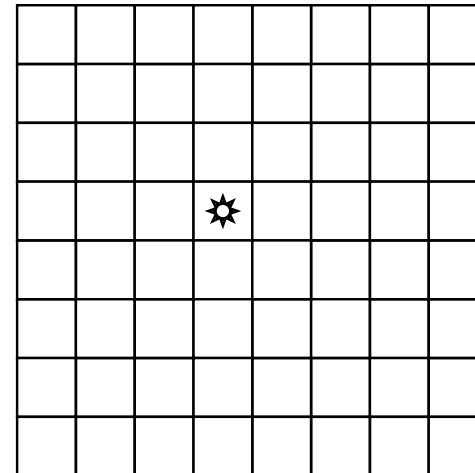


SODDISFACIMENTO DI VINCOLI

- Molti problemi di AI possono essere visti come problemi di soddisfacimento di vincoli. Obiettivo: trovare uno stato del problema che soddisfi un dato insieme di vincoli.
- **Esempio: Il Problema delle Otto Regine**
 - È data una scacchiera (8x8): il problema consiste nel posizionarvi otto regine in modo da evitare un attacco reciproco.
 - Le mosse possibili per la regina prevedono tutte le posizioni sulla stessa riga, colonna e diagonale a partire dalla casella.



PROBLEMA DELLE 8 REGINE: MODELLO

- Le otto regine vengono rappresentate con le variabili $x_1, x_2, \dots, x_8$
- Il pedice si riferisce alla colonna occupata dalla corrispondente regina.
- L'istanziamento di una variabile x_i al valore k appartenente all'insieme $[1..8]$ indica che la regina corrispondente viene posizionata sulla k -esima riga della i -esima colonna.
- Le variabili hanno come insieme di possibili valori gli interi compresi tra 1 e 8 che corrispondono alle righe occupate.

VINCOLI: DUE TIPI

- Quelli che vincolano i valori delle variabili al dominio suddetto e quelli che devono impedire un attacco reciproco e che impongono, quindi, relazioni tra i valori assunti dalle variabili.

$$1 \leq x_i \leq 8$$

$$\text{per } 1 \leq i \leq 8$$

$$x_i \neq x_j$$

$$\text{per } 1 \leq i < j \leq 8$$

$$x_i \leq x_j + (j - i)$$

$$\text{per } 1 \leq i < j \leq 8$$

$$x_i \leq x_j - (j - i)$$

$$\text{per } 1 \leq i < j \leq 8$$

- Il primo vincolo impone che i valori assunti dalle variabili del problema siano compresi tra i numeri interi 1 e 8: vincoli unari
- I tre vincoli successivi definiscono relazioni tra le variabili e, in particolare, tra due variabili alla volta: vincoli binari

Due tipi di vincoli (continua)

$$1 \leq x_i \leq 8$$

$$\text{per } 1 \leq i \leq 8$$

$$x_i \neq x_j$$

$$\text{per } 1 \leq i < j \leq 8$$

$$x_i \neq x_j + (j - i)$$

$$\text{per } 1 \leq i < j \leq 8$$

$$x_i \neq x_j - (j - i)$$

$$\text{per } 1 \leq i < j \leq 8$$

- Il secondo impone che due regine non siano posizionate sulla stessa riga. In caso contrario si attaccherebbero.
- Il terzo e il quarto vincolo riguardano le posizioni sulle due diagonali a partire dalla casella iniziale: impongono che il posizionamento di due regine sulla medesima diagonale sia scartato come soluzione non ammissibile.

POSSIBILE SOLUZIONE IN PROLOG: GENERATE AND TEST

- La soluzione è rappresentata da una permutazione della lista:
[1,2,3,4,5,6,7,8]
- La permutazione è una soluzione se le posizioni di ogni regina sono sicure.

```
solution(S) :- permutation([1,2,3,4,5,6,7,8],S),
                  safe(S).
permutation([],[]).
permutation([Head|Tail],PermList) :-
    permutation(Tail,PermTail),
    delete1(Head,PermList,PermTail).
```

(chiamando `delete1` di fatto si inserisce un elemento nella lista).

POSSIBILE SOLUZIONE IN PROLOG: GENERATE AND TEST

- Definizione di safe:
 - Se S è la lista vuota è sicuramente "safe".
 - Se S è una lista della forma [Queen|Others], è safe se Others è safe e Queen non attacca nessuna regina in Others.

```
safe([]) .
```

```
safe([Queen|Others]) :-
```

```
    safe(Others) , noattack(Queen,Others,1) .
```

```
noattack(_,[],_) .
```

```
noattack(Y,[Y1|Ylist],Xdist) :-
```

```
    Y-Y1=\=Xdist,
```

```
    Y1-Y=\=Xdist,
```

```
    Dist1 is Xdist +1
```

```
    noattack(Y,Ylist,Dist1) .
```

- Nota: questa impostazione non è particolarmente efficiente: genera una soluzione completa e poi la controlla (generate and test)

CRIPTOARITMETICA

$$\begin{array}{ccccccccc} \cdot & & S & & E & & N & & D & & + & & & & & & \\ & & & & & & M & & O & & R & & E & & = & & \\ & & & & & & \hline & & & & M & & O & & N & & E & & Y & & & & \end{array}$$

- Due lettere diverse non possono avere lo stesso valore;
- Le somme delle cifre devono corrispondere a quanto illustrato dal problema.
- Obiettivo: determinare uno stato in cui ad ogni lettera è associata una cifra in modo che i vincoli iniziali siano soddisfatti.
- I problemi di progetto sono tipicamente problemi a vincoli: un progetto deve rispettare vincoli di costo, tempo, materiali ecc.

UNA SOLUZIONE IN PROLOG

`sum(N1,N2,N) :-`

`sum1(N1,N2,N,0,0,[0,1,2,3,4,5,6,7,8,9],_).`

`N1,N2,N` sono le liste di caratteri.

- Il quarto e quinto argomento di `sum1` sono i riporti da destra e verso sinistra, ovvero il riporto prima di fare la somma e il riporto dopo aver fatto la somma, il sesto argomento è la lista di numeri disponibili prima della somma, il settimo quelli rimasti, non utilizzati nella somma.

- **Esempio:**

`:-sum1([H,E],[6,E],[U,S],1,1,[1,3,4,7,8,9],Digits).`

`H=8, E=3, S=7, U=4, Digits=[1,9]`

`(continua)`

UNA SOLUZIONE IN PROLOG

```
sum1([],[], [],0,0,Digits,Digits) .
sum1([D1|N1],[D2|N2], [D|N],C1,C,Digs1,Digs):-
    sum1(N1,N2, N,C1,C2,Digs1,Digs2) ,
    digitsum(D1,D2,C2,D,C,Digs2,Digs) .

digitsum(D1,D2,C1,D,C,Digs1,Digs):-
    del(D1,Digs1,Digs2) ,
    del(D2,Digs2,Digs3) ,
    del(D,Digs3,Digs) ,
    S is D1 +D2 +C1,
    D is S mod 10,
    C is S div 10.

del(A,L,L) :- nonvar(A),!. (se è già istanziato non fa nulla)
del(A,[A|L], L) .
del(A,[B|L],[B|L1]) :- del(A,L,L1) .
```

ALCUNI PUZZLES

```
p1([D,O,N,A,L,D],[G,E,R,A,L,D],[R,O,B,E,R,T]).
```

```
p2([O,S,E,N,D],[O,M,O,R,E],[M,O,N,E,Y]).
```

- (nota: il programma suppone per semplicità che le liste abbiano la stessa dimensione).
- Possibili goals:

```
: - p1(N1,N2,N),sum(N1,N2,N).
```

```
: - sum1([5,O,N,A,L,5],[G,E,R,A,L,5],  
[R,O,B,E,R,T],0,0,[0,1,2,3,4,6,7,8,9],_).
```

RAZIONALIZZANDO

- In generale posso utilizzare due possibili tecniche:
 - Costruisco una soluzione di tentativo e la verifico. Assegno dei valori per le lettere e poi vedo se ottengo un fallimento utilizzando i vincoli.
 - In questo caso è necessario ritrattare alcune, al limite tutte, le scelte fatte per arrivare a quel punto attraverso il meccanismo di backtracking.
 - Ritardo gli assegnamenti di valori e le scelte finché non sono costretto a farlo (più efficiente).
 - Ridurre considerevolmente il carico di ricerca da effettuare evitando di assegnare valori alle variabili fino a quando non ci si trova costretti a farlo.
- Esistono due tipi di vincoli:
 - a) vincoli semplici che esprimono possibili valori per un determinato oggetto;
 - b) vincoli più complessi che esprimono relazioni fra diversi oggetti.

CSP PIÙ FORMALMENTE

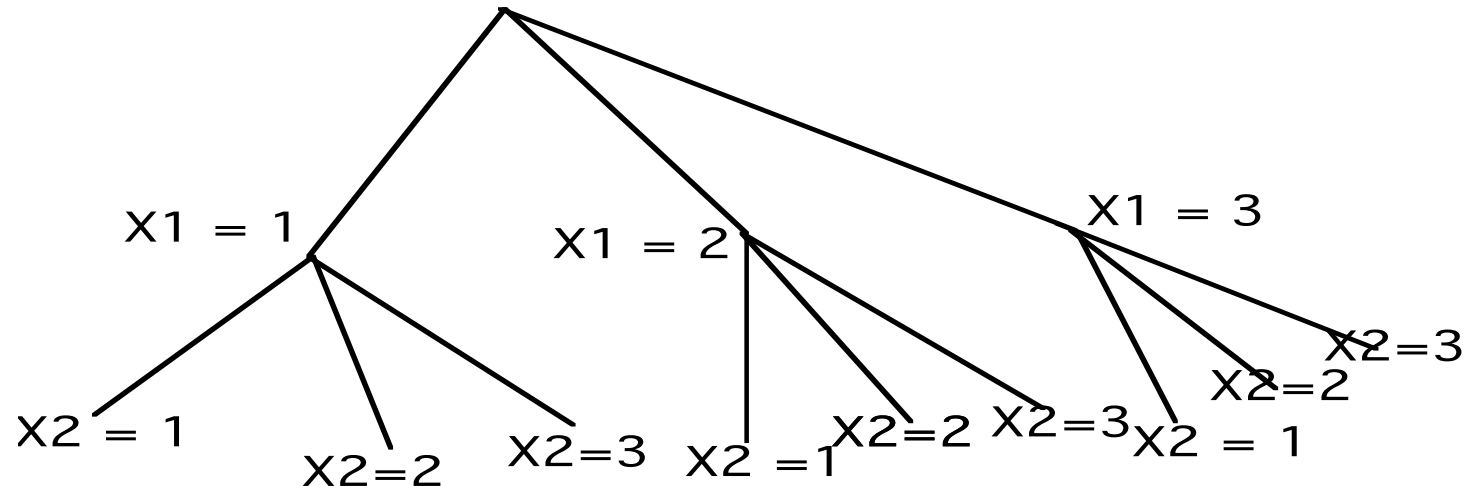
- Formalmente un CSP (Constraints Satisfaction Problem) può essere definito su un insieme finito di variabili:
 $(x_1, x_2, \dots, x_n)$ i cui valori appartengono a domini finiti di definizione $(D_1, D_2, \dots, D_n)$, e su un insieme di vincoli.
- Un vincolo $c(x_{i1}, x_{i2}, \dots, x_{ik})$ tra k variabili è un sottoinsieme del prodotto cartesiano $D_{i1} \times D_{i2} \times \dots \times D_{ik}$ che specifica quali valori delle variabili sono compatibili con le altre.
- Tale sottoinsieme non deve essere definito esplicitamente ma è rappresentato in termini di relazioni.
- Una soluzione ad un CSP prevede un assegnamento di tutte le variabili che soddisfi tutti i vincoli.

ALBERO DECISIONALE

- Qualunque tecnica di soluzione di problemi NP-difficili fa uso, almeno concettualmente, di un albero decisionale.
- Un possibile albero decisionale per un CSP si ottiene (dopo aver stabilito un ordinamento per le variabili) facendo corrispondere ad ogni livello dell'albero l'assegnamento di una variabile e ad ogni nodo la scelta di un possibile valore da dare alla variabile corrispondente al livello del nodo stesso.
- Il cammino per ogni foglia dell'albero rappresenterà quindi un assegnamento di valori a tutte le variabili. Se tale assegnamento soddisfa tutti i vincoli, allora la foglia corrispondente rappresenterà una soluzione del problema, altrimenti rappresenterà un fallimento.
- La ricerca di una soluzione è equivalente all'esplorazione dell'albero decisionale per trovare una foglia con cammino-soluzione.
- In un problema di n variabili ed in cui i domini hanno tutti la stessa cardinalità d , il numero di foglie di un albero decisionale così costruito è pari a d^n .

ESEMPIO: $N = 2$ $D=3$

- Per esempio, in un albero che rappresenta un problema di 10 variabili ed in cui ogni dominio ha cardinalità 10 esistono 10 miliardi di foglie.
- È quindi evidente che la strategia di esplorazione dell'albero risulta di importanza fondamentale al fine di trovare una soluzione per un problema complesso in tempi ragionevolmente brevi (tecniche di consistenza).



GLI ALGORITMI DI PROPAGAZIONE

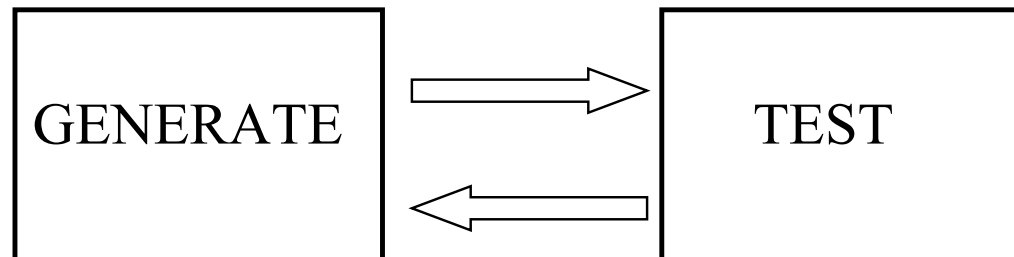
- Gli algoritmi di propagazione sono metodi di ricerca più intelligenti che tentano di prevenire i fallimenti anzichè recuperare fallimenti già avvenuti.
- *Pruning* (“potatura”) *a priori* dell'albero delle decisioni.
- Utilizzare le relazioni tra le variabili del problema, i vincoli, per ridurre lo spazio di ricerca *prima* di arrivare al fallimento.
- Vengono così eliminati rami dell'albero che porterebbero ad un sicuro insuccesso evitando ove possibile inutili backtracking.

DUE APPROCCI

- Dato un CSP esistono due possibili approcci per la sua risoluzione: uno basato sulle Tecniche di Consistenza e l'altro su Algoritmi di Propagazione.
- Senza perdita di generalità ci riferiremo, nel seguito, a CSP su vincoli binari (vincoli cioè che coinvolgono due variabili).
- Algoritmi di Propagazione
 - Basati sulla propagazione dei vincoli per eliminare a priori, durante la ricerca, porzioni dell'albero decisionale che porterebbero ad un sicuro fallimento (compatibilmente con le scelte già effettuate).
- Tecniche di Consistenza
 - Basate sulla propagazione dei vincoli per derivare un problema più semplice di quello (completo) originale.
 - Tipicamente prima si applicano Tecniche di Consistenza e poi di Propagazione.

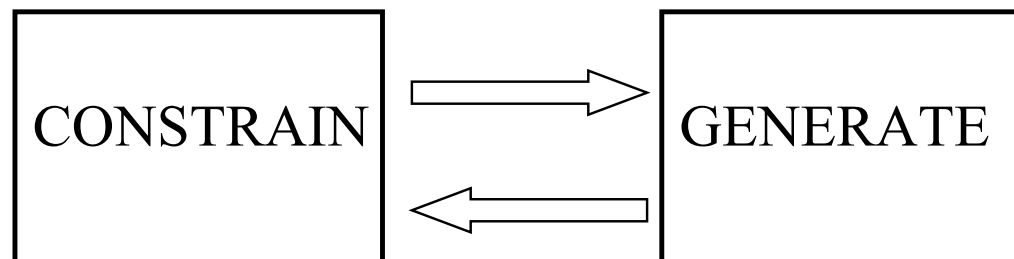
ALGORITMI DI PROPAGAZIONE

- Le due tecniche che usano i vincoli a posteriori sono:
 - Il Generate and Test (GT)
 - Lo Standard Backtracking (SB).



ALGORITMI DI PROPAGAZIONE

- Gli algoritmi di propagazione si basano sul concetto inverso.
- Tecniche quali Forward Checking (FC), il First Fail (FF) e il Looking Ahead (LA).
- Un modulo propaga i vincoli finché è possibile (*constrain*); alla fine della propagazione o si è giunti ad una soluzione o sono necessarie nuove informazioni sulle variabili libere (*generate*).



USO DEI VINCOLI A PRIORI E A POSTERIORI

- Consideriamo una ricerca depth-first in un albero decisionale. Si tende a scendere di livello nell'albero fino a quando o si sono assegnate tutte le variabili, e quindi si è trovata una soluzione, oppure non è più possibile trovare un valore (la sequenza corrente non può portare a una soluzione ammissibile); quindi si esegue un'altra scelta sull'ultima variabile della sequenza stessa.
- L'algoritmo ha tre gradi di libertà:
 - la scelta nell'ordinamento delle variabili;
 - la scelta nell'ordine di selezione del valore da attribuire alla variabile corrente;
 - la propagazione effettuata in ciascun nodo.
- I primi due riguardano le euristiche sulla strategia di ricerca.

USO DEI VINCOLI A PRIORI E A POSTERIORI

- Il terzo grado di libertà è ciò che differenzia i diversi algoritmi. Nel seguito vediamo le seguenti strategie:
 - Generate and Test
 - Standard Backtracking
 - Forward Checking
 - (Partial and Full) Look Ahead

GENERATE AND TEST

- L'interprete del linguaggio sviluppa e visita un albero decisionale percorrendolo in profondità assegnando valori alle variabili senza preoccuparsi di verificare la consistenza con gli altri vincoli.
- Consideriamo il problema delle otto regine: le variabili coinvolte nel problema prevedono, come dominio di definizione, i numeri interi compresi tra 1 e 8.
- Il Generate and Test avanza, nella ricerca di una soluzione, in modo 'miope' assegnando all'insieme delle variabili una permutazione degli interi che compongono il dominio.
- Unici vincoli considerati nella fase Generate:
 - $1 \leq X_i \leq 8$ per $1 \leq i \leq 8$
 - $X_i \neq X_j$ per $1 \leq i < j \leq 8$

Il secondo è dovuto al fatto che ogni tentativo consiste in una permutazione dei valori appartenenti ai domini e quindi ciascun valore assegnato alle variabili è diverso dagli altri.

GENERATE AND TEST

- Così facendo una soluzione al problema può essere la seguente:
 $(X_1, X_2, \dots, X_8) = (1, 2, 3, 4, 5, 6, 7, 8)$. Assegnamento di ogni regina a una casella appartenente alla diagonale principale. Sbagliato !
- Solo in un secondo tempo questa tecnica considera gli altri vincoli rifiutando la soluzione trovata perché incompatibile con i vincoli del problema.
- A questo punto inizia la procedura di backtracking tentando con la seconda permutazione e così via finché non si trova una soluzione.
- Inefficienza di base
 - I vincoli sono utilizzati per limitare lo spazio delle soluzioni dopo che la ricerca è stata effettuata, quindi a posteriori.
 - Il numero delle possibili permutazioni aumenta con il fattoriale del numero di termini da permutare. Nel caso di $n=8$ abbiamo un numero di permutazioni pari a $8! = 40320$, se $n=10$ allora $n! = 3628800$ arrivando per $n=20$ a ordini di grandezza di 10^{18} e quindi dimensioni inaccettabili dallo spazio di ricerca.

STANDARD BACKTRACKING

- Sebbene migliore della precedente anche questa tecnica prevede un utilizzo a posteriori dei vincoli.
 - A ogni istanziazione di una variabile si preoccupa di verificare la coerenza della variabile appena istanziata con quelle assegnate precedentemente.
 - Quindi l'utilizzo dei vincoli è più efficace del precedente perché non si prosegue la ricerca in rami che, ai primi livelli dell'albero, presentano delle contraddizioni.
- Il tentativo effettuato dal Generate and Test per risolvere il problema delle otto regine assegnandole ad una diagonale della scacchiera verrebbe bloccato alla seconda istanziazione.
 - Infatti il vincolo $X_i \neq X_j - (j-i)$ per $1 \leq i < j \leq 8$ verrebbe violato dalle prime due variabili: sostituendo $X_1 = 1$ e $X_2 = 2$ nella relazione si ottiene $1 \neq 2 - (2-1)$
che porta a una contraddizione.
- L'algoritmo viene quindi fermato e, con un backtracking, si tenta di assegnare a X_2 il valore 3 con successo e così via.

STANDARD BACKTRACKING

- I vincoli sono utilizzati all'indietro (*backward*) e portano a una effettiva riduzione dello spazio di ricerca.
- Tuttavia questa riduzione viene fatta a posteriori (*a posteriori-pruning*) cioè dopo aver effettuato il tentativo.
- A differenza della metodologia del Generate and Test, che traslascia il terzo e il quarto vincolo del problema delle otto regine (i vincoli sulle diagonali) durante la ricerca, per poi considerarli al termine dell'istanziamento di tutte le variabili, lo Standard Backtracking controlla la loro consistenza per ogni variabile istanziata.
- Quindi è prevista la verifica dei vincoli:
$$V_{k+1} \neq V_i + (k+1-i) \quad \text{per } 1 \leq i \leq k ;$$
$$V_{k+1} \neq V_i - (k+1-i) \quad \text{per } 1 \leq i \leq k ;$$

per ogni valore di k .

CSP E PROBLEMI DI OTTIMIZZAZIONE

- Consideriamo solo CSP in cui le variabili hanno domini discreti. La maggior parte di questi problemi sono NP-difficili, cioè sono problemi per i quali non è ancora stato trovato, e probabilmente non esiste, un algoritmo in grado di trovare la soluzione in un tempo polinomiale nella dimensione del problema.
- Un Constraint Optimization Problem (COP) è un Problema di Soddisfacimento di Vincoli in cui viene aggiunto un obiettivo.
- Un COP è quindi formalmente descrivibile come un CSP il cui scopo non è solo trovare una soluzione ammissibile, ma la soluzione ottima secondo un certo criterio di valutazione.

CSP E PROBLEMI DI OTTIMIZZAZIONE

- Dato un algoritmo generale in grado di risolvere qualsiasi CSP si può allora utilizzare tale algoritmo per risolvere anche qualsiasi COP. Infatti, dopo aver descritto il problema in termini di variabili, domini e vincoli, basta aggiungere una variabile ulteriore che rappresenta la funzione obiettivo.
- Ogni volta che si trova una soluzione al CSP viene aggiunto un nuovo vincolo che garantisce che ogni soluzione futura avrà un valore della funzione obiettivo migliore. Questo procedimento continua finché non sarà più possibile trovare alcuna soluzione. L'ultima soluzione trovata è la soluzione ottima.
- Non sempre questo procedimento è efficiente

ESEMPIO

- Siano date due variabili booleane x_1 e x_2 i cui domini di definizione contengono i valori: $\{V, F\}$ dove V significa VERO e F FALSO
- Il prodotto cartesiano dei domini contiene le coppie:
$$\{(V,V), (V,F), (F,V), (F,F)\}$$
- Il vincolo **diverso**(x_1, x_2) è un sottoinsieme del prodotto cartesiano dei domini contenente solo le coppie $(V,F), (F,V)$
- Prodotto logico tra due variabili booleane: date tre variabili booleane X, Y e Z ; il vincolo **and**(x, y, z) ha successo se Z è l'and logico tra X e Y e può essere definito (“propagato”) come segue:
 - . Se $Z=1$ allora X e Y devono essere 1.
 - . Se $X=0$ e/o $Y=0$ allora Z deve essere 0.
 - . Se $X=1$ e $Y=1$ allora Z deve essere 1.
 - . Se $Z=0$ e $X=1$ allora Y deve essere 0.
 - . Se $Z=0$ e $Y=1$ allora X deve essere 0.
 - . Se $X=1$ allora $Z=Y$.
 - . Se $Y=1$ allora $Z=X$.

ALGORITMO

- Queste regole (che esprimono la **PROPAGAZIONE DI UN VINCOLO**) esprimono quali informazioni possono essere dedotte una volta che sia noto il valore di una o più variabili.
- Spazio formato da un insieme di vincoli.
- Spazio iniziale: contiene i vincoli dati inizialmente alla descrizione del problema.
- Goal: stato che è stato sufficientemente vincolato (dipende dal problema). Es: nella crittoaritmetica a ogni lettera è stata assegnata un'unica cifra.
- Due passi:
 - Dapprima si individuano i vincoli e si propagano all'interno del sistema il più possibile.
 - Se non si è trovata una soluzione si inizia la ricerca. Si fa un tentativo in base a un certo criterio generando un nuovo vincolo e poi si riparte con la propagazione.

ALGORITMO

- Esempio:
Iniziamo con un solo vincolo $N = E + 1$. Se successivamente aggiungiamo il vincolo $N = 3$ possiamo propagarlo in modo da rafforzare il vincolo per E ottenendo $E=2$.
- La propagazione di vincoli è una regola di inferenza che consente di derivare nuovi vincoli da quelli noti o di limitare i valori di alcune variabili.
- La propagazione può terminare per due motivi:
 - 1. Si individua una contraddizione nel qual caso non esiste una soluzione coerente con i vincoli dati e/o ricavati quindi alcuni di essi devono essere ritrattati.
 - 2. Si sono propagati i vincoli il più possibile. A questo punto o si è giunti a una soluzione oppure si procedere all'aggiunta di nuove relazioni facendo ipotesi sul valore di una o più variabili.

ALGORITMO

- In un algoritmo di soddisfacimento di vincoli, è necessaria la compresenza di due tipi di regole:
 - quelle che definiscono come propagare correttamente i vincoli e
 - regole che suggeriscono tentativi per procedere quando è necessario.
- Esistenza di due tipi di vincoli: i primi limitano i valori assegnabili a una variabile e costituiscono il suo dominio; i secondi descrivono relazioni tra variabili.
- Entrambi giocano un ruolo identico nel processo di propagazione mentre solo i primi intervengono nella scelta dei tentativi da effettuare per proseguire l'algoritmo di soddisfacimento dei vincoli.

ALGORITMO CSP

- 1. Si propagano i vincoli disponibili. A tal fine si inizializza APERTO all'insieme degli oggetti cui, in una soluzione completa, deve essere assegnato un valore. Si ripetono le seguenti operazioni fino a che si individua una contraddizione o aperto è vuoto:
 - (a) Si seleziona un oggetto OB da APERTO e si rafforza il più possibile l'insieme di vincoli su OB;
 - (b) Se tale insieme è diverso da quello assegnato a OB l'ultima volta che si è esaminato o se è la prima volta che si esamina OB si aggiungono all'insieme APERTO tutti gli oggetti che condividono vincoli con OB;
 - (c) Si rimuove OB da APERTO
- 2. Se l'unione dei vincoli trovati al passo 1 costituisce una soluzione si termina riportando tale soluzione.
- 3. Se l'unione dei vincoli trovati al passo 1 determina una contraddizione si termina riportando un fallimento.

ESEMPIO

- Criptoarimetica
 - Si propagano i vincoli utilizzando regole che sfruttano proprietà aritmetiche;
 - Si prova ad assegnare una cifra ad una lettera il cui valore non è ancora stato determinato.
- Nel secondo passo è fondamentale conoscenza euristica sul problema.
 - Quale cifra assegnare per prima ?
 - Quale lettera assegnare alla cifra selezionata ?
- Nel nostro caso conviene fare l'assegnamento di tentativo a lettere a cui si possono assegnare pochi valori o che sono coinvolte in molti vincoli.

PRIME ITERAZIONI DEL PROCEDIMENTO

- C1, C2 e C3 sono i riporti relativi alle colonne contando da destra verso sinistra.
- All'inizio le regole per la propagazione dei vincoli generano i seguenti vincoli aggiuntivi:
 - $M=1$, poiché i numeri di una sola cifra con riporto possono produrre come somma al più 19 (si suppone $M \neq 0$);
 - $S=8$ o 9 poiché $S + M + C3 > 9$ per generare riporto, $M=1$ e $C3$ è al più 1;
 - $O = 0$, poiché $S + M + C3$ deve dare almeno 10 per generare un riporto. Ma $M = 1$ e $C3 \leq 1$, quindi $O = 0$. Non può valere 1 poiché tale valore è già associato a M ;
 - $N = E$ o $E+1$ a seconda del valore di $C2$. Ma poiché non possono essere uguali N ed E , $N = E+1$ e $C2 = 1$;
 - poiché $C2 = 1$, $N + R + C1 > 9$ e quindi $N + R > 8$;

ITERAZIONI SUCCESSIVE

- $N + R$ non può essere maggiore di 18 (anche con riporto), quindi E non può essere 9, (Non possono essere entrambi 9)
- Se a questo punto non si possono generare altri vincoli si assegna di tentativo ad E il valore 2.
- Inizia di nuovo il propagatore di vincoli:
 - - $N=3$ ($N=E+1$)
 - - $R = 8$ o 9 poiché $R + N(3) + C1$ (1 o 0) = 2 o 12 .
 - - Dalla somma della colonna più a destra $2 + D = Y$ oppure $2 + D = 10 + Y$.
- Nuovo tentativo: $C1 = 1 \implies$ vicolo cieco;
- $C1 = 0$ ecc.....
- Nota: Si richiede alle regole di propagazione dei vincoli di non essere sbagliate, non di essere complete. Trade-off fra tempo di propagazione dei vincoli e tempo di ricerca.

ESEMPIO: LE OTTO REGINE

- Nella risoluzione del problema delle otto regine supponiamo di istanziare sei variabili ai valori:
 - $(X_1, X_2, X_3, X_4, X_5, X_6) = (1, 3, 5, 7, 2, 4)$
- con il seguente procedimento:
 - L'assegnazione $X_1=1$ è la prima scelta fatta.
 - Se ora si assegnasse alla seconda variabile il valore $X_2=2$ verrebbe violato il vincolo
 - $V_{k+1} \neq V_i - (k+1-i)$ per $1 \leq i \leq k$.
 - Infatti la seconda regina verrebbe posizionata sulla stessa diagonale della prima. Allora viene tentato l'assegnamento $X_2=3$ che verifica con successo i vincoli:
 - $3 \neq 1$;
 - $1 \neq 3 \neq 8$;
 - $3 \neq 1 + (2 - 1)$;
 - $3 \neq 1 - (2 - 1)$.
 - Per la terza variabile è ancora disponibile il valore 2 che però viola il vincolo sulla diagonale con la seconda variabile, infatti: $2 \neq 3 - (3 - 2)$ non è soddisfatto.

ESEMPIO: LE OTTO REGINE

- Allora si procede all'assegnazione del valore 4 alla variabile X_3 che risulta però incompatibile con il valore assegnato alla variabile X_2 a causa del vincolo sulla diagonale.
- Si tenta allora l'istanziatura $X_3=5$ che ha successo verificando i vincoli:

$$\begin{aligned}5 &\neq 1 ; 5 \neq 3; & 1 &\neq 5 \neq 8 ; \\5 &\neq 1 + (3 - 1) ; 5 \neq 3 + (3 - 2) ; \\5 &\neq 1 - (3 - 1) ; 5 \neq 3 - (3 - 2) .\end{aligned}$$

- Procediamo ora all'istanziatura della quarta variabile al valore $X_4=7$ che soddisfa i vincoli:

$$\begin{aligned}7 &\neq 1 ; 7 \neq 3 ; 7 \neq 5 ; & 1 &\neq 7 \neq 8 ; \\7 &\neq 1 + (4 - 1) ; 7 \neq 3 + (4 - 2) ; & 7 &\neq 5 + (4 - 3) \\7 &\neq 1 - (4 - 1) ; 7 \neq 3 - (4 - 2) ; & 7 &\neq 5 - (4 - 3)\end{aligned}$$

- Procedendo in questo modo si può facilmente verificare che anche le istanziazioni $X_5=2$ e $X_6=4$ soddisfano tutti i vincoli e quindi vengono accettate dallo Standard Backtracking.

PROBLEMA

- L'ultima colonna, corrispondente alla variabile X_8 , non ha più posizioni disponibili assegnabili alla regina.
- Lo Standard Backtracking procede comunque all'assegnazione della settima variabile all'ultimo valore ancora disponibile $X_7=6$, ne controlla la compatibilità con i valori già assegnati alle variabili $X_1, X_2, \dots, X_6$ e procede all'istanziamento dell'ultima regina.
- Solo dopo si 'accorge' di non avere più caselle disponibili e quindi fallisce.
- La ricerca procede anche nel caso in cui una variabile ancora libera, nel nostro caso l'ultima, non presenta più posizioni disponibili.

LIMITI DELL'USO A POSTERIORI DEI VINCOLI

- Questo è un difetto da attribuire a tutti i metodi che utilizzano i vincoli passivamente cioè posteriormente ad un tentativo di istanziamento.
- Utilizzando anche i vincoli che coinvolgono variabili ancora libere il problema sarebbe stato rilevato in anticipo evitando così costosi backtracking.
- L'idea che sta alla base delle tecniche di consistenza consiste in un utilizzo attivo dei vincoli nella guida della computazione e nel cosiddetto *pruning* a priori dell'albero decisionale associando, a ciascuna variabile, l'insieme di valori ammissibili rimanenti dopo ogni assegnazione.
- Questi insiemi (domini) vengono perciò ridotti nel corso della computazione permettendo di scegliere per le variabili ancora libere valori ammissibili con le variabili già istanziate senza più considerare i vincoli che le legano.

FORWARD CHECKING

- Viene utilizzata, dopo ogni assegnamento, la propagazione dei vincoli che consiste nell'eliminazione dei valori incompatibili con quello appena istanziato dai domini delle variabili non ancora istanziate.
- Questo metodo si rivela molto efficace soprattutto quando le ultime variabili ancora libere sono associate ad un insieme di valori ammissibili ridotto e perciò risultano molto vincolate e facilmente assegnabili.
 - Se il dominio associato ad una variabile libera presenta un solo valore l'assegnamento può essere effettuato senza sforzo computazionale.
 - Se ad un certo punto della computazione ci si accorge che un dominio associato ad una variabile risulta vuoto il meccanismo del Forward Checking fallisce senza proseguire in tentativi e backtracking.
- L'assegnazione di un valore ad una variabile ha ripercussioni sull'insieme dei valori disponibili per le variabili ancora libere. In questo modo i vincoli agiscono in avanti (*forward*) e limitano lo spazio delle soluzioni prima che vengano effettuati tentativi su di esso.

FORWARD CHECKING

- L'esempio che porta all'assegnazione dei valori (1,3,5,7,2,4) alla sequenza di variabili ($X_1, \dots, X_6$).
- Inizialmente l'insieme dei domini di tutte le variabili contiene gli interi compresi tra 1 e 8.
- Dopo l'istanziamento di X_1 al valore 1 vengono eliminati, dagli insiemi relativi a $X_2, \dots, X_8$, tutti i valori incompatibili con questo. Ovviamente il valore 1 viene tolto da tutti i domini a causa del vincolo

$$X_1 \neq X_i \text{ per } i = 2, \dots, 8.$$

- Dopo l'istanziamento del valore 1 alla variabile X_1 , vengono ridotti gli insiemi dei valori ammissibili per le variabili ancora libere. Infatti X_2 ha, come valori ammissibili (3,4,5,6,7,8) in quanto il valore 2 risulta incompatibile con l'assegnazione già effettuata e pertanto viene eliminato.

FORWARD CHECKING

- Procedendo con questo ragionamento elenchiamo gli insiemi associati a ciascuna variabile:
 - X_2 è associata al dominio $D_2=(3,4,5,6,7,8)$,
 - X_3 è associata al dominio $D_3=(2,4,5,6,7,8)$,
 - X_4 è associato al dominio $D_4=(2,3,5,6,7,8)$,
 - X_5 è associato al dominio $D_5=(2,3,4,6,7,8)$,
 - X_6 è associato al dominio $D_6=(2,3,4,5,7,8)$,
 - X_7 è associato al dominio $D_7=(2,3,4,5,6,8)$,
 - X_8 è associato al dominio $D_8=(2,3,4,5,6,7)$.
- Ora si procede istanziando X_2 al valore 3. I domini delle variabili libere diventano:
 - X_3 è associata al dominio $D_3=(5,6,7,8)$,
 - X_4 è associato al dominio $D_4=(2,6,7,8)$,
 - X_5 è associato al dominio $D_5=(2,4,7,8)$,
 - X_6 è associato al dominio $D_6=(2,4,5,8)$,
 - X_7 è associato al dominio $D_7=(2,4,5,6)$,
 - X_8 è associato al dominio $D_8=(2,4,5,6,7)$.

FORWARD CHECKING

- Ora si prosegue nella ricerca assegnando un valore a X_3 (il valore 5) e si propagano i vincoli ottenendo:
 - X_4 è associato al dominio $D_4=(2,7,8)$,
 - X_5 è associato al dominio $D_5=(2,4,8)$,
 - X_6 è associato al dominio $D_6=(4)$,
 - X_7 è associato al dominio $D_7=(2,4,6)$,
 - X_8 è associato al dominio $D_8=(2,4,6,7)$.
- Per la variabile X_6 il dominio contiene ora un solo valore. Si procede all'istanziamento della variabile X_4 al valore 2, il primo appartenente al suo dominio.
- Gli insiemi di valori rimanenti:
 - X_5 è associato al dominio $D_5=(4,8)$,
 - X_6 ha dominio D_6 vuoto,
 - X_7 è associato al dominio $D_7=(4,6)$,
 - X_8 è associato al dominio $D_8=(4,7)$.
- Si ha quindi il fallimento in seguito all'assenza di valori ammissibili per la variabile X_6 .

FORWARD CHECKING

- Per la variabile X_6 il dominio contiene ora un solo valore. Si procede all'istanziamento della variabile X_4 al valore 2, il primo appartenente al suo dominio.
- Gli insiemi di valori rimanenti:
 - X_5 è associato al dominio $D_5=(4,8)$,
 - X_6 ha dominio D_6 vuoto,
 - X_7 è associato al dominio $D_7=(4,6)$,
 - X_8 è associato al dominio $D_8=(4,7)$.
- Si ha quindi il fallimento in seguito all'assenza di valori ammissibili per la variabile X_6 .

Forward Checking : l'esempio (continua)

- Il backtracking porta alla ritrazione del valore assegnato all'ultima variabile istanziata, della propagazione causata da quest'ultimo e al successivo tentativo con il valore $X_4=7$.
- L'algoritmo restringe i domini nel modo seguente:
 - X_5 è associato al dominio $D_5=(2,4)$,
 - X_6 è associato al dominio $D_6=(4)$,
 - X_7 è associato al dominio $D_7=(6)$,
 - X_8 è associato al dominio $D_8=(4,6)$.
- Poi si procede all'istanziamento della variabile X_5 al valore 2 e X_6 al valore 4 eliminando ogni possibile istanziamento per la variabile X_8 .

LOOK AHEAD

- La tecnica più completa per quel che riguarda il *pruning* a priori dell'albero decisionale.
- Ad ogni istanziazione viene controllata, come per il Forward Checking, la compatibilità dei vincoli contenenti la variabile appena assegnata con le precedenti (istanziate) e le successive (libere).
- In più viene sviluppato il *look ahead* (sguardo in avanti) che controlla l'esistenza, nei domini associati alle variabili ancora libere, di valori compatibili con i vincoli contenenti solo variabili non istanziate.
- I domini associati a ogni variabile vengono ridotti propagando anche le relazioni contenenti coppie di variabili non istanziate. Viene verificata quindi la possibilità di una futura assegnazione consistente fra le variabili libere.

LOOK AHEAD

- Tuttavia, la propagazione di vincoli contenenti variabili ancora libere si differenzia nel caso in cui si adotti una strategia di tipo Partial Look Ahead (PLA) o Full Look Ahead (FLA). Nel primo caso (PLA) si ha una propagazione dei vincoli contenenti la variabile X_h , non ancora istanziata e le variabili "più future", ossia le variabili $X_{h+1}, \dots, X_n$
- Per ogni variabile non ancora assegnata $X_{k+2}, \dots, X_n$, un valore per il quale sia possibile trovare, per tutte le altre variabili "successive" non ancora assegnate, almeno un valore compatibile con esso.
- Nel secondo caso (FLA), se V_k è il valore appena assegnato alla variabile X_k , si ha una propagazione dei vincoli contenenti la variabile X_h , non ancora istanziata, e tutte le variabili non ancora assegnate, ossia le variabili $X_{k+1}, \dots, X_{h-1}, X_{h+1}, \dots, X_n$.
- Quest'ultima proprietà è soddisfatta se esiste, per ogni variabile non ancora assegnata $X_{k+2}, \dots, X_n$ un valore per il quale sia possibile trovare, per tutte le variabili non ancora assegnate, almeno un valore compatibile con esso.

ESEMPIO DI PLA: LE OTTO REGINE

- Supponiamo di avere assegnato alle prime tre variabili X_1, X_2, X_3 rispettivamente i valori 1,3,5.
- Domini restanti (identici a quelli ridotti dalla tecnica del Forward Checking):
 - X_4 è associato al dominio $D_4=(2,7,8)$,
 - X_5 è associato al dominio $D_5=(2,4,8)$,
 - X_6 è associato al dominio $D_6=(4)$,
 - X_7 è associato al dominio $D_7=(2,4,6)$,
 - X_8 è associato al dominio $D_8=(2,4,6,7)$.
- Eliminiamo quei valori per i quali non esiste, nei domini associate alle variabili "più future" almeno un posizionamento compatibile con il valore estratto.
- Quindi: Estraiamo da D_4 il valore 2. I valori 4 in D_5 , 6 in D_7 e 4 in D_8 sono compatibili con il valore 2 di X_2 mentre il valore 4, unico assegnabile a X_6 , non è compatibile. Dal dominio D_4 viene eliminato il valore 2.
- Estraiamo da D_4 il valore 7. I valori 2 in D_5 , 4 in D_6 , 6 in D_7 e 6 in D_8 sono compatibili con il valore 7 che pertanto non va eliminato dal dominio come, del resto, il valore 8. Il dominio associato a X_4 diventa pertanto: $D_4=(7,8)$.

ESEMPIO DI PLA: LE OTTO REGINE

- Si estrae da D_5 un valore e si controlla che esista, per D_6 , D_7 e D_8 , almeno un valore compatibile con questo. Per l'estrazione 2 in D_5 esistono i valori 4 in D_6 , 6 in D_7 e 6 in D_8 che rispettano tutti i vincoli.
- Estraendo il valore 4 in D_5 si vede subito che l'unico valore associato a X_6 non ha più alcuna possibilità di soddisfare i vincoli quindi da D_5 si può eliminare il valore 4.
- Non dà problemi invece il valore 8 in D_5 per la presenza dei valori 4 in D_6 , 2 in D_7 e 2 in D_8 che rispettano tutti i vincoli. Il dominio associato a X_5 diventa pertanto:

$$D_5=(2,8).$$

- Gli insiemi D_6 e D_7 non vengono modificati.
- Evitiamo quindi l'assegnazione, tentata dal Forward Checking, alla variabile X_4 del valore 2 che falliva immediatamente a causa di D_6 , che rimaneva vuoto.

ESEMPIO DI FLA: LE OTTO REGINE

- Supponiamo di avere già propagato i domini con la tecnica del PLA.
 - X_4 è associato al dominio $D_4=(7,8)$,
 - X_5 è associato al dominio $D_5=(2,8)$,
 - X_6 è associato al dominio $D_6=(4)$,
 - X_7 è associato al dominio $D_7=(2,4,6)$,
 - X_8 è associato al dominio $D_8=(2,4,6,7)$.
- Si noti che per il valore 8 in D_5 , non esiste alcun valore appartenente a D_4 che soddisfi i vincoli imposti dal problema.
- Il PLA non si "accorge" di questa inconsistenza perchè verifica la consistenza dei valori appartenenti ad un dominio D_i con i valori appartenenti ai domini D_j solo se $j>i$.
- Nel dominio D_5 , pertanto, resta il valore 2. Il valore 4 in D_6 risulta compatibile con 4 in D_6 e con i valori 7,8 in D_4 .
 - X_4 è associato al dominio $D_4=(7,8)$,
 - X_5 è associato al dominio $D_5=(2)$,
 - X_6 è associato al dominio $D_6=(4)$,

ESEMPIO DI FLA: LE OTTO REGINE

- Analizziamo ora il dominio D_7 : il valore 2 in D_7 non è compatibile con 2 in D_5 (unico valore di D_5) quindi viene eliminato. La stessa cosa avviene per 4 in D_7 incompatibile con 2 in D_5 .
 - D_7 risulta pertanto: $D_7 = \{6\}$
- Il dominio D_8 , già a questo punto della computazione, non contiene più valori compatibili con quelli dei domini precedenti. Infatti il valore 2 in D_8 risulta incompatibile con 2 in D_5 , 4 in D_8 risulta incompatibile con 4 in D_6 , 6 in D_8 risulta incompatibile con 6 in D_7 . La computazione, pertanto, fallisce senza tentare ulteriori assegnazioni.
- Il carico computazionale dovuto alle continue verifiche della consistenza dei vincoli, e quindi alla propagazione piuttosto pesante, non porta al raggiungimento di vantaggi quando le dimensioni del problema diventano considerevoli ai primi livelli dell'albero.
 - Nell'esempio delle otto regine infatti i domini ridotti, dopo le prime due istanziazioni, dalle tecniche in esame sono identici ma, mentre il Look Ahead verifica la consistenza dei vincoli su tutte le coppie di variabili ancora libere (la maggioranza), il Forward Checking esegue molte meno verifiche guadagnando in efficienza.

PROBLEMA DELLE 8 REGINE

- Standard Backtracking in Prolog

```
solution(X) :-  
    queens(X, [], [1,2,3,4,5,6,7,8]).
```

```
queens([], Placed, []).  
queens([X|Xs], Placed, Values) :-  
    delete1(X, Values, NewValues),  
    noattack(X, Placed),  
    queens(Xs, [X|Placed], NewValues).
```

- La `delete1/3` istanzia la variabile `X` a un valore contenuto nel suo dominio.
- La `noattack/2` controlla (a posteriori) che il valore scelto per `X` sia compatibile con le variabili già istanziate.

FORWARD CHECKING IN PROLOG

```
queens ([X1,X2,X3,X4,X5,X6,X7,X8], [1,2,3,4,5,6,7,8]) :-  
    L=[1,2,3,4,5,6,7,8],  
    queens_aux ([X1,L], [X2,L], [X3,L], [X4,L],  
                [X5,L], [X6,L], [X7,L], [X8,L])).
```

```
queens_aux ([]).
```

```
queens_aux ([X1,D] | Rest) :-  
    member(X1,D), % istanzia la variabile X1  
    forward(X1,Rest,Newrest), %propagazione  
    queens_aux(Newrest).
```

```
forward(X,Rest,Newrest) :-  
    forward(X,Rest,1,Newrest).
```

```
forward(X,[],Nb,[]).
```

```
// segue prossima
```

FORWARD CHECKING IN PROLOG

```
forward(X, [[Var,Dom]|Rest],Nb, [[Var,[F|T]]|Newrest]):-  
    remove_value(X,Dom,Nb,[F|T]),  
    Nb1 is Nb +1,  
    forward(X,Rest,Nb1,Newrest) .
```

```
remove_value(X,[],Nb,[]) .
```

```
remove_value(X,[Val|Rest],Nb,[Val|Newrest]):-  
    compatible(X,Val,Nb), !,  
    remove_value(X,Rest,Nb,Newrest) .
```

```
remove_value(X,[Val|Rest],Nb,Newrest):-  
    remove_value(X,Rest,Nb,Newrest) .
```

```
compatible(Value1,Value2,Nb):-  
    Value1 =\= Value2 +Nb,  
    Value1 =\= Value2 - Nb,  
    Value1 =\= Value2 .
```

CSP ED EURISTICHE

- La scelta dell'ordinamento delle variabili e scelta dell'ordine di selezione dei valori) rimangono a disposizione del programmatore e possono essere gestiti attraverso la procedura di labeling.
- Le euristiche potranno agire quindi su questi due gradi di libertà per cercare di garantire il raggiungimento di una buona soluzione in tempi ragionevoli anche per i problemi più complessi.
- Due principi di euristica comunemente accettati sono il First-Fail Principle (FFP) e il Least Constraining Principle (LCP).
 - Il First-Fail Principle consiglia di risolvere prima i sottoproblemi più difficili che quindi hanno più probabilità di portare a fallimenti.
 - Il Least Constraining Principle consiglia invece di fare le scelte meno vincolanti per il sottoproblema rimanente.

FIRST FAIL

- Il Forward Checking e le tecniche di look-ahead sono molto efficienti ma non permettono, al programmatore, di scegliere quale variabile istanziare successivamente.
- La tecnica del First Fail si basa sulla conoscenza di algoritmi euristici che permettono di definire quale variabile risulta più conveniente da istanziare.
- Il First Fail propone l'istanziamento, a ogni passo, della variabile più vincolata cioè della variabile che, più velocemente, può portare al fallimento.
- Nell'esempio precedentemente illustrato per l'algoritmo Forward Checking dopo le istanziazioni
 $X_1=1, X_2=3, X_3=5$
proseguirebbe con la scelta per la variabile X_6 perché il suo dominio è il più ristretto.
- Oltre alla scelta della variabile da istanziare è molto importante anche la scelta del valore da assegnare tra quelli disponibili. Esistono tecniche euristiche per questa scelta.

CLASSIFICAZIONE DELLE EURISTICHE

Le euristiche possono essere classificate in:

- euristiche per la selezione della variabile:
 - determinano quale deve essere la prossima variabile da istanziare. Le due euristiche più comunemente usate sono il first-fail che scegliere la variabile con il dominio di cardinalità minore, e il most-constraining principle che sceglie la variabile legata a più vincoli. Entrambe queste euristiche decidono di istanziare prima le variabili più difficili da assegnare.
- euristiche per la selezione del valore:
 - determinano quale valore assegnare alla variabile selezionata. Si segue in genere il principio di scegliere prima il valore che si ritiene abbia più probabilità di successo (least constraining principle).

CLASSIFICAZIONE DELLE EURISTICHE

Un'ulteriore classificazione è la seguente:

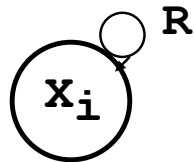
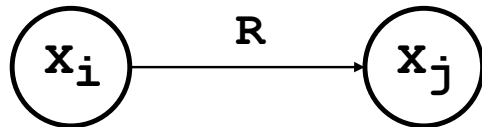
- euristiche statiche:
 - determinano l'ordine in cui le variabili (o i valori) vengono scelti prima di iniziare la ricerca; tale ordine rimane invariato durante tutta la ricerca.
- euristiche dinamiche:
 - scelgono la prossima selezione da effettuare ogni volta che una nuova selezione viene richiesta (quindi ad ogni passo di labeling).
- Le euristiche dinamiche sono potenzialmente migliori (meno backtracking). La determinazione dell'euristica perfetta (che non richiede backtracking) e' un problema che ha, in genere, la stessa complessità del problema originale. Bisognerà quindi trovare un compromesso.

TECNICHE DI CONSISTENZA

- Differenza fondamentale: al contrario degli algoritmi di propagazione che propagano i vincoli in seguito a istanziazioni delle variabili coinvolte nel problema, le tecniche di consistenza riducono il problema originale eliminando dai domini delle variabili i valori che non possono comparire in una soluzione finale.
- Tutte le tecniche di consistenza sono basate su una rappresentazione del problema come una rete (grafo) di vincoli. Gli archi possono essere orientati o non orientati: ad esempio il vincolo $>$ viene rappresentato da un arco orientato, mentre il vincolo $\neq$ da un arco semplice (non orientato o doppiamente orientato).

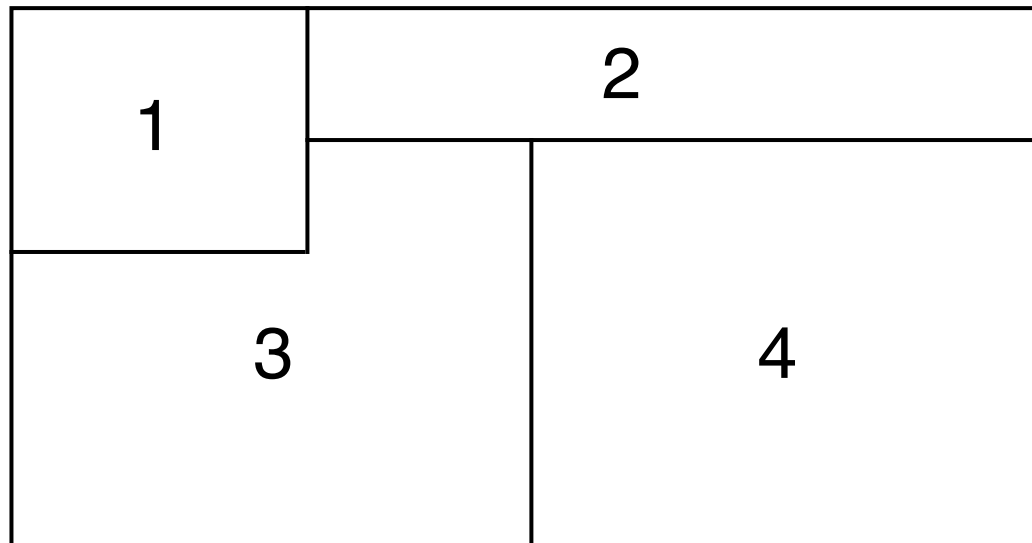
CONSTRAINT GRAPH

- Per ogni CSP esiste un grafo (constraint graph) in cui i nodi rappresentano le variabili e gli archi i vincoli tra le variabili costituenti i nodi del grafo.
 - I vincoli binari (R) collegano due nodi X_i e X_j :
 - I vincoli unari sono rappresentati da archi che iniziano e terminano sullo stesso nodo X_i



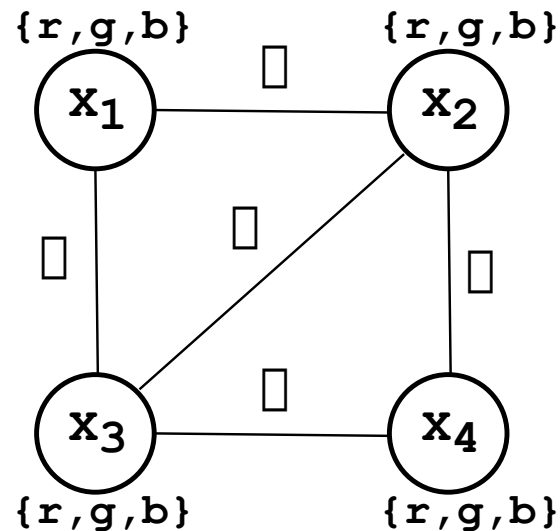
ESEMPIO: MAP COLORING PROBLEM

- Supponiamo di dover colorare delle porzioni di un piano, caratterizzate da un numero, in modo tale che due regioni contigue siano colorate da colori diversi. Supponiamo anche di aver a disposizione i colori red (r), green (g) e blu (b)

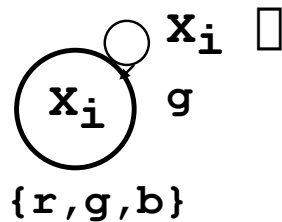


ESEMPIO: MAP COLORING PROBLEM

- Il constraint-graph corrispondente è il seguente. tuttavia, esistono combinazioni di valori non compatibili tra loro (es: $X_1=r$, $X_2=r$, $X_3=r$, $X_4=r$).
- Esistono diversi algoritmi che realizzano gradi diversi di consistenza.

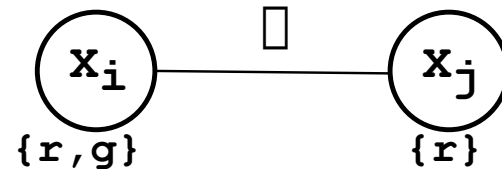


NODE CONSISTENCY



- NODE-CONSISTENCY: consistenza di grado 1
 - Un nodo di un grafo di vincoli è consistente se per ogni valore $X_i \in D_i$ i vincoli unari su X_i sono soddisfatti.
- Nell'esempio il nodo non è node consistent perchè il valore $g \in D_i$ viola il vincolo unario $P(i)$ su X_i .
- Per rendere il nodo consistente è necessario eliminare dal dominio di X_i il valore g .
- Un grafo è node consistente se tutti i suoi nodi sono consistenti.

ARC CONSISTENCY

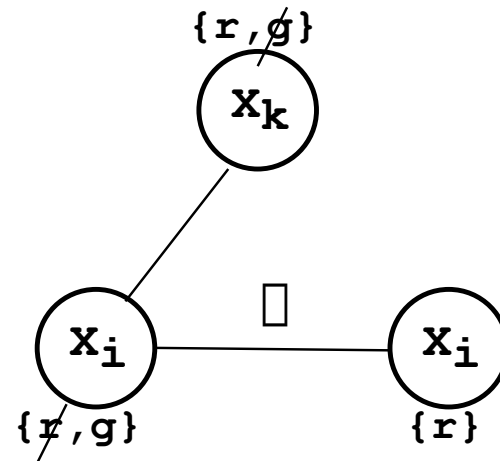


- La consistenza di grado 2 si ottiene partendo da un grafo node-consistente. Tale consistenza verifica se un arco $A(i,j)$ è consistente.

• **ARC CONSISTENCY:** un arco $A(i,j)$ è consistente se per ogni valore $X \in D_i$ esiste almeno un valore $Y \in D_j$ tale che il vincolo tra i e j $P(i,j)$ sia soddisfatto

- L'arco in figura non è consistente perché, considerando il valore $r \in D_i$, non esiste un valore appartenente a D_j che soddisfi il vincolo $P(i,j)$.
- Per rendere consistente l'arco tra x_i e x_j è necessario eliminare il valore r dal dominio di x_i : questo valore non comparirebbe in nessuna soluzione ammissibile.

PROCEDIMENTO ITERATIVO



- La rimozione di alcuni valori dal dominio di una variabile rende necessarie ulteriori verifiche che coinvolgono i vincoli contenenti la variabile stessa.
- Quindi questo procedimento deve essere ripetuto finché la rete non raggiunge una configurazione stabile, detta situazione di QUIESCENZA
- La rimozione del valore $X_i = r$ a causa del vincolo $\square$ tra X_i e X_j comporta la rimozione del valore $X_k = g$ a causa del vincolo tra X_i e X_k
- Quindi l'arc consistency è un procedimento iterativo che converge ad una rete stabile e arc-consistente.

ARC CONSISTENCY IN PROLOG

```
ac(VarList, ArcList, NewVarList):-
    ac(VarList,ArcList,ArcList,0,NewVarList).
ac(VarList,OrigArcList,ArcList,Bit,NewVarList):-
    select((V1,V2,P), ArcList, NewArcList),
    revise(V1,V2,P,VarList, TempVarList1),
    revise(V2,V1,P,TempVarList1,TempVarList2),
    (Bit=0,same_set(VarList,TempVarList2) ->
        NewBit=0; NewBit=1),
    (NewArcList=[], NewBit=0 ->
        NewVarList=TempVarList2;
    (NewArcList=[] ->
        ac(TempVarList2, OrigArcList,
            OrigArcList,0, NewVarList);
    ac(TempVarList2, OrigArcList,
        NewArcList,NewBit,
    NewVarList))) .
```

ARC CONSISTENCY IN PROLOG

```
revise(V1,V2,P,VarLIn,[ (V1,NewDL) | RestVars]) :-
    select((V1,D1List), VarLIn, RestVars),
    member((V2,D2List), RestVars),
    findall(D1, (member(D1,D1List), member(D2,D2List),
                apply(P,D1,D2,V1,V2)), NewDL1),
    remove_duplicates(NewDL1, NewDL),
    (NewDL=[] ->
        (writeln('No consistent assignment to',
                D1),
         !, fail); true).

same_set([], []).
same_set([X|Tail], [X|Tail1]) :-
    same_set(Tail, Tail1).

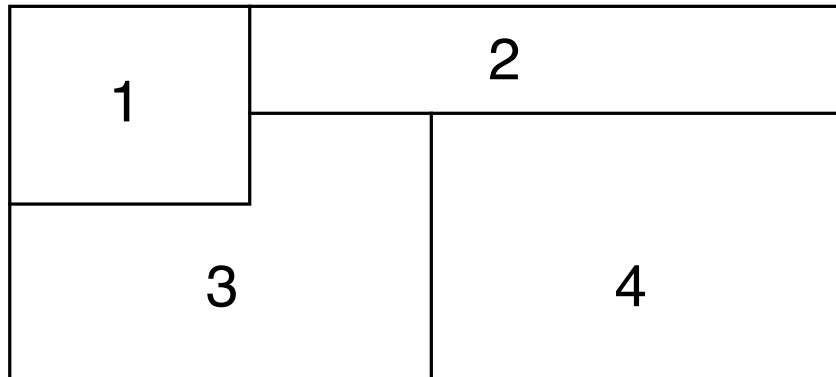
apply(P,D1,D2,V1,V2) :-
    Claus=..[P,D1,D2,V1,V2],
    call(Claus).
```

ARC CONSISTENCY IN PROLOG

```
select(Element,[Element|Tail], Tail).  
select(Element,[Head|Tail1], [Head|Tail2]):-  
    select(Element, Tail1, Tail2).
```

- **ac/5** realizza l'arc-consistency selezionando, tramite la clausola **select/3**, una coppia di variabili. In un secondo tempo, **revise/5** realizza l'arc-consistency eliminando dai domini delle variabili selezionate i valori incompatibili con i vincoli.
- La findall contenuta nella **revise/5** applica, per ciascun elemento dei domini delle variabili considerate, **apply/5**. La **apply/5** deve essere specializzata per i vincoli del problema. Infatti, questa versione generale dell'arc-consistency può essere specializzata per diversi problemi.

MAP COLORING PROBLEM



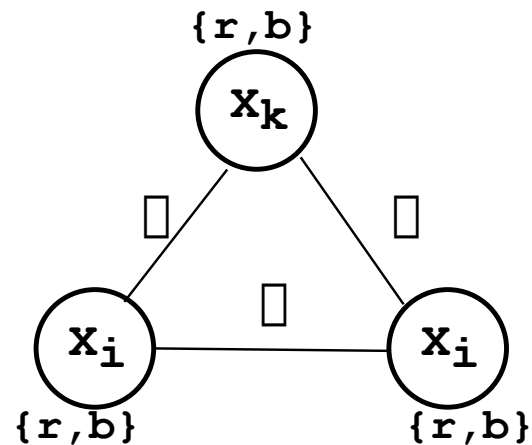
```
apply(noattack,D1,D 2,V1,V2) .  
noattack(D1,D2,V1,V2) :-  
    (adiac(V1,V2) ; adiac(V2,V1)) , ! , D1\==D2 .  
noattack(_,_,V1,V2) :- not adiac (V1,V2) .  
adiac (1,2) .  
adiac(1,3) .  
adiac(2,3) .  
adiac(2,4) .  
adiac(3,4) .
```

STRUTTURE DATI

- Lista di Variabili:
`[(X1, [v1, v2, ..., vn]), (X2, [k1, k2, ..., km]), ...]`
- Lista di Archi:
`[(X1, X2, vincolo), .....]`
- In `ac/5` la variabile `Bit` indica la condizione di stabilità `findall (X,P,L)`.
produce una lista di oggetti che soddisfano `P`.
- Differenza rispetto a `bagof`: tutti gli oggetti di `x` sono collezionati, non considerando che si possono istanziare in modo diverso altre variabili di `P`.
`age(peter,7) . age(ann,5) . age(pat,8) .`
`:- bagof(C,age(C,A),L) .`
`A=7,`
`L=[peter] .`
`:- findall(C,age(C,A),L) .`
`L=[peter, ann, pat] .`

PATH CONSISTENCY

- La consistenza di grado 3 si ottiene partendo da un grafo arc-consistente.
- Supponiamo di considerare la rete di vincoli relativa al map coloring problem:

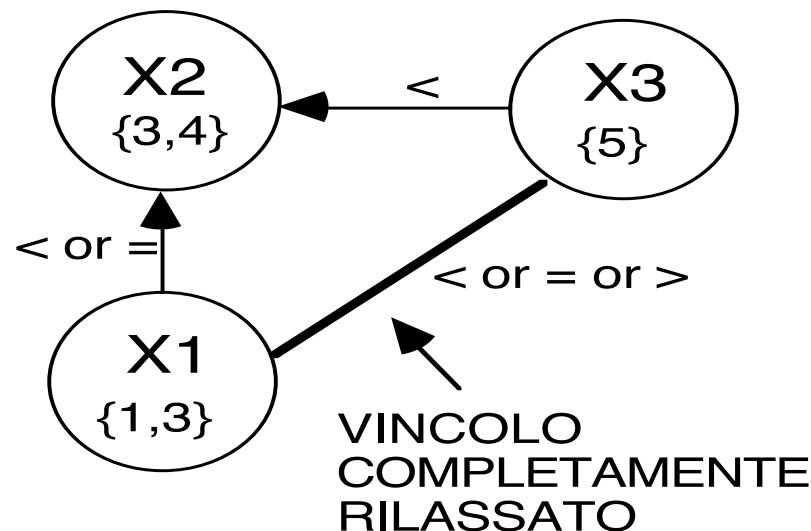


PATH-CONSISTENCY: grado 3

- Questa rete è arc-consistente: infatti, per ogni valore di ciascun dominio, esiste almeno un valore in ogni altro dominio che soddisfa il vincolo esistente tra i due nodi.
- Tuttavia, è immediato verificare che la rete non presenta soluzioni -> non è path-consistente.
- PATH CONSISTENCY: Un cammino tra i nodi (i,j,k) è path consistente se, per ogni valore $x \in D_i$, e $y \in D_j$ (che rispettano la node e la arc-consistenza) esiste un valore $z \in D_k$ che soddisfa i vincoli $P(i,k)$ e $P(k,j)$.
(la consistenza del vincolo unario $P(k)$ è garantita dalla node consistenza della rete)
- Nell'esempio precedente, presi i valori $r \in D_1$ e $b \in D_3$, non esiste nessun valore appartenente a D_2 che soddisfi contemporaneamente i vincoli $P(1,2)$ e $P(2,3)$.

PATH-CONSISTENCY

- Nota: la definizione originale di path-consistenza si riferisce a cammini di lunghezza arbitraria m .
- Tuttavia, esiste il seguente teorema:
- TEOREMA: Se ogni cammino di lunghezza 2 di un grafo completo è path-consistente allora l'intera rete è path-consistente.
- Un constraint-graph non completo può essere reso completo aggiungendo, tra le variabili non vincolate, vincoli completamente rilassati: sempre veri



K-CONSISTENZA

- Scelti valori per ogni $k-1$ -pla di variabili consistenti con i vincoli imposti dal problema, si cerca un valore per ogni k -esima variabile che soddisfa i vincoli tra tutte le k variabili. Se tale valore esiste allora le k variabili sono k consistenti.
- In generale, se un grafo contenente n variabili è k - consistente con $k < n$, allora per trovare una soluzione è necessaria una ricerca nello spazio restante.
- Se un grafo contenente n variabili è n consistente, allora si può trovare una soluzione senza ricerca.
- Freuder nel 1978 ha definito un algoritmo generale per rendere una rete k -consistente con k qualunque.
- Tuttavia, rendere una rete di vincoli contenente n variabili n -consistente ha una complessità esponenziale in n .

I VINCOLI NEI LINGUAGGI

- Linguaggi di programmazione che combinano la dichiaratività della Programmazione Logica e l'efficienza della Risoluzione di Vincoli.
- Limitazioni della programmazione logica:
 - gli oggetti manipolati dai programmi logici sono strutture non interpretate per cui l'unificazione ha successo solo tra oggetti sintatticamente identici.
 - strategia di ricerca del tipo depth-first con backtracking cronologico nello spazio delle soluzioni e conducono a strategie del tipo Generate and Test.

PROGRAMMAZIONE LOGICA A VINCOLI

- Programmazione logica a vincoli: Constraint Logic Programming CLP
- 1989 Jaffar & Lassez
- CLP permette di:
 - associare a ciascun oggetto la sua semantica e le operazioni primitive che agiscono su di esso (domini di computazione quali reali, interi, razionali, booleani e domini finiti di ogni genere).
 - sfruttare procedure di ricerca nello spazio delle soluzioni più intelligenti che conducono ad una computazione guidata dai dati e ad uno sfruttamento attivo dei vincoli.

LO SCHEMA CLP

- Sviluppato nel 1989 da Jaffar e Lassez.
- Aspetto chiave: l'aumento di flessibilità derivante dall'introduzione di oggetti semantici primitivi su cui il linguaggio può inferire.
- L'unificazione è solo un caso particolare di risoluzione di vincoli.
- Superamento di una delle lacune presenti nello schema tradizionale della PL che ne inficia il meccanismo fondamentale dell'unificazione.
- Due termini, intesi come strutture non interpretate, sono unificabili solo se sintatticamente identici.
- In tal modo la struttura 5 e la struttura $2+3$ non sono considerate lo stesso oggetto.
- Predicato `is/2` che consente di rispondere affermativamente alla query `5 is 2+3` e alla query `A is 2+3` con `A=5`, yes ma non si comporta correttamente nel caso in si ponga la query `5 is A+3` rispondendo `A=2`.

LO SCHEMA CLP (continua)

- Lo schema $\text{CLP}(X)$ (dove X è il generico dominio di computazione) permette la definizione di oggetti semantici appartenenti ad X , di operazioni primitive e di relazioni (vincoli) su questi.
 - Estrarre dalle relazioni $Y=2+3$, $5=A+3$ le informazioni corrette sull'istanziamento della variabile libera.
 - Possibilità di trattare relazioni del tipo $X+3=Y-2$.
 - Questo vincolo viene mantenuto in forma implicita fino a quando una istanziamento di una delle due variabili libere fornisce informazioni sull'altra.
- Tra i principali domini di computazione per i quali è stato costruito un constraint solver troviamo i reali e lo schema $\text{CLP}(\mathbb{R})$, i razionali con $\text{CLP}(\mathbb{Q})$, gli interi $\text{CLP}(\mathbb{Z})$, i booleani e i domini finiti.
- Domini come strumento realizzativo delle Tecniche di Consistenza.

LO SCHEMA CLP

- Le Tecniche di Consistenza possono essere realizzate con la PL costruendo programmi strutturati in modo tale da garantire il *pruning* a priori dell'albero decisionale.
- Gli insiemi di valori ammissibili per ciascuna variabile possono essere trattati utilizzando le liste e alcune primitive per agire su di esse.
- Tuttavia le operazioni sulle liste sono molto pesanti computazionalmente.

LO SCHEMA CLP

- Un algoritmo che realizza il Forward Checking con gli strumenti offerti dalla PL risulta, in media, meno efficiente di uno che utilizza lo Standard Backtracking per problemi di dimensione ridotta ($n < 12$).
- Estendere la PL con meccanismi specifici per il trattamento delle Tecniche di Consistenza.
- Introdurre il concetto di *dominio* associato ad una variabile che ne definisca un *range* cioè un insieme di valori ammissibili.
- La realizzazione dei domini deve essere tale da permettere un loro utilizzo efficiente quindi deve essere svincolata dalle liste e dal loro trattamento.

DOMINIO NELLA CLP

- Supponiamo di avere a disposizione una primitiva

`domain(X, D)` che associa alla variabile `X` un insieme di valori possibili contenuto in `D`, e una primitiva

`indomain(X)` che istanzia `X`, in modo backtrackabile, con un elemento del suo dominio.

- Un vincolo su una variabile con dominio agisce sul dominio stesso riducendolo.

DOMINIO NELLA CLP

- Esempio
`:- domain(X, [1,2,3,4,5]), X<3.`
- ha come effetto la riduzione del dominio di x a $[4,5]$ mentre il vincolo $x>7$ conduce al fallimento non essendoci, nel dominio x , alcun valore che soddisfa il vincolo imposto.
- Arricchendo la PL in questo modo è possibile esprimere relazioni tra le variabili con dominio:
`:- domain(X, [1,2,3,4,5]), domain(Y, [3,4,5,6]), X=Y.`
- L'unificazione delle variabili con dominio deve essere opportunamente gestita e la query precedente effettua la riduzione dei domini di x e di y ai soli valori $[3,4,5]$, intersezione dei due domini.

FORWARD CHECKING CON DOMINI

- Molto più leggibile ed efficiente

```
queens ([]) .  
queens ([X|Y]) :-  
    indomain(X) ,  
    noattack(X,Y) ,  
    queens(Y) .  
noattack(X,Y) :-  
    noattack(X,Y,1) .  
noattack(X,[],Nb) .  
noattack(X,[Y|Ys],Nb) :-  
    X  $\neq$  Y ,  
    X  $\neq$  Y-Nb ,  
    X  $\neq$  Y+Nb ,  
    Nb1=Nb+1 ,  
    noattack(X,Ys,Nb1) .
```

- I vincoli contenuti in `noattack/3` agiscono eliminando i valori dai domini di `Y` non consistenti.
- Chiaramente tutte le variabili devono essere dotate di dominio tramite la primitiva