Problemi di soddisfacimento di vincoli

Maria Simi

a.a. 2015/2016

Problemi di soddisfacimento di vincoli (CSP)

- Sono problemi con una struttura particolare, per cui conviene pensare ad algoritmi specializzati
- È un esempio di rappresentazione fattorizzata, in cui si comincia a dire qualcosa sulla struttura dello stato
- Esistono euristiche generali che si applicano e che consentono la risoluzione di problemi di dimensioni significative per questa classe
- La classe di problemi formulabili in questo modo è piuttosto ampia: layout di circuiti, scheduling, …

Strategie di soluzione di CSP

- Per questa classe di problemi individueremo:
  - Le formulazione come problema di ricerca e le caratteristiche dello spazio di ricerca
  - Algoritmi di ricerca non informati
  - Le euristiche da utilizzare, che sono di *uso generale* per questa classe di problemi: la logica è quella di eliminare combinazioni di variabili-valori che non soddisfano i vincoli, limitando l'esplorazione.
  - Le strategie di ricerca.

Formulazione di problemi CSP

- *Problema*: descritto da tre componenti
  1. $X = \{X_1\ X_2 \ldots X_n\}$ insieme di variabili
  2. $D = \{D_1\ D_2 \ldots D_n\}$ insieme di domini
     dove $D_i = \{v_1, \ldots, v_k\}$ è l' insieme dei valori possibili per X_i
  3. $C = \{C_1\ C_2 \ldots C_m\}$ insieme di vincoli (relazioni tra le variabili)
- *Stato*: un assegnamento [parziale | completo] di valori a variabili

  $\{X_i = v_i\ , X_j = v_j \ldots\}$
- *Stato iniziale*: { }
- *Azioni*: assegnamento di un valore a una variabile
- *Soluzione (goal test)*: un assegnamento completo (le variabili hanno tutte un valore) e consistente (i vincoli sono tutti soddisfatti)

Come esprimere i vincoli

- Forma generale:

  <*ambito*, *relazione*>

  Ambito: tupla di variabili che partecipano nel vincolo

  Relazione: un insieme di tuple di valori

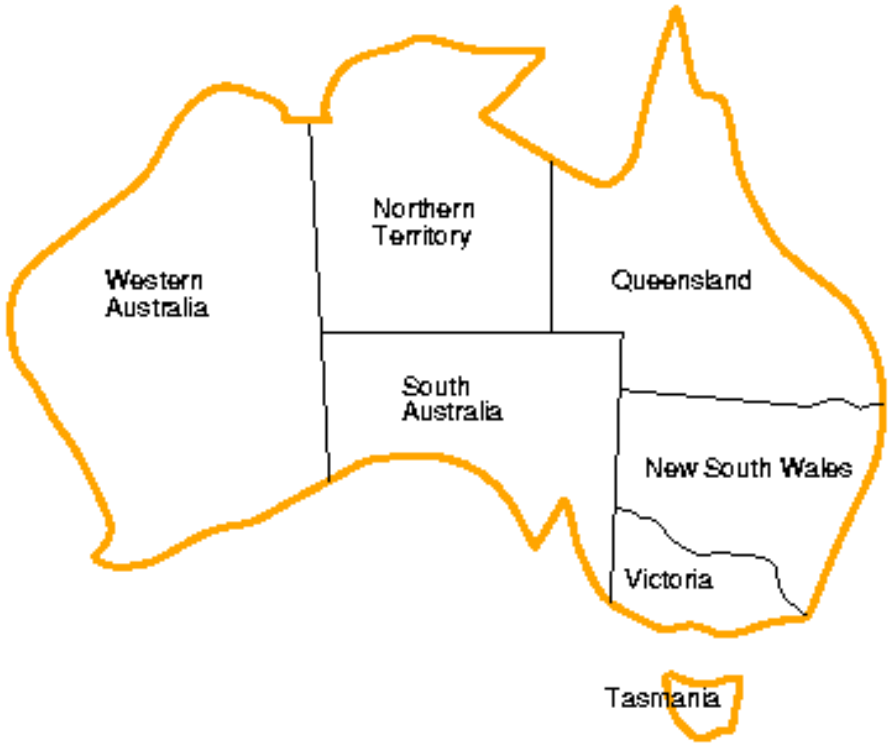
- Esempio: $X=\{X_1, X_2\}$ $D_1=D_2=\{A, B\}$
  - Vincolo che le variabili devono avere valori diverse:

  $<(X_1, X_2), \{(A, B), (B, A)\}>$ oppure

  $<(X_1, X_2), X_1 \neq X_2>$

Colorazione di una mappa

Si tratta di colorare i diversi paesi sulla mappa con tre colori in modo che paesi adiacenti abbiano colori diversi



Variabili

$X = \{WA, NT, SA, Q, NSW, V, T\}$

Domini

$D_{WA} = D_{NT} = D_{SA} = D_Q = D_{NSW} = D_V = D_T = \{red, green, blue\}$

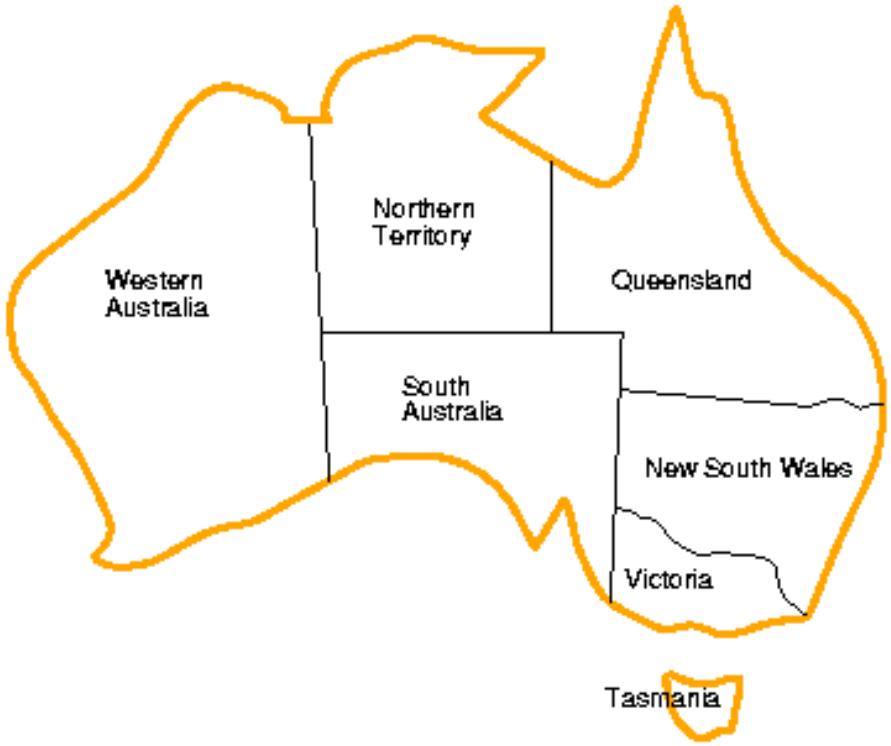
Vincoli

$C = \{WA \neq NT, WA \neq SA,$

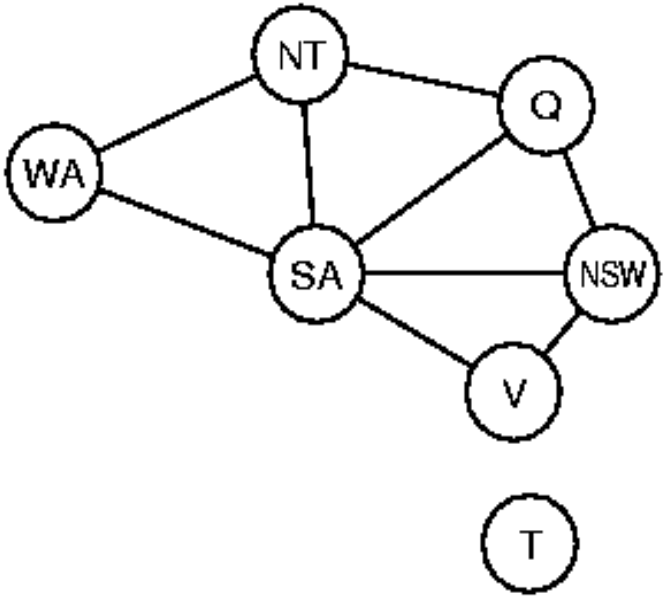
$NT \neq Q, NT \neq SA, SA \neq Q,$

$SA \neq NSW, SA \neq V, NSW \neq V\}$

Colorazione di una mappa



Grafo dei vincoli



Altri problemi: le 8 regine

- Il problema delle 8 regine
  - $X=\{Q_1, \ldots, Q_8\}$ *una regina per colonna*
  - $D_i=\{1, 2, 3, 4, 5, 6, 7, 8\}$ *numero di riga*
  - C, i vincoli di non attacco
- Esempio di vincolo tra Q_1 e Q_2

  $<(Q_1, Q_2)$, {(1, 3) (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (2, 4), … }>
- Formulazione incrementale o a stato completo

Altri problemi: *scheduling*

- Il problema di pianificare una serie di lavori
  - X i tempi di inizio dei lavori da compiere
  - D tempi in un certo intervallo (numero finito)
  - C, vincoli tra i lavori

- vincoli di *precedenza*

  $X_i + d_i < X_j$ *dove* d_i *è la durata del lavoro* i

  - *vincolo di mutua esclusione tra* X_i e X_j

  $X_i + d_i < X_j$ *OR* $X_j + d_j < X_i$

- vincolo di *durata complessiva* dei lavori

  limitiamo il dominio dei tempi

Tipi di problemi CSP

- Variabili con domini discreti e finiti
  - CSP booleani (valori *vero* e *falso*)
- Variabili con domini discreti e infiniti
  - Si tratta di cercare soluzioni in un range di valori
- Variabili con domini continui (ricerca operativa)
  - programmazione lineare (vincoli espressi con uguaglianze e disuguaglianze lineari)

Tipi di vincoli

- I vincoli possono essere:
  - unari (es. “x pari”)
  - binari (es. “x > y”)
  - di grado superiore (es. x+y = z)
    comunque riconducibili a vincoli binari, intoducendo più variabili
- Vincoli globali
  - Es. TuttiValoriDiversi, come nelle righe e colonne del Sudoku
  - Es. Ogni lettera una cifra distinta nella cripto-aritmetica
- Vincoli assoluti o di preferenza
  - Problemi di ottimizzazione di vincoli (COP)

Cercare vs "propagare"

- Finora potevamo solo ricercare la soluzione nel grafo degli stati.
- Adesso possiamo fare anche un minimo di ragionamento che ci porta a restringere i domini e quindi al limitare la ricerca: *propagazione di vincoli*
- Tipicamente un misto delle due cose.

Ricerca in problemi CSP

- Ad ogni passo si assegna una variabile
  - La massima profondità della ricerca è fissata dal numero di variabili *n*
- L'ampiezza dello spazio di ricerca è

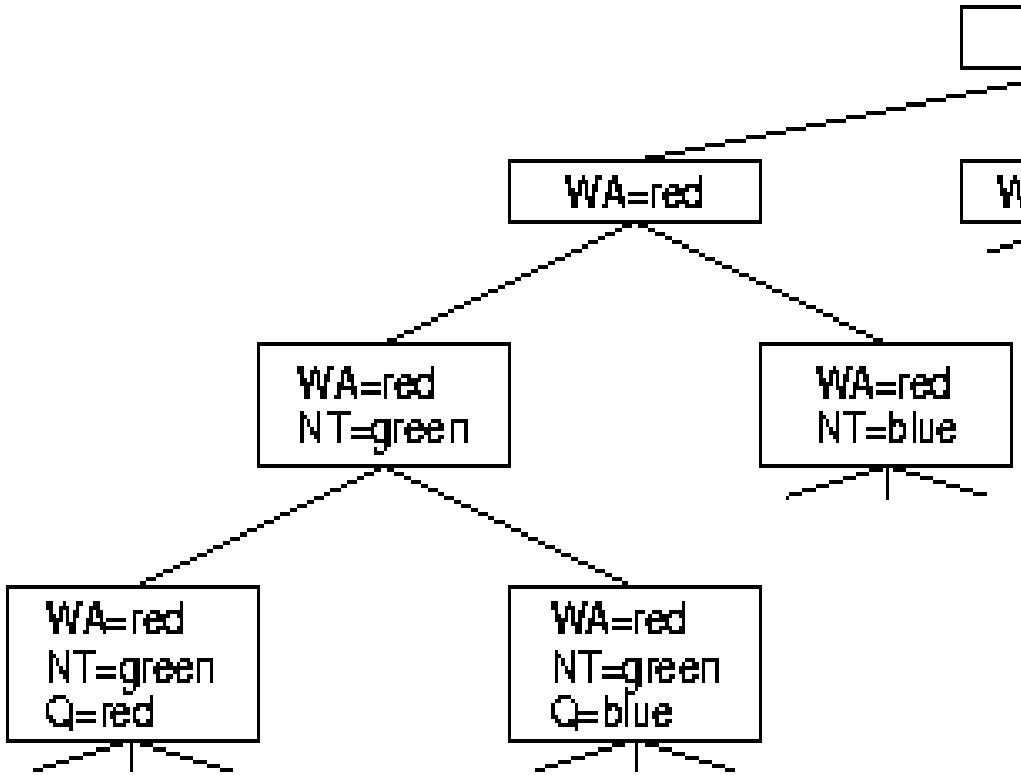
$$|D_1| \times |D_2| \times \ldots \times |D_n|$$

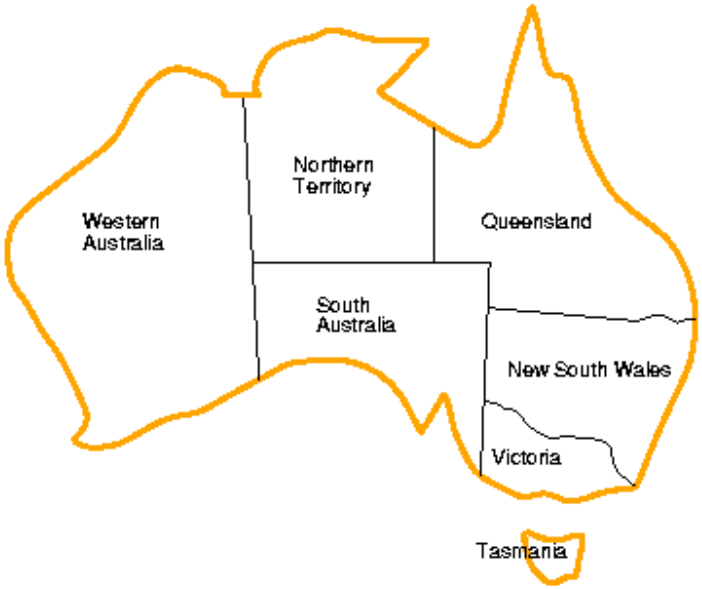
  dove $|D_i|$ è la cardinalità del dominio di X_i
- Il fattore di diramazione
  - Teoricamente pari a *nd al primo passo; (n-1) d al secondo ... le foglie sarebbero* $n! \cdot d^n$
  - Riduzione drastica dello spazio di ricerca dovuta al fatto che il *goal-test* è commutativo (l'ordine non è importante)
  - *In realtà: pari* alla dimensione dei domini *d* (d^n *foglie*)

Strategie di ricerca

- *Generate and Test*. Si genera in profondità una soluzione e si controlla: poco efficiente
- *Ricerca con backtracking* (BT): ad ogni passo si assegna una variabile e si controllano i vincoli; si torna indietro in caso di fallimento
  - *Controllo anticipato* della violazione dei vincoli: è inutile andare avanti fino alla fine e poi controllare; si può fare *backtracking* non appena si scopre che un vincolo è stato violato.
  - La ricerca è limitata naturalmente in profondità dal numero di variabili quindi il metodo è completo.

Esempio di backtracking





Backtracking ricorsivo per CSP

```
function Ricerca-Backtracking (csp) returns una soluzione o fail
    return Backtracking-Ricorsivo({ }, csp)   //un assegnamento vuoto

function Backtracking-Ricorsivo(ass, csp) returns una soluzione o fail
    if ass è completo then return ass
    var ← Scegli-var-non-assegnata(csp)
    for each val in Ordina-valori-dominio(var, ass, csp) do
        if val consistente con ass then              //controllo anticipato
            aggiungi {var = val } a ass
            risultato ← Backtracking-Ricorsivo(ass, csp)
            If risultato ≠ fail then return risultato
        rimuovi {var = val } da ass                  // si disfa lo stato
    return fail
```

Euristiche e strategie per CSP

- Scegli-var-non-assegnata: Quale variabile scegliere?
- Ordina-valori-dominio: Quali valori scegliere?
- Qual è l'influenza di un assegnamento sulle altre variabili? Come restringe i domini?

  ➔ *propagazione di vincoli*

  (tecniche per la *consistenza locale*)
- Come evitare di ripetere i fallimenti?

  ➔ b*acktracking intelligente*

Scelta delle variabili

1. MRV (*Minimum Remaining Values*)

   scegliere la variabile che ha meno valori legali [residui], la variabile *più vincolata*. Si scoprono prima i fallimenti (*fail first*)
2. Euristica *del grado:* scegliere la variabile coinvolta in più vincoli con le altre variabili (la *variabile più vincolante* o di *grado maggiore*)

   Da usare a parità di MRV

Scelta dei valori

- Una volta scelta la variabile come scegliere il valore da assegnare?

1. Valore *meno vincolante*: quello che esclude meno valori per le altre variabili direttamente collegate con la variabile scelta
   - Meglio valutare prima un assegnamento che ha più probabilità di successo
   - Se volessimo tutte le soluzioni l'ordine non sarebbe importante

CSP con ricerca BT + inferenza

```
function Backtracking-Ricorsivo(ass, csp) returns una soluzione o fail
    if ass è completo then return ass
    var ← Scegli-var-non-assegnata(csp)
    for each val in Ordina-valori-dominio(var, ass, csp) do
        if val consistente con ass then
            aggiungi {var=val } a ass
            inferenze ← Inferenza(csp, var, val)
            if inferenze ≠ fail then
                aggiungi inferenze a ass
                risultato ← Backtracking-Ricorsivo(ass, csp)
                If risultato ≠ fail then return risultato
        rimuovi {var=val } e inferenze da ass
    return fail
```

Propagazione di vincoli

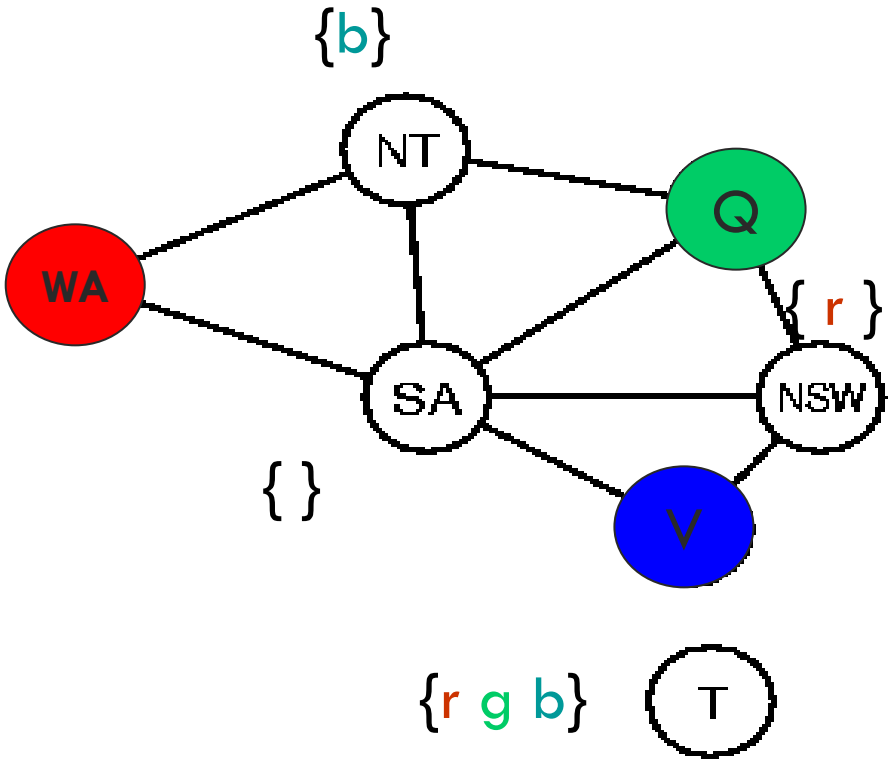
1. Verifica in avanti (*Forward Checking* o *FC*)
   - assegnato un valore ad una variabile si possono eliminare i valori incompatibili per le altre var. *direttamente collegate* da vincoli (non si itera)
2. Consistenza di nodo e d'arco
   - si restringono il valori dei domini delle variabili tenendo conto dei vincoli unari e binari su tutto il grafo (si itera finché tutti i nodi ed archi sono consistenti)

Esempio di FC

WA=r

Q=g

V=b



Stesso esempio in forma tabellare

	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	(R)	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	(R)	B	(G)	R B	R G B	B	R G B
After *V=blue*	(R)	B	(G)	R	(B)		R G B

Consistenza di nodo

- Un nodo è consistente se tutti i valori nel suo dominio soddisfano i vincoli unari
- Una rete di vincoli è *nodo-consistente* se tutti i suoi nodi sono consistenti
- I vincoli unari quindi possono essere risolti restringendo opportunamente i domini delle variabili

Consistenza d'arco

- Nel grafo di vincoli, un arco tra X e Y, (X, Y), è *consistente* rispetto a X, se per ogni valore *x* di X c'è almeno un valore *y* di Y consistente con *x*.
- Si dice anche: X è *arco-consistente* rispetto a Y.
- Se un arco (X, Y) non è consistente rispetto a X (o Y) si cerca di renderlo tale, rimuovendo valori dal dominio di X (o di Y).
- Se il dominio di una variabile viene modificato vanno ricontrollati tutti gli archi con i vicini. Si itera fino a che tutte le variabili sono *arco-consistenti*.

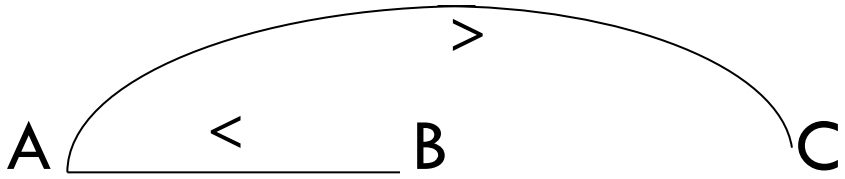
AC-3 e MAC

function AC-3 (*csp*) **returns** *false* o *true* // *csp* = {X, D, C}
    var locale: *coda* // inizializzata con tutti gli archi in *csp*
    while *coda* non è vuota **do**
        $(X_i, X_j) \leftarrow$ POP(*coda*);
        if *Rimuovi-valori-inconsistenti (csp,* X_i, X_j) **then**
            if dimensione $D_i = 0$ **then return** *false*
            for each X_k **in** X_i.nodi-adiacenti – $\{X_j\}$ **do**
                aggiungi (X_k, X_i) a *coda*
    return *true* // *la coda è vuota*

- MAC (Maintaining Arc Consistency) controlla la consistenza degli archi con AC-3 all'inizio e dopo ogni assegnamento

Consistenza d'arco: esempio

- Variabili A {1, 2, 3, 4} B {1, 2, 3, 4} C {1, 2 ,3 4}
- Vincoli A < B; A > C



Coda	Arco	Dominio arco
{(A, B), (B, A), (A, C), (C, A)}	*coda iniziale*	
{(B, A), (A, C), (C, A)}	(A, B)	A = {1, 2, 3, 4}
{(A, C), (C, A)}	(B, A)	B = {1, 2, 3, 4}
{(C, A)}	(A, C)	A = {1, 2, 3}
{(B, A), (C, A)}	*aggiungi* (B, A) per ricontrollarlo	
{(C, A)}	(B, A)	B = {2, 3, 4}
{ }	(C, A)	C = {1, 2, 3, 4}

Alla fine di AC-3: A ={2, 3} B={3, 4} C= {1, 2}

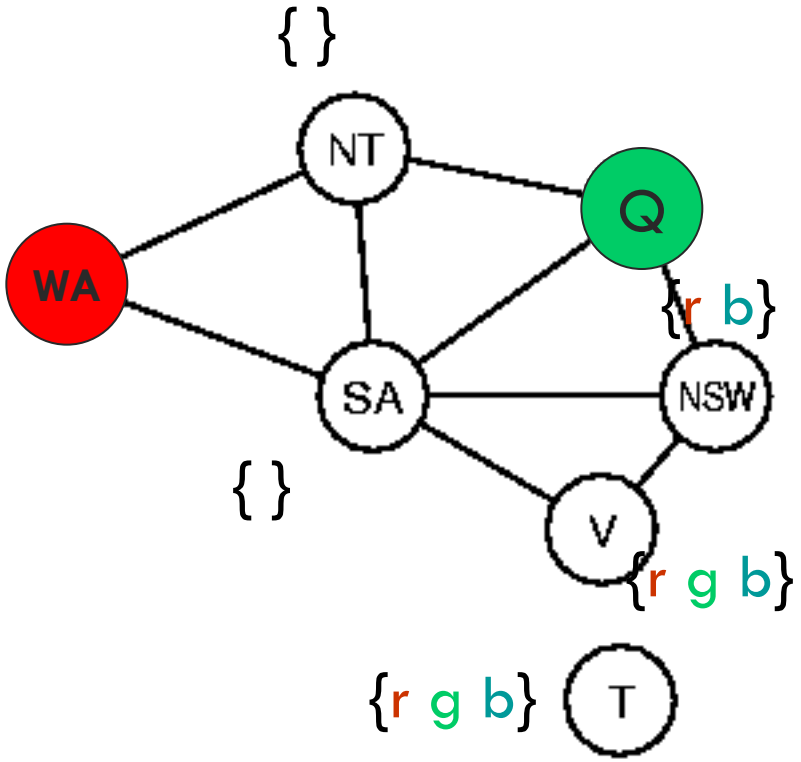
Esempio di MAC

WA=r

AC-3 riduce i domini

Q=g

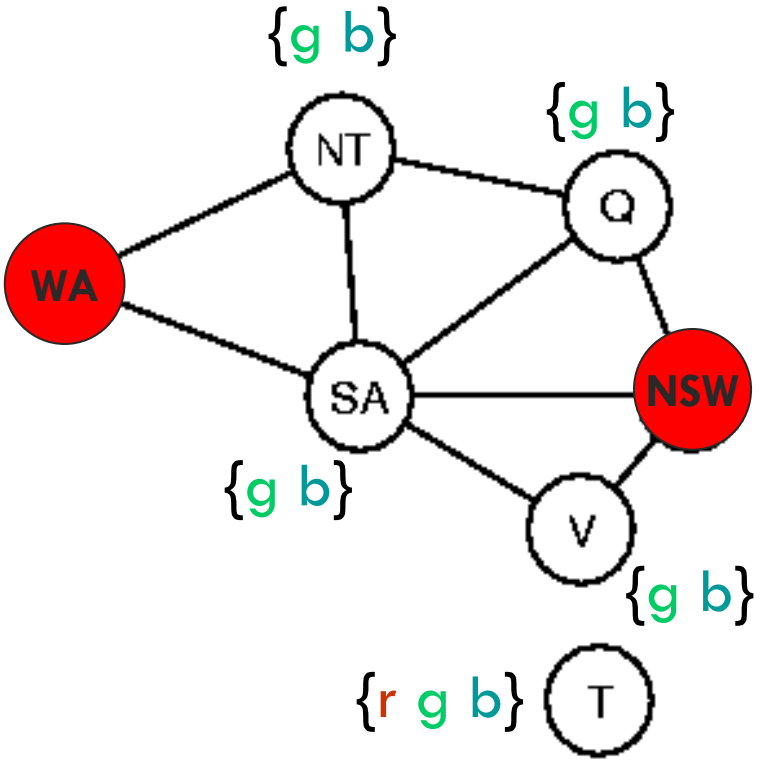
Si scopre subito che non va bene con AC-3



MAC non completo

- Più efficace di forward-checking, ma non rileva tutte le inconsistenze Esempio:

  WA=red
  NSW=red
  ... non viene rilevata inconsistenza

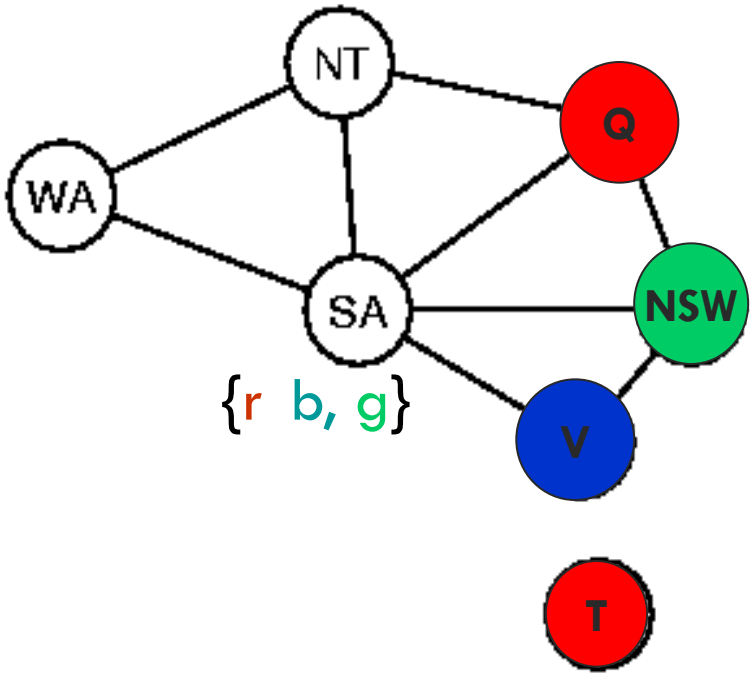


Complessità di AC-3

- Un metodo più efficace ma più costoso di FC per propagare i vincoli.
- Devono essere controllati tutti gli archi (sia c il #archi)
- Se durante il controllo di un arco (X, Y) il dominio di X si restringe vanno ricontrollati tutti gli archi adiacenti (Z, X) con $Z \neq Y$.
- Il controllo di consistenza di un arco ha complessità d^2, se d è la dimensione massima dei domini
- Un arco deve essere controllato al massimo d volte
- Complessità: $O(c\, d^3)$ ... polinomiale

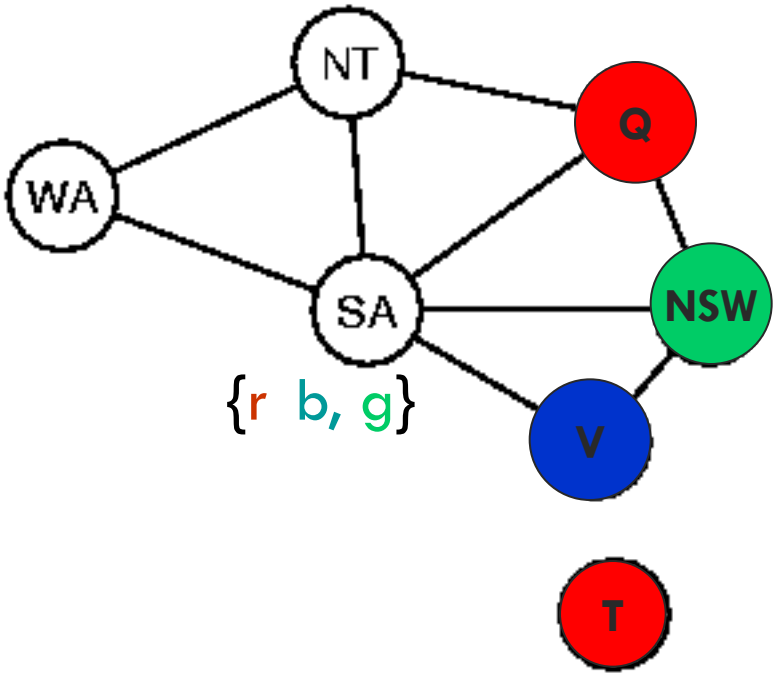
Backtracking "cronologico"

- Supponiamo di avere {Q=red, NSW=green, V=blue, T=red}
- Cerchiamo di assegnare SA
- Il fallimento genera un backtracking "cronologico"
- … e si provano tutti i valori alternativi per l'ultima variabile, T, continuando a fallire



Backtracking "intelligente"

- Si considerano alternative solo per le variabili che hanno causato il fallimento {Q, NSW, V}, *l'insieme dei conflitti*
- *Backtracking guidato dalle dipendenze*

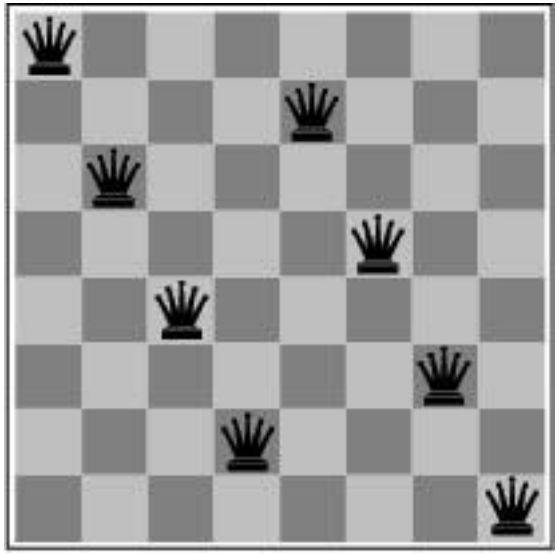


Le 8 regine come CSP (richiamo)

Formulazione come CSP:

- Q_i: posizione della regina nella colonna i-esima
- D_i: {1 … 8}
- Vincoli di "*non-attacco*" tra Q_1 e Q_2:

  < (Q1, Q2), {(1, 3) (1, 4) (1, 5) (1, 8) …
  (2, 4) (2, 5) … (2, 8) …}>
- Nella formulazione a stato completo si può usare un metodo locale

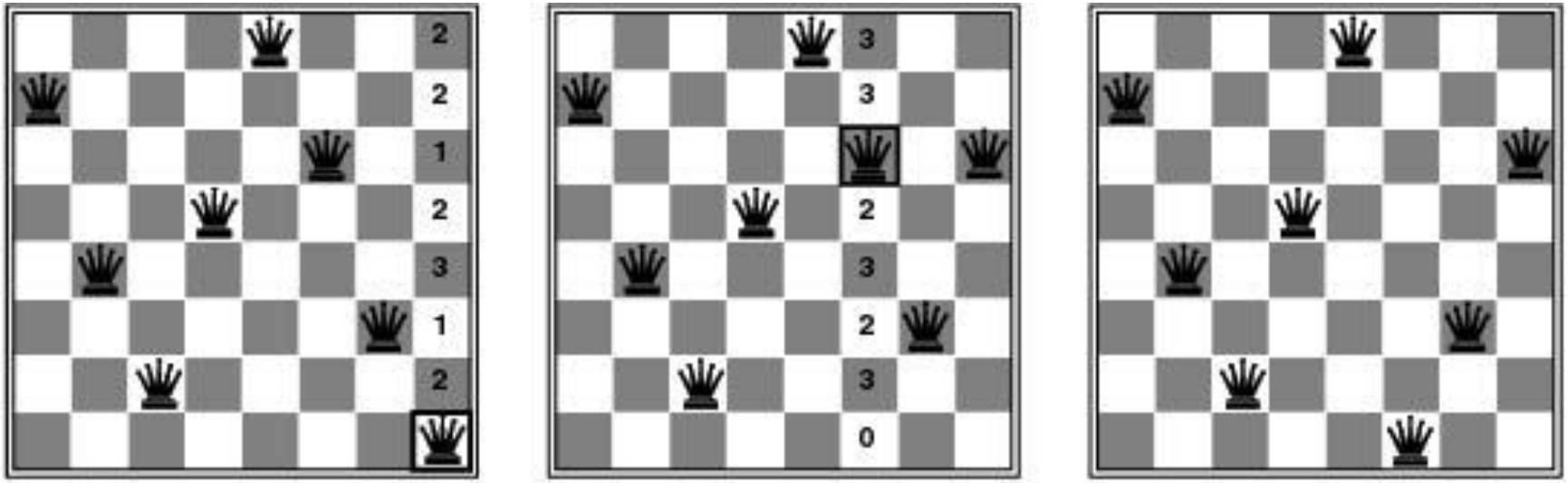


Metodi CSP locali: le regine

- Si parte con tutte le variabili assegnate (tutte le regine sulla scacchiera)
- ad ogni passo si modifica l'assegnamento ad una variabile per cui un vincolo è violato (si muove una regina minacciata su una colonna).
- È un algoritmo di *riparazione euristica*.

Min-conflicts

- Un'euristica nello scegliere un nuovo valore potrebbe essere quella dei *conflitti minimi*: si sceglie il valore che crea meno conflitti.



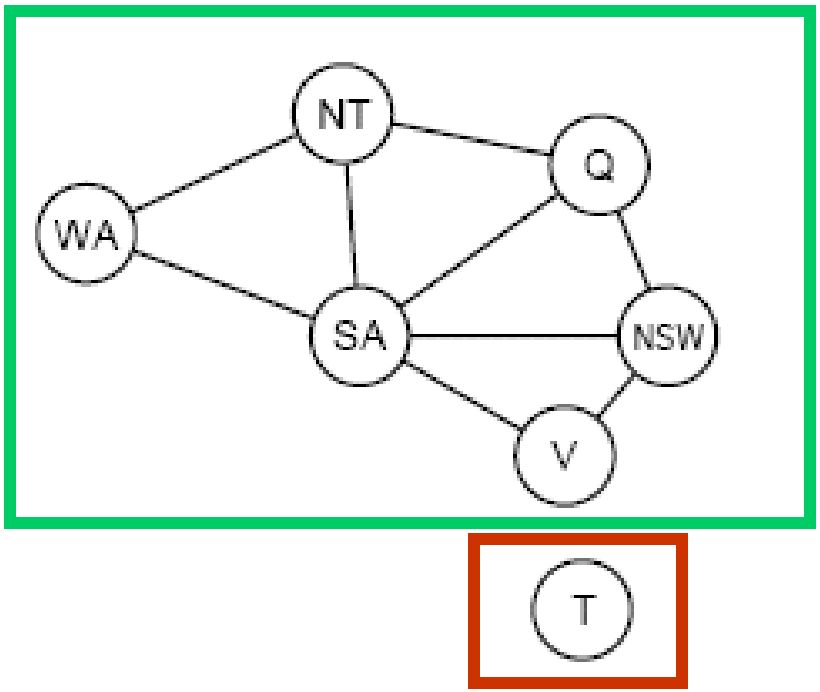
- *Molto efficace*: 1 milione di regine in 50 passi!

Sottoproblemi indipendenti

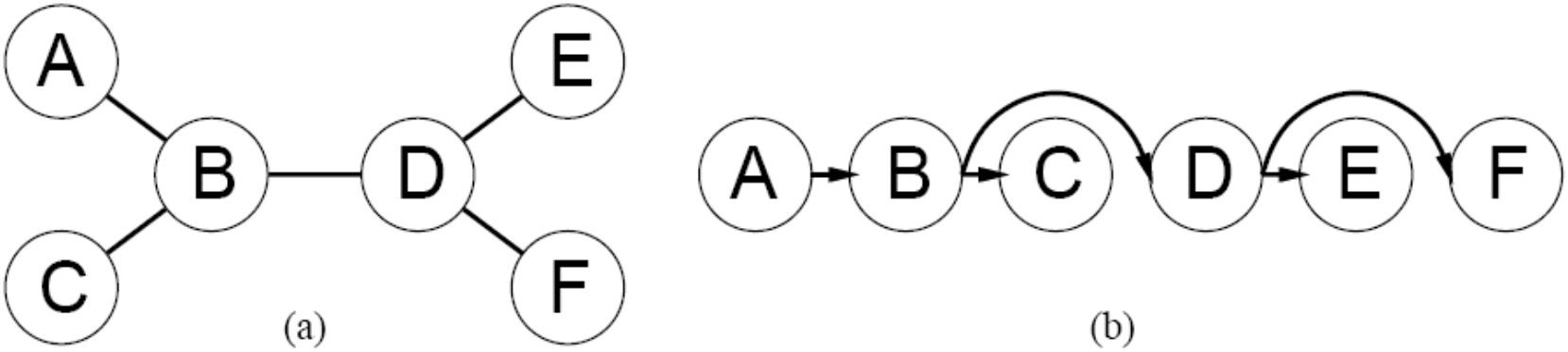
- *n* # variabili
- c # variabili per sottoproblema
- *n*/c problemi indipendenti
- *d* dimensione domini
- $O(d^c\ n/c)$ *lineare* nel numero di variabili *n*

  piuttosto che

  $O(d^n)$ *esponenziale*!



Struttura dei problemi: albero



- In un grafo di vincoli ad albero, due variabili sono collegate da un solo cammino (a)
- Scelto un nodo come radice, l'albero induce un ordinamento "topologico" sulle variabili (b)
- Consistenza d'arco orientato (DAC)
  - Dato un ordinamento per le variabili: $X_1, X_2, \ldots X_n$

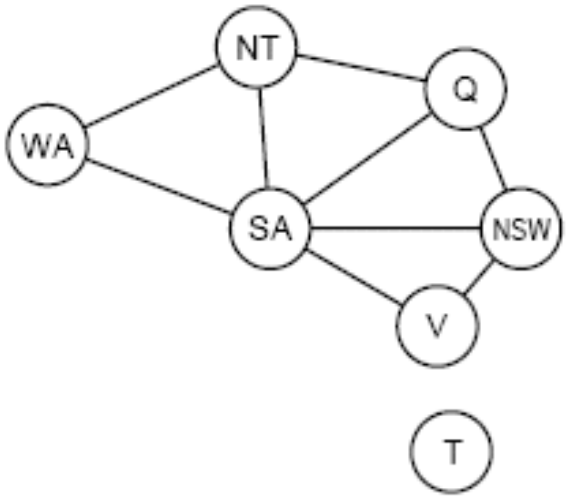
  ogni arco $X_i \rightarrow X_j$ con $i < j$ è consistente

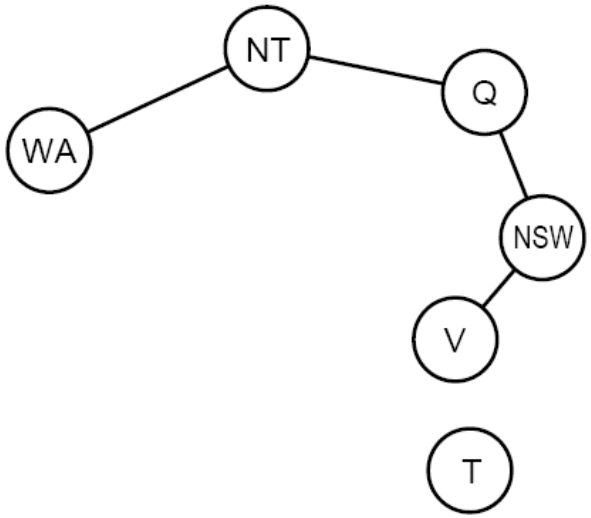
Algoritmo basato su DAC

1. Procedendo da X_n a X_2 rendere gli archi $X_i \rightarrow X_j$ consistenti *riducendo il dominio di* X_i se necessario.
   Può essere fatto in una sola passata.
2. Procedendo da X_1 a X_n, si assegnano i valori alle variabili, *senza dover fare backtracking* (ogni valore per un padre ha almeno un valore legale per il figlio)

- Complessità: $O(nd^2)$, lineare in n
  - d^2 è il costo di rendere consistente un arco

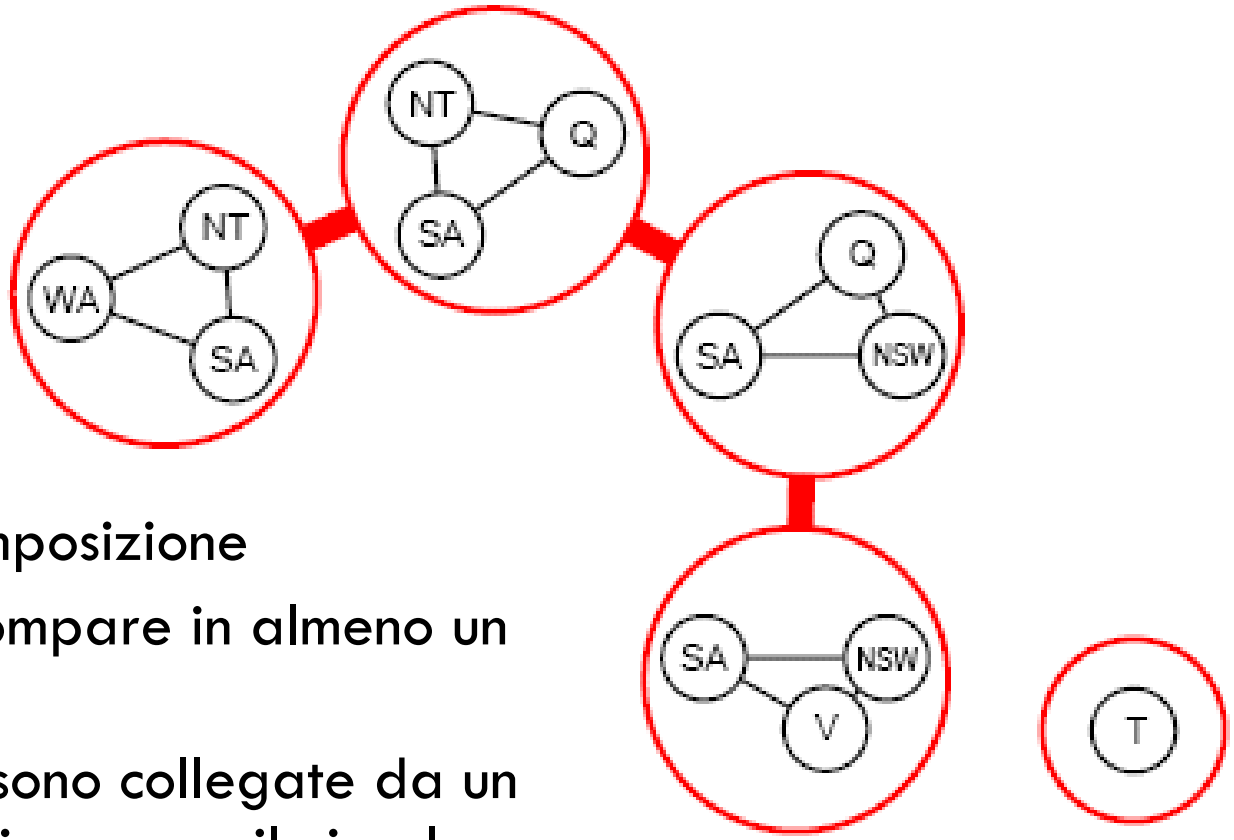
Riduzione ad albero





- Es. Assegnare SA, e ridurre i domini delle variabili collegate. Provare con diversi valori di SA.
- In generale eliminare un insieme *minimo* S di variabili, fino a ottenere un albero (*insieme di taglio dei cicli*) e provare con tutti gli assegnamenti possibili di S.
- *Condizionamento con insieme di taglio*

Scomposizione ad albero



- Requisiti della scomposizione
  1. ogni variabile compare in almeno un sottoproblema
  2. se due variabili sono collegate da un vincolo vanno insieme, con il vincolo.
  3. se una variabile compare in due sottoproblemi deve anche comparire nei sottoproblemi sul cammino che le congiunge

Soluzione

- Ogni sotto-problema viene risolto in maniera indipendente (in maniera efficiente)
- Possiamo vedere il problema originario come un *Mega-problema* con la seguente formulazione:
  - *Mega-variabili* in corrispondenza a sotto-problemi, con dominio le soluzioni ai sottoproblemi

    Es. Dom(X1) ={[WA=r, SA=b, NT=g] ...} 6 sol.

  Vincoli: i valori assegnati alle variabili nei diversi sotto-problemi devono essere gli stessi

Conclusione

- Abbiamo visto come iniziando a “guardare dentro” lo stato si possono migliorare le strategie
- La classe dei problemi di ricerca CSP si presta ad ottimizzazioni *ad hoc* ed è molto vasta
- Prossimamente:
  - rappresentazioni dello stato più ricche
  - sistemi basati su “conoscenza”