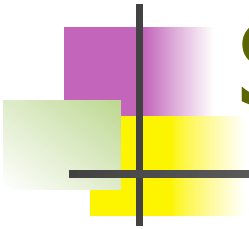
Constraint Satisfaction Problems



Sommario

- **C**onstraint **S**atisfaction **P**roblems (CSP)

  Constraint Programming = rappresentazione e ricerca soluzione di CSP

- Backtracking search for CSPs

- Local search for CSPs

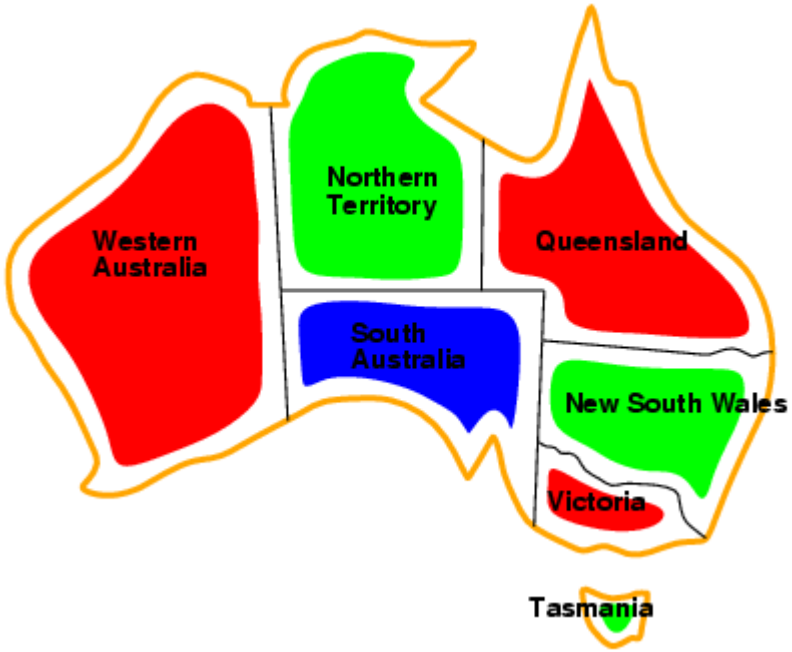
Constraint satisfaction problems (CSPs)

- Standard search problem:
  - Stato: "scatola nera" – any data structure that supports successor function, heuristic function, and goal test
- CSP:
  - Stato definito da variablili X_i con valori in un dominio (cont o disc) D_i
  - goal test insieme di constraints che specifica combinazioni possibili di valori per sottoinsimi di variabili (constraints *espliciti* e *impliciti*)
  - Esempio 8-regine
- Semplice esempio di linguaggio formale di rappresentazione
- Permette lo sviluppo di general-purpose algorithms con più potere degli algoritmi standard di ricerca

Example: Map-Coloring

- Variables *WA, NT, Q, NSW, V, SA, T*
- Domains D_i = {red,green,blue}
- Constraints: regioni adiacienti devono avere colori
- Ad es., WA ≠ NT (implicito) o, esplicitamente, (WA,NT) in {(red,green), (red,blue),(green,red), (green,blue),(blue,red),(blue,green)}

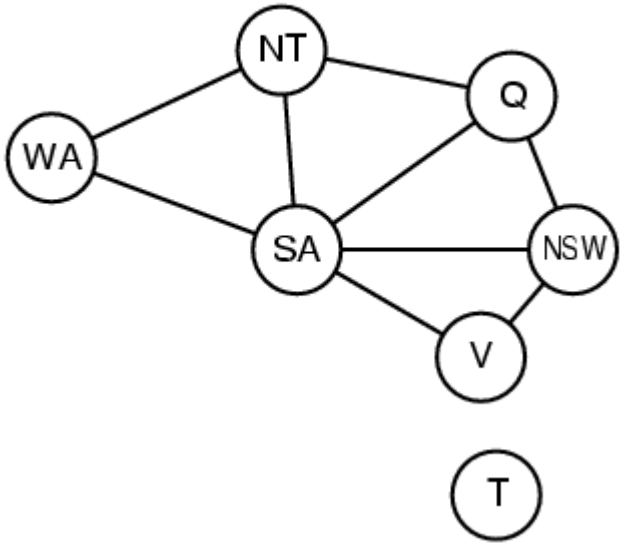
Example: Map-Coloring



- Solutions are complete and consistent assignments, e.g., WA = red, NT = green,Q = red,NSW = green,V = red,SA = blue,T = green

Constraint graph

- Binary CSP: each constraint relates two variables
- Constraint graph: nodes are variables, arcs are constraints



Varieties of CSPs

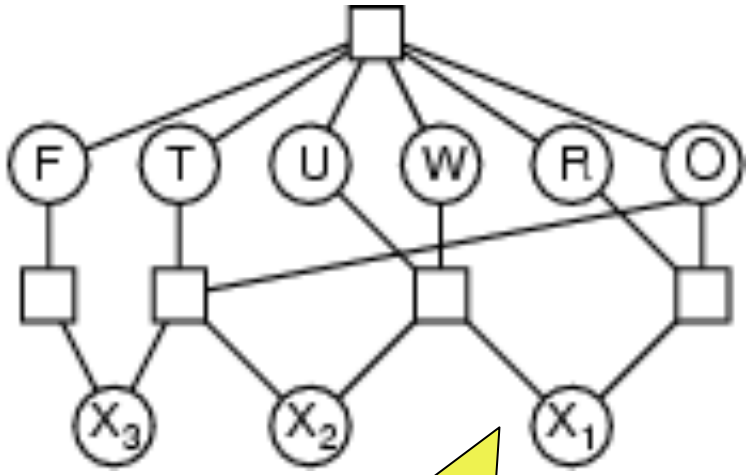
- **Discrete variables**
  - *finite domains*:
    - *n* variables, domain size *d* → $O(d^n)$ complete assignments
    - e.g., Boolean CSPs, incl. Boolean satisfiability (NP-complete)
  - *infinite domains*:
    - integers, strings, etc.
    - e.g., job scheduling, variables are *start/end days for each job*
    - need a ***constraint language***, e.g., $StartJob_1 + 5 \leq StartJob_3$
- **Continuous variables**
  - e.g., start/end times for Hubble Space Telescope observations
  - **linear constraints** solvable in polynomial time by *linear programming*

Varieties of constraints

- Unary constraints involve a single variable,
  - e.g., SA ≠ green
- Binary constraints involve pairs of variables,
  - e.g., SA ≠ WA
- Higher-order constraints involve 3 or more variables,
  - e.g., cryptarithmetic column constraints

Example: Cryptarithmetic

```
    T W O
+   T W O
-------
F O U R
```



hyper constraint graph

- 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)* *variabili tutte con valori diversi!*
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F,\ T \neq 0,\ F \neq 0$

Many Real-world CSPs

- *Assignment problems*
  - e.g., who teaches what class
- *Timetabling problems*
  - e.g., which class is offered *when* and *where*?
- *Transportation scheduling*
- *Factory scheduling*
- *Boolean satisfiability*
- *….*
- Notice that many real-world problems involve *real-valued* variables

Standard search formulation (incremental)

Let's start with the straightforward approach, then fix it
States are defined by the values assigned so far

- Initial state: the empty assignment { }
- Successor function: assign a value to an unassigned variable **that *does not conflict* with current assignment**
  → *fail if no legal assignments*
- Goal test: the current assignment is complete

1. This is the same for all CSPs!
2. Every solution appears at depth *n* with *n* variables
   → use **depth-first** search
3. b = (n - l)d at depth l, hence n! · d^n leaves !!

Backtracking search

- Variable assignments are commutative:

[prima WA = red poi NT = green] stessa cosa di:
[prima NT = green poi WA = red]

- Only need to consider assignments *to a **single** variable* at each node → b = d and there are d^n leaves

- Depth-first search for CSPs with single-variable assignments is called **backtracking** search

- Backtracking search is the basic **un**informed algorithm for CSPs

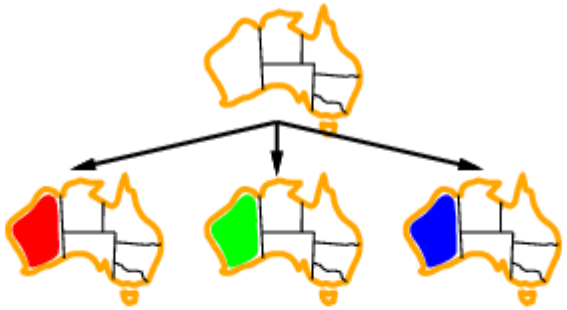
- Can solve *n*-queens for $n \approx 25$

Backtracking search

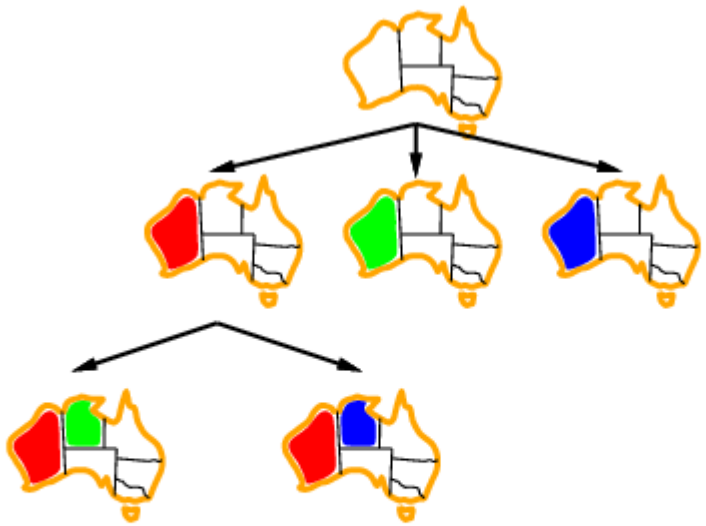
```
function BACKTRACKING-SEARCH( csp) returns a solution, or failure
    return RECURSIVE-BACKTRACKING({}, csp)

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
```

Backtracking example

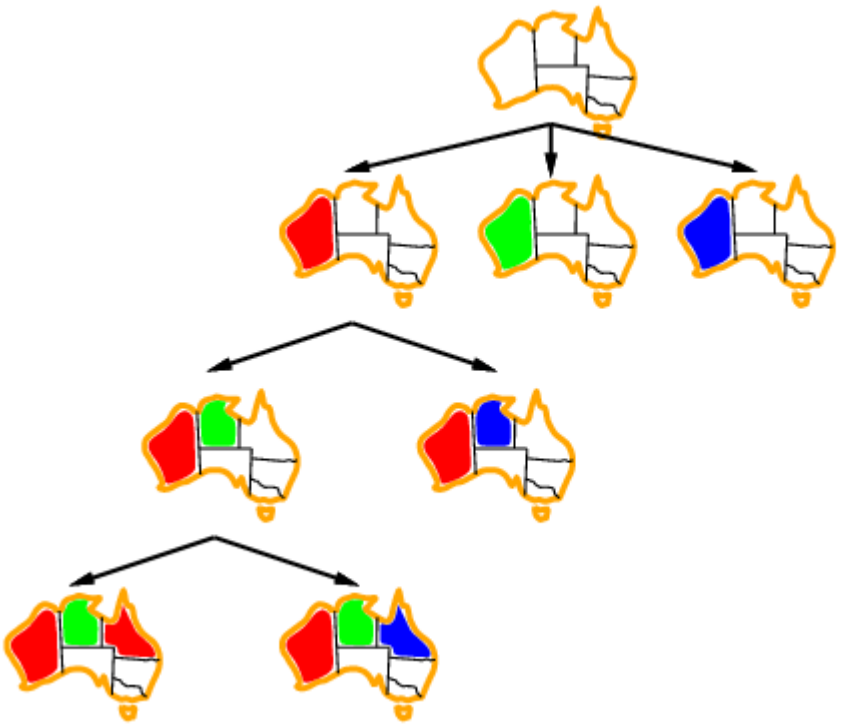
Backtracking example

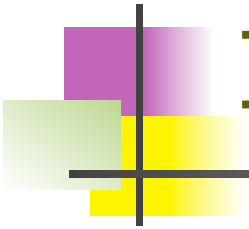


Backtracking example



Backtracking example





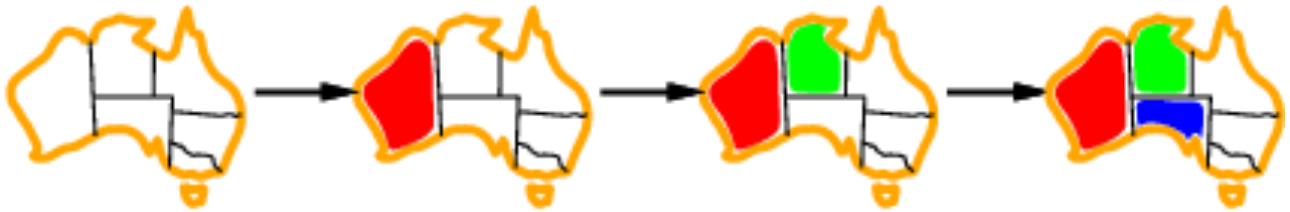
Improving backtracking efficiency

- General-purpose methods can give huge gains in speed:
  - *Which variable should be assigned next?*
  - *In what order should its values be tried?*
  - *Can we detect inevitable failure early?*

Most constrained variable

- Most constrained variable:

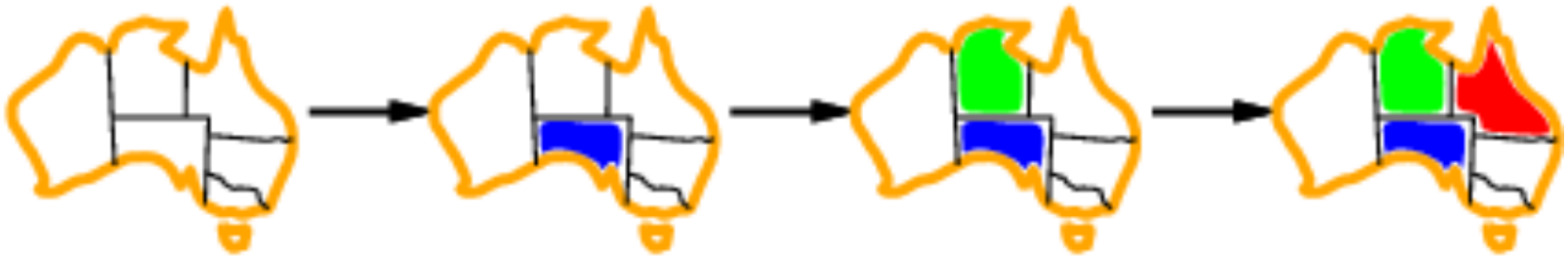
  Scegli la var con meno valori accettabili



- minimum remaining values (MRV) heuristic

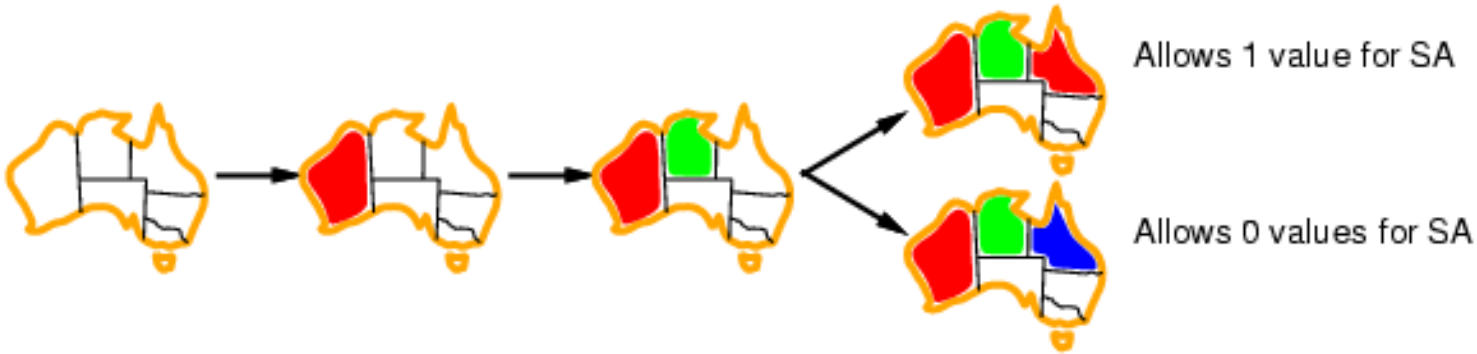
Most constrain*ing* variable

- Criterio aggiuntivo di preferenza tra most constrained variables
- Most constraining variable (degree heuristic):
  - Scegli la variabile con il maggior numero di vincoli con le variabili rimanenti



Least constraining *value*

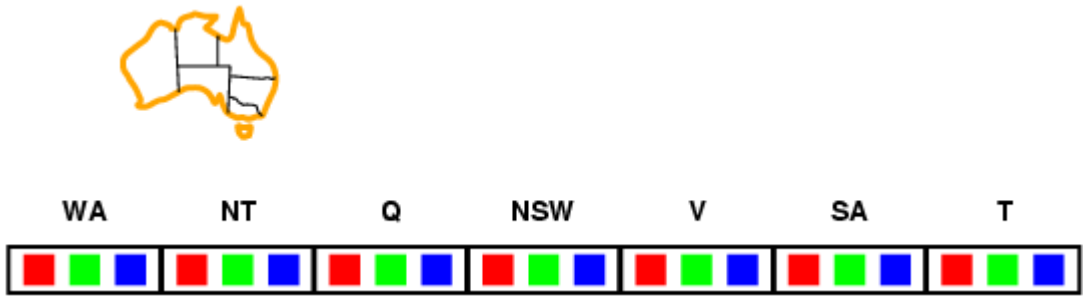
- Data una variabile, scegli il valore *meno vincolante:*
  - the one that rules out the fewest values in the remaining variables



- Combining these heuristics makes *1000 queens feasible!*

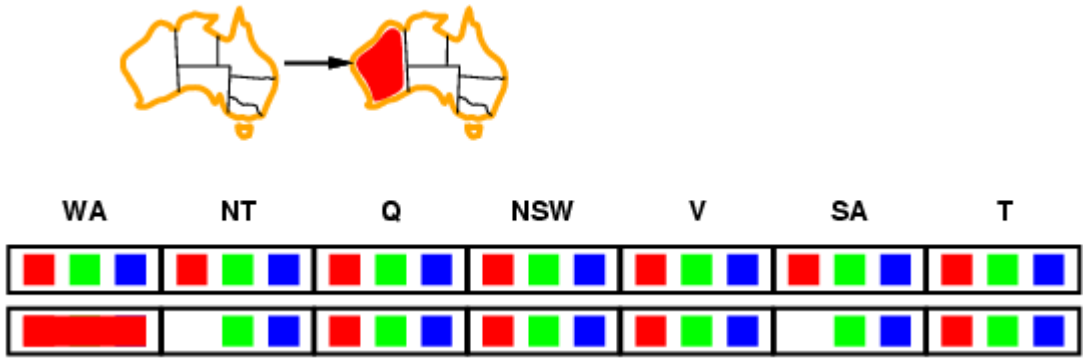
Forward checking

- Idea:
  - Keep track of *remaining* legal values for *unassigned* variables
  - Terminate search when any variable has no legal values



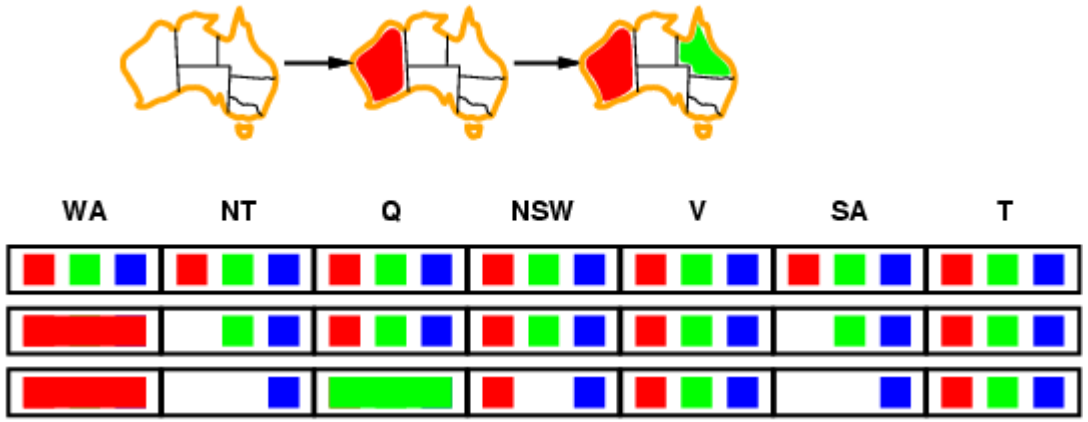
Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



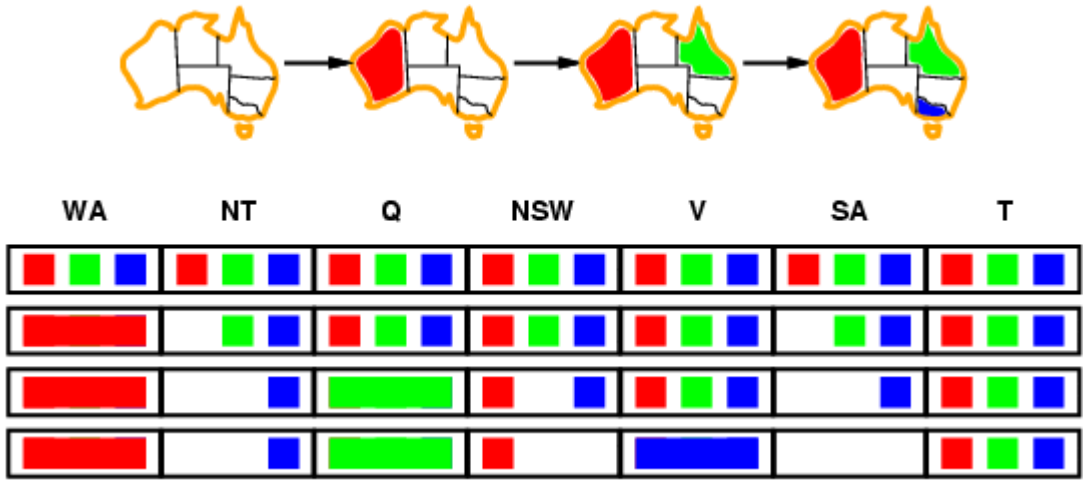
Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



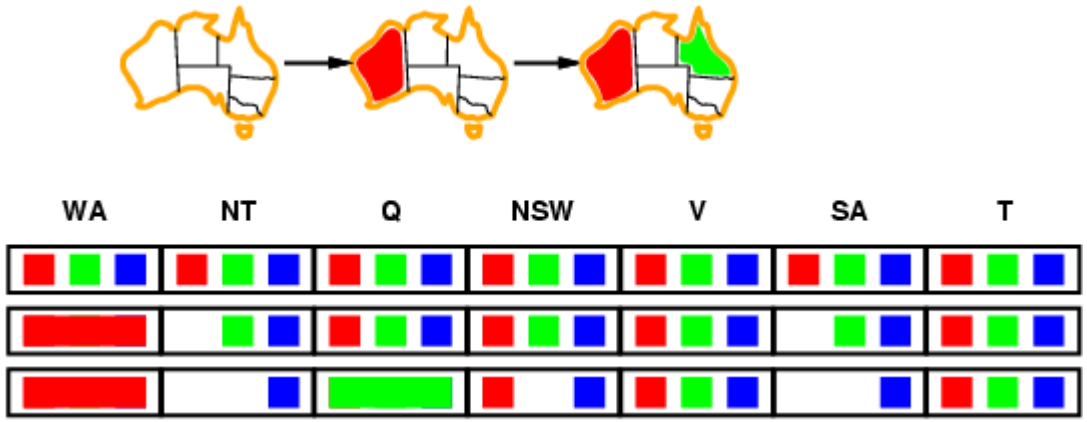
Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



Constraint propagation (Inference)

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

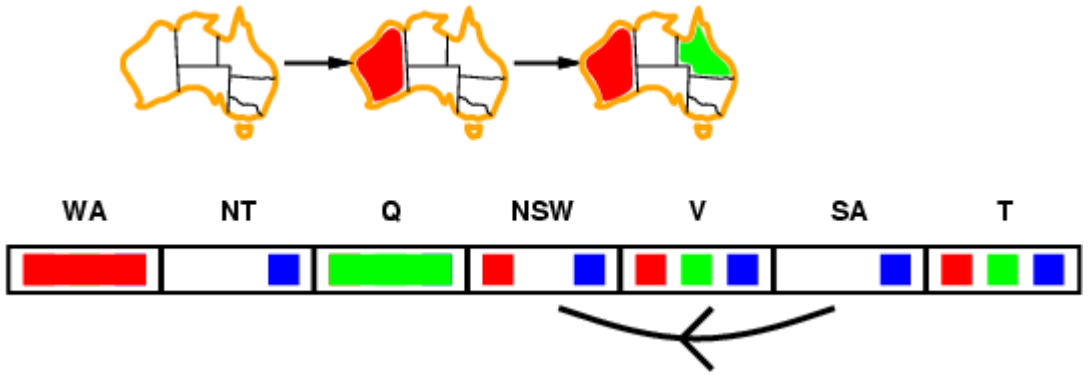


- NT and SA cannot both be blue!
- Constraint propagation repeatedly enforces constraints *locally*

Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff

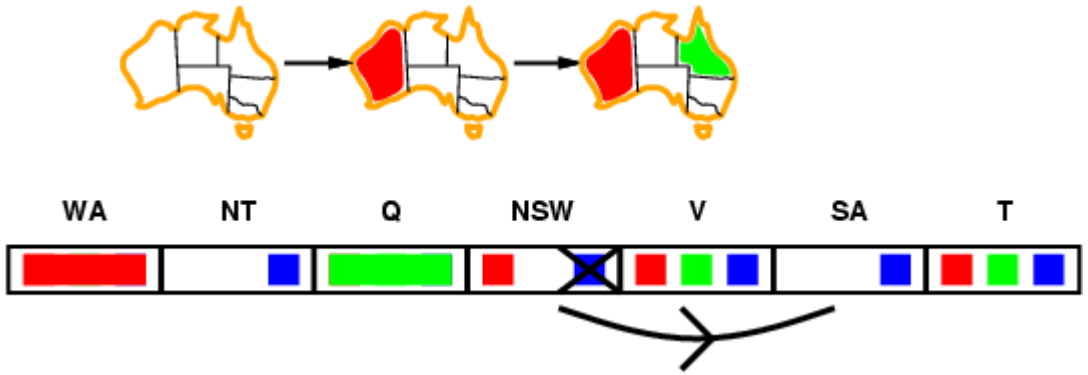
  for every value x of X there is some allowed y of Y



Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff

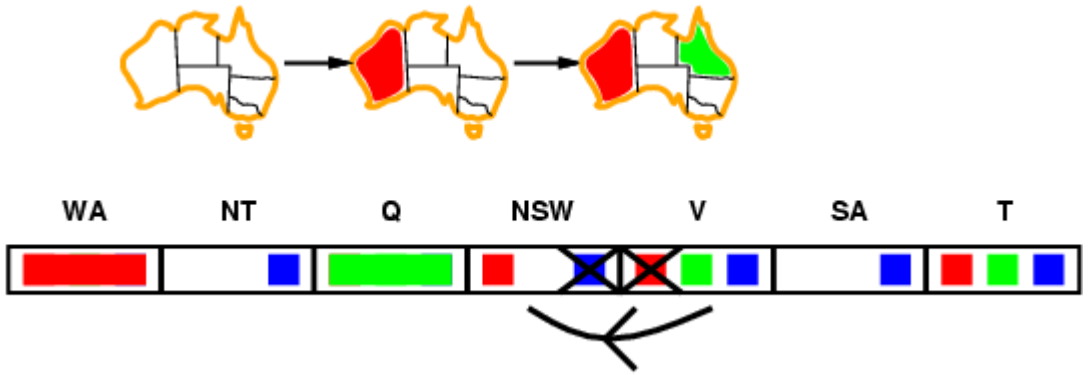
  for every value x of X there is some allowed y of Y



Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff

  for every value x of X there is some allowed y

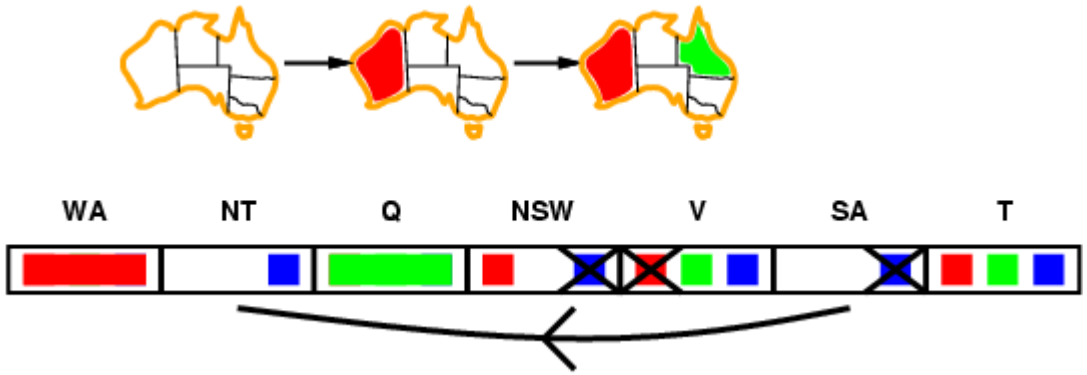


- If X loses a value, neighbors of X need to be rechecked

Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff

  for every value x of X there is some allowed y



- If X loses a value, neighbors of X need to be rechecked
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment

Arc consistency algorithm AC-3

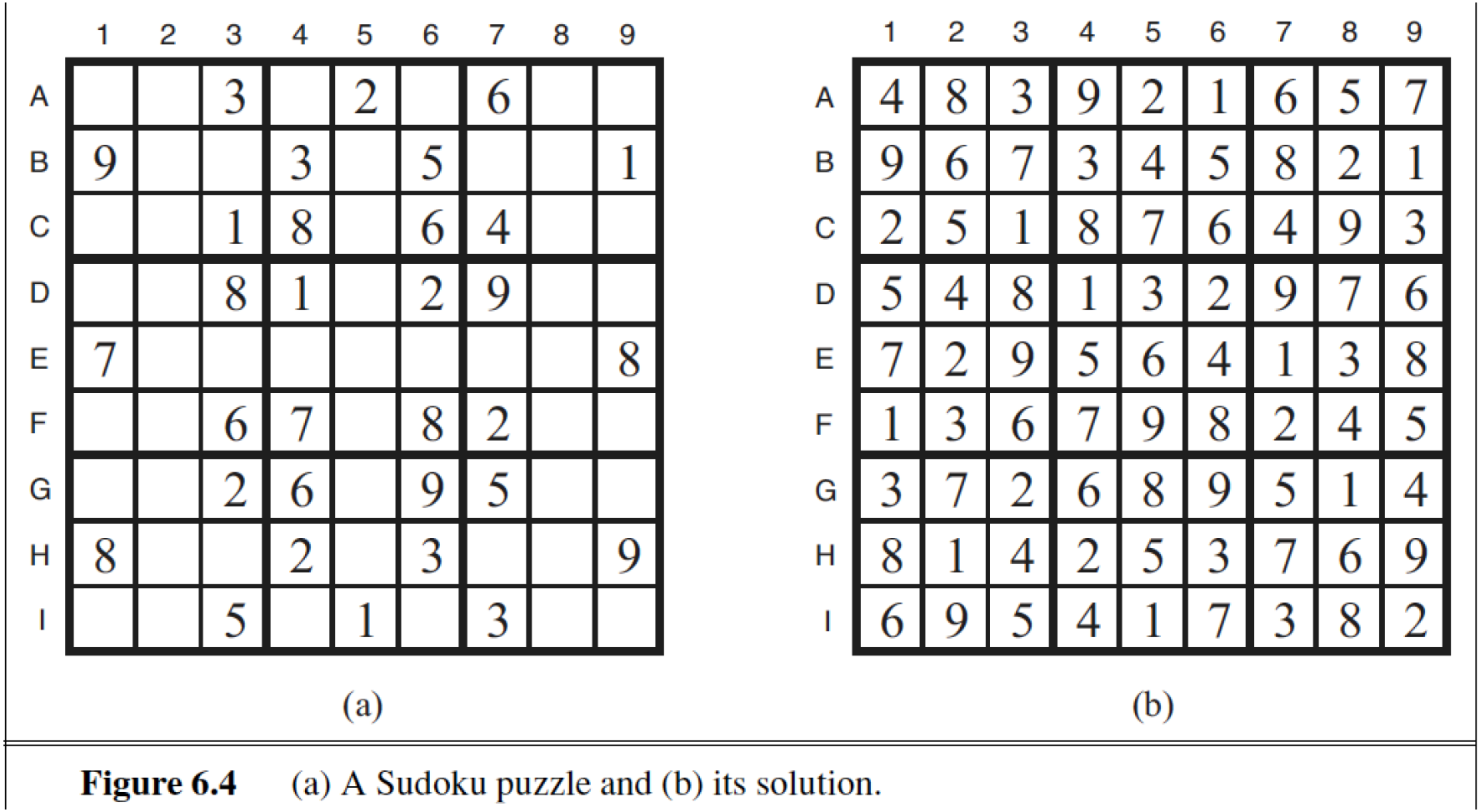
```
function AC-3( csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X_1, X_2, ..., X_n}
  local variables: queue, a queue of arcs, initially all the arcs in csp

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

---

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

Time complexity: $O(n^2d^3)$

Backtraching with Inference

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

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment then
          add {var = value} to assignment
          inferences ← INFERENCE(csp, var, value)
          if inferences ≠ failure then
              add inferences to assignment
              result ← BACKTRACK(assignment, csp)
              if result ≠ failure then
                  return result
      remove {var = value} and inferences from assignment
  return failure
```

Esempio Sudoku: AC-3 utile?



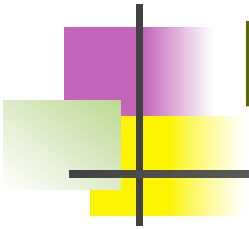
	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)

	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

(b)

Figure 6.4 (a) A Sudoku puzzle and (b) its solution.



Esempio Sudoku

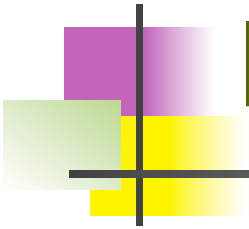
- Formulare il problema come CSP (variabili, domini, vincoli)
- Applicare AC-3 al problema precedente: provare ordine E6, I6, A6, ….
- Quanto è utile AC-3 nell'esempio?

Esercizio AC-3

- 3 variabili: x, y, z
- Dom(x)={3,5}, Dom(y)={7}, Dom(z)={4,7}
- Vincoli: x ≠ y, x < z, y ≠ z
- AC(x,z): si/no?
- AC(z,x): si/no?
- AC(z,y): si/no?
- ….

Esercizio AC-3: soluzione

- ....



Local search for CSPs

- Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned
- To apply to CSPs:
  - allow states with unsatisfied constraints
  - operators reassign variable values
- **Variable selection**: randomly select any conflicted variable
- **Value selection** by min-conflicts (general) heuristic:
  - choose value that violates the fewest constraints
  - i.e., hill-climb with *h(n)* = total number of violated constraints

Example: 4-Queens

- States: 4 queens in 4 columns ($4^4 = 256$ states)
- Actions: move queen in column
- Goal test: no attacks
- Evaluation: *h(n)* = number of attacks

- Given random initial state, can solve *n*-queens in almost constant time for arbitrary *n* with high probability: *n* = 10.000.000 is solvable!!

Esempio: 8 regine

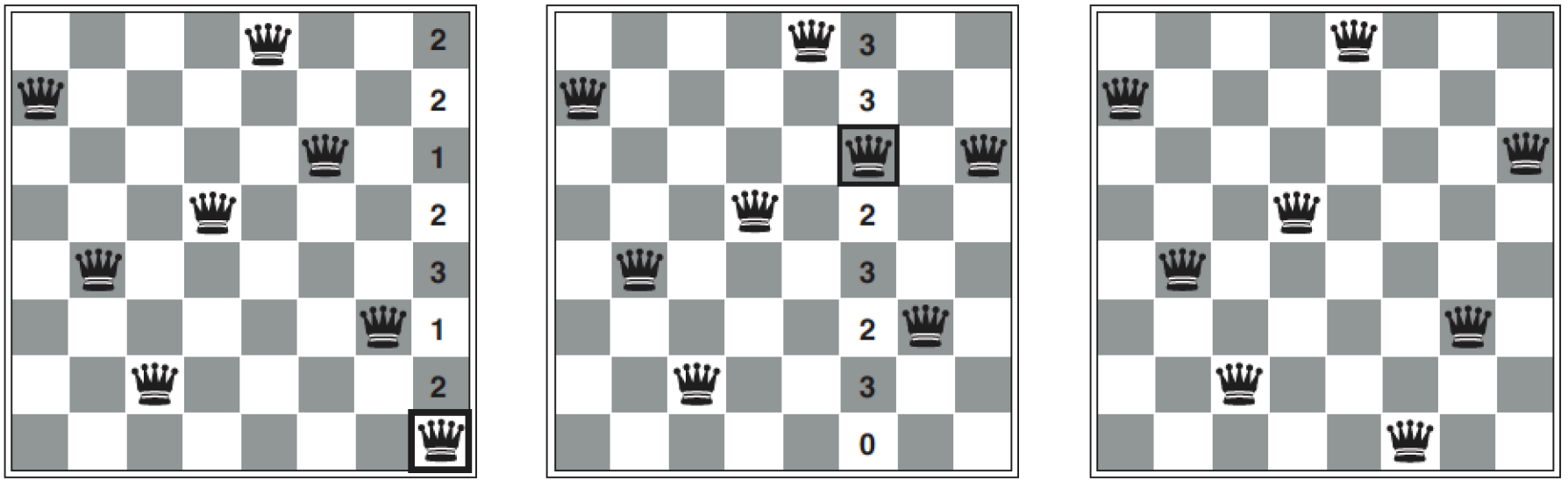


Figure 6.9 A two-step solution using min-conflicts for an 8-queens problem. At each stage, a queen is chosen for reassignment in its column. The number of conflicts (in this case, the number of attacking queens) is shown in each square. The algorithm moves the queen to the min-conflicts square, breaking ties randomly.

Local Search: Min-Conflic

- Funziona molto bene quando il problema contiene molte soluzioni

- Usato per schedulare le osservazioni del telescopio Spaziale Hubble:

  Drastica riduzione dei costi computazionali: una settimana di osservazioni schedulata in 10 minuti (prima in tre settimane!)

Da arc-consistency a *n*-consistency

- Estensione del concetto di arc(2)-consistenza

- **3-consistenza**:

  per ogni tripla (X,Y,Z):

  per ogni (X=v, Y=w) che soddisfa C_{xy}

  Esiste Z=k tale che

  C_{xz} *e* C_{zy} *sono soddisfatti*

- Anche chiamata **path-consistenza** perchè considera un cammino da X a Y attraverso Z

- Imporre la path-consistenza vuol dire eliminare gli assegnamenti a X e Y che non soddisfano la proprietà, cioè rendere più forte C_{xy}

Esempi di 3-consistenza (1)

- Si consideri il problema colorazione mappa australia con due colori: blu e rosso
- Il CSP ha soluzione?
- Imporre la proprietà di arc-consistency al CSP come modifica I domini?
- Il CPS arc-consistente è anche path-consistente?
- Se non lo è come si modificano i vincoli binari? (Cioè gli assegnamenti binari ammessi dai relativi vincoli)

Esempi di 3-consistenza (3)

- Si consideri il seguente CSP:

  X <= Y, Y < Z, X <= Z

  Dom(X)={1,2}, Dom(Y)={1,3}, Dom(Z)={2,4}

- E' arc-consistente?

- E' path-consistente?

- Se non lo è, quali sono gli assegnamenti binari da rimuovere?

K-consistenza e strong consistency

- La 1-2-3 consistenza si generalizza a **k-consistenza** con k>=1:

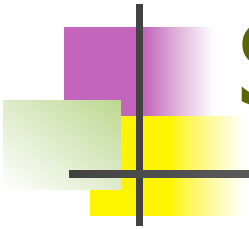
  Ogni assegnamento consistente per X1, …, X(k-1) può essere esteso a un assegnamento consistente per

  X1, …, Xk

- **Strong k-consistency**:

  vale q-concistency per ogni valore di q in {1…k}

- **Theorema**: *Se il CSP ha n variabili e vale strong n-consistency allora posso trovare soluzione, se esiste, con un algoritmo polinomiale (alla lavagna)*



Summary

- CSPs are a special kind search problem:
  - Molte applicazioni
  - *states* defined by values of a **fixed set of variables**
  - *goal test* defined by **constraints on variable values**
- **Backtracking** = *depth-first search with one variable assigned per node*
- **Variable ordering** and **value selection** heuristics help significantly
- **Forward checking** prevents assignments that guarantee later failure
- Constraint propagation (e.g., **arc consistency**) does additional work to constrain values and detect inconsistencies
- **Iterative min-conflicts** is usually effective in practice