Arc consistency

- Un grafo di vincoli è **arc consistent** quando rispetta tutti i vincoli binari
- È una proprietà importante perché: dato un vincolo a tre o più variabili è sempre possibile trasformarlo in un insieme di vincoli binari
- **NB**: il forward checking realizza un controllo di arc consistency per i vincoli che legano una variabile, alla quale è stato assegnato un valore, e i suoi vicini diretti sul grafo dei vincoli ma non propaga il ragionamento sui vicini dei vicini, ecc.

Arc consistency

- Dato un grafo di vincoli si intende per **arco** un *lato orientato*
- **Esempio**: WA → NSW sarà diverso da NSW → WA
- Un arco è **consistente** quando:
  per ogni valore possibile della variabile corrispondente al vertice sorgente, esiste un qualche valore del vertice destinazione che è ad esso consistente
- **Esempio**:
  - se il dominio di **WA** è **{R, G}** e il dominio di **NSW** è **{R}**
  - **WA → NSW** non è consistente
    se WA assume valore R, NSW non ha valori assegnabili
  - **NSW → WA** è consistente
- Si può cercare di rendere consistenti gli archi che non lo sono cancellando valori dai loro domini

AC-3

- Algoritmo di arc consistency sviluppato nel 1977 da Alan Macworth[§] (AC-3 = Arc Consistency algorithm #3)
- Viene insegnato perché è più efficiente dei precedenti e più semplice dei successori
- Si può usare come **preprocessing** oppure a valle degli assegnamenti per **propagare le scelte** fatte tramite i vincoli. Quest' ultimo uso realizza l' algoritmo MAC (Maintaining Arc Consistency)

[§] se può essere utile sono accessibili delle slide introduttive ai CSP dello stesso Mackworth: http://www.cs.ubc.ca/~mack/CS322/lectures/3-CSP2.pdf

AC-3

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

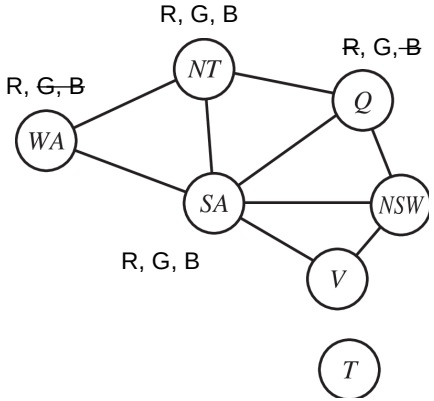
  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if REMOVE-INCONSISTENT-VALUES(X_i, X_j) **then**
      for each X_k **in** NEIGHBORS[X_i] **do**
        add (X_k, X_i) to *queue*

---

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff we remove a value
  removed $\leftarrow$ *false*
  for each x **in** DOMAIN[X_i] **do**
    if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraint between X_i and X_j
      then delete x from DOMAIN[X_i]; *removed* $\leftarrow$ *true*
  return *removed*

Figure 5.7 The arc consistency algorithm AC-3. After applying AC-3, either every arc is arc-consistent, or some variable has an empty domain, indicating that the CSP cannot be made arc-consistent (and thus the CSP cannot be solved). The name "AC-3" was used by the algorithm's inventor (Mackworth, 1977) because it's the third version developed in the paper.

Esempio



VARIABILI: WA, NT, SA, Q, …

DOMINI PER LE VRB NON ASSEGNATE:
{R, G, B}

QUEUE:
~~WA → NT~~ NT → WA
NT → Q Q → NT
WA → SA SA → WA
NT → WA WA → NT
SA → Q Q → SA
… ecc …

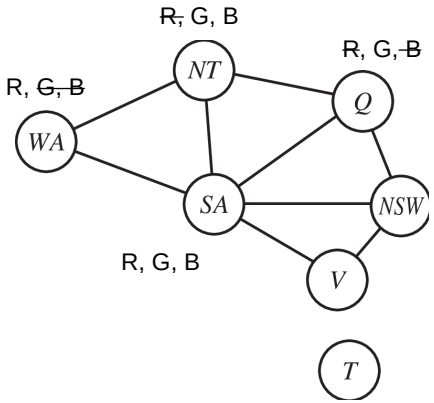
Supponiamo di avere assegnato **WA=R, Q=G**
Gli archi sono consistenti?

Seleziono WA → NT

È arc-consistent (il valore R di WA è consistente con i valori G e B di NT)

Togliamo l'arco da queue

Esempio



QUEUE:
~~WA → NT~~ ~~NT → WA~~
NT → Q Q → NT
WA → SA SA → WA
NT → WA WA → NT
SA → Q Q → SA,
WA → NT, … ecc …

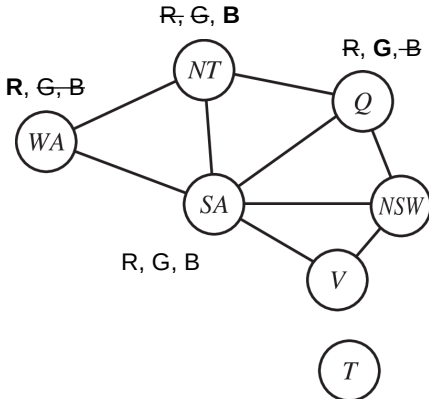
Seleziono NT → WA

Non è arc-consistent:
il valore R di NT è inconsistente con il valore R di WA

Togliamo R dai valori possibili di NT

Togliamo NT → WA da queue ma aggiungiamo gli archi di tipo NODO → NT, quindi:
WA → NT (rientra, era stato rimosso),
SA → NT, Q → NT (ci sono già)

Esempio



QUEUE:
~~WA → NT~~ ~~NT → WA~~
~~NT → Q~~ Q → NT
WA → SA SA → WA
NT → WA WA → NT
SA → Q Q → SA,
WA → NT, … ecc …

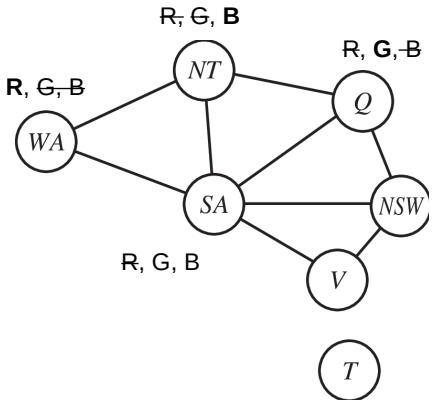
Seleziono NT →Q

Non è arc-consistent:
il valore G di NT è inconsistente con
il valore G di Q

Togliamo G dai valori possibili di NT

Togliamo NT → Q da queue ma aggiungiamo gli archi di tipo NODO → NT. Questi sono però già presenti, li abbiamo aggiunti all'iterazione precedente e non ancora esaminati

Esempio



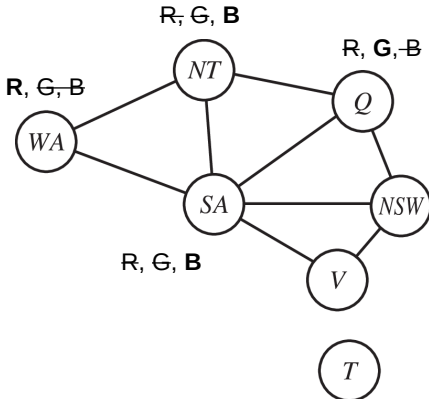
QUEUE:
~~WA → NT~~ ~~NT → WA~~
~~NT → Q~~ ~~Q → NT~~
~~WA → SA~~ ~~SA → WA~~
NT → WA WA → NT
SA → Q Q → SA,
WA → NT **WA → SA**
… ecc …

Seleziono Q →NT: è consistente
Seleziono WA → SA: è consistente
Seleziono SA → WA
Non è arc-consistent:
il valore R di SA è inconsistente con
il valore R di WA

Togliamo R dai valori possibili di SA

Togliamo SA → WA da queue ma aggiungiamo gli archi di tipo NODO → SA. Fra questi vi è WA → SA che era stato rimosso prima

Esempio



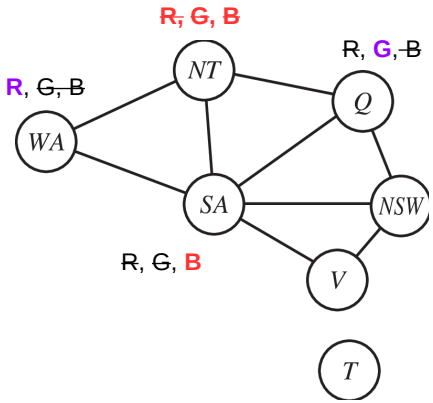
QUEUE:
~~WA → NT~~ ~~NT → WA~~
~~NT → Q~~ ~~Q → NT~~
~~WA → SA~~ ~~SA → WA~~
~~NT → WA~~ ~~WA → NT~~
~~SA → Q~~ Q → SA,
WA → NT WA → SA
NT → SA
… ecc …

Seleziono NT →WA: è consistente
Seleziono WA → NT: è consistente
Seleziono SA → Q
Non è arc-consistent:
il valore G di SA è inconsistente con
il valore G di Q

Togliamo G dai valori possibili di SA

Togliamo SA → Q da queue ma aggiungiamo gli archi di tipo NODO → SA.

Esempio



QUEUE:
...
~~NT → WA~~ ~~WA → NT~~
~~SA → Q~~ ~~Q → SA~~,
~~WA → NT~~ ~~WA → SA~~
NT → SA
… ecc …

Seleziono Q →SA: è consistente
Seleziono WA → NT: è consistente
Seleziono WA → NT: è consistente
Selezioni NT → SA
Non è arc-consistent:
il valore B di NT è inconsistente con
il valore B di SA

Togliamo B dai valori possibili di NT

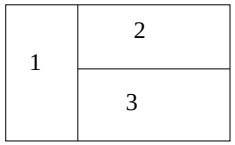
NT ha dominio vuoto! L'assegnamento che stiamo verificando e cioè **WA=R, Q=G** è inconsistente

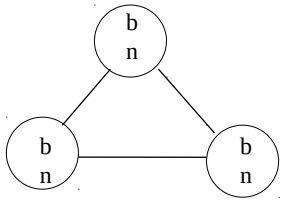
Commento su arc consistency

- La propagazione dei valori tramite arc consistency è un esempio di tecnica di inferenza
- La proprietà di arc consistency:
  - Consente di ridurre i domini delle variabili di molti CSP
  - Quando riduce i domini di tutte le variabili a un solo valore trova anche una soluzione
  - Quando rende vuoto un dominio scopre che un particolare CSP non può essere risolto
  - Di per sè non è però generalmente sufficiente a determinare una soluzione o ad accorgersi dell' irresolvibilità di un CSP, vedremo un esempio per AC-3 (usata da sola è incompleta)

Incompletezza di AC-3

Esempio: **colorazione con due colori**





È arc consistent eppure non c'è soluzione

Si può definire la **path consistency**, ma l'algoritmo aumenta di complessità

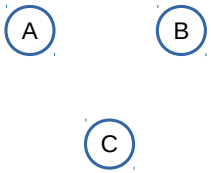
Valutazione di AC-3

- Un CSP con **n** variabili e **vincoli binari** ha al più n^2 archi
- Sia **d** il numero massimo di valori di una variabile: Il tempo per la verifica della consistenza sarà nel caso peggiore $O(n^2d^3)$
- **Più costoso della verifica forward ma più efficace**
- **AC-3 è incompleto**:
  vi sono alcuni assegnamenti inconsistenti che non vengono rilevati

Path consistency

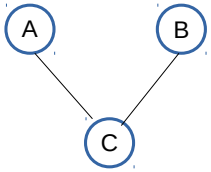
- Proprietà più forte di arc consistency
- Identifica vincoli impliciti, inferiti da triplette di variabili
- Una coppia di variabili {X1, X2} è **path consistent rispetto a una terza variabile** X3 quando:
  1) per ogni assegnamento {X1=a, X2=b} consistente con i vincoli su X1 e X2,
  2) c' è un assegnamento di X3 che soddisfa i vincoli esistenti sulle coppie di variabili {X1, X3} e {X2, X3}

Path consistency



{A=a, B=b}

In assenza di vincoli qualsiasi assegnamento di A e di B è consistente con una variabile C



{A=a, B=b}

Lo stesso assegnamento può invece avere un impatto quando A o B sono legate da vincoli a C

Path consistency, esempio

- Consideriamo il problema dell' Australia ma per semplicità supponiamo di avere solo **2 colori** (R e B)
- Consideriamo la coppia **{WA, SA}** e la variabile **NT,** Il CSP ha soluzione?
  - Ci sono due casi o {WA=R, SA=B} oppure {WA=B, SA=R}
  - Consideriamo i valori possibili per NT e rimuoviamo quelli che non rispettano la path consistency
  - Se {WA=R, SA=B}, NT non può essere R perché {WA=R, NT=R} non è consistente. Non può neanche essere B perché {SA=B, NT=B} non è consistente
  - Simili ragionamenti si possono fare anche partendo da {WA=B, SA=R}
  - Di conseguenza il CSP non ha soluzione
- L' algoritmo PC-2 (di Mackworth) propaga la proprietà di path consistency come AC-3 propaga quella di arc consistency

Generalizzazione: k-consistency

- Un CSP è k-consistent quando:
  - per ogni sottoinsieme costituito da k-1 delle sue variabili e ogni loro assegnamento consistente, è possibile individuare un assegnamento consistente per qualsiasi k-ma variabile
- 1-consistency: node consistency
- 2-consistency: arc consistency
- 3-consistency: path consistency

CSP fortemente k-consistent

- Un CSP è fortemente k-consistent quando è:
  - k-consistent,
  - (k-1)-consistent,
  - (k-2)-consistent,
  - …,
  - 1-consistent

Risolvere CSP senza backtracking

- **È dimostrato che un CSP fortemente k-consistent può essere risolto senza dover mai fare backtracking**
- Intuitivamente:
  - Poiché è 1-consistent basta iniziare da un valore consistente per una variabile X1
  - Poiché è 2-consistent è possibile individuare un valore consistente per le variabili direttamente in relazione con X1
  - Poiché è 3-consistente è possibile individuare un valore consistente per le variabili in relazione con coppie delle precedenti
  - …
- Complessità temporale: **O(nd)**
  dove: **n**= numero di variabili, **d**=numero di valori del dominio
- Problema: la complessità temporale per decidere se un CSP è fortemente k-consistent è esponenziale

Vincoli speciali

- Alcuni tipi di vincoli sono usati di frequente e sono quindi stati studiati in modo specifico:
  - Alldifferent(X1, …, Xn): i valori delle variabili elencate devono essere tutti differenti
  - Atmost (N, A1, …, Ak): le attività A1, …, Ak possono impegnare complessivamente al più N risorse

Alldifferent

- Alldifferent(X1, …, Xn)
- Se le n variabili possono assumere al più m valori distinti con m < n il vincolo non può essere soddisfatto
- Altrimenti:
  - Si comincia ad assegnare un valore alle variabili che possono assumerne uno solo
  - Se il vincolo rimane soddisfatto si propagano le scelte riducendo via via i domini delle variabili rimanenti

Atmost: assegnamento di risorse

- Atmost (N, A1, …, Ak):
  1) N è il numero di risorse disponibili
  2) A1, …, Ak rappresentano le quantità di risorse assegnate alle varie attività
- Supponiamo che ciascuna variabile Ai abbia un dominio $\{v_{i1}, v_{i2}, \ldots, v_{im}\}$, i cui valori sono ordinati in ordine crescente
- Si considerano i valori minimi $v_{11}, \ldots, v_{k1}$ nei domini di A1, …, Ak:
  1) Se la somma supera il limite imposto da atmost il vincolo non può essere soddisfatto
  2) Altrimenti si possono cancellare i valori massimi che non sono consistenti con quelli minimi degli altri domini

Atmost per domini su grandi numeri

- Applicazioni come la logistica richiedono di gestire grandi quantità di risorse: non è possibile scrivere esplicitamente array che enumerano tutti i possibili valori delle variabili
- In questo caso i domini sono rappresentati come intervalli $[v_{i1}, v_{im}]$
- La risoluzione del CSP passa attraverso la ridefinizione e la propagazione degli estremi di questi intervalli

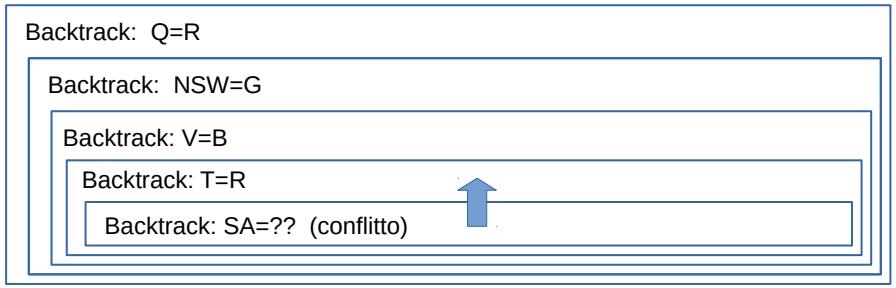
Migliorare ulteriormente il backtracking

Backjumping

Limite del backtracking

- Quando la ricerca con backtracking raggiunge un vicolo cieco, torna indietro alla variabile che era stata considerata al passo precedente (backtracking cronologico)
- Non sfrutta i vincoli, ha i limiti delle strategie di ricerca cieche
- Esempio:
  - Sia {Q=R, NSW=G, V=B, T=R} l' assegnamento corrente
  - Scegliamo SA: tutti i valori vanno in conflitto (Q, NSW e V coprono lo spettro dei colori)
  - T è la variabile scelta alla chiamata ricorsiva precedente: si fa backtracking a questa variabile che però non crea il conflitto!!

Limite del backtracking: esempio



- Scegliamo SA: confina con Q, NSW e V. Dominio vuoto.
- T è la variabile scelta alla chiamata ricorsiva precedente: si fa backtracking a questa variabile che però non crea il conflitto!!

Backjumping: un backtracking non cronologico

- **Strategia più intelligente**:
  fare backtracking a una variabile che potrebbe risolvere il problema (nell' esempio: V)
- Per ogni variabile occorre mantenere un insieme di assegnamenti che creano il conflitto (**conflict set**), nell' esempio {Q=R, NSW=G, V=B}
- **Backjumping**:
  variante del backtracking che utilizza i conflict set per decidere a quale variabile ritornare in caso di vicolo cieco

Backjumping: conflict set

- Quando si raggiunge un vicolo cieco, il backjumping torna indietro alla variabile assegnata più di recente che ha un valore che partecipa a creare il conflitto, Gli assegnamenti che portano al conflitto costituiscono il conflict set di SA
- *Definizione:*
  Sia A un assegnamento parziale consistente, sia X una variabile non ancora assegnata. Se l' assegnamento A U $\{X=v_i\}$ risulta inconsistente per qualsiasi valore v_i appartenente al dominio di X si dice che A è un **conflict set** di X

Backjumping: conflict set minimo

- Un conflict set per una variabile è minimo quando togliendo uno qualsiasi degli assegnamenti che lo costituiscono non si ottiene più un conflict set
- Esempio: {Q=R, NSW=G, V=B} è un conflict set minimo per SA
- Quando si ha un fallimento su una variabile il backjumping usa il suo insieme dei conflitti per identificare la variabile da restituire insieme all' indicazione di fallimento (nell' esempio: V)

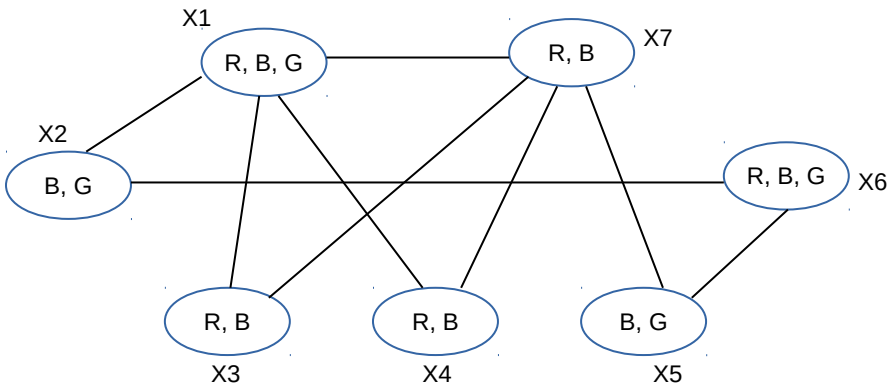
Backjumping (BJ) e forward checking (FC)

- Il FC può essere modificato in modo da costruire gli insiemi dei conflitti per ogni variabile:
  - Quando si assegna un valore a una variabile, FC propaga questa scelta attraverso i vincoli cancellando valori dai domini di altre variabili, esempio: V=B cancella il valore B dal dominio di SA
  - basta arricchre FC in modo che registri la relazione fra la variabile assegnata e quelle che hanno subito una riduzione di dominio, quindi conflict(SA) ← conflict(SA) U {V=B}
  - NOTA:
    FC rileva esattamente gli stessi conflitti rilevati da BJ, quindi l' uso di BJ congiunto a FC è ridondante

Non solo fallimenti: assegnamenti NOGOOD

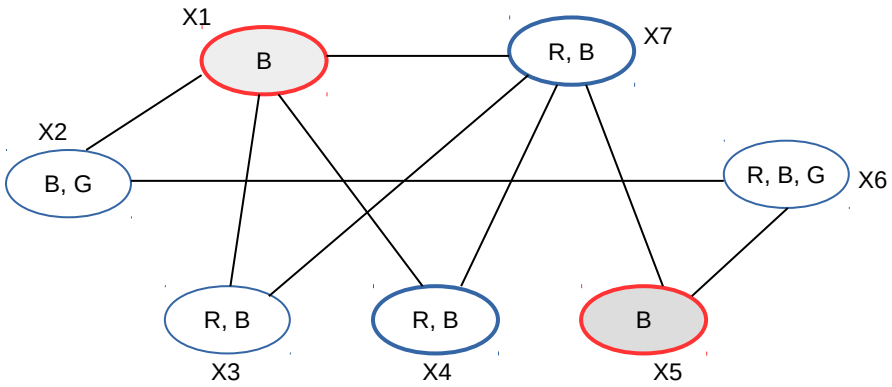
- Ricordate sistemi operativi e il deadlock?
- Ricordate le nozioni di stato sicuro, insicuro e di deadlock? Uno stato insicuro non corrisponde a un blocco ma porta necessariamente a un blocco
- l' analogo degli stati insicuri dei CSP è dato dagli **assegnamenti NOGOOD**: sono assegnamenti parziali che non appartengono all' insieme delle possibili soluzioni

Esempio



Vincoli: due variabili fra le quali c'è un vincolo non devono assumere lo stesso colore

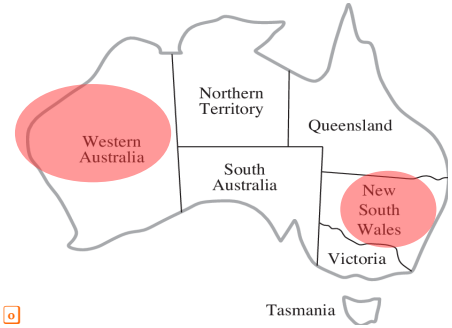
Esempio di stato NOGOOD



L'assegnamento parziale {X1=B, X5=B} non è uno stato di fallimento ma è uno stato NOGOOD perché impedisce di trovare assegnamenti consistenti delle variabili x4 e x7.

NOGOOD per Australia

- **{WA=R, NSW=R}** è un assegnamento NOGOOD per il problema dell' Australia
- Non è un vicolo cieco, la ricerca può esplorare molte alternative, per esempio procederà con T=R
- Inoltre i conflict set di V, Q e NT conterranno o l' assegnamento di NSW o quello di WA, non entrambi
- Ma quando poi tenta di assegnare NT, SA, Q o V fallirà sempre (tutte dovranno essere o B o G): quando fallisce, come capire che la colpa era di questa scelta iniziale?

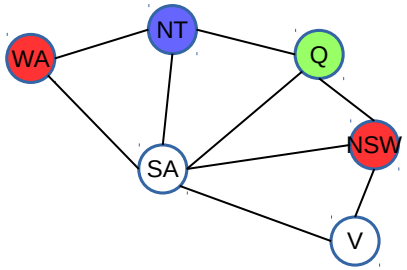


Conflict-directed backjumping

- Il problema non sussisterebbe se dovessimo assegnare valori solo a uno degli stati rimanenti, sussiste perché dobbiamo assegnarli a tutti e questi stati sono fra loro confinanti
- In altri termini il conflitto dipende da due fattori:
  (1) dagli assegnamenti precedenti e
  (2) dalle variabili rimaste a cui dovremo assegnare valori in futuro
- È possibile calcolare i conflict set in modo in modo tale che guidino in modo più efficace il backtracking (evitando di considerare T=R da cui non dipende l'inconsistenza e saltando direttamente ai "colpevoli" veri)?

Australia

- Supponiamo di assegnare le variabili in quest'ordine:
  WA = R;
  NSW = R;
  T = B;
  NT = B;
  Q = G



- SA = ?
  Non è possibile fare l'assegnamento senza creare conflitti!
- Vediamo i conflict set …

Australia. Conflict set

- Costruiti tramite FC come assegnamenti precedenti di variabili legate da un vincolo (si riportano solo i nomi di variabile per brevità). I seguenti conflict set sono stati costruiti man mano che si facevano gli assegnamenti:
  - conf(WA) = {}
  - conf(NSW) = {}
  - conf(T) = {}
  - conf(NT) = {WA}
  - conf(Q) = {NSW,NT}
  - **conf(SA) = {WA,NSW,NT,Q}**

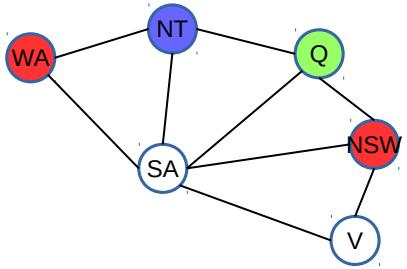
Backjump e aggiornamenti dei conflict set

- **backjump**:
  - Sia Xj la variabile corrente
  - Indichiamo conf(X) il conflict set della generica variabile X
  - Se tutti i valori possibili di Xj falliscono si fa un backjump alla variabile Xi che è stata aggiunta a conf(Xj) più di recente e si aggiorna conf(Xi) in questo modo:

    conf(Xi) ← conf(Xi) U conf(Xj) - { Xi }

Australia

- Supponiamo di assegnare le variabili in quest' ordine:
  WA = R;
  NSW = R;
  T = B;
  NT = B;
  Q = G



- SA = ?
  conf(SA) = {WA,NSW,NT,Q}
- La vrb aggiunta più di recente è Q

Si parte da SA

- Backjump a Q:
  - conf(Q) = {NSW, NT}
  - Lo aggiorniamo arricchendolo dell' informazione di conflitto fornita da SA: conf(Q) ← conf(Q) U conf(SA) – {Q}
    conf(Q) ← {NSW, NT} U {WA,NSW,NT,Q} – {Q}
  - Quindi **conf(Q)** = {**WA**, NSW, NT}
- **NB:** WA non confina con Q ma prende parte al generarsi del conflitto e allora viene ricordato anche in associazione a Q. Viene rilevato un "vincolo implicito"
- Q=G si è dimostrato un assegnamento non consistente. Q=B e Q=R non sono possibili perché non si avrebbe arc consistency. Il dominio di Q diventa vuoto
- …

Backjump a NT

- Backjump a NT:
  conf(NT) ← conf(NT) U conf(Q) – {NT}, quindi:
  conf(NT) ← { WA, NSW }
- L’ assegnamento NT=G si è dimostrato non portare a soluzioni
- Si prova ad assegnare NT=B. Saltiamo i passi: l’ assegnamento si dimostrerà altrettanto infruttuoso.
- Il dominio di NT diventa vuoto occorre quindi fare un backjump a NSW
- **conf(NSW) ← {} U {WA} cioè: conf(NSW) ← {WA}**
  Il metodo ha scoperto la relazione fra NSW e WA!
- Tenta NSW=G e arriverà a costruire una soluzione

Osservazione

- La simulazione è partita da uno stato NOGOOD
- Ha scoperto che si trattava di uno stato NOGOOD solo tramite l’ esplorazione
- I vincoli hanno fornito una guida che ha permesso di evidenziare relazioni implicite fra le variabili
- Gli stati NOGOOD minimali possono essere registrati dall’ algoritmo che potrà così evitare di ripetere questi assegnamenti in futuro
- Si tratta di una forma di **apprendimento**

Applicazioni

Esempio: Sudoku

	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

(b)

Vincoli:
ogni cifra deve comparire una volta sola:
- In ogni riga
- In ogni colonna
- In ogni quadretto 3x3

Figure 6.4 **FILES: figures/sudoku.eps (Tue Nov 3 13:49:46 2009).** (a) A Sudoku puzzle and (b) its solution.

Quante regine riusciamo a gestire?

- Alcuni algoritmi attuali hanno risolto il problema delle N regine con N=6.000.000

Esempio: sudoku

- **Variabili**: ciascuna casella è rappresentata da una variabile x_{ij}, dove i è la riga e j la colonna che identificano la casella
- **Valori**: {1, …, 9} alcune variabili hanno valore prefissato
- I valori prefissati sono **vincoli unari**
- **Altri vincoli**:
  - Sulle righe: per ogni i in {1, …, 9} alldifferent(x_{i1}, x_{i2}, …, x_{i9}}
  - Sulle colonne: per ogni j in {1, …, 9} alldifferent(x_{1j}, x_{2j}, …, x_{9j}}
  - Nei riquadri: sia S_{kl} il generico riquadro di lato 3, k e l compresi fra 0 e 2: per ogni i, j dove i = k*3+1 e j = l*3 + 1
    alldifferent(x_{ij}, $x_{i(j+1)}$, $x_{i(j+2)}$, $x_{(i+1)j}$, $x_{(i+1)(j+1)}$, $x_{(i+1)(j+2)}$, $x_{(i+2)j}$, $x_{(i+2)(j+1)}$, $x_{(i+2)(j+2)}$)

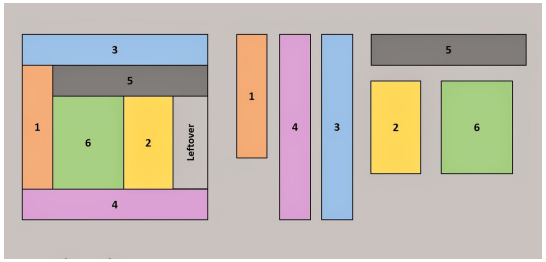
Applicazioni reali

I CSP si sono dimostrati vincenti in diversi contesti applicativi dello stesso genere di quelle affrontate in ricerca operativa, alcuni esempi:

- **Location of facilities** (es. warehouses): dato un elenco di magazzini e date delle richieste fatte da clienti, identificare un percorso che permetta di soddisfare le richieste visitando i magazzini così da minimizzare i costi
- **Job scheduling**: supponiamo che una fabbrica produca un range di prodotti, ogni tipologia richiede la sequenzializzazione di determinate operazioni svolte da macchinari. Ogni operazione richiede un tempo di esecuzione. Lo scopo è trovare una sequenza di produzione che minimizzi i tempi di produzione
- **Car sequencing**: nell' industria automobilistica la catena di montaggio assembla automobili partendo da un modello base a cui sono aggiunte caratterizzazioni scelte dai clienti (es. Optional). Automobili diverse avranno insiemi di optional diversi. Le auto sono portate da un nastro trasportatore e gli optional sono montati in aree di lavoro, ognuna delle quali ha una capacità massima. Il problema consiste nel definire un ordine di costruzione tale da non eccedere le capacità delle aree di lavoro

Applicazioni reali

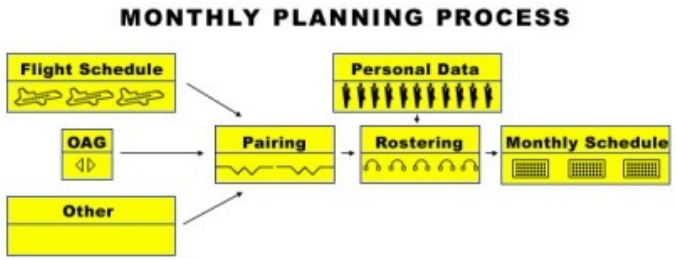
- **Cutting stock problem**: un materiale deve essere tagliato in pezzi più piccoli per un cliente minimizzando lo spreco



- **Vehicle routing**: n clienti vengono riforniti da uno stesso deposito. Il problema consiste nel trovare il percorso di costo minimo

Applicazioni reali

- **Timetabling**: costruzione automatica di orari (esempio per corsi di studi) tenendo conto dell' allocazione di risorse (aule, laboratori) e altri vincoli
- **Rostering e crew scheduling**: definizione di turni e di equipaggi (esempio: per compagnie aeree). Nel 1998 sono stati usati CSP realizzati in ECLiPSe per definire gli equipaggi dei treni delle ferrovie dello stato italiane



Software

- I linguaggi di programmazione a vincoli sono dichiarativi e in parte sviluppati come moduli del linguaggio Prolog
- ECLiPSe (http://eclipseclp.org/):
  un' introduzione passo passo alla programmazione in ECLIPSE della durata di circa 1:30 fatta da Sergii Dymchenko (in inglese):
  https://www.youtube.com/watch?v=84amHOgCEe8
- **Febbraio 2015**: Opel vince il premio VDA (German Association of the Automotive Industry) Logistics Award grazie allo strumento di ottimizzazione della catena di distribuzione sviluppato in cooperazione con Flexis AG e basato su CSP ed ECLiPSe

Software

- **CHR**: sviluppato dall' università di Leuven (Belgio) https://dtai.cs.kuleuven.be/CHR/download.shtml
- Disponibile in diversi linguaggi di programmazione:
  - Vari prolog (SWI-prolog, CIAO, ecc.)
  - Java
  - Haskell
  - Linguaggio C
- Applicato per esempio alla navigazione robotica

Software

- **SAT solvers**
- **Risolvono problemi a variabili booleane**
- Molti SAT solver sono basati sull' algoritmo DPLL arricchito con risoluzione di conflitti, backjumping, propagazione lineare, apprendimento di clausole e altre tecniche di ottimizzazione
- MiniSat (http://minisat.se/) vincitore della competizione 2005 è open source: "MiniSat is a minimalistic, open-source SAT solver, developed to help researchers and developers alike to get started on SAT"
- Sito della competizione annuale per SAT solver: http://www.satcompetition.org/

Software

- **GECODE**
  http://www.gecode.org/
- Progetto open source con interfaccia grafica, basato su C++
- Molta documentazione che spiega tecniche, modelli, tutto nell' ottica di realizzare sistemi

Software

- **GECODE/R**
- Gecode nel linguaggio Ruby
- Trovate implementati diversi esempi che abbiamo visto insieme ed altri: http://gecoder.rubyforge.org/examples/index.html
- Trovate anche una miniguida alla modellazione di problemi come CSP:
  http://gecoder.rubyforge.org/documentation/modelling/index.html

Answer set programming

- ASP non è un approccio specificamente pensato per risolvere CSP ma è uno strumento (concettuale e anche pratico) insegnato nel corso “Intelligenza Artificiale e Laboratorio” della laurea magistrale con cui si possono programmare e risolvere CSP
- Risolverete CSP come il problema della zebra