Università di Parma
Dipartimento di Ingegneria e Architettura

Introduzione all'Intelligenza Artificiale

Big Data & Business Intelligence

A.A. 2022/2023

Corso di «Introduzione all'Intelligenza Artificiale»
Corso di «Big Data & Business Intelligence»

Agenti Intelligenti

Monica Mordonini (monica.mordonini@unipr.it)

PROBLEM SOLVING AGENTS

ALGORITMI DI RICERCA LOCALE E PROBLEMI DI OTTIMIZZAZIONE

Ricerca locale e problemi di ottimizzazione (Algoritmi di Miglioramento Iterativo)

- Gli algoritmi analizzati fin qui sono progettati per esplorare sistematicamente gli spazi di ricerca.
- Ciò si ottiene memorizzando uno o più cammini e registrando quali alternative sono state esplorate.
- Quando viene raggiunto uno stato obiettivo il *cammino* verso quel stato costituisce una *soluzione* del problema.
- ***Per molti problemi il cammino verso l'obiettivo è irrilevante, ciò che conta è la configurazione finale e non l'ordine con cui è stata raggiunta***
- Si possono considerare allora algoritmi di ricerca locale che operano su un singolo stato corrente invece che su cammini multipli

Ricerca locale e problemi di ottimizzazione
(Algoritmi di Miglioramento Iterativo)

- Più precisamente:
  - Gli algoritmi visti fino ad ora **generano** una soluzione ex-novo aggiungendo ad una situazione di partenza delle componenti in un particolare ordine fino a “costruire” la soluzione completa.
  - Gli algoritmi di ricerca locale partono da una soluzione iniziale e iterativamente cercano di rimpiazzare la soluzione corrente con una “migliore” in un intorno della soluzione corrente. In questi casi non interessa la “strada” per raggiungere l’obiettivo.

Ricerca locale e problemi di ottimizzazione (Algoritmi di Miglioramento Iterativo)

- Ci sono dei problemi (es. 8 regine) per cui la descrizione dello stato contiene tutte le informazioni necessarie per la soluzione indipendentemente dal percorso che ha condotto a quello stato.
- In questi casi l'idea generale per la soluzione può essere quella di partire da uno stato iniziale "completo" (es. 8 regine già sulla scacchiera) e poi modificarlo per raggiungere una soluzione (o una soluzione migliore, se esiste).
- Questi metodi fanno parte dei cosiddetti *algoritmi di ottimizzazione* il cui scopo è massimizzare (minimizzare) il valore di una funzione di valutazione che misura la bontà (costo) di una soluzione.

Ricerca locale e problemi di ottimizzazione (Algoritmi di Miglioramento Iterativo)

- La ricerca locale, quindi, è basata sull'esplorazione iterativa di "soluzioni vicine", che possono migliorare la soluzione corrente mediante modifiche locali.

- La soluzione trovata da un algoritmo di ricerca locale non è detto sia ottima globalmente, ma può essere ottima rispetto ai cambiamenti locali.

Ricerca locale e problemi di ottimizzzione (Algoritmi di Miglioramento Iterativo)

Struttura dei "vicini" (neighborhood).

- Una struttura dei "vicini" è una funzione *F* che assegna a ogni soluzione *s* dell'insieme di soluzioni *S* un insieme di soluzioni *N(s)* sottoinsieme di S.
- La scelta della funzione *F* è fondamentale per l'efficienza del sistema e definisce l'insieme delle soluzioni che possono essere raggiunte da s in un singolo passo di ricerca dell'algoritmo.
- Tipicamente è definita implicitamente attraverso le possibili mosse.

Ricerca locale e problemi di ottimizzazione (Algoritmi di Miglioramento Iterativo)

- Questi algoritmi partono con una soluzione di tentativo generata in modo casuale o mediante qualche algoritmo non ottimo e non completo ma a complessità computazionale modesta e cerca di migliorare la soluzione corrente muovendosi verso i vicini.

- Se fra i vicini c'è una soluzione migliore, tale soluzione va a rimpiazzare la corrente. Altrimenti termina in un massimo (minimo) locale di ricerca.

- Questo metodo può essere rappresentato come il movimento su una superficie che rappresenta lo spazio degli stati alla ricerca della vetta più elevata o più bassa

Ricerca locale e problemi di ottimizzazione (Algoritmi di Miglioramento Iterativo)

- Cercheremo un minimo globale se l'altezza corrisponde ad un costo, un massimo globale se l'altezza corrisponde alla funzione obiettivo

- ✓ Un algoritmo di ricerca locale ***completo*** trova sempre un minimo/massimo

- ✓ Un algoritmo di ricerca locale ***ottimo*** trova sempre un minimo/massimo globale

Ricerca locale e problemi di ottimizzazione (Algoritmi di Miglioramento Iterativo)

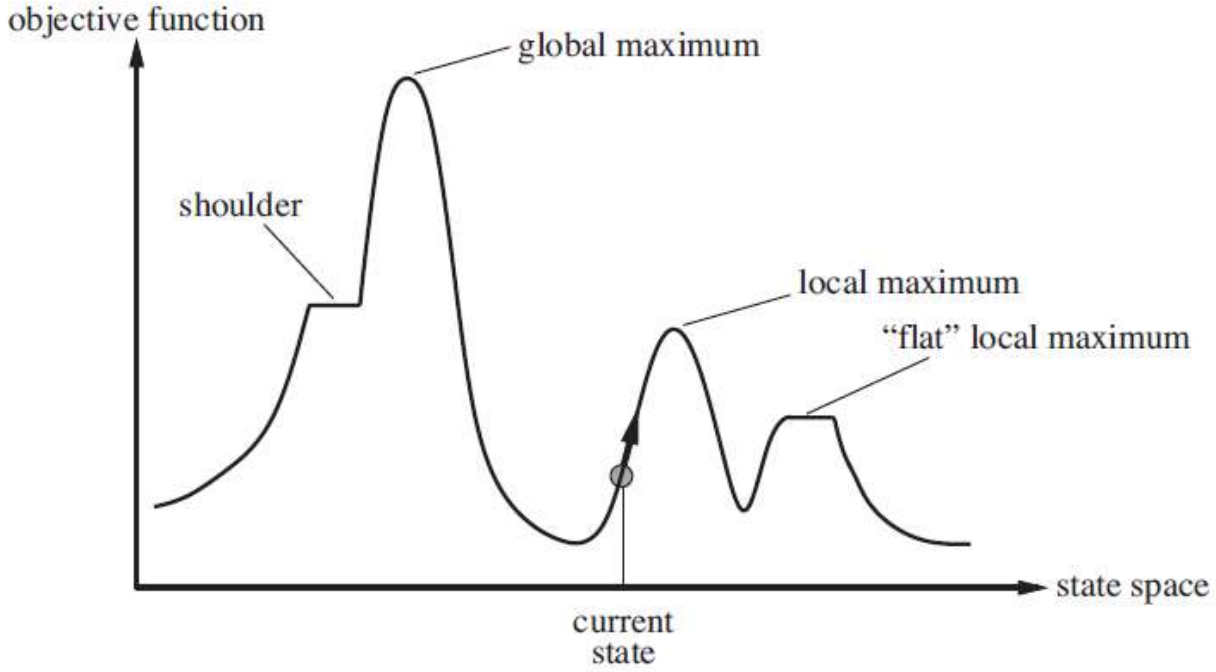


Figure 4.1 A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text.

Ricerca in Salita (Hill Climbing)

- L'algoritmo di ricerca in salita è un algoritmo ciclico che, ad ogni passo, raggiunge una posizione migliore all'interno di un'area di ricerca, se esiste, seguendo la direzione di massima pendenza.
- Possiamo definire l'algoritmo di ricerca in salita come segue:

```
function Hill-Climbing(problem) {
  current = Make-Node(Initial-State(problem));
  while true {
    next = 'a higher valued successor';
    if Value(next) < Value(current)
     return current ;
    current = next;
  }
}
```

Ricerca in Salita: toy problem – 8 regine

- Gli algoritmi di ricerca locale in genere utilizzano una formulazione a stato completo, in cui ogni stato ha 8 regine sul tabellone, una per colonna.
- I successori di uno stato sono tutti i possibili stati generati spostando una singola regina in un altro quadrato nella stessa colonna (quindi ogni stato ha 8×7=56 successori).
- La funzione di costo euristica h è il numero di coppie di regine che si attaccano a vicenda, direttamente o indirettamente.
- Il minimo globale di questa funzione è zero, che si verifica solo a soluzioni perfette.

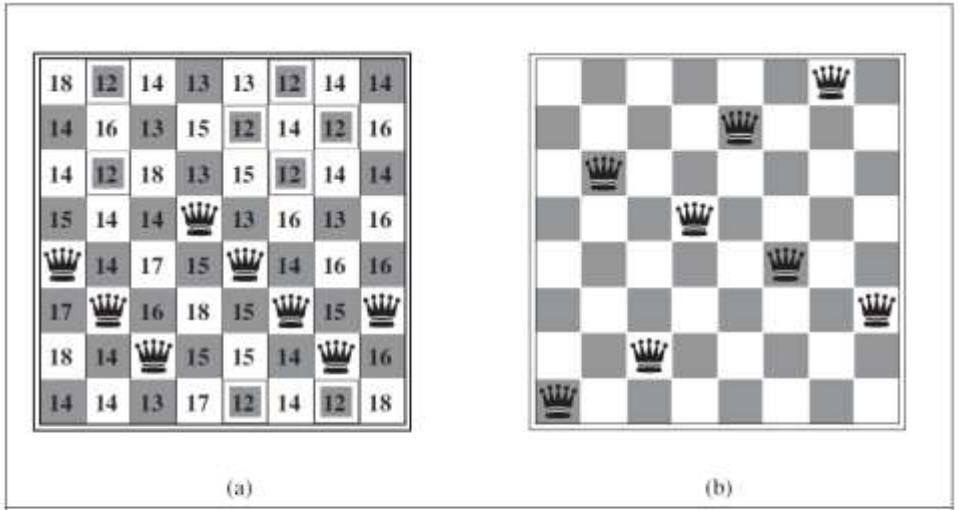


Figure 4.3 (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked. (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost.

Ricerca in Salita: toy problem – 8 regine

- La figura mostra uno stato con h=17
- La figura mostra anche i valori di tutti i suoi successori, con i migliori successori che hanno h=12.
- Gli algoritmi di arrampicata in collina in genere scelgono casualmente tra l'insieme dei migliori successori se ce n'è più di uno.

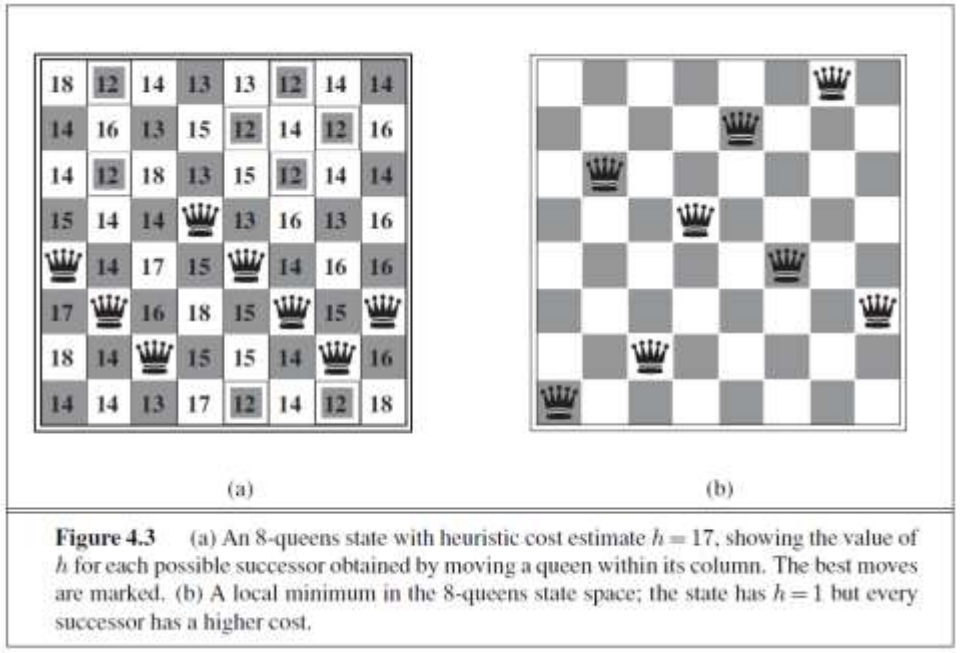


Figure 4.3 (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked. (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost.

Ricerca in Salita

- Si noti che l'algoritmo non tiene traccia dell'albero di ricerca.
- Hill climbing is sometimes called **greedy local search** because it grabs a good neighbor state without thinking ahead about where to go next.

- Questo algoritmo deve affrontare tre tipi di situazioni che lo possono "bloccare".
  Nella rappresentazione dello spazio di ricerca come una superficie, corrispondono al raggiungimento di:
  - massimi locali
  - pianori
  - crinali

Ricerca in Salita

- Il metodo per superare questi problemi è fare ripartire la ricerca da un altro punto scelto a caso
- Questo è ciò che fa una variante dell'algoritmo di ricerca in salita (*Random-restart-hill-climbing*).
- Questa variante ad ogni iterazione mantiene in memoria il risultato migliore e, periodicamente o al verificarsi di condizioni di "blocco", riparte con una nuova iterazione da uno stato scelto in modo random.

Ricerca in Salita

- Dopo un certo numero fisso di iterazioni o dopo che per un certo numero di iterazioni questo risultato non viene migliorato, l'algoritmo termina ritornando il risultato migliore.

- In genere una buona soluzione può essere trovata dopo poche iterazioni.

- Se il problema è NP-completo (generalmente un problema intrattabile), la soluzione non può che essere fornita in un tempo esponenziale poiché, per tale classe di problemi, lo spazio degli stati ha un numero esponenziale di massimi locali.

Simulated Annealing

- L'algoritmo di simulated annealing ha un comportamento molto simile all'algoritmo di ricerca in salita.
- La differenza consiste nell'eseguire, ad ogni passo, una mossa casuale.
- Se la mossa migliora la situazione, allora viene sempre eseguita, altrimenti viene eseguita con probabilità e $^{\Delta E/T}$ dove:
  - ΔE è il peggioramento (<0) causato dall'esecuzione della mossa.
  - T è un valore che tende a zero con l'aumentare dei cicli e quindi rende sempre meno probabile l'esecuzione di mosse negative.

Simulated Annealing

- cambiamo il nostro punto di vista dalla ricerca in salita alla ricerca minimizzando il gradiente di discesa discesa in pendenza e immaginiamo il compito di far entrare una pallina da ping-pong nella fessura più profonda di una superficie accidentata.
- Se lasciamo rotolare la palla, si fermerà al minimo locale.Se scuotiamo la superficie, possiamo far rimbalzare la palla fuori dal minimo locale.Il trucco è scuotere abbastanza forte da far rimbalzare la palla fuori dai minimi locali ma non abbastanza da rimuoverla dal minimo globale.
- L'algoritmo di simulated-annealing consiste nell'iniziare agitando forte (cioè ad alta temperatura) e quindi ridurre gradualmente l'intensità dell'agitazione (cioè abbassare la temperatura).

Simulated Annealing

- L’algoritmo è così chiamato perché simula l’annealing, cioè il processo *graduale* di raffreddamento di una sostanza che passa dallo stato liquido allo stato solido.

- Se la temperatura viene diminuita abbastanza lentamente, allora il liquido raggiunge una configurazione a minima energia.

Simulated Annealing

- Possiamo definire l'algoritmo di simulated annealing come segue:

```
function Simulated-Annealing(problem) {
  current = Make-Node(Initial-State(problem));
  for t = 1 to ∞ {
    T = schedule[t];     schedule = mappatura tempo/temperatura
    if (T = = 0)
     return current;
    next = 'a randomly selected successor';
    ΔE = Value(next) - Value(current);
    if (ΔE > 0)
     current = next;
    else 'current = next with probability e^(ΔE/T)'
  }
}
```

Simulated Annealing

- In pratica cerca di evitare il problema dei massimi locali, seguendo la strategia in salita, ma ogni tanto fa un passo che non porta a un incremento in salita.
- Quando rimaniamo bloccati in un massimo locale, invece di cominciare di nuovo casualmente potremmo permettere alla ricerca di fare alcuni passi in discesa per evitare un massimo locale.
- Ci suggerisce quindi di esaminare, ogni tanto nella ricerca un nodo anche se sembra lontano dalla soluzione.
- Esempio di **meta-euristiche**

Meta-Euristiche

- Si definiscono ***meta-euristiche*** l'insieme di algoritmi, tecniche e studi relativi all'applicazione di criteri euristici per risolvere problemi di ottimizzazione (*quindi migliorando la ricerca locale con criteri abbastanza generali*).

META-CONOSCENZA

- **META-CONOSCENZA = CONOSCENZA SULLA CONOSCENZA**

- Può risolvere i problemi precedentemente citati (vasta applicazione non solo per il controllo).
- ***Usa delle meta-regola:***
  - **Es:** "Usa regole che esprimono situazioni di allarme prima di regole che esprimono situazioni di funzionamento normale".
- ESEMPI: Sistemi Esperti
- VANTAGGI:
  - il sistema è maggiormente flessibile e modificabile; un cambiamento nella strategia di controllo implica semplicemente il cambiamento di alcune meta-regole.
  - la strategia di controllo è semplice da capire e descrivere.
  - potenti meccanismi di spiegazione del proprio comportamento.

Meta-Euristiche : altri esempi

- **ANT colony optimization:**
  - ispirata al comportamento di colonie di insetti. Tali insetti mostrano capacità globali nel trovare, ad esempio, il cammino migliore per arrivare al cibo dal formicaio (algoritmi di ricerca cooperativi)
- **Tabu search**
  - La sua caratteristica principale è l'utilizzo di una memoria per guidare il processo di ricerca in modo da evitare la ripetizione di stati già esplorati.
- **Algoritmi genetici**
  - algoritmi evolutivi ispirati ai modelli dell'evoluzione delle specie in natura. Utilizzano il principio della selezione naturale che favorisce gli individui di una popolazione che sono più adatti aduno specifico ambiente per sopravvivere e riprodursi.
  - Ogni individuo rappresenta una soluzione con il corrispondente valore della funzione di valutazione (*fitness*). I tre principali operatori sono: *selezione*, *mutazione* e *ricombinazione*.

Meta-Euristiche : altri esempi


```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
      new_population ← empty set
      for i = 1 to SIZE(population) do
          x ← RANDOM-SELECTION(population, FITNESS-FN)
          y ← RANDOM-SELECTION(population, FITNESS-FN)
          child ← REPRODUCE(x, y)
          if (small random probability) then child ← MUTATE(child)
          add child to new_population
      population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

```
function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

Meta-Euristiche: Altre tecniche euristiche

Scelta in base agli operatori

(regole, task inseriti in un'agenda):

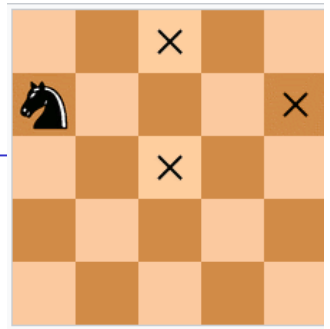
a) Insita nel controllo e influenzata IMPLICITAMENTE dall'utente:

b) Influenzata ESPLICITAMENTE dall'utente:

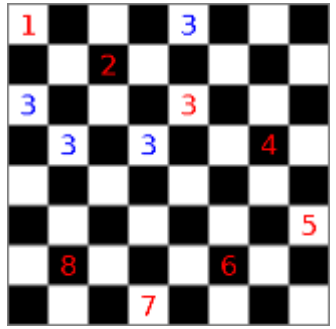
Attribuendo alle regole dei valori di priorità (visti come numeri interi; in questo modo l'interprete selezionerà sempre la regola applicabile a più alta priorità.

- lo stato della memoria di lavoro;
- la presenza di particolari regole applicabili;
- la precedente esecuzione di regole correlate;
- l'andamento dell'esecuzione per particolari scelte fatte precedentemente.

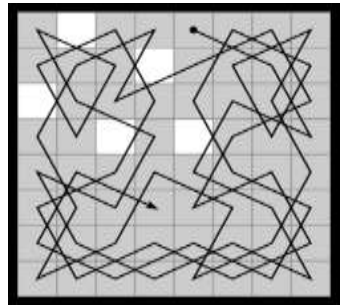
Knight's tour



- È una sequenza di mosse di un cavallo su una scacchiera in modo tale che il cavaliere visiti ogni casella solo una volta.
  - Il cavallo può spostarsi dalla posizione i alla posizione i + 1 se lo spostamento include un movimento di due righe e una colonna o viceversa.



- knight's tour problem è un'istanza (un toy problem) del più generale problema del percorso hamiltoniano nella teoria dei grafi
  - A differenza del problema generale del percorso Hamiltoniano, il problema del percorso del cavallo può essere risolto in tempo lineare.

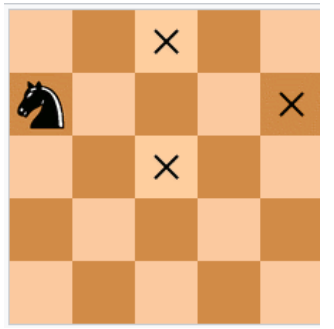


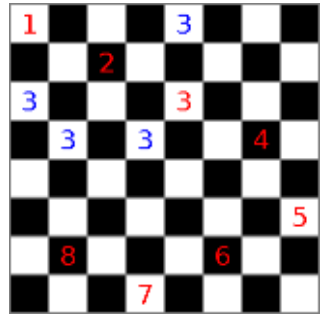
Knight's tour

- Il problema ha una soluzione, ma una ricerca con la forza bruta per esso è impraticabile su tutte le scacchiere tranne quelle più piccole, perché dopo pochi passi finirà ripetutamente in vicoli ciechi.

  Nella figura, alla 36a mossa il cavallo finisce in a1 dove non ci sono mosse disponibili

1							
		2					
	32			3			
34			27			4	
31	10	33			26	23	
18	35	28	11	24	21	14	5
9	30	19	16	7	12	25	22
36	17	8	29	20	15	6	13

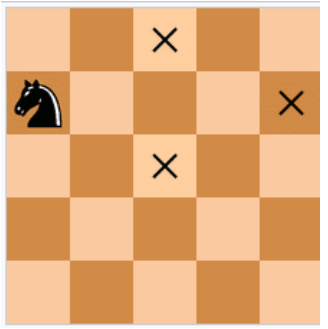


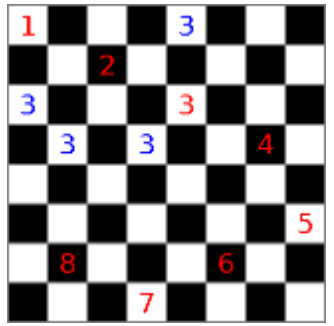


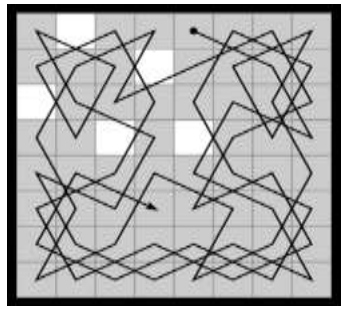
We need heuristics

Knight's tour: heuristics

- Divide and conquer algorithms
  - Dividendo la scacchiera in pezzi più piccoli, costruendo tour su ogni pezzo e rattoppando i pezzi insieme
- Neural network solutions
  - La rete è impostata in modo tale che ogni mossa legale del cavallo cavaliere sia rappresentata da un neurone e ogni neurone viene inizializzato casualmente per essere "attivo" o "inattivo" (output di 1 o 0), con 1 che implica che il neurone fa parte di la soluzione finale.
- *Warnsdorf's rule*
  - Il cavallo viene spostato in modo che proceda sempre nella casella da cui il cavallo avrà il minor numero di mosse in avanti.
  - *ad ogni passo il cavallo deve scegliere la cella più isolata, o più vicina al bordo, prima che diventi troppo isolata...*







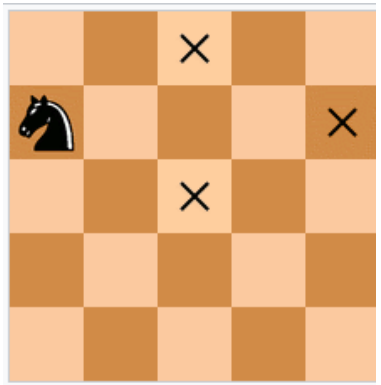
Knight's tour:
Warnsdorf's rule

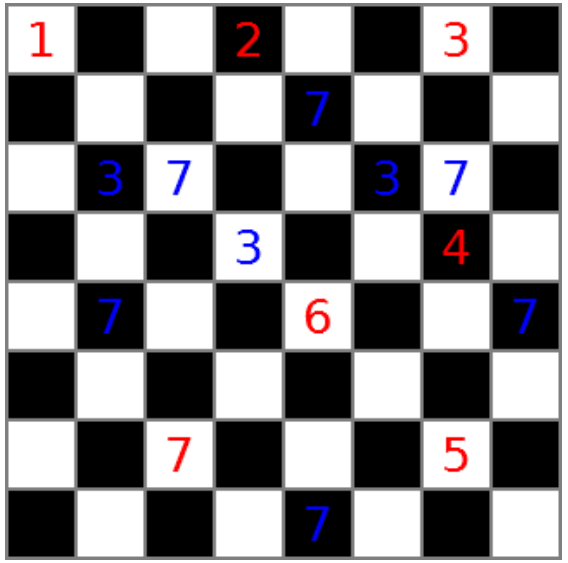
```python
#!/usr/bin/env python3
'''
@author Michele Tomaiuolo - http://www.ce.unipr.it/people/tomamic
@license This software is free - http://www.gnu.org/licenses/gpl.html
'''

import sys
sys.setrecursionlimit(15000)


delta = [(1, -2), (2, -1), (2, 1), (1, 2),
         (-1, 2), (-2, 1), (-2, -1), (-1, -2)]


def find_moves(board, x0, y0) -> list:
    w, h = len(board[0]), len(board)
    moves = []
    for dx, dy in delta:
        x, y = x0 + dx, y0 + dy
        if (0 <= x < w and 0 <= y < h and board[y][x] == 0):
            moves.append((x, y))
    return moves


def border_distance(x, y, w, h):
    dist_x = min(x, w - x)
    dist_y = min(y, h - y)
    # return dist_x + dist_y
    return sorted((dist_x, dist_y))
```

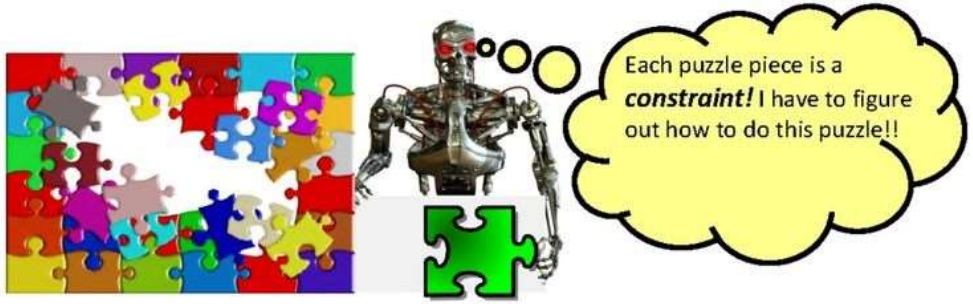
```python
def knight_tour(board, x, y) -> bool:
    w, h = len(board[0]), len(board)
    n = board[y][x]
    if n == w * h: return True

    moves = find_moves(board, x, y)
##    moves.sort(key=lambda m: border_distance(m[0], m[1], w, h))
    moves.sort(key=lambda m: len(find_moves(board, m[0], m[1])))

    # try each possible move (recursively)
    for xm, ym in moves:
        board[ym][xm] = n + 1

        if knight_tour(board, xm, ym): return True

        # no luck this way, go back (backtracking)
        board[ym][xm] = 0
    return False


def main():
    w = h = int(input('size? '))
    board = [[0 for x in range(w)] for y in range(h)]
    board[0][0] = 1

    print(knight_tour(board, 0, 0))

    for y in range(h):
        for x in range(w):
            print(("{:5}").format(board[y][x]), end='')
        print()

if __name__ == '__main__':
    main()
```

Knight's tour Exercise

- Il cavaliere deve visitare l'intera scacchiera (di qualsiasi dimensione...) e può spostarsi dalla posizione i alla posizione i+1 se la mossa prevede un salto di due linee / due colonne / due linee e due colonne.







PROBLEM SOLVING AGENTS

PROBLEMI CON VINCOLI - CONSTRAINT SATISFACTION PROBLEMS (CPS)

Problemi con vincoli

- Fino adesso sappiamo che i problemi possono essere risolti cercando uno spazio di stati, i quali possono essere valutati dall'euristica specifica del dominio e testati per vedere se sono il goal.
- Aal punto di vista dell'algoritmo di ricerca ogni stato è atomico e indivisibile: una scatola nera senza struttura interna.

- Problemi con vincoli (CSP-Constraint Satisfction Problems) cercano di trovare soluzioni più efficienti andando a incrementare la definizione di un singolo stato
- Esistono euristiche generali che si applicano e che consentono la risoluzione di problemi di dimensioni significative per questa classe

Problemi con vincoli

- Usiamo una rappresentazione fattorizzata per ogni stato: un insieme di variabili, ognuna delle quali ha un valore.
- Un problema è risolto quando ogni variabile ha un valore che soddisfa tutti i vincoli sulla variabile.
- Se gli algoritmi di ricerca sfruttano la struttura dello spazio degli stati e utilizzano una euristica più o meno generica per consentire la soluzione di problemi complessi.
- Qui l'idea è quella di eliminare in una volta grandi porzioni dello spazio di ricerca identificando le combinazioni di variabile/valore che violano i vincoli.L
- classe dei problemi così formulata è piuttosto ampia: layout dei circuiti, schedulazione, ...

Problemi con vincoli: Formulazione di problemi CSP

- Problema: descritto da tre componenti
  1. 1.X= {X1 X2 … Xn} insieme di variabili
  2. 2.D= {D1 D2 … Dn} insieme di domini
     dove Di={v1, …, vk} è l' insieme dei valori possibili per Xi
  1. 3.C= {C1 C2 … Cm} insieme di vincoli (relazioni tra le variabili)

- Stato: un assegnamento [parziale|completo] di valori a variabili

  {Xi = vi , Xj = vj …}
- 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)

Problemi con vincoli: come esprimere i vincoli

- Forma generale:

  <*ambito, relazione*>

  Ambito: tupla di variabili che partecipano nel vincolo

  Relazione: un insieme di tuple di valori

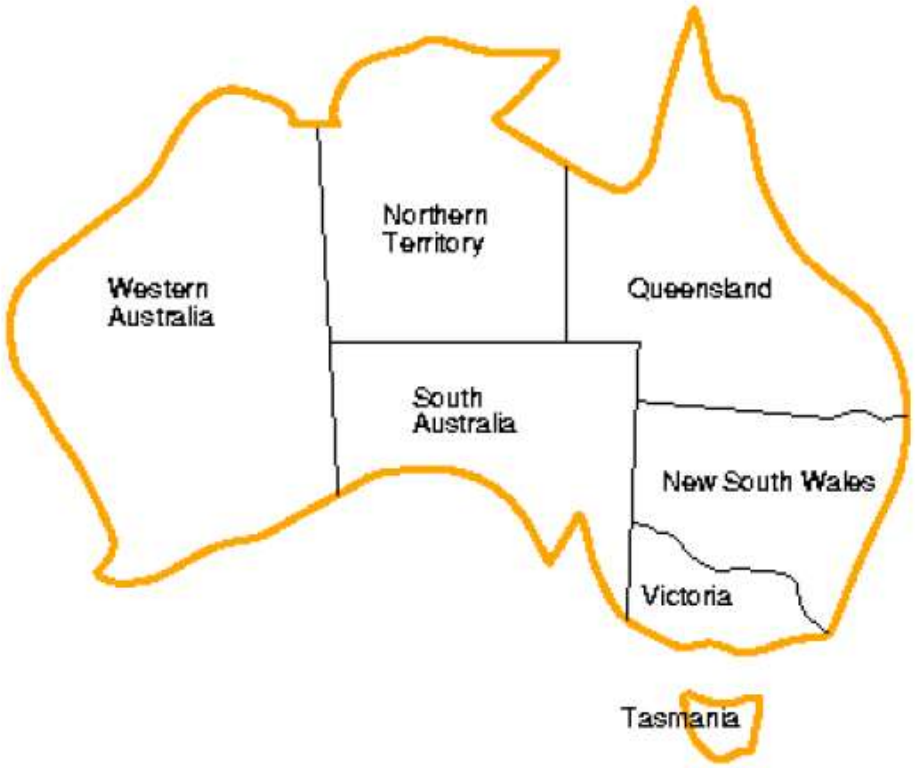
- Esempio: $X=\{X_1, X_2\}$ $D_1=D_2=\{A, B\}$

  Vincolo che le variabili devono avere valori diverse:

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

Problemi con vincoli: esempio colorare 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

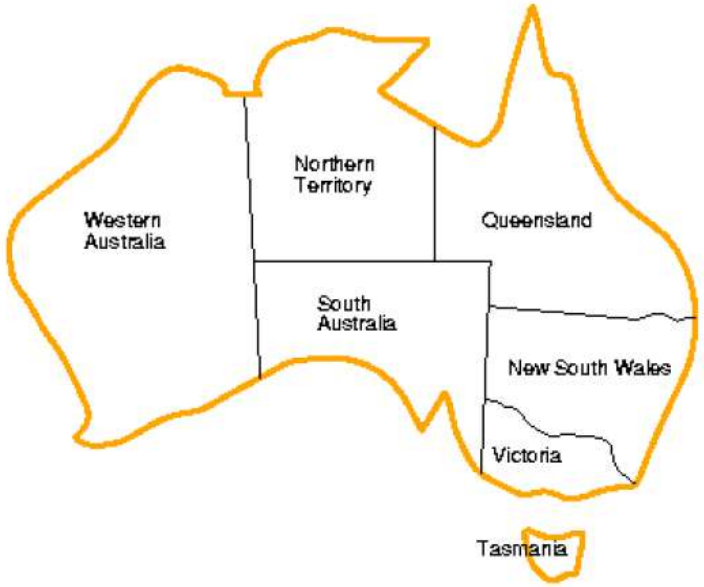
$C = \{WA \neq NT, WA \neq SA, NT \neq Q, NT \neq SA, SA \neq Q, SA \neq NSW, SA \neq V, NSW \neq V\}$

Problemi con vincoli: esempio colorare una mappa

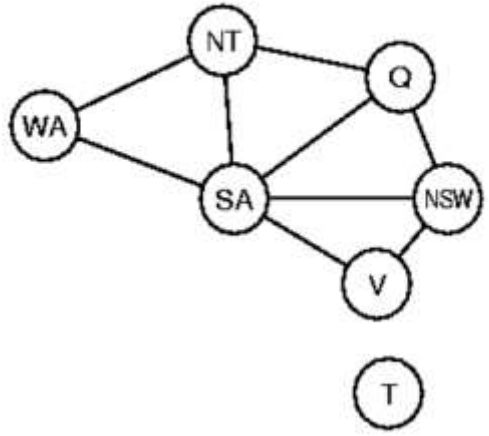
Può essere utile visualizzare un CSP come grafo di vincoli.

I nodi del grafo corrispondono a variabili del problema

Un collegamento collega due variabili qualsiasi che partecipano a un vincolo.



Grafo dei vincoli



Problemi con vincoli

- Perché formulare un problema come CSP?
- Uno dei motivi è che i CSP forniscono una rappresentazione naturale per un'ampia varietà di problemi;
- Se si dispone già di un algoritmo efficiente di risoluzione per un generico problema CSP, è più facile risolvere un problema utilizzando questo che progettare una soluzione personalizzata utilizzando un'altra tecnica di ricerca.
- Inoltre, i risolutori CSP possono essere più veloci di una ricerca nello spazio degli stati perché il risolutore CSP può eliminare rapidamente grandi campioni dello spazio di ricerca.

Problemi con vincoli

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

Problemi con vincoli

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.

Problemi con vincoli

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

  $$|D1| \times |D2| \times \ldots |Dn|$$

  - dove $|Di|$ è la cardinalità del dominio di Xi
- Il fattore di diramazione
  - Teoricamente pari a *nd al primo passo; (n-1) d al secondo ... le foglie sarebbero n! . dn*
  - 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 (dn foglie)*

Problemi con vincoli

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

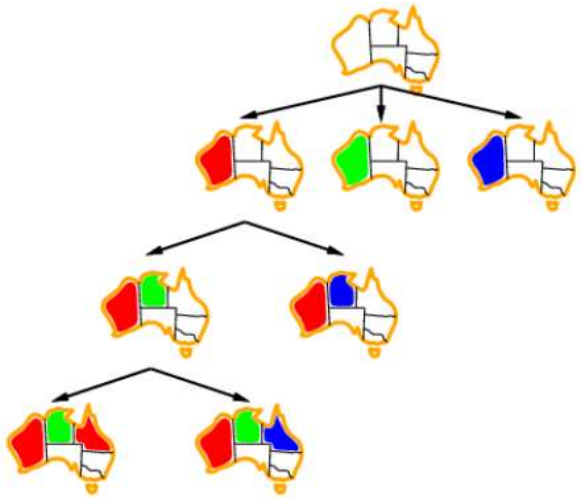
Problemi con vincoli

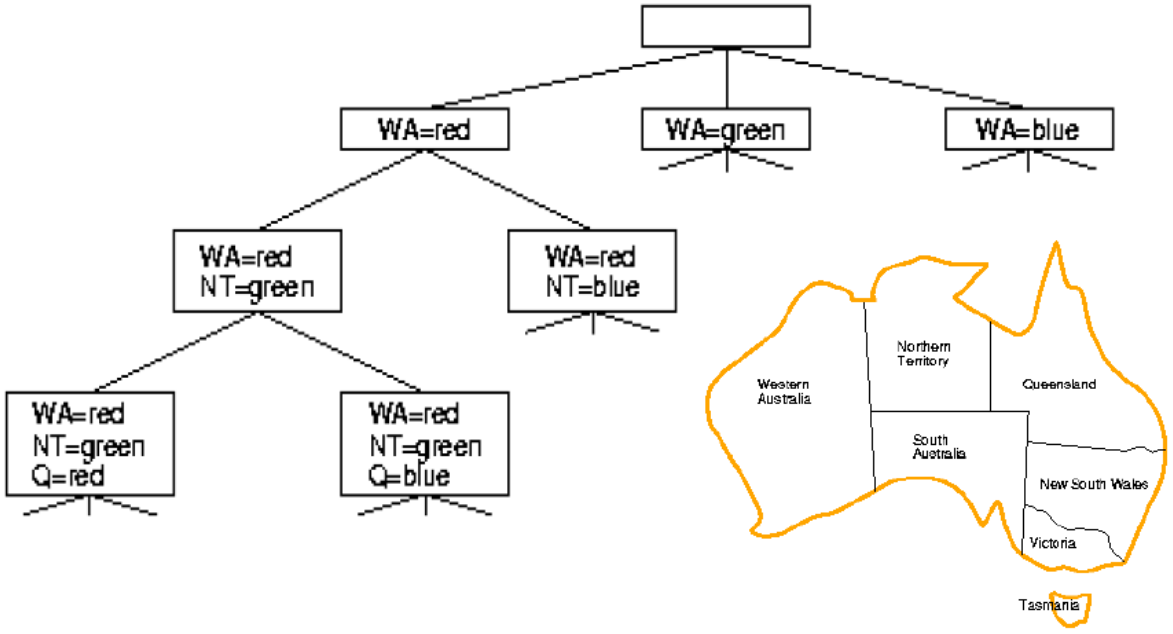
Strategie di ricerca

- Nel backtracking si ha il ***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.

Problemi con vincoli

Strategie di ricerca: esempio di backtracking





Problemi con vincoli

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
```

Problemi con vincoli

Problemi nell'suo del backtracking

- Trashing: continua a ripetere le stesse assegnazioni di variabili fallite
- Inefficienza: può esplorare aree dello spazio di ricerca che probabilmente non avranno successo

Problemi con vincoli

Improving backtracking efficiency: euristiche e strategie

- 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*

Problemi con vincoli

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

Problemi con vincoli

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

Problemi con vincoli

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
```

Problemi con vincoli

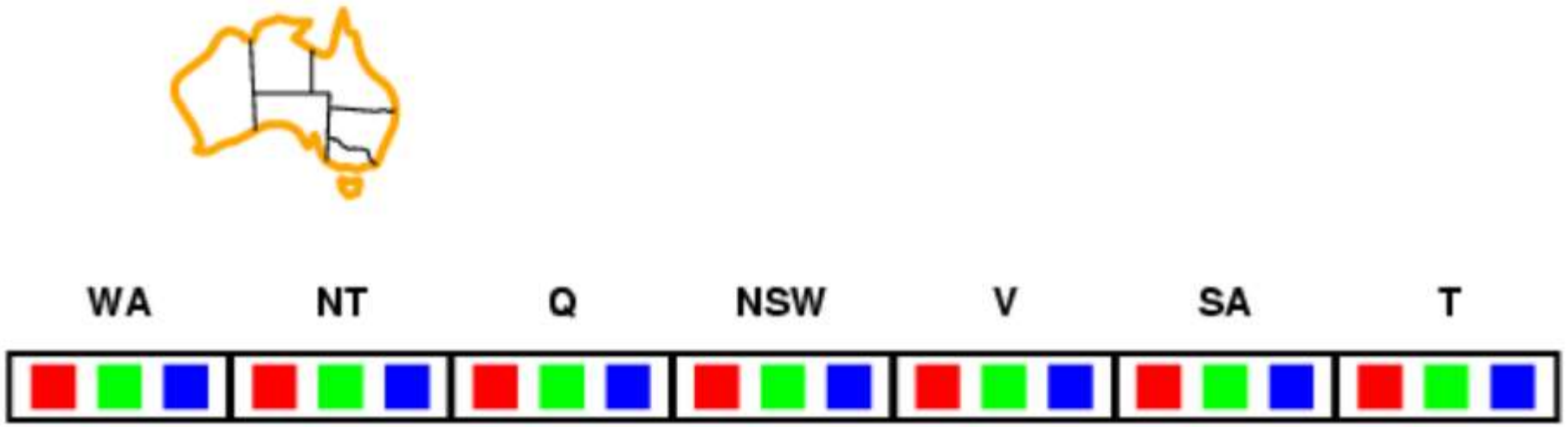
Propagazione di vincoli

- 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)
- 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)

Problemi con vincoli

Forward checking

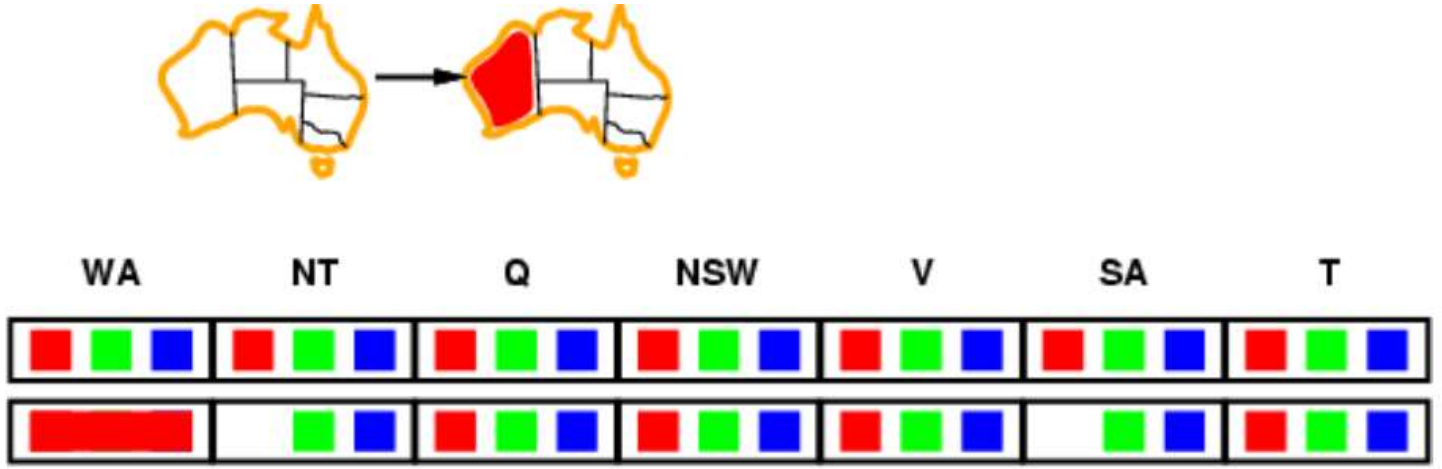
- ✓ Tieni traccia dei valori legali rimanenti per le variabili non assegnate
- ✓ Termina la ricerca quando una variabile non ha valori legali



Problemi con vincoli

Forward checking

- ✓ Tieni traccia dei valori legali rimanenti per le variabili non assegnate
- ✓ Termina la ricerca quando una variabile non ha valori legali



Problemi con vincoli

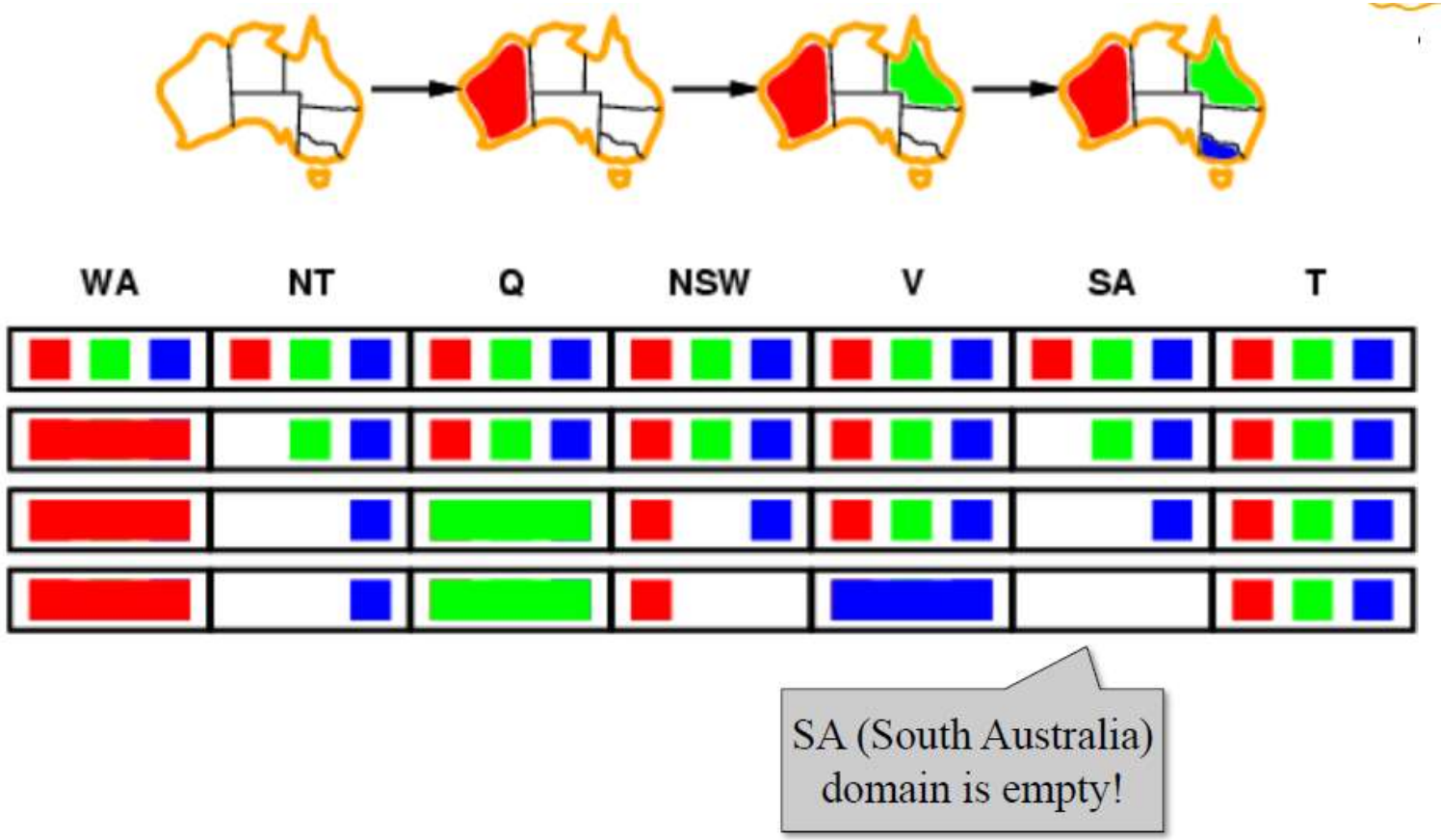
Forward checking

- ✓ Tieni traccia dei valori legali rimanenti per le variabili non assegnate
- ✓ Termina la ricerca quando una variabile non ha valori legali

Problemi con vincoli

Forward checking

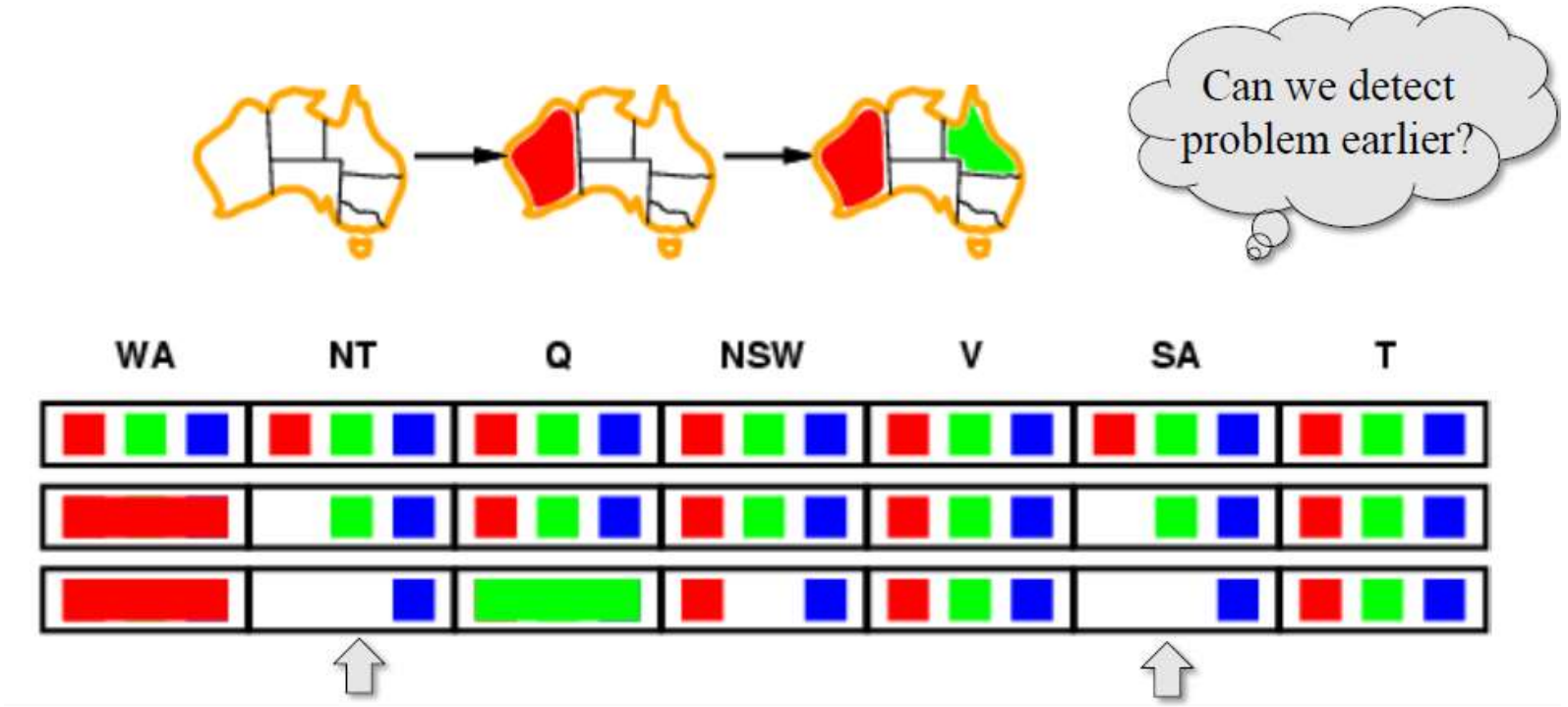
- ✓ Tieni traccia dei valori legali rimanenti per le variabili non assegnate
- ✓ Termina la ricerca quando una variabile non ha valori legali



Problemi con vincoli

Forward checking e constraint propagation

- Forward checking propaga le informazioni da variabili assegnate a variabili non assegnate, ma non fornisce il rilevamento anticipato per tutti gli errori
- NT e SA non possono essere blu!



Problemi con vincoli

Propagazione di vincoli

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

Problemi con vincoli

Propagazione di vincoli

Consistenza d'arco

- Nel grafo a 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*.

Problemi con vincoli

Propagazione di vincoli

Consistenza d'arco

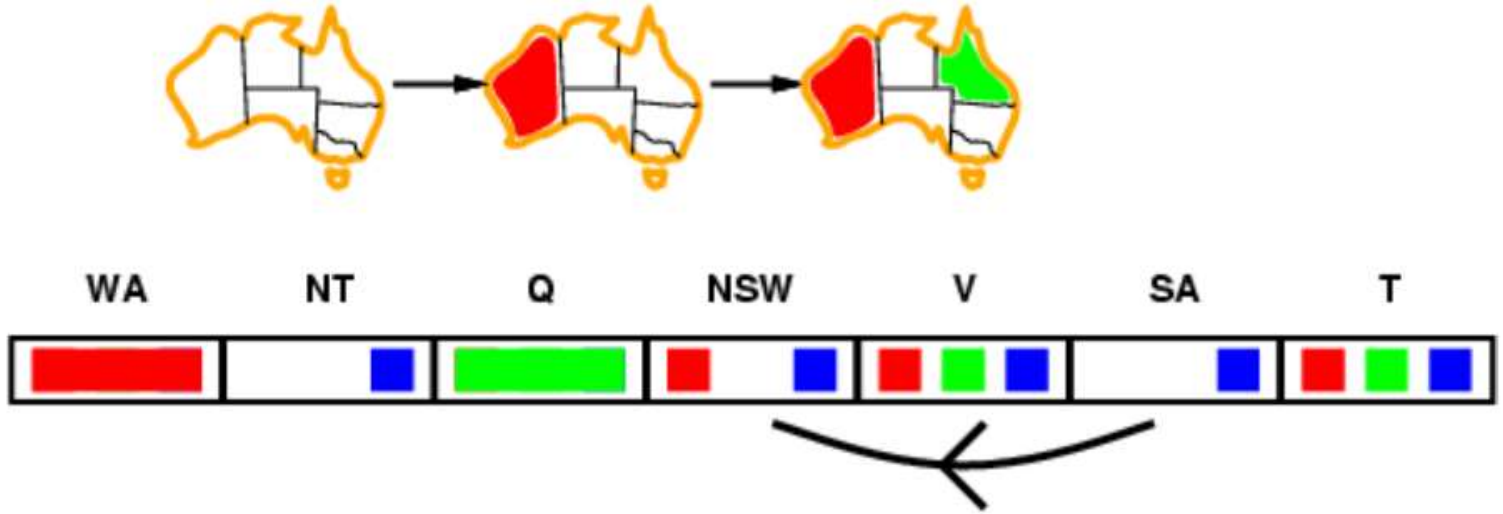
- Nel grafo di vincoli, un arco tra X e Y, (X, Y), è *consistente* rispetto a X, se per ogni valore *x* di X c'è almeno un valore *y* di Y consistente con *x*.
- Si dice anche: X è *arco-consistente* rispetto a Y.
- Se un arco (X, Y) non è consistente rispetto a X (o Y) si cerca di renderlo tale, rimuovendo 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*.

=>La consistenza d'arco viene realizzato tramite l'algoritmo AC-3(by A. Mackworth-1977)

Problemi con vincoli

Consistenza d'arco

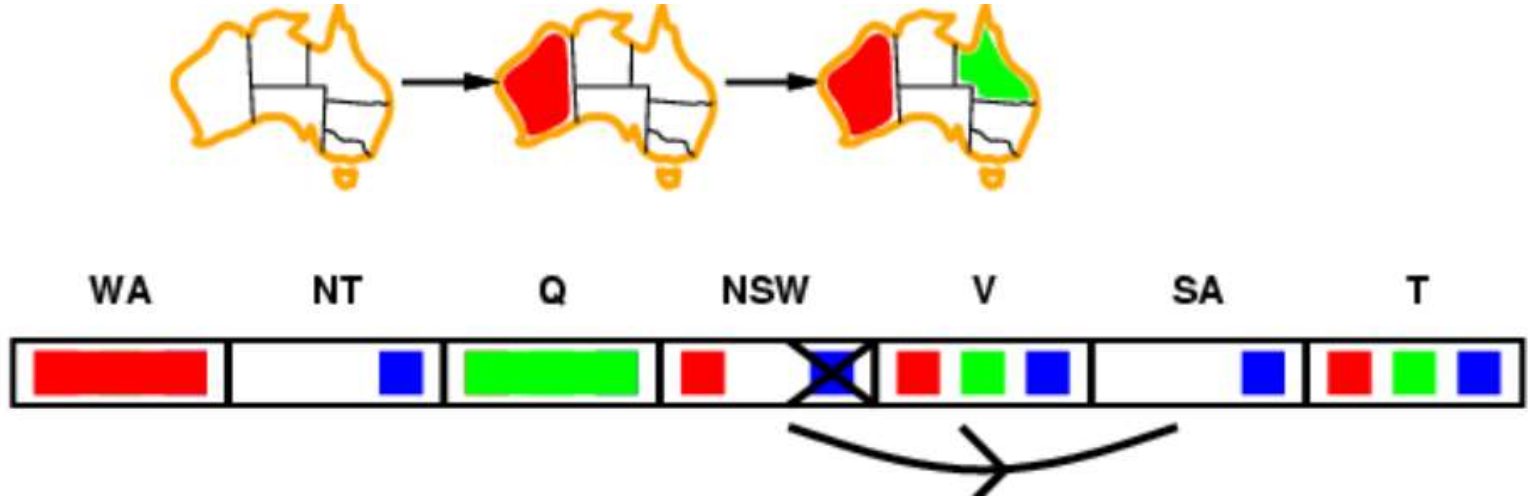
- La forma più semplice di propagazione che rende ogni arco coerente
- X → Y è consistente se e solo s per ogni valore di x_i di X
- c'è un valore consentito y_i in Y



Problemi con vincoli

Consistenza d'arco

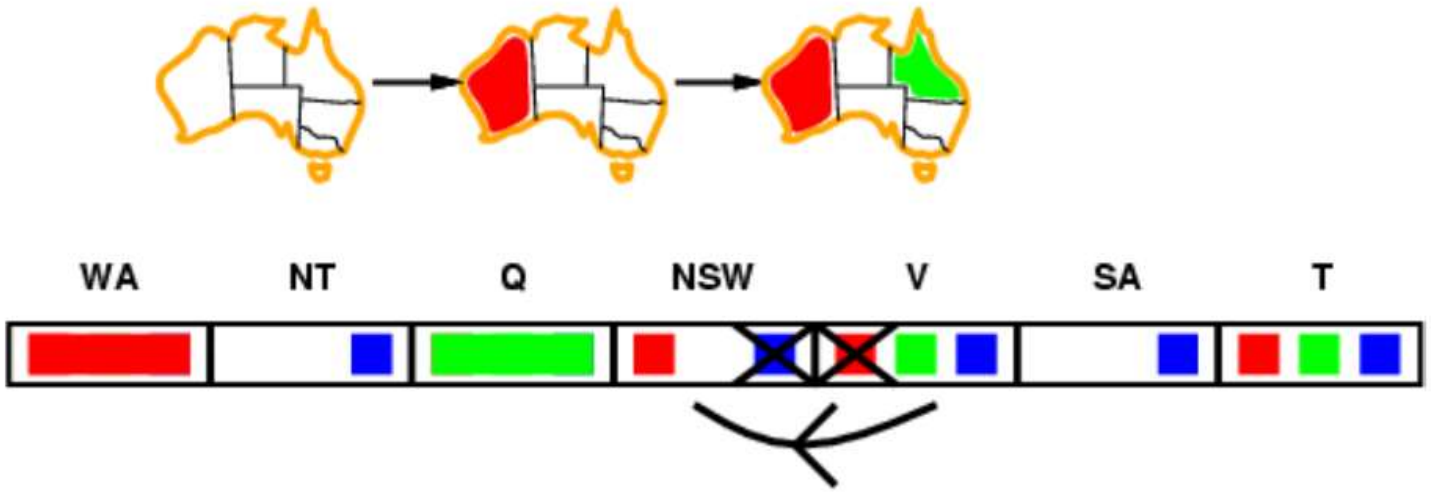
- La forma più semplice di propagazione che rende ogni arco coerente
- X → Y è consistente se e solo s per ogni valore di x_i di X
- c'è un valore consentito y_j in Y



Problemi con vincoli

Consistenza d'arco

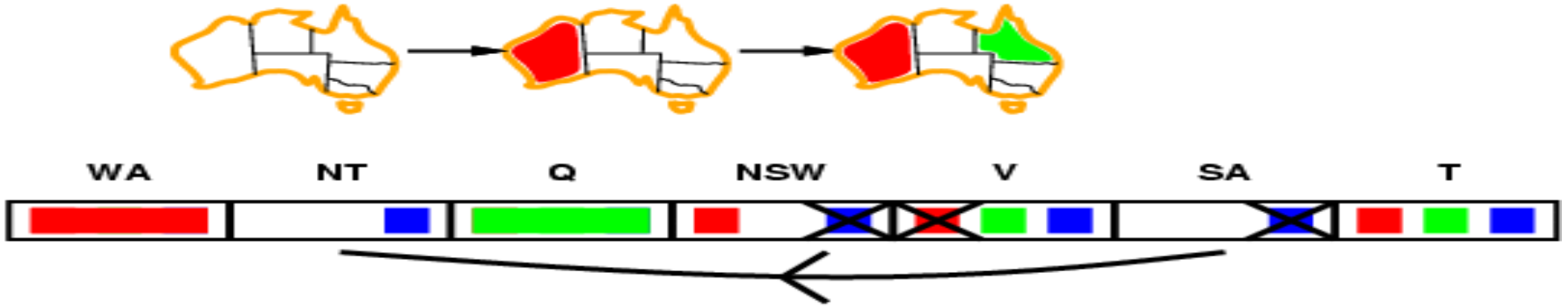
- Se X perde un valore, i vicini di X devono essere ricontrollati



Problemi con vincoli

Consistenza d’arco

- La coerenza dell'arco rileva gli errori prima del semplice forward checking
- WA=red and Q=green viene subito riconosciuto come un vicolo cieco, cioè un'istanza parziale impossibile
- L'algoritmo di consistenza d’arco può essere eseguito sia prima che dopo ogni assegnaziome



Problemi con vincoli

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≠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

Problemi con vincoli

La risoluzione di un CSP richiede ancora la ricerca …

- Ricerca:
  - può trovare buone soluzioni, ma deve esaminare le non-soluzioni lungo il cammino
- Constraint Propagation:
  - può escludere non-soluzioni, ma questo non è lo stesso che trovare soluzioni
- Si possono entrambi -- constraint propagation & search:
  - eseguendo la propagazione del vincolo in ogni fase della ricerca

Problemi con vincoli

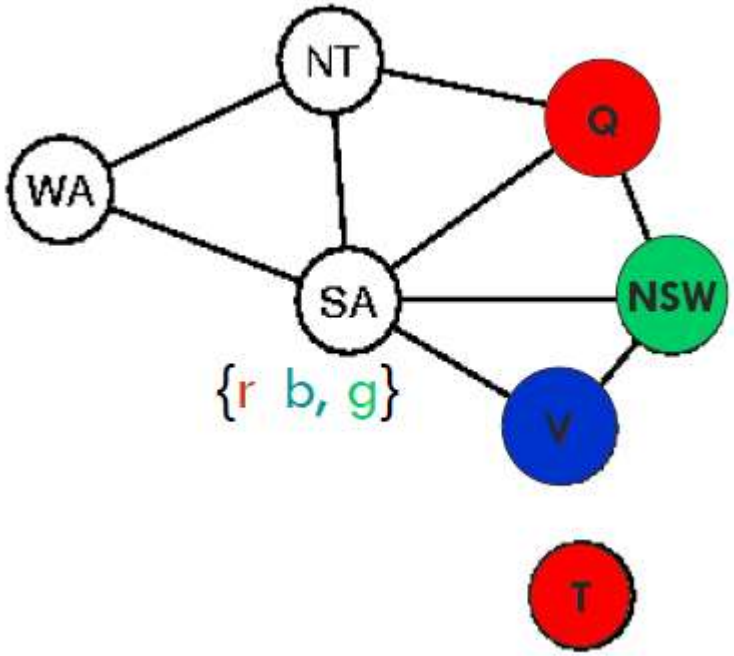
Tecniche euristiche

- Fino ad ora
  - Le variabili vengono assegnate sempre nel loro ordine naturale
  - I valori dei domini vengono assegnati alle variabili nel loro ordine naturale
  - Se una variabile non ha più valori nel proprio dominio si torna a quella immediatamente precedete nell'ordine naturale
- Non è necessario che sia così
  - Le tecniche euristiche spesso forniscono miglioramenti significativi

Problemi con vincoli

Backtracking “cronologico”

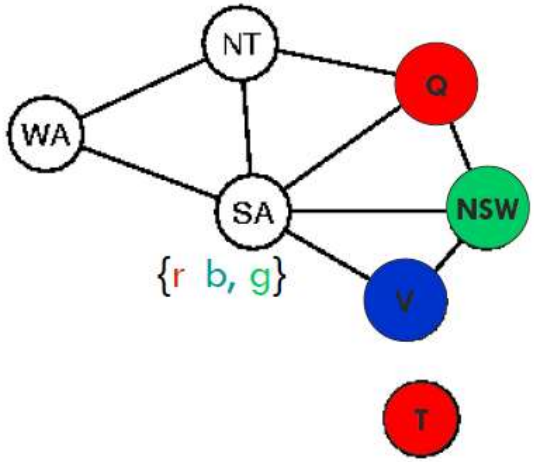
- Supponiamo di avere {Q=red, NSW=green, V=blue, T=red}
- Cerchiamo di assegnare SA
- Il fallimento genera un backtracking “cronologico”
- … e si provano tutti i valori alternativi per l’ultima variabile, T, continuando a fallire



Problemi con vincoli

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***



Problemi con vincoli

- Concludendo si è visto come iniziando a “guardare dentro” lo stato si possono migliorare le strategie
- Si cercano quindi
- rappresentazioni dello stato più ricche che tengano conto delle relazioni fra oggetti
- I cosidetti sistemi basati su “conoscenza”