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns a solution, or
failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(Variables[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment according to Constraints[csp] then
      add { var = value } to assignment
      result ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failue then return result
      remove { var = value } from assignment
  return failure
```


Differenti CSP

- Variabili Discrete
  - Domini Finiti:
    - *n* variables, con dimensione d (quelli che vedremo)
  - Domini Infiniti:
    - Interi, stringhe ecc,
    - e.g., job- scheduling, variabili rappresentano giorni di inizio-fine per ogni lavoro
    - *vincoli di durata* $StartJob_1 + 5 \leq StartJob_3$
- Variabili Continue
  - Programmazione lineare (Ricerca Operativa)

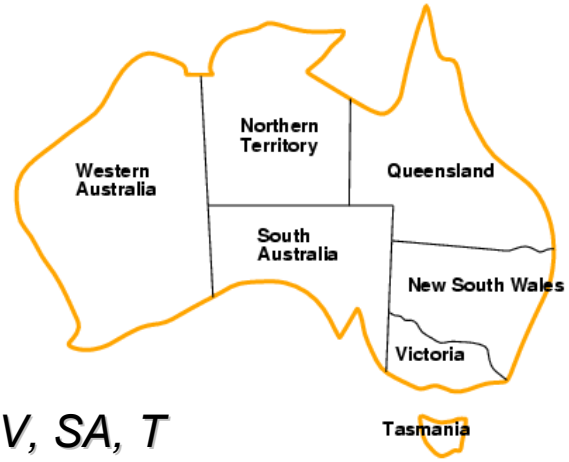
Esempio: Criptoaritmetica

$$\begin{array}{r} \text{T W O} \\ +\ \text{T W O} \\ \hline \text{F O U R} \end{array}$$



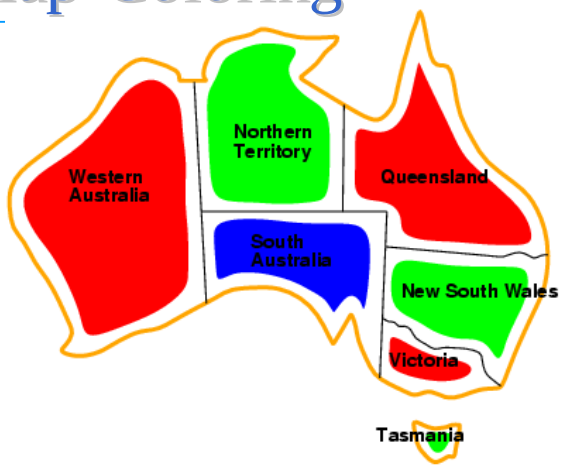
- Variables: *F T U W R O* X_1 X_2 X_3
- Domains: {*0,1,2,3,4,5,6,7,8,9*}
- Constraints: *Alldiff (F,T,U,W,R,O)*
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F$, $T \neq 0$, $F \neq 0$

Esempio: Map-Coloring



- variabili *WA, NT, Q, NSW, V, SA, T*
- Domini D_i = {red,green,blue}
- vincoli: regioni adiacenti devono avere colori diversi
- e.g., WA ≠ NT, or (WA,NT) in {(red,green),(red,blue),(green,red), (green,blue),(blue,red),(blue,green)}

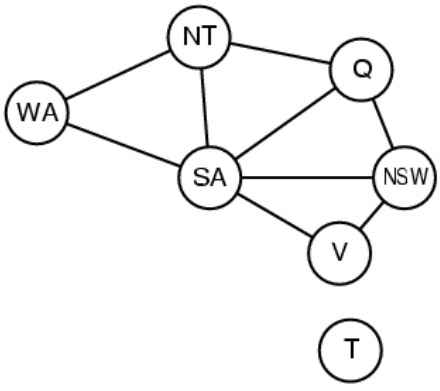
Esempio: Map-Coloring



- Le soluzioni sono assegnamenti completi e consistenti, e.g., WA = red, NT = green,Q = red,NSW = green,V = red,SA = blue,T = green

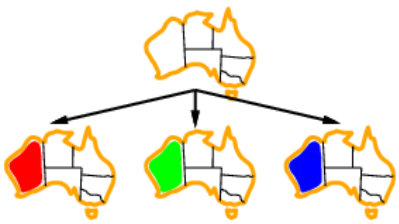
Constraint graph

- CSP binario: ogni vincolo si correla a due variabili
- Constraint graph: nodi sono variabili, archi sono vincoli

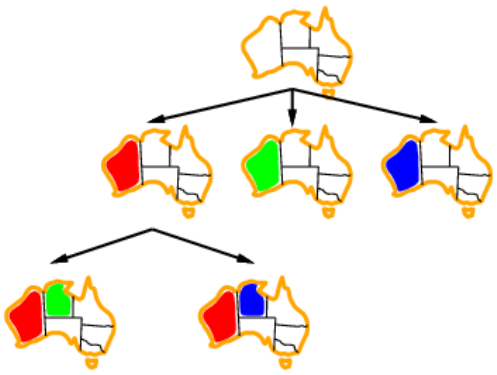


Backtracking : esempio

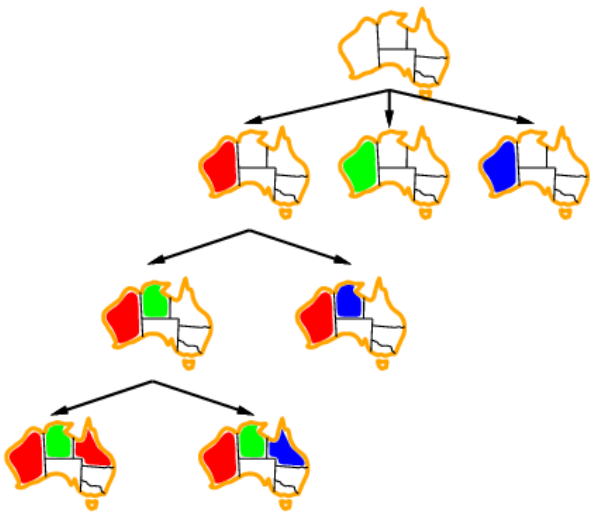
Backtracking : esempio



Backtracking : esempio



Backtracking : esempio



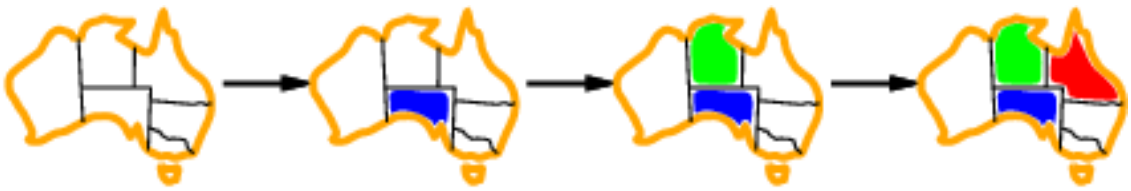
Most constrained variable

- Most constrained variable:
  Scegli la variabile con meno valori ammessi

- minimum remaining values (MRV)

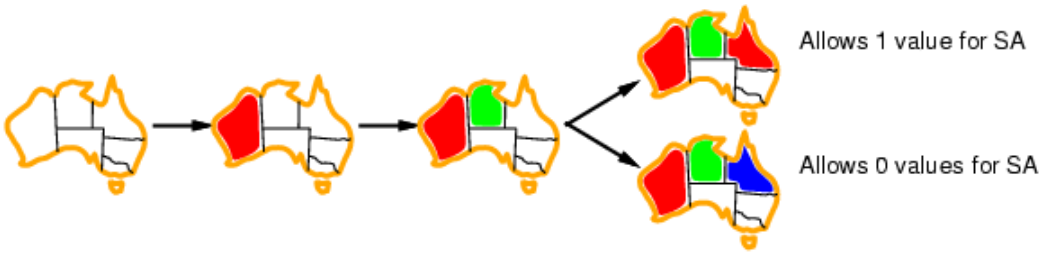
Most constraining variable

- Most constraining variable:
  - Scegli la variabile con il maggior numero di vincoli sulle altre variabili rimaste (si applica quando ci sono piu' variabili con uguale numero di valori nel dominio.



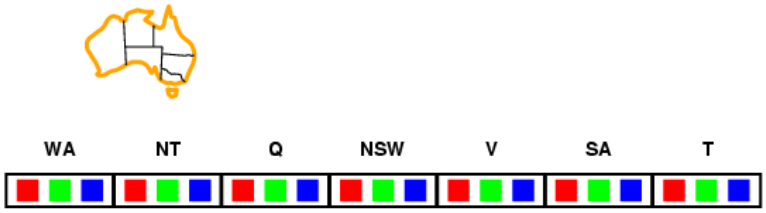
Least constraining value

- Data una variabile, scegli il valore meno vincolante, cioe' che rende impossibili o inconsistenti meno assegnamenti dellle variabili rimaste.

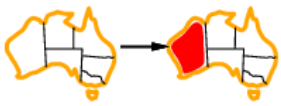


Forward checking

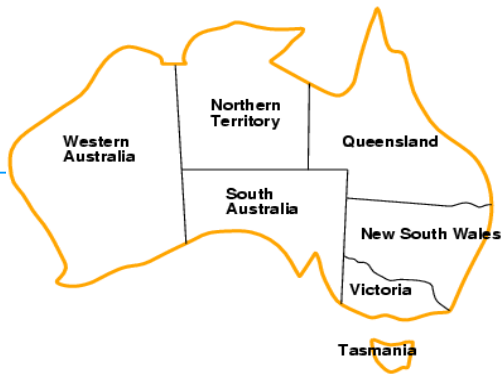
- Idea:
  - Tenere traccia dei valori legali per le variabili rimanenti
  - Fallire quando non ci sono piu' valori legali.



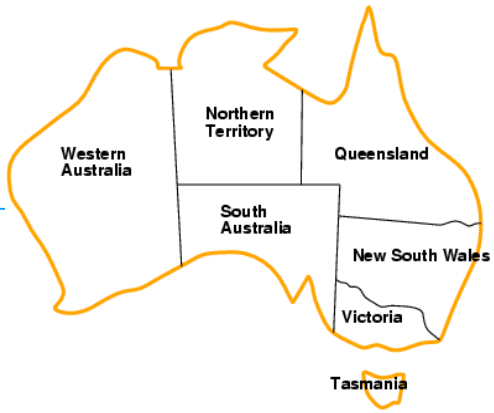
Forward checking

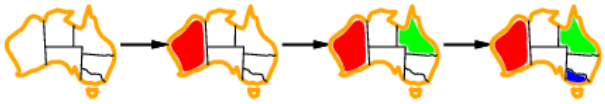


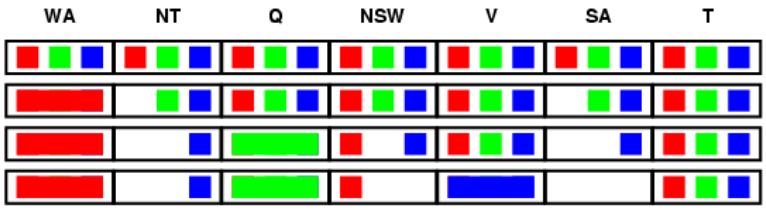
Forward checking



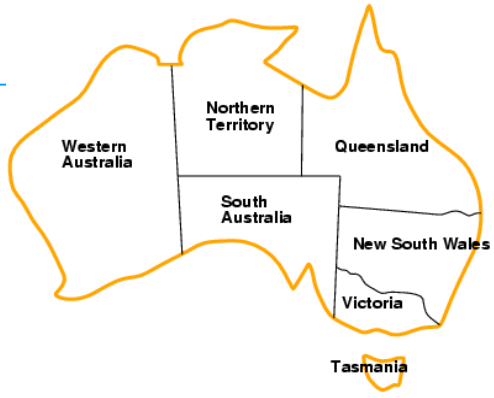
Forward checking







Constraint Propagazione di vincoli e forward checking

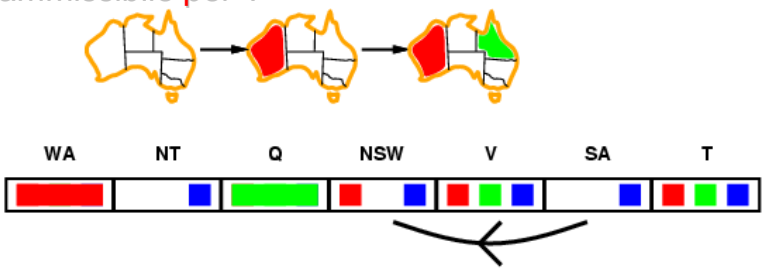


- Forward checking propaga l'informazione da variabili assegnate a variabili non assegnate, ma non consente di individuare subito situazioni inconsistenti.
- NT e SA non possono essere entrambe blue
- Constraint propagation fra variabili non assegnate!

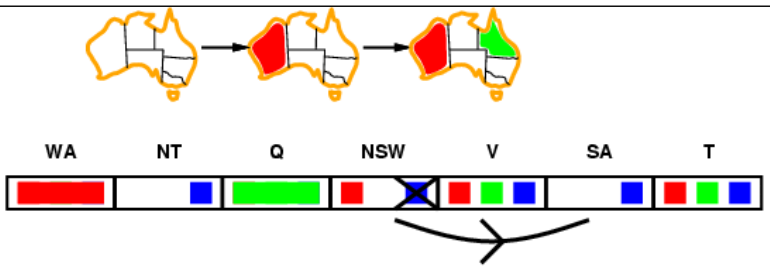
Arc consistency

- La piu' semplice forma di propagazione, rende ogni arco consistente.
- *X* → *Y* e' consistente se e solo se

  Per ogni valore di *X* *c'e' almeno un* valore ammissibile per Y

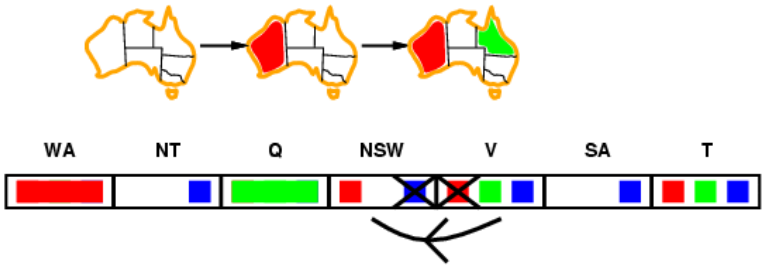


Arc consistency



Arc consistency

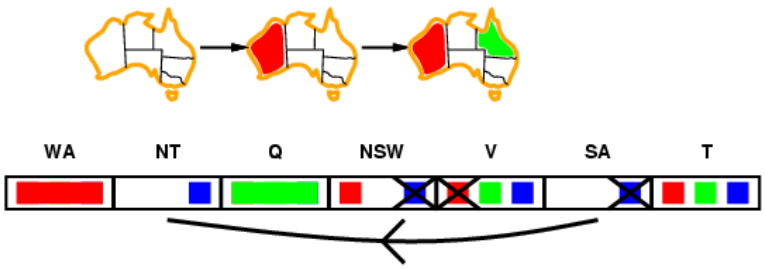
- SE *X* perde un valore, devo ricontrollare I "vicini" di X.



Arc consistency

I fallimenti sono trovati con l'Arc consistency prima che con il forward checking

Puo' girare come pre-processore, oppure dopo ogni assegnamento.



Arc-Consistency: Algoritmo

```
function AC-3( csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (Xi, Xj) ← REMOVE-FIRST(queue)
    if RM-INCONSISTENT-VALUES(Xi, Xj) then
      for each Xk in NEIGHBORS[Xi] do
        add (Xk, Xi) to queue

---

function RM-INCONSISTENT-VALUES( Xi, Xj) returns true iff remove a value
  removed ← false
  for each x in DOMAIN[Xi] do
    if no value y in DOMAIN[Xj] allows (x,y) to satisfy constraint(Xi, Xj)
      then delete x from DOMAIN[Xi]; removed ← true
  return removed
```