Constraint Satisfaction Problems

Cercheremo ancora soluzioni a problemi. Ora però gli stati assumono una struttura e contengono informazione che viene utilizzata nella ricerca di una soluzione. Si introduce la nozione di vincolo e si vede come alcuni metodi di ricerca già studiati possono essere applicati con successo. Si introducono anche nuovi metodi di ricerca, specifici.

Qualche esempio dell'uso di vincoli

- **Design di oggetti:**
  la specifica della struttura di un oggetto deve rispettare dei vincoli funzionali e fisici
- **Allocazione di risorse:**
  occorre tener conto di quantità, priorità, tempi di utilizzo, tempi per entrare nella disponibilità, politiche d'uso
- **Progettazione di circuiti:**
  date un'occhiata a
  ftp://ftp.cs.brown.edu/pub/techreports/96/cs96-34.p
- **Pianificazione percorsi robotici:**
  tutti i movimenti fisici sono sottoposti a vincoli (per manovre per parcheggiare)

https://www.flickr.com/photos/pio1976/3130250099

Quali tipi di problemi?

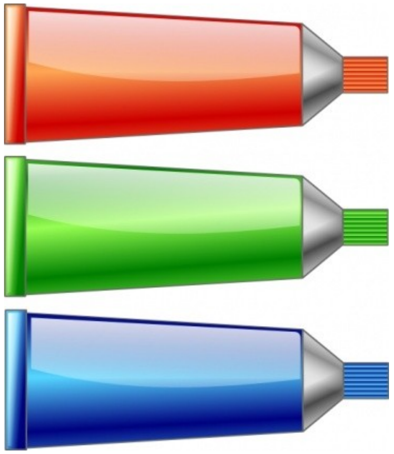
- Un constraint satisfaction problem (CSP) è definito da:
  - Un insieme di variabili X1, ..., Xn
  - Un insieme di vincoli C1, ..., Cm
  - Opzionale: in alcuni casi è richiesta la massimizzazione di una funzione obiettivo

Stato del problema e assegnamento

- Gli stati di un CSP sono dati da tutti gli assegnamenti possibili per le variabili del CSP
- Un assegnamento $\{X_{i1} = v_{i1}, X_{i2} = v_{i2}, \dots \}$ è un'attribuzione di valori a un sottoinsieme delle variabili del CSP
- Un assegnamento è detto:
  - Completo: se assegna valori a tutte le variabili del CSP
  - Consistente: se non viola alcun vincolo del CSP
  - Soluzione: se è completo e consistente
- Quando esiste una soluzione per un vincolo si dice anche che esiste un mondo possibile che soddisfa il vincolo

Esempio: colorature di mappe

Colorare la mappa di rosso, verde e blu evitando che territori confinanti siano dello stesso colore



Esempio: coloratura di mappe

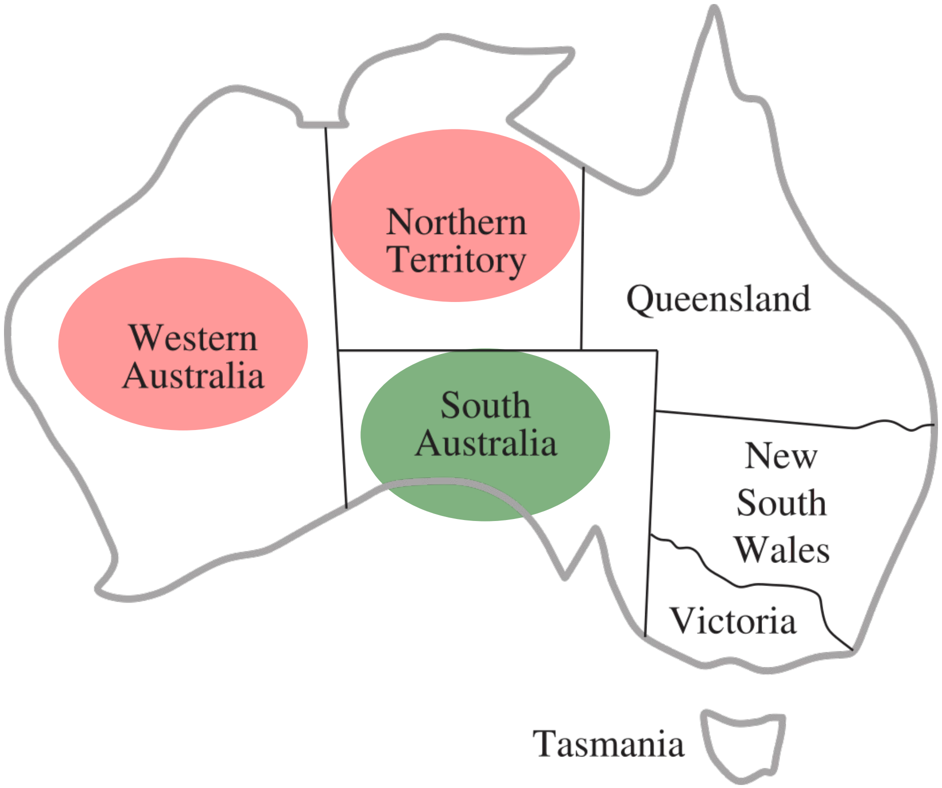
7 variabili, una per territorio, i cui domini sono tutti uguali {R, G, B}

WA ∈ {R, G, B}
NT ∈ {R, G, B}
SA ∈ {R, G, B}
Q ∈ {R, G, B}
NSW ∈ {R, G, B}
V ∈ {R, G, B}
T ∈ {R, G, B}

Esempio: un possibile assegnamento

7 variabili, una per territorio, i cui domini sono tutti uguali {R, G, B}

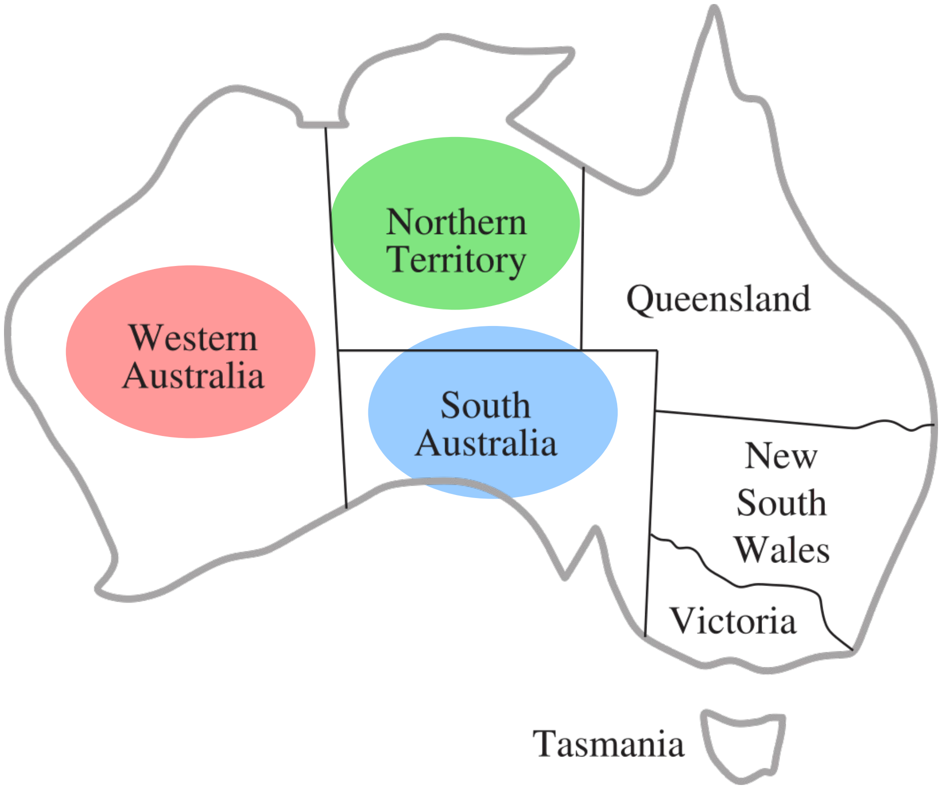
WA = R
NT = R
SA = G
Q
NSW
V
T



Esempio: assegnamento consistente

7 variabili, una per territorio, i cui domini sono tutti uguali {R, G, B}

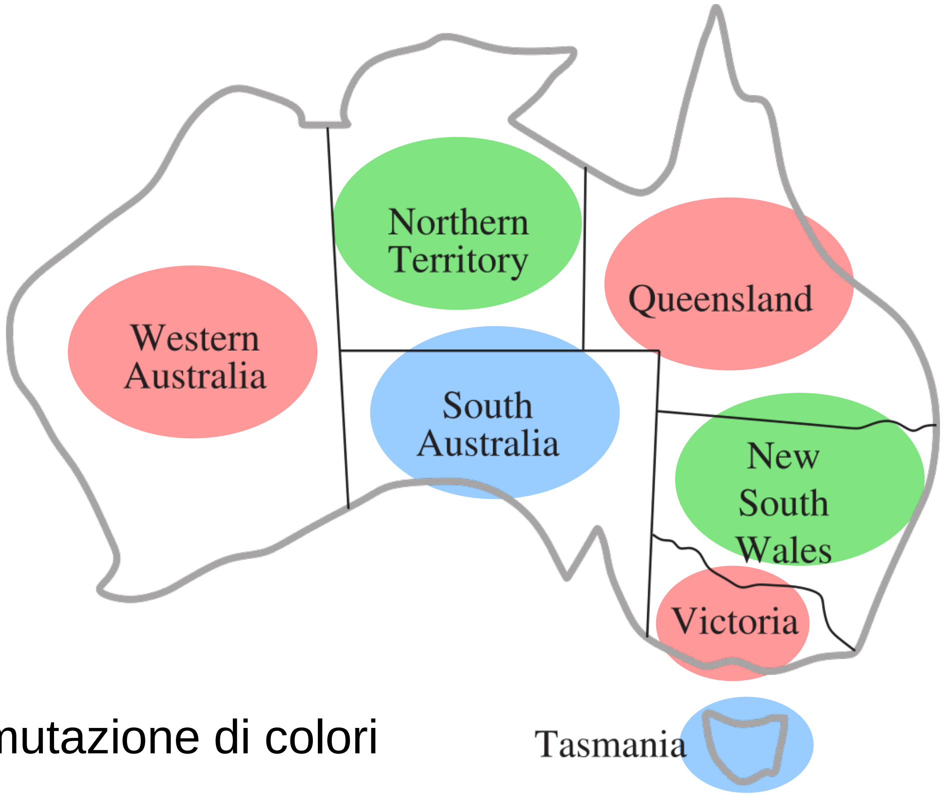
WA = R
NT = G
SA = B
Q
NSW
V
T



Esempio: una possibile soluzione

7 variabili, una per territorio, i cui domini sono tutti uguali {R, G, B}

WA = R
NT = G
SA = B
Q = R
NSW = G
V = R
T = B



NB: qualunque permutazione di colori
è una soluzione

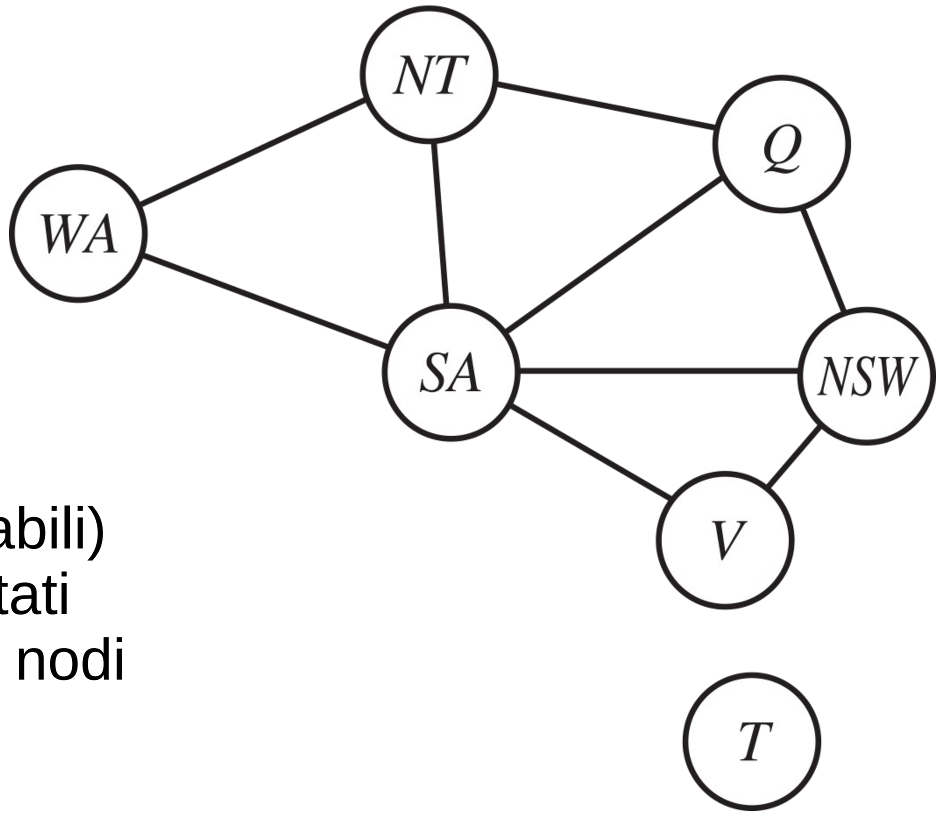
Esempio: vincoli

I vincoli riguardano coppie di territori confinanti. Supponendo che il linguaggio di rappresentazione contenga l'operatore ≠ (diverso) avremo quindi:

WA ≠ NT
WA ≠ SA
NT ≠ SA
NT ≠ Q
SA ≠ Q
SA ≠ NSW
SA ≠ V
Q ≠ NSW
NSW ≠ V

Tasmania è un'isola

Grafo di vincoli



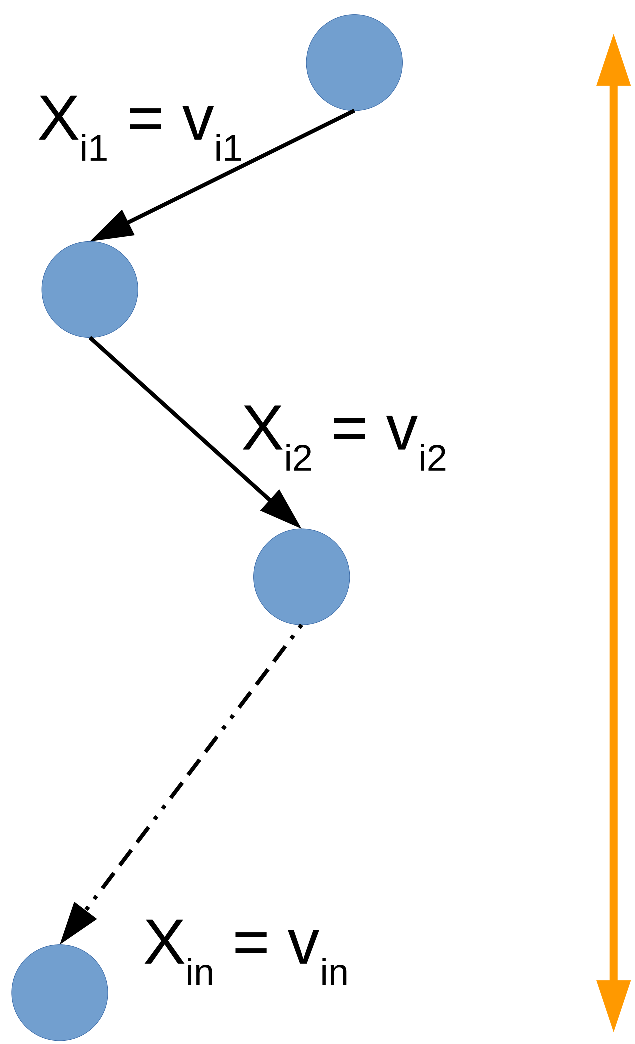
I vincoli binari (fra due variabili) possono essere rappresentati come archi di un grafo i cui nodi sono le variabili del CSP

CSP e problemi di ricerca

- È possibile formulare i CSP come problemi di ricerca in uno spazio degli stati:
  - **Stato iniziale** = { } assegnamento vuoto
  - **Successore** = assegnamento di un valore a una delle variabili che non ce l'hanno facendo attenzione che non sorgano conflitti
  - **Test obiettivo** = assegnamento completo
  - **Costo** = ogni passo ha costo costante (ad esempio 1)

Caratterizzazione

Profondità massima



Poiché ogni passo valorizza una variabile senza valore, i cammini saranno al più lunghi quanto il numero delle variabili del CSP. Sia n questo valore.

Stati dell'albero di ricerca

- Uno **stato** è un assegnamento di valori:
  {X1 = v1, X2, = v2, ..., Xn = vn}
- L'**ordine** (il cammino) con cui i valori vengono assegnati alle variabili è irrilevante sul risultato, ad esempio:

{X1 = **?**, **X2 = v2**, X3 = **?**}

{**X1 = v1**, X2 = v2, X3 = **?**}

{X1 = v1, X2 = v2, X3 = v3}

{**X1 = v1**, X2 = **?**, X3 = **?**}

{X1 = v1, X2 = **?**, **X3 = v3**}

{X1 = v1, X2 = v2, X3 = v3}

Stesso risultato

Stati dell'albero di ricerca

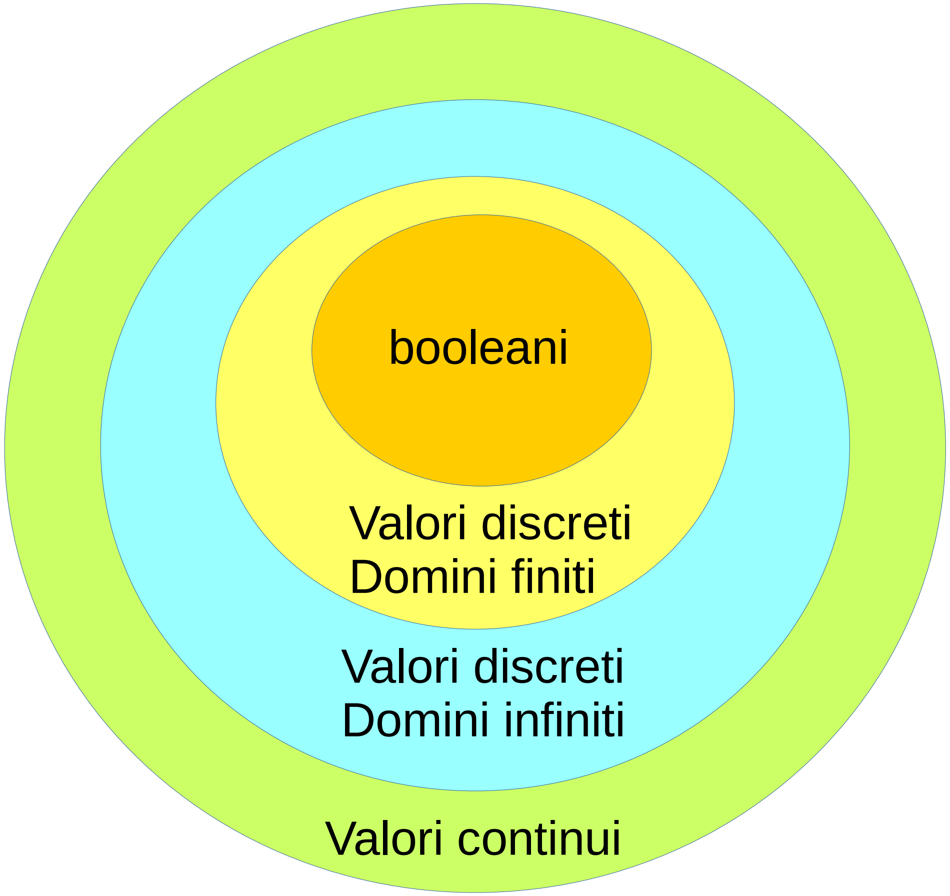
- Uno **stato** è un assegnamento di valori:
  {X1 = v1, X2, = v2, ..., Xn = vn}
- L'**ordine** (il cammino) con cui i valori vengono assegnati alle variabili è irrilevante sul risultato
- È possibile applicare metodi di ricerca già visti (esempio: blind)

Tipi di domini

- **Domini finiti:**
  è possibile enumerare i vincoli mettendo in relazione i diversi valori (es. Australia)
- **Domini infiniti:**
  non è possibile enumerare i vincoli,
  si usano **linguaggi di specifica**, esempio per dire che Lavoro2 deve iniziare almeno 5 giorni dopo Lavoro1 si potrebbe scrivere: Lavoro1 + 5 < Lavoro2
- **Domini continui**
  la programmazione lineare permette di risolvere CSP in cui i **vincoli sono disuguaglianze lineari** che specificano una regione convessa

Variabili

- Possono essere:
  - A valori discreti
    - Booleani (NP-completa)
    - A domini finiti
    - A domini infiniti
  - A valori continui



Arità dei vincoli

- **Vincoli unari**
  coinvolgono una variabile e un valore, esempio $T \neq G$ il colore della Tasmania deve essere diverso da verde

- **Vincoli binari**
  coinvolgono due variabili e possono essere rappresentati come archi di un grafo, esempio $WA \neq NT$

- **Vincoli a tre o più variabili**
  coinvolgono un numero qualsiasi di variabili, possono essere rappresentati da ipergraf (grafi con archi che connettono più di due nodi)i, esempio diverse(WA, NT, SA). A volte possono essere scomposti in vincoli binari

Vincoli a 3 e più variabili, esempio

Vincoli a 3 e più variabili, esempio

- Criptoaritmetica: gioco in cui a ogni lettera corrisponde una cifra diversa, bisogna trovare la sostituzione corretta

```
    S E N D
  + M O R E
  ---------
  M O N E Y
```

Vincoli a 3 e più variabili, esempio

- Criptoaritmetica: gioco in cui a ogni lettera corrisponde una cifra diversa, bisogna trovare la sostituzione corretta

```
    S E N D
+   M O R E
-----------
  M O N E Y
```

Dominio di S e M: [1,2,3,4,5,6,7,8,9]
Dominio di E,N,D,O,R,Y: [0,1,2,3,4,5,6,7,8,9]

Vincoli: a lettere diverse valori diversi

Vincoli a 3 e più variabili, esempio

- Criptoaritmetica: gioco in cui a ogni lettera corrisponde una cifra diversa, bisogna trovare la sostituzione corretta

```
    S E N D
  + M O R E
  _________
  M O N E Y
```

questo vincolo coinvolge tutte le variabili (non è binario):

```
(1000*S+100*E+10*N+D+
 1000*M+100*O+10*R+E) = (10000*M+1000*O+100*N+10*E+Y)
```

Vincoli e criteri di preferenza

- I vincoli possono essere più o meno rigidi
- Distinguiamo fra vincoli veri e propri e criteri di preferenza
- Vincoli e criteri di preferenza si differenziano sul modo in cui la loro violazione impatta sulla soluzione:
  - Una soluzione deve soddisfare tutti i vincoli
  - Una soluzione può violare uno o più criteri di preferenza
  - Il soddisfacimento dei criteri di preferenza permette di ordinare le soluzioni identificando quelle preferibili e quelle meno preferibili

Esempio, criterio di preferenza

- Nella definizione dell’orario di lezione un docente può dire di preferire le lezioni del mattino
- Questo “vincolo”va composto con quelli degli altri docenti
- Risultato: l’orario prevede che il docente faccia lezione sempre al pomeriggio ☹

11-12	(Aula F)	Svil AS T3 (Laboratorio Dijkstra)	(Aula F)	(Aula A)	(Laboratorio Turing)
12-13	Sic (Aula F)	Svil AS T2 (Laboratorio Von Neumann) Svil AS T3 (Laboratorio Dijkstra)	Sic (Aula F)	Svil AS (Aula A)	Prog III Lab1 (Laboratorio Turing)
13-14					
14-15	Sist Int (Aula F)	LogicaInf (Aula C) Sic (Aula F)	Svil AS (Aula C)	LogicaInf (Aula B) Sist Int (Aula F)	LogicaInf (Aula F)
15-16	Sist Int (Aula F)	LogicaInf (Aula C) Sic (Aula F)	Svil AS (Aula C)	LogicaInf (Aula B) Sist Int (Aula F)	LogicaInf (Aula F)
16-17	Svil AS T2 (Laboratorio Von Neumann)				
17-18	Svil AS T2 (Laboratorio Von Neumann)				
18-19					

La soluzione non è ottimale, perché non soddisfa la preferenza, ma potrebbe essere la migliore fra quelle possibili

Rappresentare gli stati

- Stato = assegnamento
- Modi diversi di rappresentare lo stato influiscono sul tempo della ricerca
- Approccio di forza bruta: generate-and-test

  Usa solo informazioni di stato
- Ricerca blind con backtracking

  Usa anche i vincoli

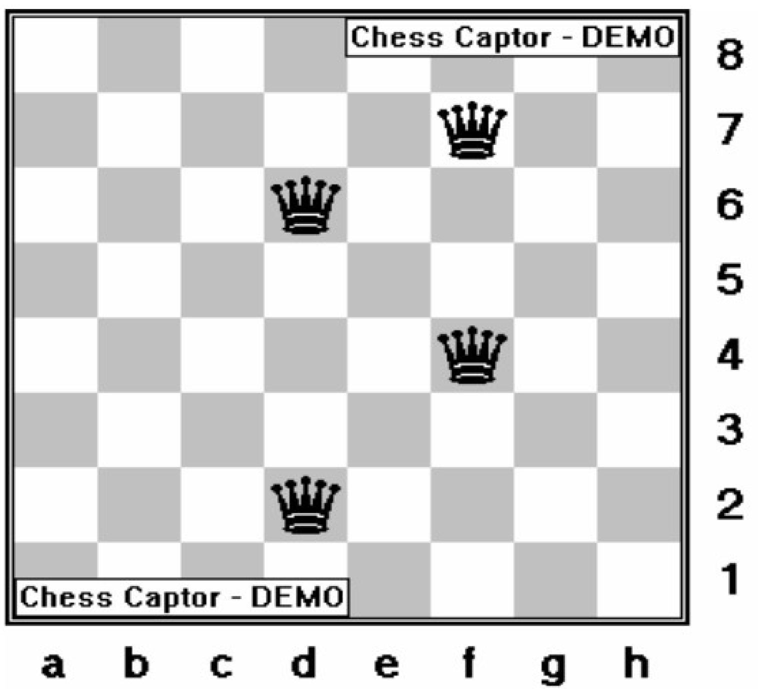
Usare solo informazioni di stato è pratico?

Generate-and-test

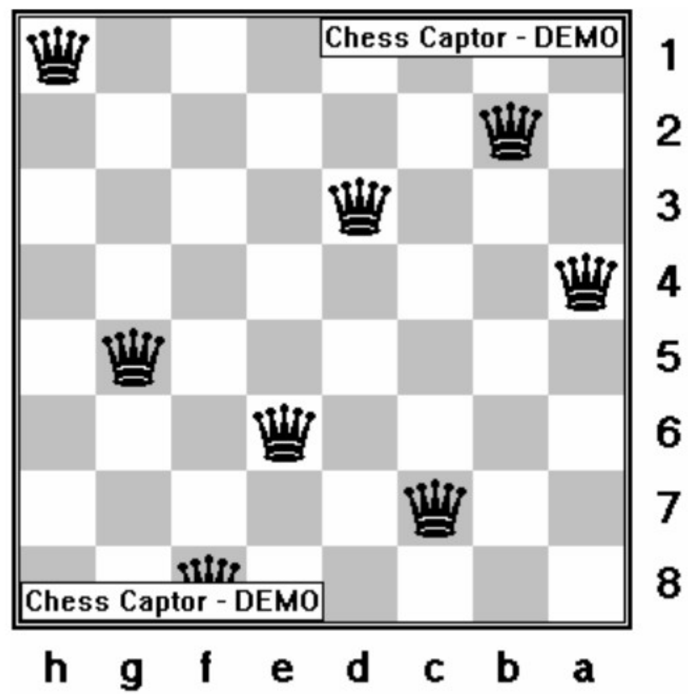
- Si può adottare una rappresentazione a stato completo, cioè utilizzare solo **assegnamenti completi**, che possono o meno essere consistenti
- **Generate-and-test**, metodo per la risoluzione di CSP molto semplice (ma costoso):
  - Finché non hai una soluzione e ci sono delle alternative:
    1) Genera un assegnamento completo
    2) Controlla se è consistente
    3) Se sì è una soluzione, esci dal ciclo
    4) Se no torna al passo 1
  - Se hai una soluzione restituiscila
  - Altrimenti fallimento
- Può richiedere l'esplorazione dell'intero spazio degli assegnamenti!

Esempio: problema delle 8 regine

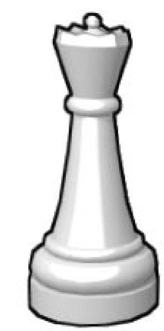
- Posizionare 8 regine su una scacchiera 8x8 in modo tale che nessuna risulti sotto attacco di un’altra
- Il problema ha 92 soluzioni distinte (12 se si considerano solo quelle non ottenibili da altre per simmetria)



NO



SOLUZIONE



Generate-and-test: 8 regine

Q			
Q			
Q			
Q			

	Q		
Q			
Q			
Q			

		Q	
Q			
Q			
Q			

			Q
Q			
Q			
Q			

Q			
	Q		
Q			
Q			

Q			
		Q	
Q			
Q			

…

NB: le scacchiere sono 4x4 per rendere l'idea del procedimento senza occupare troppo spazio

Quanto tempo ci vorrà?

La risposta dipende da come abbiamo rappresentato il problema

8 regine come CSP

- **Primo modo (esempio)**:
  - **Variabili**: Q1, Q2, …, Q8
    ognuna rappresenta la posizione di una regina
  - **Domini**: le posizioni sono numerate in modo crescente
    1, 2,3, …, 16, 17, 18, …, 64
  - La soluzione riportata prima corrisponde all'assegnamento:
    Q 1 = 1, Q2 = 15, Q3 = 21, Q4 = 32, Q5 = 34, Q6 = 44, Q7 = 54, Q8 = 59
  - **Problema**:
    - se ci sono **n variabili** ognuna delle quali può assumere **d valori**, si hanno **d^n possibili assegnamenti**
    - Nell'esempio ben oltre 28.100.000.000.000 possibili assegnamenti

8 regine come CSP, versione 2

- **Secondo modo**:
  ogni regina deve per forza occupare una diversa colonna quindi **assegnamo ciascuna regina alla colonna di numero uguale** (esempio: Q1 sta nella colonna 1)
- **Variabili**: Q1, …, Q8
- **Dominio**: 1, … 8
  il numero di riga che completa le coordinate della regina
- La soluzione di prima sarà quindi rappresentata come:
  Q1 = 1, Q2 = 5, Q3 = 8, Q4 = 6, Q5 = 3, Q6 = 7, Q7 = 2, Q8 = 4
- **Va molto meglio**: abbiamo "solo" 8^8 = 16.777.216 assegnamenti alternativi ma sono ancora tanti …

Generate-and-test: 8 regine

- E se avessimo un numero maggiore di regine?
- In generale per N regine si hanno N^N configurazioni
- Se avessimo 16 regine le configurazioni sarebbero 18.446.744.073.709.551.616
- ~ 1200 anni di computazione

2) Usiamo più conoscenza: introduciamo i vincoli

Per passi, a partire da una semplice ricerca in profondità con backtracking, vedremo come si può rendere più efficiente la ricerca usando più e meglio l'informazione a disposizione

Generate-and-test: 8 regine

- Fino ad ora non abbiamo rappresentato i **vincoli**:
  - Date due qualsiasi regine Qi e Qj queste devono essere posizionate in modo che nessuna delle due sia attaccata dall'altra
  - Due regine si attaccano quando:
    - Occupano la stessa colonna: **impossibile** nella nostra rappresentazione
    - Occupano la stessa riga: **aggiungiamo il vincolo Qi ≠ Qj**
    - Occupano la stessa diagonale: **aggiungiamo il vincolo |i - j|≠ |Qi - Qj|**

Ricerca di una soluzione in profondità

- Esploriamo lo spazio degli stati (dei possibili assegnamenti) utilizzando una ricerca depth-first con backtracking
- Ricerca non informata + vincoli per decidere quando potare un cammino
- La limitatezza dei cammini rende ragionevole la scelta della ricerca in profondità, tuttavia spesso branching factor elevato

Ricerca di una soluzione con backtracking

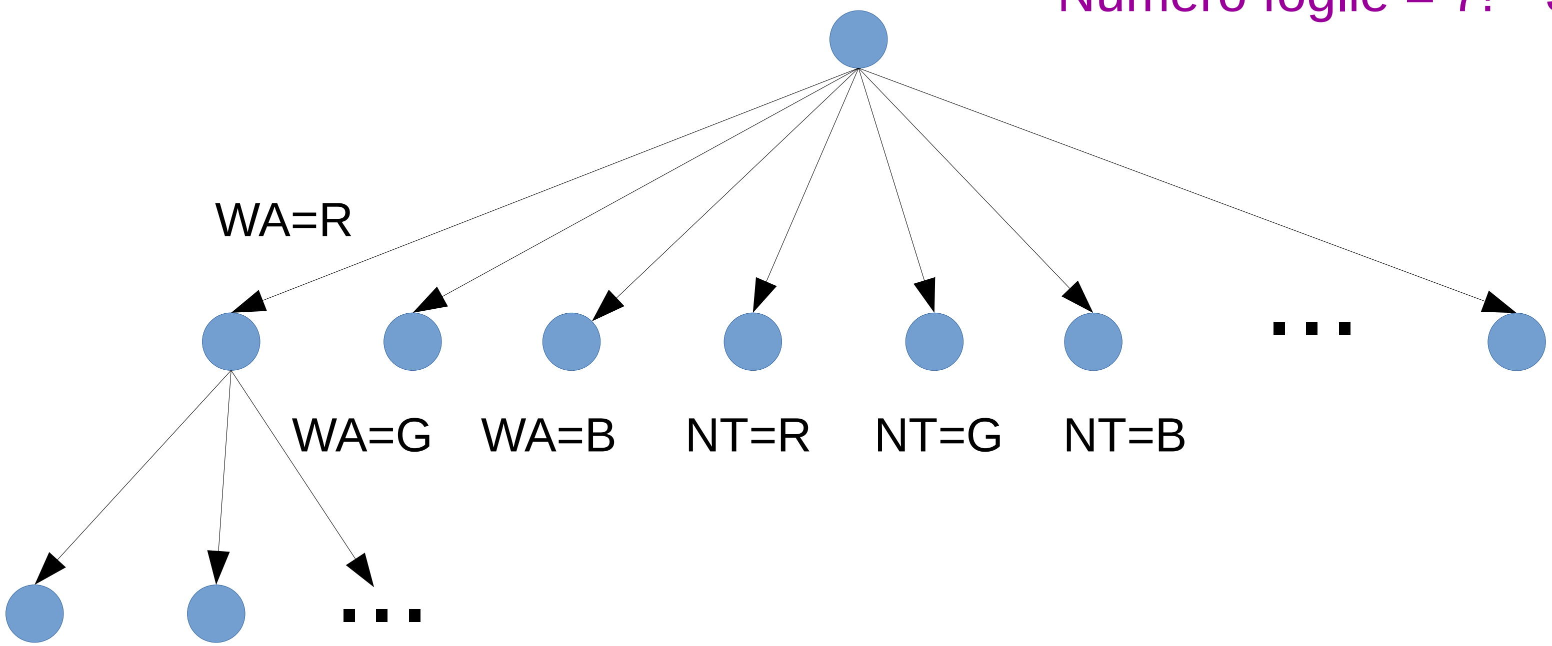
- Siano:
  - **n** = numero delle variabili
  - **d** = numero medio dei valori possibili per ciascuna variabile
  - Uno qualsiasi dei valori può essere assegnato a una qualsiasi delle variabili
- Il branching factor sarà $n*d$ al primo livello, $(n-1)*d$ al secondo (perché una variabile è stata fissata), eccetera
  - l'albero avrà $\mathbf{n! * d^n}$ foglie

Esempio Australia

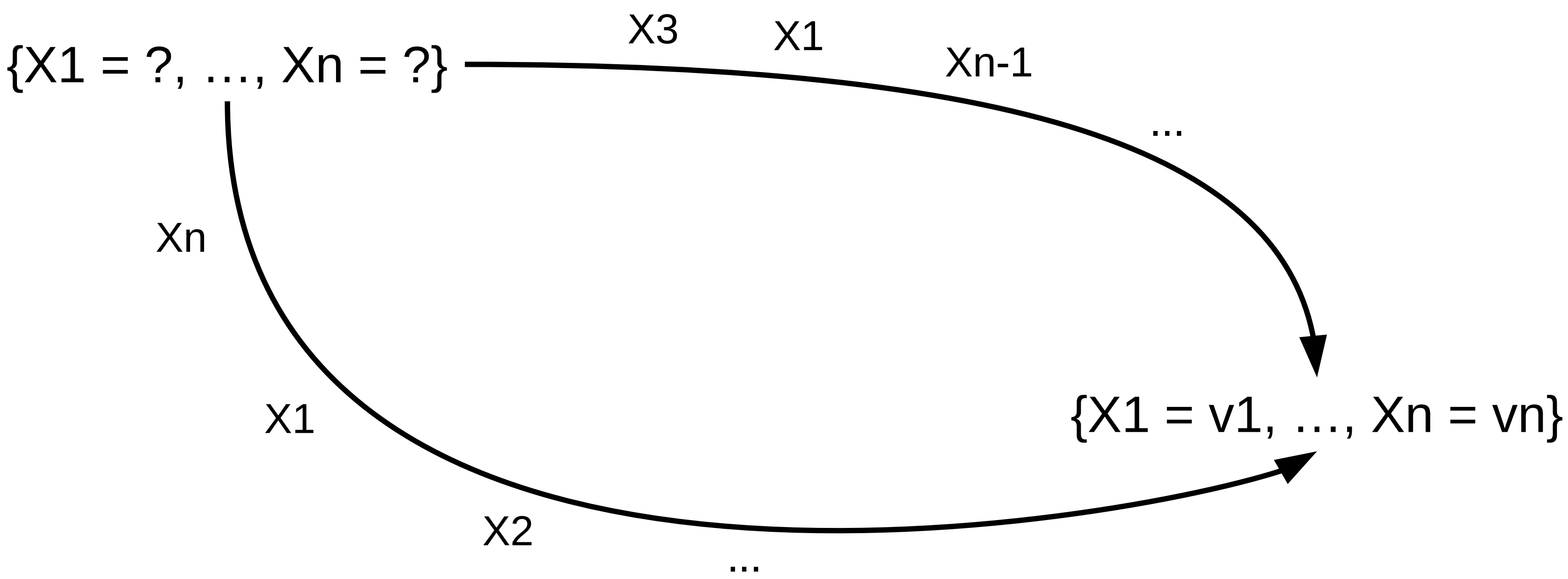
7 variabili
3 valori per variabile

Branching factor liv. 1 = 21
Branching factor liv. 2 = 18
…

Numero foglie = 7! * 3^7 =11.022.480



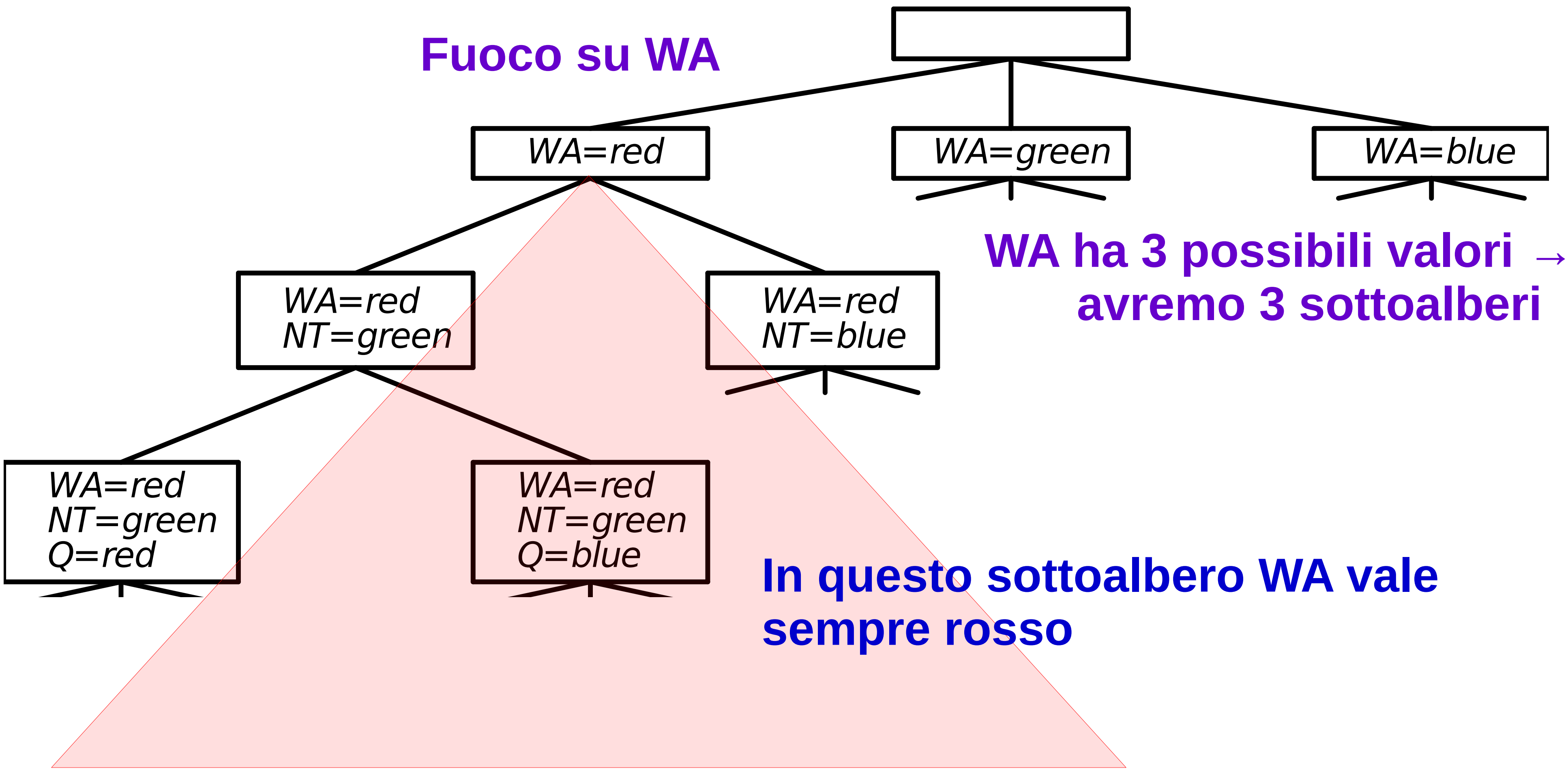
I CSP sono commutativi



La stessa soluzione è ottenibile tramite più cammini, permutando l'ordine con cui le variabili vengono assegnate. La soluzione è sempre la stessa e l'ordine è ininfluente sulla bontà della soluzione.

Gli algoritmi quindi **prima scelgono una variabile** e **poi un valore** per questa variabile

Esempio, Australia



Con questa restrizione **il numero di foglie si riduce a d^n**, nel nostro esempio a 2.187 (un bel risparmio rispetto a 11.022.480)

Ricerca con backtracking: algoritmo

```
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 ≠ failure then return result
         remove {var = value} from assignment
   return failure
```

Figure 5.3 A simple backtracking algorithm for constraint satisfaction problems. The algorithm is modeled on the recursive depth-first search of Chapter 3. The functions SELECT-UNASSIGNED-VARIABLE and ORDER-DOMAIN-VALUES can be used to implement the general-purpose heuristics discussed in the text.

Questo algoritmo è riportato nella II edizione del libro ma non nella III

Ricerca con backtracking: algoritmo

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

function RECURSIVE-BACKTRACKING(                                        ailure
  if assignment is complete then retur
  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-BACKTRACK
        if result ≠ failure then return resu
        remove {var = value} from assignment
  return failure
```

Fuoco su una variabile per volta

Cerca un valore consistente con l'assegnamento parziale attuale, secondo i vincoli

Arriva qui solo quando ha esaminato tutti i valori per la variabile e nessuno va bene. Failure causa backtracking ed esplorazione di alternative

Figure 5.3 A

algorithm is modeled on the recursive depth-first search of Chapter 3. The functions SELECT-UNASSIGNED-VARIABLE and ORDER-DOMAIN-VALUES can be used to implement the general-purpose heuristics discussed in the text.