PROBLEMI DI VINCOLI
CSP (Constraint satisfaction problems)

Dato un insieme finito di variabili $X_1, X_2, \ldots, X_n$ i cui valori appartengono a *domini* $D_1, D_2, \ldots, D_n$

e un insieme di *vincoli* $c(X_{i1}, X_{i2}, \ldots, X_{ik})$ che specificano quali valori delle variabili sono compatibili tra loro,

trovare una assegnazione di valori alle variabili che soddisfa i vincoli.

In alcuni casi si chiede di trovare una *assegnazione ottima* (rispetto a qualche criterio)

Esempio: N regine

Disporre, su una scacchiera N X N, N regine in modo che queste non si attacchino fra di loro.
Una regina può muoversi in orizzontale, verticale o diagonale.

Una soluzione può essere rappresentata con una lista di N interi compresi tra 1 e N $[X_1, \ldots, X_N]$, che indicano la posizione delle regine nelle N righe (le regine devono stare necessariamente in righe diverse).

I vincoli sono *binari* (fra coppie di regine).
Per ogni coppia X_i X_j $(i < j)$

$X_i \neq X_j$: colonne diverse

$X_i + (j - i) \neq X_j$
$X_i - (j - i) \neq X_j$: diagonali diverse

Criptoaritmetica

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

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

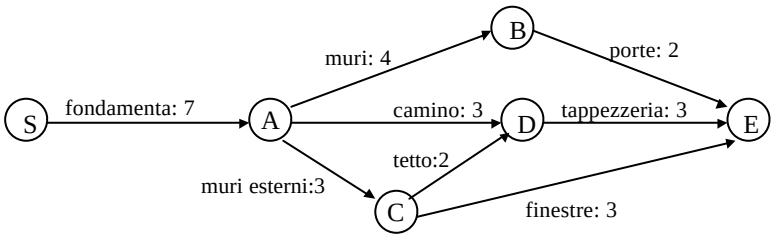
Vincoli:

tutti i valori sono diversi

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

questo vincolo coinvolge tutte le variabili

Scheduling: Costruire una casa



Vincoli:

$T_S = 0$

$T_A \geq T_S + 7$

$T_B \geq T_A + 4$

.....

Problema di ottimizzazione: Trovare la soluzione che minimizza T_E

Tutte le variabili degli esempi hanno **domini finiti**

Altri problemi:

- colorazione di mappe
- interpretazione di immagini (algoritmo di Waltz)
- analisi di circuiti
- orari
- problemi di ricerca operativa (con ottimizzazione)

Come risolvere CSP in Prolog

Soluzione *generate and test*: si genera una proposta di soluzione e si verifica che questa soddisfi i vincoli.

```
queens(N, L) :-
   length(L,N),
   label(L,1,N),
   constrain_all(L).

label([],_,_).
label([X|L],M,N):
   in_domain(X,M,N),
   label(L,M,N).

in_domain(M,M,N).
in_domain(X,M,N) :-
   M<N, M1 is M+1,
   in_domain(X,M1,N).
```

```
constrain_all([]).
constrain_all([X|Xs]) :-
     constrain_between(X,Xs,1),
     constrain_all(Xs).

constrain_between(_X,[],_N).
constrain_between(X,[Y|Ys],N) :-
     no_threat(X,Y,N),
     N1 is N+1,
     constrain_between(X,Ys,N1).

no_threat(X,Y,I) :-
     X =\= Y,
     X+I =\= Y,
     X-I =\= Y.
```

Il metodo *generate and test* è molto inefficiente.

Per rendere la computazione più efficiente conviene verificare i vincoli appena possibile, ossia appena tutte le variabili coinvolte sono istanziate.

Nell'esempio delle N regine, ogni volta che si aggiunge una regina si possono verificare i vincoli che legano questa regina con tutte quelle che sono già state inserite nelle righe precedenti.

Criptoaritmetica

```
mm([S,E,N,D,M,O,R,Y]):-
        label([S,M], [1,2,3,4,5,6,7,8,9]),
        label([E,N,D,O,R,Y], [0,1,2,3,4,5,6,7,8,9]),
        all_diff([S,E,N,D,M,O,R,Y]),
        sum(S,E,N,D,M,O,R,Y).

label([],_).
label([X|L],D):-
        member(X,D), label(L,D).

member(M,[M|_]).
member(M,[_|L]):- member(M,L).

all_diff([]).
all_diff([X|L]):-
        different(X,L),
        all_diff(L).

different(X,[]).
different(X,[Y|L]) :-
        X=\=Y,
        different(X,L).
```

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

```
sum(S,E,N,D,M,O,R,Y) :-
   (1000*S+100*E+10*N+D +
   1000*M+100*O+10*R+E) =:=
   (10000*M+1000*O+100*N+10*E+Y).
```

In questo caso non è possibile anticipare la verifica dei vincoli, perché questi coinvolgono tutte le variabili.

Si può migliorare un po' l'efficienza introducendo i riporti.

D + E = Y + 10 × R4

R4 + N + R = E + 10 × R3

……

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

Il numero delle variabili aumenta, ma i vincoli coinvolgono meno variabili, e quindi possono essere verificati prima.

Altri tipi di problemi con vincoli

Vincoli aritmetici sui reali

Si può modellare un circuito elettrico con variabili reali che rappresentano tensioni, correnti e resistenze.

I vincoli modellano le leggi

di Ohm : $V = I \times R$

e di Kirchhoff

Vincoli booleani

per modellare circuiti logici

Problemi di vincoli

La forma ed il significato dei vincoli dipendono dal **dominio** (reali, interi, booleani, domini finiti, ...).

Un insieme di vincoli su un dominio D è **soddisfacibile** se esiste un assegnamento di valori del dominio D alle variabili che rende veri i vincoli. Altrimenti è **insoddisfacibile**.

Es. Dominio dei **numeri reali**

$X \leq 5 \ \Lambda \ \ Y \geq 3 \ \Lambda \ \ X \geq Y$
è soddisfacibile

$X \leq 5 \ \Lambda \ \ Y \geq 3 \ \Lambda \ \ X > Y + 2$
non lo è

Un algoritmo per determinare la soddisfacibilità di un insieme di vincoli in un certo dominio è detto risolutore di vincoli (**constraint solver**).

Es. vincoli aritmetici lineari:

$e_1 = e_2, \quad e_1 < e_2, \ldots.$

Dove le espressioni e_i hanno la forma $a_1x_1 + a_2 x_2 + \ldots$

Un constraint solver è *completo* se risponde sempre SI o NO.

In molti casi un algoritmo completo è troppo costoso o non esiste.
Ad esempio, la soddisfacibilità di formule booleane è NP-completa.

Un constraint solver *incompleto* può restituire NON SO.

Operazioni importanti su insiemi di vincoli:

Semplificazione: sostituire un insieme di vincoli con uno equivalente più "semplice"

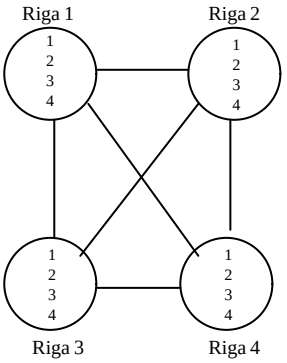
Ottimizzazione: trovare la soluzione "migliore"
Es. algoritmo del simplesso per vincoli aritmetici lineari

Algoritmi di consistenza per domini finiti

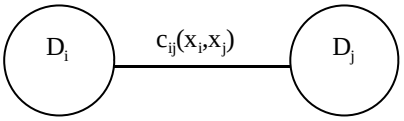
Se i vincoli sono *binari*, un problema di vincoli può essere rappresentato con un grafo:

un nodo per ogni dominio e un arco per ogni vincolo.

Esempio 4 regine



Arc consistency



Un arco (i,j) è *arc consistent* se

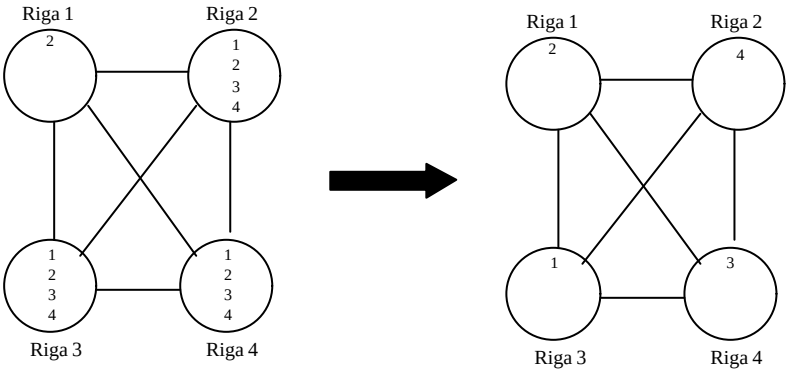
per ogni $x \in D_i$ esiste un $y \in D_j$ tale che $c_{ij}(x,y)$

Si può rendere un arco arc consistent riducendo i domini dei nodi.

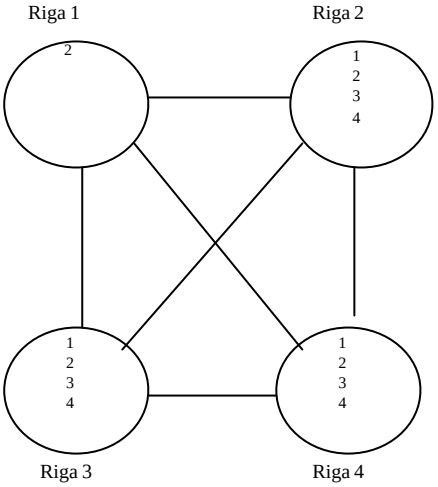
Un grafo è arc consistent se tutti gli archi lo sono.

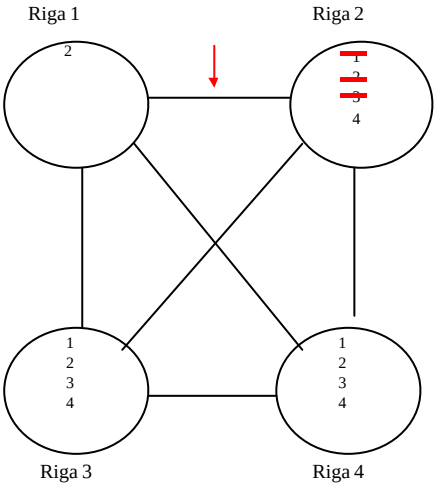
Algoritmo polinomiale per arc consistency: visitare iterativamente tutti gli archi per renderli arc consistent finché non cambia niente.

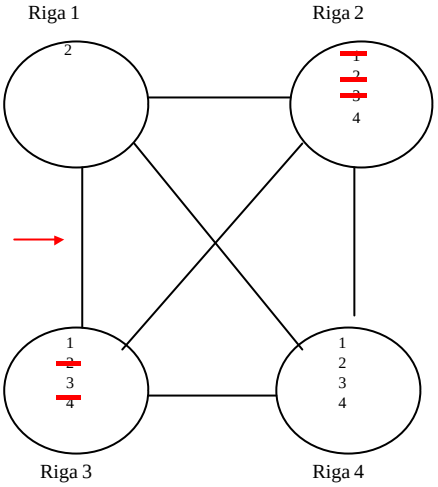
Se alla fine un dominio è vuoto, il problema è insoddisfacibile.

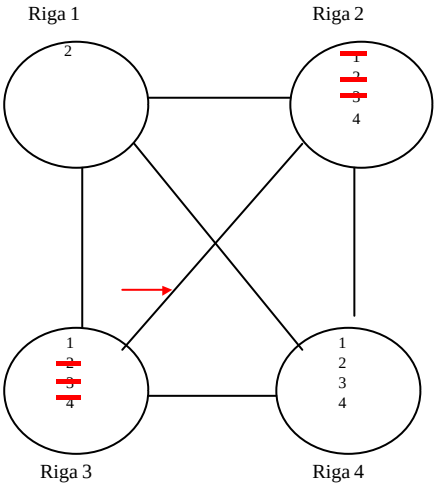


soddisfacibile

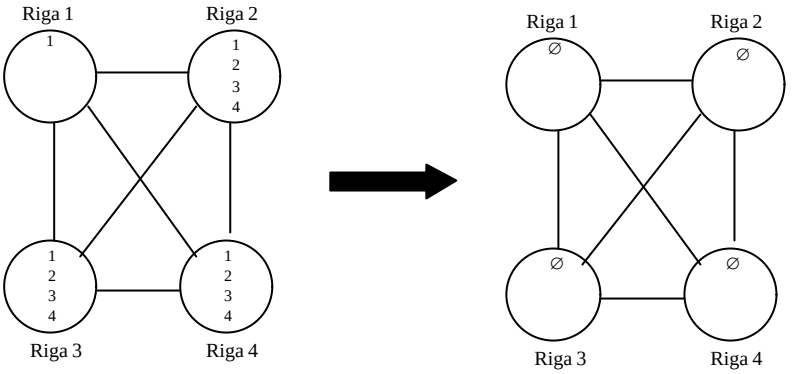

Riga 1
2
Riga 2
1
2
3
4
1
2
3
4
1
2
3
4
Riga 3
Riga 4


Riga 1
2
Riga 2
1
2
3
4
1
2
3
4
1
2
3
4
Riga 3
Riga 4


Riga 1
Riga 2
2
1
2
3
4
1
2
3
4
1
2
3
4
Riga 3
Riga 4


Riga 1
Riga 2
2
1
2
3
4
1
2
3
4
1
2
3
4
Riga 3
Riga 4

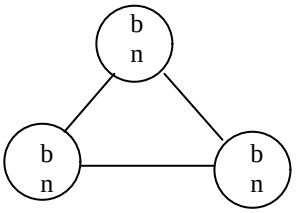
ecc.



insoddisfacibile

L'algoritmo per arc consistency *non è completo*.

Es. colorazione con due colori.



E' arc consistent ma non c'è soluzione.

Si può definire la path consistency, ma l'algoritmo aumento di complessità.

Spesso, per risolvere un problema di vincoli, si combina l'algoritmo di arc consistency con un algoritmo di backtracking.

Se i vincoli non sono binari (vedi SEND + MORE = MONEY) si devono usare altre tecniche.

Ad esempio, se i vincoli sono aritmetici ed i domini sono degli intervalli [i..j], si possono usare tecniche di *bounds consistency*.

Per l'ottimizzazione si usano tecniche di ricerca operativa, come il *branch and bound*.