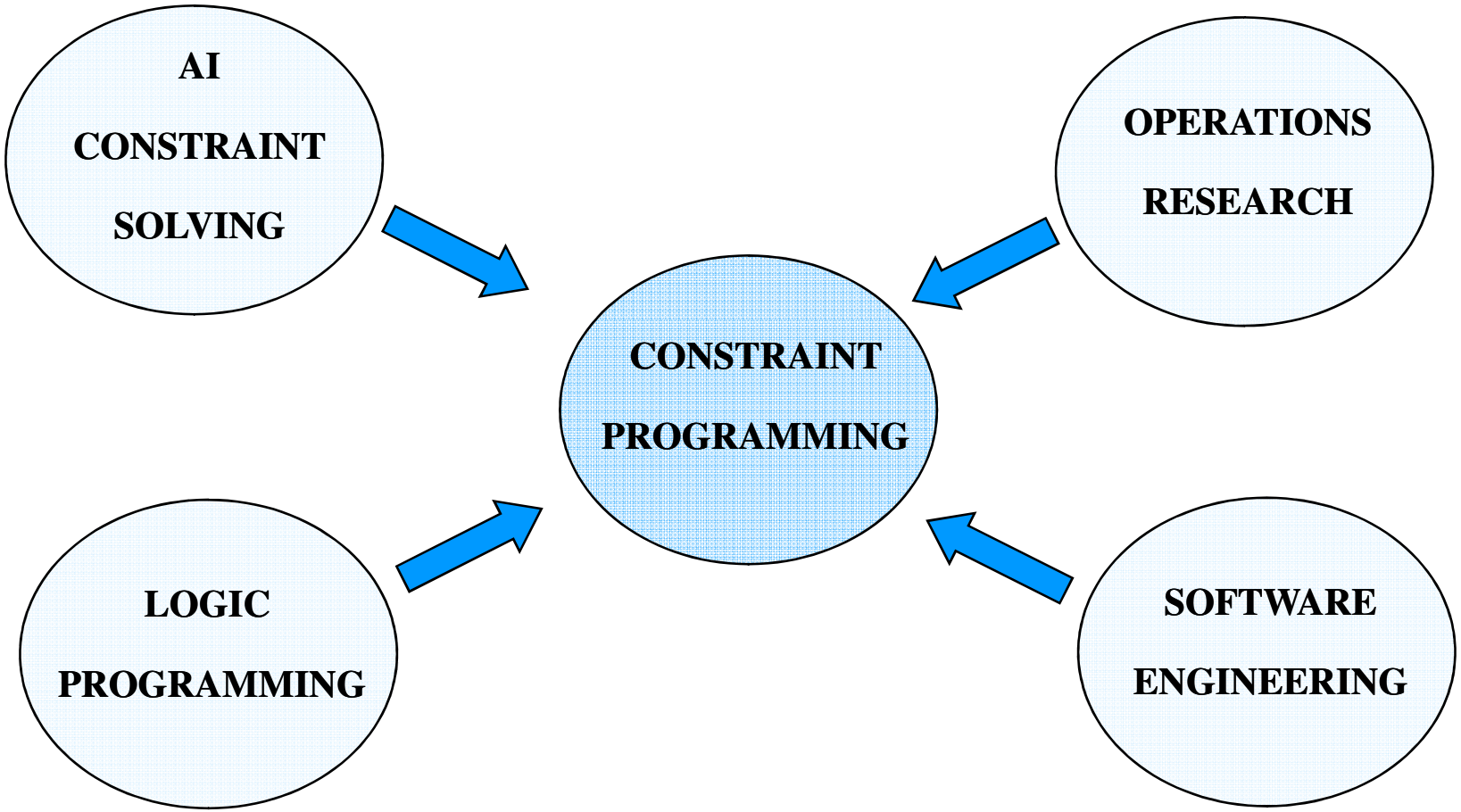
PROGRAMMAZIONE LOGICA A VINCOLI



PROGRAMMAZIONE LOGICA A VINCOLI

- Problemi di soddisfacimento di vincoli:
  - concetti generali
- Programmazione Logica
  - vantaggi
  - limiti
- Programmazione Logica a Vincoli
  - dominio e interpretazione
  - controllo
  - modello computazionale

PROBLEMA DI SODDISFACIMENTO DI VINCOLI

•Un problema di soddisfacimento di vincoli (Constraint Satisfaction Problem – CSP) è definito da:

$$CSP = <X, D, C>$$

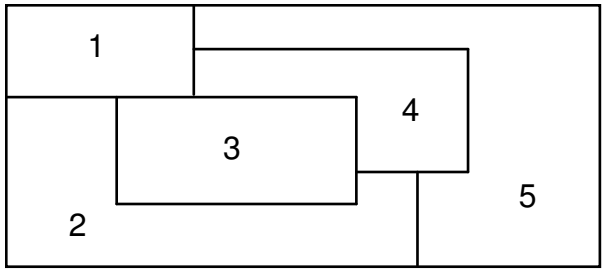
– *un insieme di variabili* $V = \{X_1, X_2 \ldots X_n\}$
– *un dominio discreto per ogni variabile* $D = \{D_1, D_2 \ldots D_n\}$
– *un insieme di vincoli su queste variabili*:

vincolo: una relazione tra variabili che definisce un sottoinsieme del prodotto cartesiano dei domini $D_1 x D_2 x \ldots x D_n$

Soluzione di un Problema di Soddisfacimento di vincoli: un assegnamento di valori alle variabili consistente con i vincoli

E. Tsang: "Foundations of Constraint Satisfaction" Academic Press, 1992.

ESEMPIO: Map Coloring



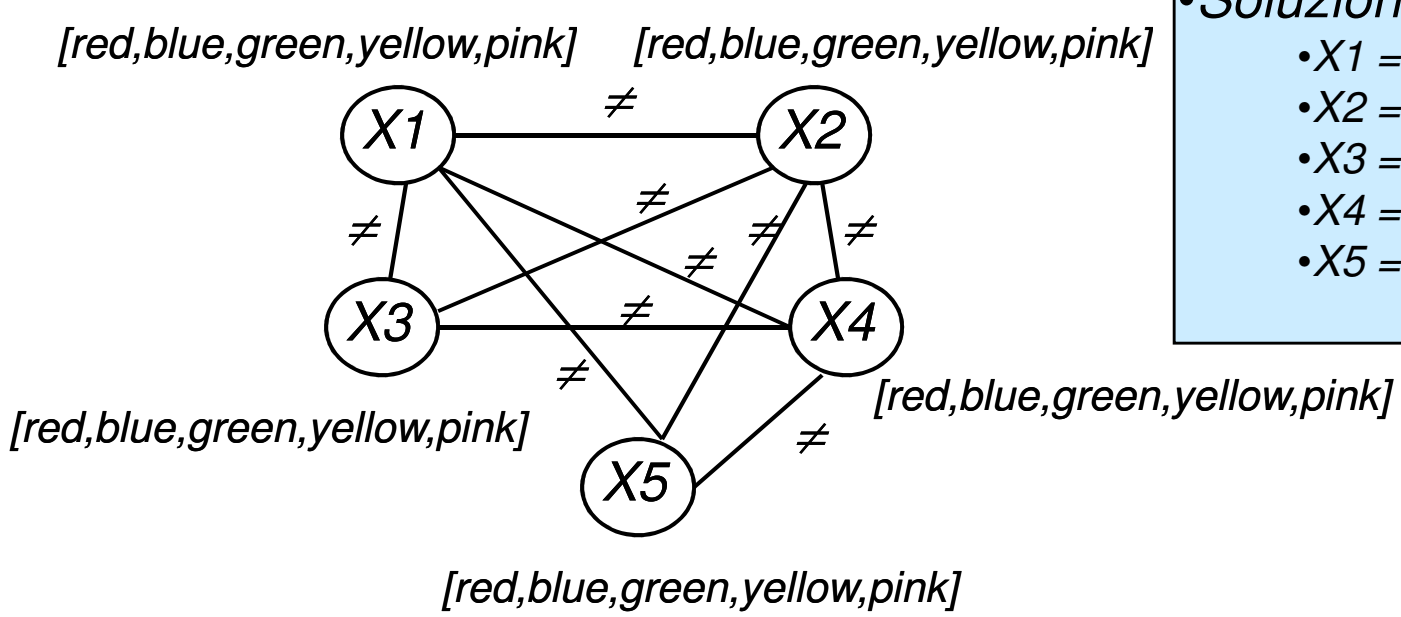
• Trovare un assegnamento di colori alle variabili consistente con i vincoli

– *variabili X1, X2, X3, X4, X5: zone*
– *domini D1, D2, D3, D4, D5: [red, blue, green, yellow,pink]*
– *vincoli : near(i, j) ⇒ Xi ≠ Xj*

CONSTRAINT GRAPHS

Un problema di soddisfacimento di vincoli si può rappresentare con un grafo detto constraint graph:

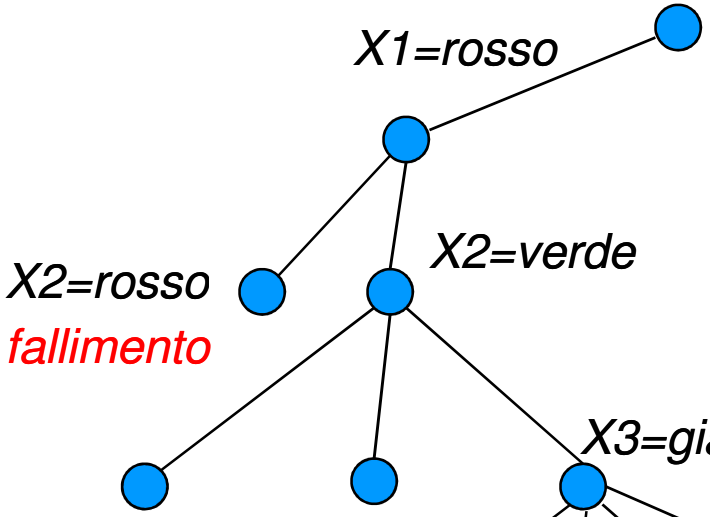
- *variabili* ⟺ *nodi*
- *vincoli* ⟺ *(iper)-archi*

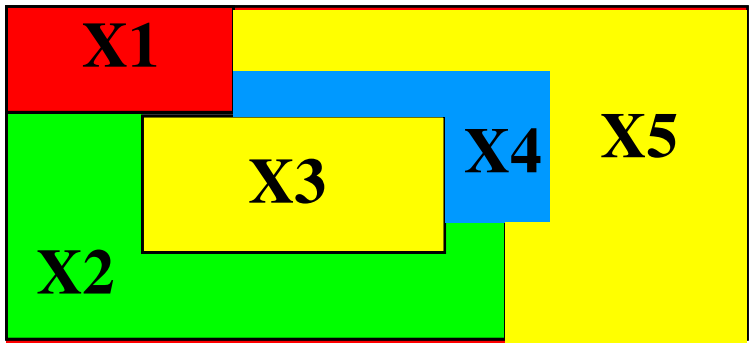


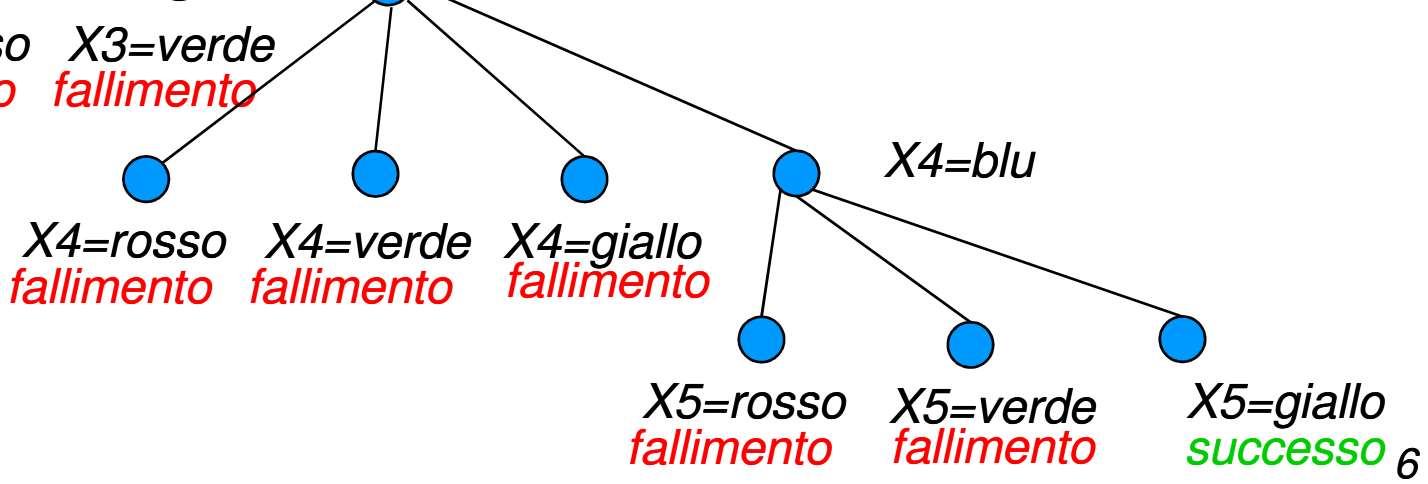
- *Soluzione ammissibile:*
  - *X1 = red*
  - *X2 = green*
  - *X3 = blue*
  - *X4 = yellow*
  - *X5 = pink*

ESEMPIO: Map Coloring

- Algoritmo semplice (ma a volte inefficiente)







PROBLEMI DI OTTIMIZZAZIONE CON VINCOLI

Un problema di ottimizzazione con vincoli (Constraint Optimization Problem – COP) è definito da:

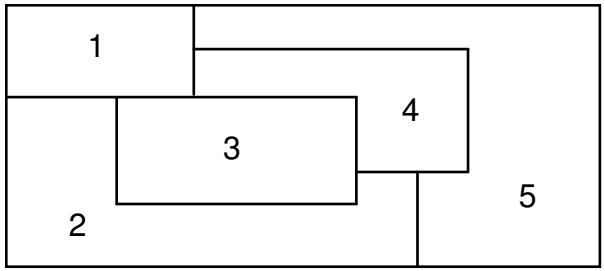
$$COP = \langle X, D, C, f \rangle$$

In altre parole, un CSP + una funzione obiettivo:

$$f(X_1, X_1 ... X_n)$$

Soluzione di un problema di ottimizzazione: un assegnamento di valori alle variabili compatibile con i vincoli del problema che ottimizza la funzione obiettivo

EXAMPLE: Map Coloring



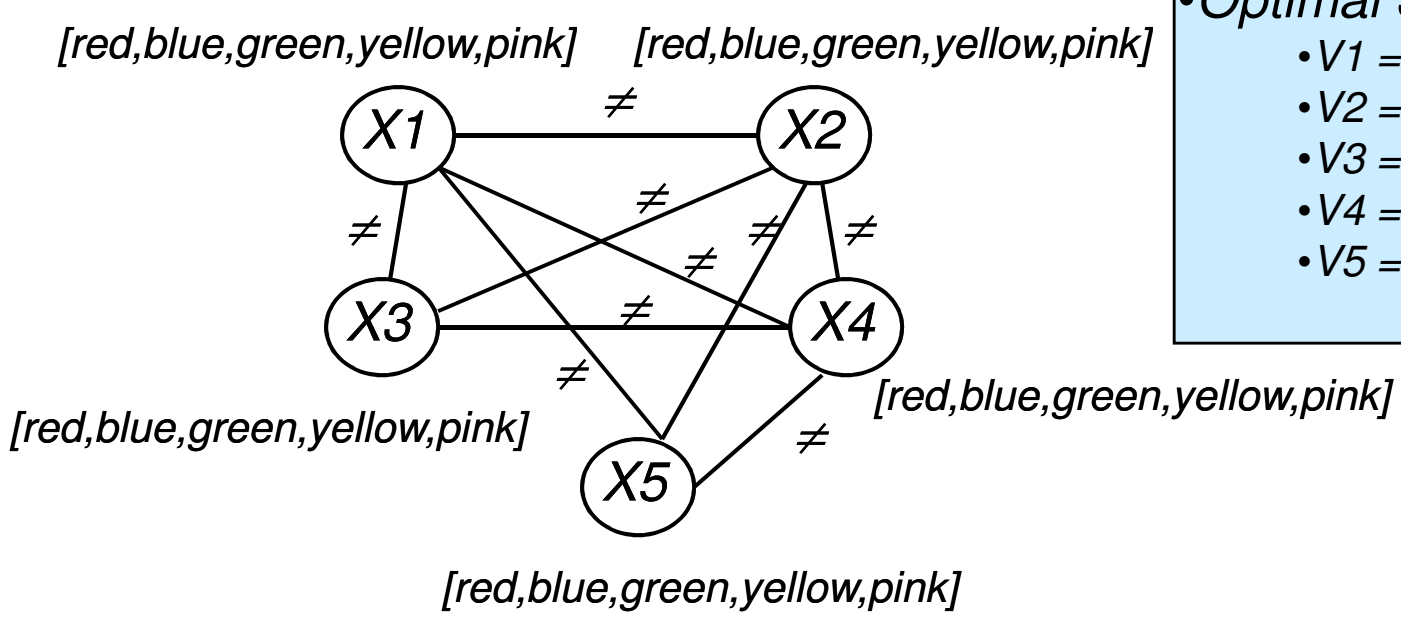
• Trovare un assegnamento di colori alle zone tale che due zone adiacenti sono colorate con colori diversi, e MINIMIZZANDO il numero di colori usati

– *variabli X1, X2, X3, X4, X5: zone*
– *domini D1, D2, D3, D4, D5: [red, blue, green, yellow,pink]*
– *vincoli: near(i, j) $\Rightarrow$ Xi $\neq$ Xj*

CONSTRAINT GRAPHS

Un problema di ottimizzazione si può rappresentare con un grafo detto constraint graph:

- *variabili* ⟺ *nodi*
- *vincoli* ⟺ *(iper)-archi*



- *Optimal Solution:*
  - *V1 = red*
  - *V2 = green*
  - *V3 = yellow*
  - *V4 = blue*
  - *V5 = yellow*

“Prolog è un buon linguaggio per risolvere CSP?”

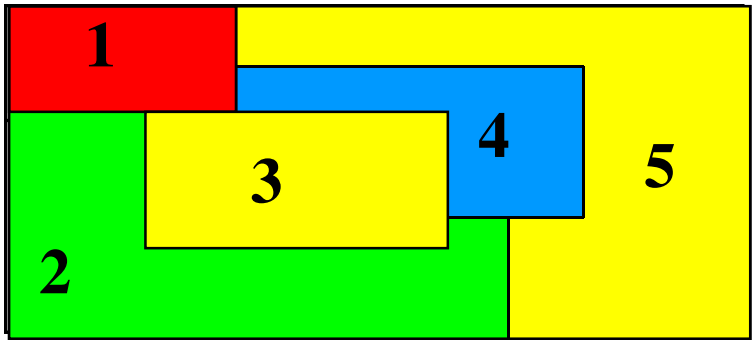
CONCETTI GENERALI

- **Capacità di esprimere vincoli**
- **Constraint store**
  - *collezione dei vincoli del problema*
- **Soddisfacibilità**
  - *una collezione di vincoli è soddisfacibile (consistente) se esiste (almeno) una soluzione*
- **Propagazione di vincoli**
  - *meccanismo di inferenza di nuovi vincoli a partire da quelli dati*

ESEMPIO

- Constraint store
  - vincoli unari: $\forall i \in [1,5], X_i \in \{r,g,b,y\}$
  - vincoli binari: $\forall\ i,j \in [1,5]$, near(i, j) $\rightarrow X_i \neq X_j$

- Soddisfacibilità
  - *Una possibile soluzione*



ESEMPIO

- Propagazione dei vincoli
  - *Modifica del constraint store (nuovo vincolo)*

$$(near(i,j) \Rightarrow X_i \neq X_j) \wedge near(1,2)$$

implica

$$X_1 \neq X_2$$

  - *Modifica del constraint store (riduzione di un dominio)*

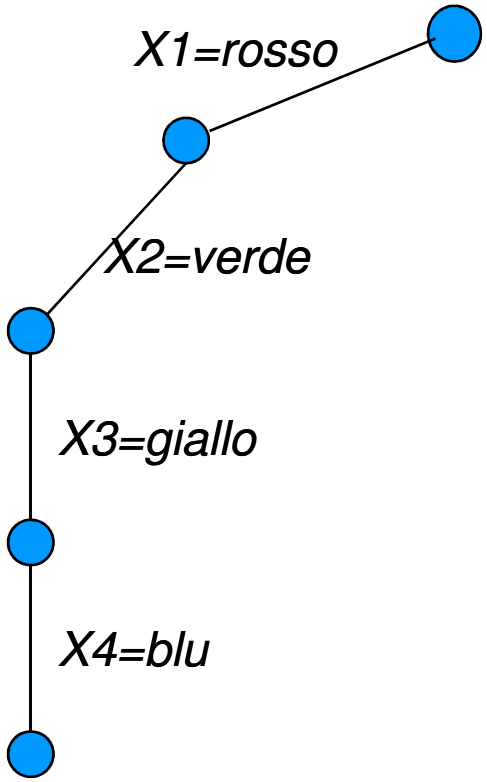
$$X_1 = r \wedge \quad X_1 \neq X_2 \quad \wedge \quad X_2 \in \{r,g,b,y\}$$

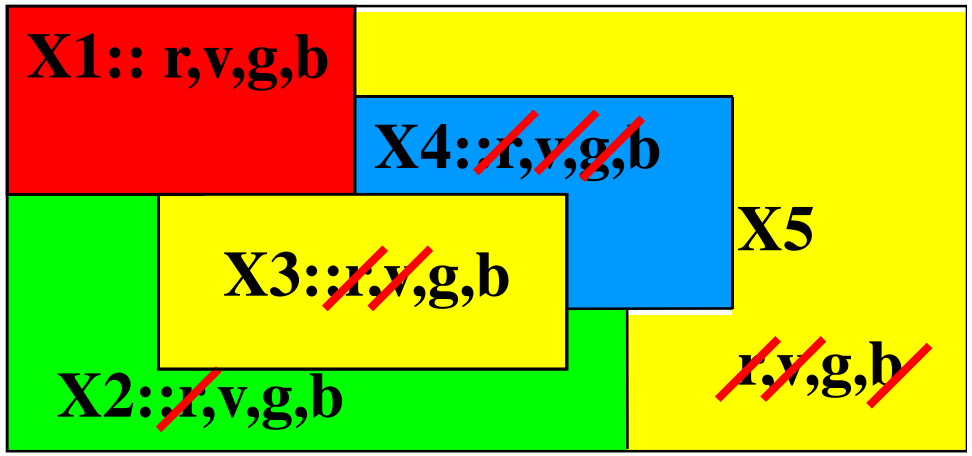
implica

$$X_2 \in \{g,b,y\}$$

PROPAGAZIONE DI VINCOLI

- Eliminazione a priori dei valori inconsistenti





LINGUAGGI LOGICI

- **Esprimere relazioni**
  - basati su relazioni
  - dominio non interpretato: Universo di Herbrand
  - unificazione interpretabile come vincolo (= sintattico)
  - sostituzione come store dei vincoli

- Universo di Herbrand di un programma P:
  - insieme di tutti i termini "ground" che possono essere costruiti a partire dai simboli di costante e di funzione di P
  - es. p(X) :- q(f(X)).
  - p(a).
  - p(b).
  - q(f(c)).
  - H={a,f(a),f(f(a)),f(f(...f(a))...,b,f(b),...,c,f(c), f(f(c))}

LINGUAGGI LOGICI

- **Test di consistenza**
  - test di consistenza sul dominio di Herbrand
    - uguaglianza sintattica
  - variabili write-once => incrementalità del test

- **Capacità inferenziali**
  - passo di risoluzione
  - algoritmo di unificazione

LINGUAGGI LOGICI

- `p(X,X).`
  - *clausola (fatto) come invariante*
- `:- p(Z,Y), …`
  - *unificazione del goal con la testa della clausola*
  - *vincoli risultanti: Z = X, Y = X, consistenti*
  - *propagazione: Z = Y*
- `:- …, Z = 3, …`
  - *nuovo vincolo Z = 3*
  - *test di consistenza ok*
  - *propagazione: X = Y = 3*
- `:- …, Y = 4.`
  - fallimento test di consistenza

LIMITI DEI LINGUAGGI LOGICI

- ***Dominio non interpretato***
  - *Universo di Herbrand: dominio "sintattico"*
  - *unificazione sintattica*
  - *assenza di tipi (domini) predefiniti*
- ***Esempio***

```
x = 2 + 5
```

  - *X istanziato alla struttura 2+5 =>X = 7 fallisce*
  - *Soluzione parziale: predicato is*
    - *X is 2 + 5 => X = 7 ha successo*
    - *Problema: X is Y + 5. Se Y è libera fallisce*
- Direzionalità: is introduce una *valutazione*

LIMITI DEI LINGUAGGI LOGICI

- ***Dominio non interpretato***

```
X > Y
```

  - *> visto come test e non come vincolo*
  - *valutazione espressioni (come is)*

- ***Cosa ci serve ?***
  - *X > Y, X = Y + 5: vincoli collezionati nel constraint store*
  - *verificati a ogni passo della computazione*

LIMITI DEI LINGUAGGI LOGICI

- ***Ordine testuale dei goal significativo***
  - `X > 5, X = 7` *fallisce*
  - `X = 7, X > 5` *ha successo*

- ***Necessità di intervenire sull'ordine di valutazione dei goal (secondo lo stato delle variabili)***
  - *freeze + metapredicati (var, ground, etc.)*
  - *a carico del programmatore !!!*

- ***Cosa ci serve ?***
  - *Che il risolutore "addormenti" i goal che non e' in grado di valutare per selezionarli non appena sono disponibili nuove informazioni*

Soluzione parziale: freeze, when

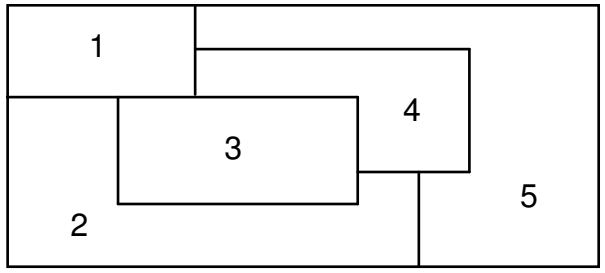
`freeze(X,atomo(X))`

- *Indica a Prolog che l'*`atomo` *deve essere selezionato (dalla risoluzione SLD) solo quando* `X` *non è variabile.*

```
freeze(Y, X is Y+1), Y=3.
        Y/3 |
freeze(3, X is 3+1).
        X/4 |
        []
```

Map Coloring usando **when**

```
diverso(A,B):-
   when((nonvar(A),nonvar(B)),A\=B).
coloring([X1,X2,X3,X4,X5],Dom):-
   diverso(X2,X1), diverso(X3,X1),
   diverso(X4,X1), diverso(X5,X1),
   diverso(X3,X2), diverso(X4,X2),
   diverso(X5,X2), diverso(X4,X3),
   diverso(X4,X5),
   member(X1,Dom),
   member(X2,Dom),
   member(X3,Dom),
   member(X4,Dom),
   member(X5,Dom).
```



Molto dichiarativo! Che algoritmo viene usato?

Ancora su freeze / when

- *E se Y non diventa ground?*

risposta condizionale!

```
?- freeze(Y, X is Y+1).
Delayed goals: X is Y + 1

Yes
```

- *Non semplifica le equazioni, ma è già qualcosa*
- *A carico del programmatore*
- ***Cosa ci serve ?***
  - *Che il risolutore "addormenti" i goal che non è in grado di valutare per selezionarli non appena sono disponibili nuove informazioni*

LIMITI DEI LINGUAGGI LOGICI

- ***Inefficienza dell'esplorazione dello spazio delle soluzioni***
  - *relazioni come test: le relazioni sono controllate a posteriori, dopo che le variabili coinvolte sono istanziate.*
  - *Prima istanzio e poi verifico*

- ***Cosa ci serve ?***
  - *relazioni come vincoli: le relazioni devono agire in avanti sullo spazio di ricerca. Ogni volta che una variabile viene istanziata, i vincoli vengono propagati per eliminare strade a priori inconsistenti.*

Es reversibilità

```
member(X,[X|_]).
member(X,[_|T]):- member(X,T).
```

- *verifica se un elemento appartiene ad una lista*

```
member(1,[4,1,2]).
yes
```

- *Se metto l'elemento variabile, in backtracking gli vengono assegnati i vari elementi della lista*

```
member(X,[4,1,2]).
yes, X=4, more?
yes, X=1, more?
yes, X=2.
```

- *Utile per generare assegnamenti!*

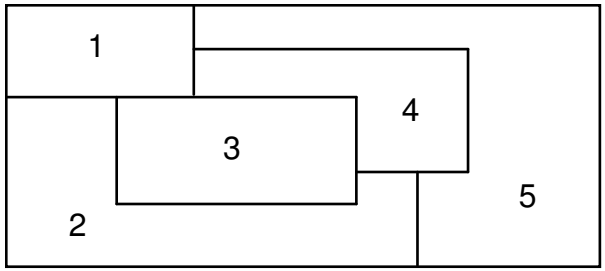
Esempio di CSP

- *Variabili: X, Y - Domini: da 1 a 4 - Vincoli: X>Y*

```
csp(X,Y):-
   member(X,[1,2,3,4]),
   member(Y,[1,2,3,4]),
   X>Y.
```

- *Elenca, in backtracking, tutte le soluzioni del CSP*
- *Molto dichiarativo: dichiaro le variabili, i domini (member) e i vincoli*
- *Quale algoritmo viene usato?*
- *Che cosa devo fare se voglio cambiare euristica?*
  - *selezione del valore*
  - *selezione della variabile*

ESEMPIO: Map Coloring

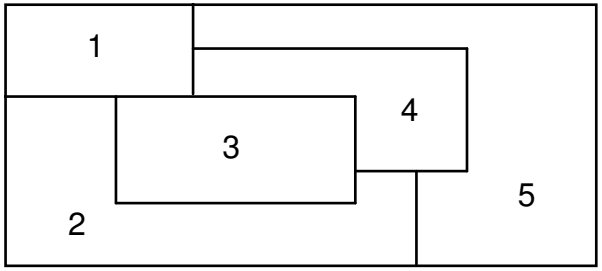


• Trovare un assegnamento di colori alle variabili consistente con i vincoli

– *variabili V1, V2, V3, V4, V5: zone*
– *domini D1, D2, D3, D4, D5: [red, blue, green, yellow]*
– *vincoli : near(Vi, Vj) ⇒ Vi ≠ Vj*

Esempio

```
coloring([X1,X2,X3,X4,X5],Dom):-
    member(X1,Dom),
    member(X2,Dom),
    member(X3,Dom),
    member(X4,Dom),
    member(X5,Dom),
    X2 \= X1, X3 \= X1,
    X4 \= X1, X5 \= X1,
    X3 \= X2, X4 \= X2,
    X5 \= X2, X4 \= X3,
    X4 \= X5.
```

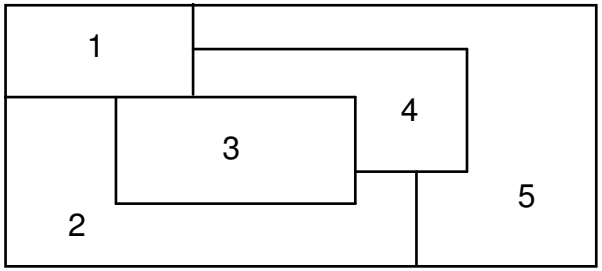


Molto dichiarativo!

Che algoritmo viene usato?

Standard backtracking

```
coloring([X1,X2,X3,X4,X5],Dom):-
    member(X1,Dom),
    member(X2,Dom),
    X2 \= X1,
    member(X3,Dom),
    X3 \= X1, X3 \= X2,
    member(X4,Dom),
    X4 \= X1, X4 \= X2, X4 \= X3,
    member(X5,Dom),
    X5 \= X1,
    X5 \= X2, X4 \= X5.
```

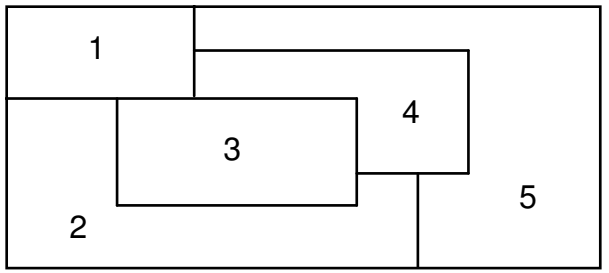


- L'ordine è importante ☹
- un po' meno dichiarativo
- Risolve solo questa istanza

Standard backtracking: generale

```
% colori(+LStati, -Assegnam, +NodiGiaAssegnati,
  +ValoriGiaAssegnati, Dom)
% ?- colori([1,2,3,4,5], X,[],[],[r,b,g,y]).
colori([],[],_,_,_).
colori([N1|Ntail], [X|Tail], NPlaced, Placed, Values):-
  member(X,Values),
  compatible(X,N1,Placed,NPlaced),
  colori(Ntail, Tail, [N1|NPlaced], [X|Placed], Values).

compatible(_,_,[],[]).
compatible(X,N1,[P|Tail],[NP|NTail]):-
  (near(N1,NP); near(NP,N1)), !,
  X\==P,
  compatible(X,N1,Tail,NTail).
compatible(X,N1,[P|Tail],[NP|NTail]):-
  compatible(X,N1,Tail,NTail).
```




```
near(1,2).
near(1,3).
…
```

ESEMPIO

- ***Esempio: Inefficienza dell'esplorazione dello spazio delle soluzioni***
  - *In PL si genera una soluzione poi si esegue il test*
  - $X_1 = r$, $X_1 \neq X_2$, $X_2 \in \{r,g,b,y\}$
  - $X_2 = r$ *fail (backtracking)*
  - $X_2 = g$ *consistente (proseguo)*

- ***Cosa ci serve?***
  - *Propagazione dei vincoli a priori*
  - $X_1 = r$, $X_1 \neq X_2$, $X_2 \in \{r,g,b,y\}$
  - $X_2 \in \{g,b,y\}$
  - *Pruning dello spazio delle soluzioni =>* prevenzione dei fallimenti

PROPAGAZIONE DI VINCOLI

- *Enumerazione: algoritmi basati su backtracking*

– *Assegna un valore di tentativo e controlla i vincoli*
– *Inefficienza dovuta alle dimensioni dello spazio di ricerca*
– *Trashing*

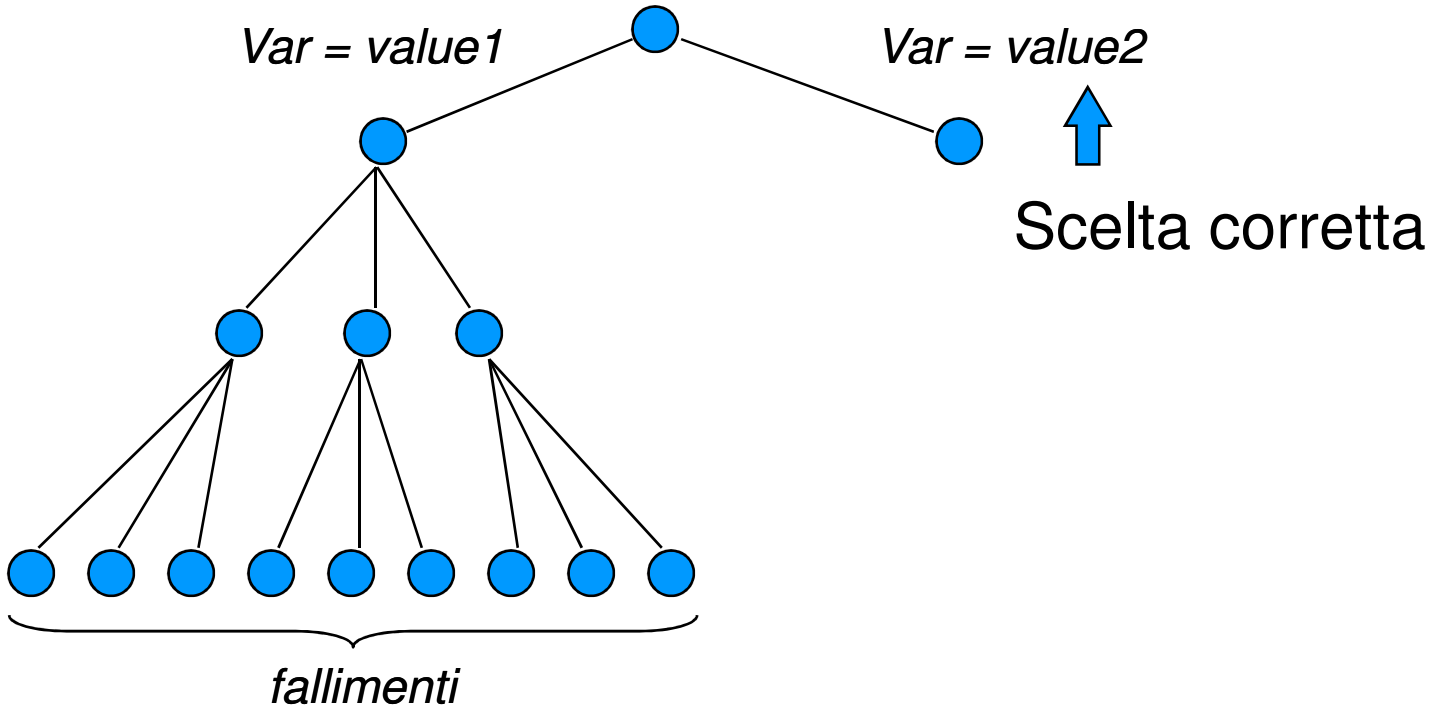
- *Propagazione di Vincoli*

– *scopo: ridurre lo spazio di ricerca*
– *la propagazione di vincoli rimuove A PRIORI combinazioni di assegnamenti che non possono condurre ad alcuna soluzione consistente*
–*prevenzione dei fallimenti e dei successivi backtracking*

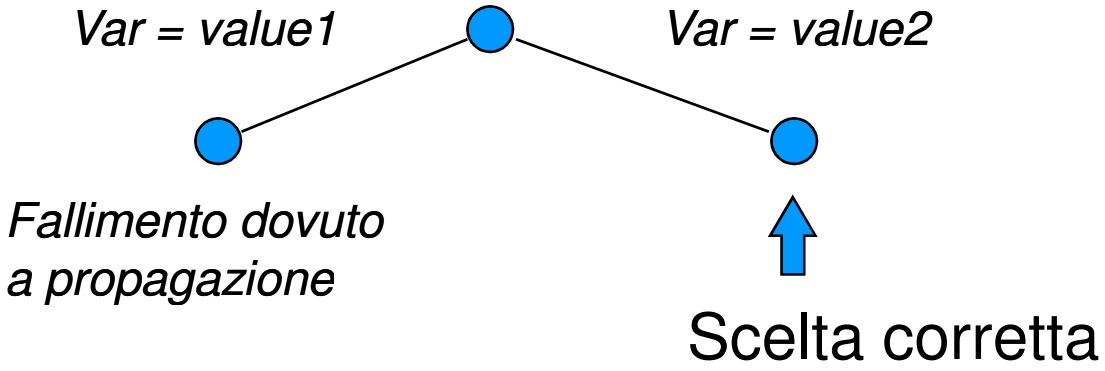
ALGORITMI DI BACKTRACKING

Modo intuitivo per risolvere problemi a vincoli CSP

Trashing

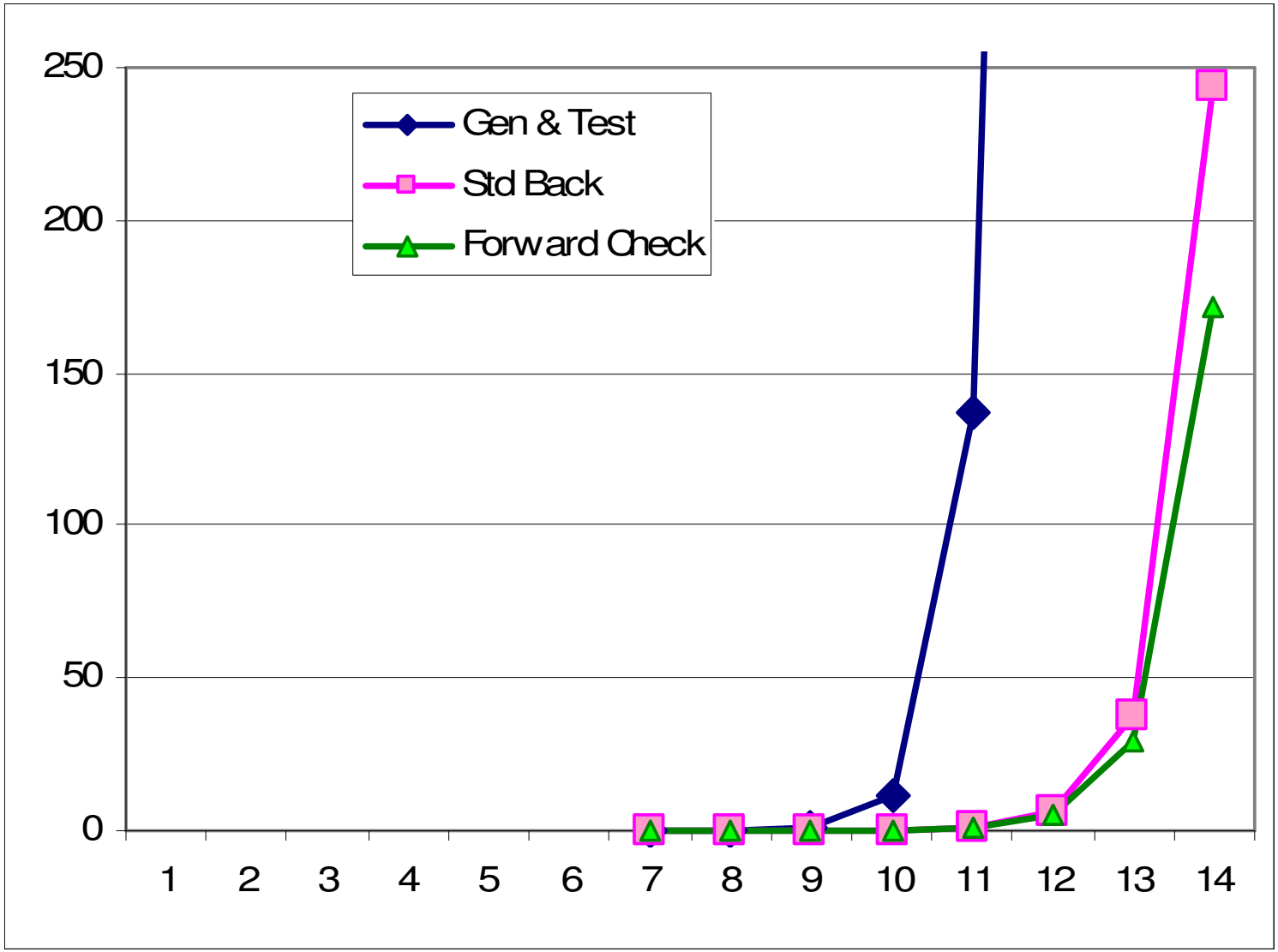


PROPAGAZIONE DI VINCOLI



Algoritmi di Propagazione evitano i fallimenti piuttosto di recuperare lo stato da situazioni in cui il fallimento si e' gia' verificato

N-Queens: Efficienza



secondi necessari per trovare tutte le soluzioni

LIMITI DEI LINGUAGGI LOGICI

- ***Vincoli come risultato di una computazione***
  - *In PL: risposta calcolata come insieme di vincoli d'uscita*
  - *funziona con il predicato = (unificazione)*

```
p(X,X).                    DB Prolog
?- p(Z,Y).                 Goal
yes Z = Y                  Risposta calcolata
```

  - *non funziona con un predicato come >*

```
p(X,Y) :- X > Y.           DB Prolog
?- p(Z,Y).                 Goal
fail
```

- ***Cosa ci serve ?***
  - *Risposta: constraint store al termine della computazione*

```
yes if Z > Y               Risposta calcolata
```

Constraint Logic Programming

- *Nuovo paradigma di programmazione, estensione della Programmazione Logica [Jaffar e Lassez, 1998]*
- *è uno schema per produrre nuovi linguaggi*
- *durante la risoluzione SLD, alcuni atomi speciali, detti vincoli (constraints), non vengono risolti, ma vengono accumulati in un'area di memoria esterna (constraint store) ed elaborati da un risolutore esterno (constraint solver).*
- *Ci possono essere diversi risolutori, basati su tecnologie diverse. Ciascuno di questi dà luogo ad un linguaggio della classe CLP:*
  - *CLP(FD): sui domini finiti basato su consistency*
  - *CLP(R): sui reali basato sul simplesso*
  - *CLP(Bool): sui booleani basato su BDD*
  - *…*

PROGRAMMAZIONE LOGICA A VINCOLI

- ***Domini definiti e interpretati nel linguaggio CLP***
  - *CLP(R): dominio dei numeri reali X > 4.3 + 7.2*
    - *4.3 e 7.2 sono numeri reali, non generici simboli*
    - *> è la relazione d'ordine sui reali, non un generico predicato o un "test"*
    - *+ è la somma sui reali e non un generico funtore*

- ***Capacità di esprimere e trattare relazioni sul dominio come vincoli***
  - *CLP(R): X > Y + 5*

PROGRAMMAZIONE LOGICA A VINCOLI

- ***Nozione di consistenza (semantica dichiarativa) e di propagazione (semantica operazionale) incorporata nel risolutore***
  - *CLP(R):* $X > Y + 5, Y = 3$
    - *passo di inferenza:* $X > 8$
    - *nozione di consistenza: specializzata su R*

- ***Vincoli come ingressi e risultati***
  - *CLP(R):*

```
p(A,B) :- A > B + 5.

:- p(X,Y), Y = 3.
yes X > 8, Y = 3
```

PROGRAMMAZIONE LOGICA A VINCOLI

- ***Risolutori su domini diversi:***
  - *Reali: CLP(R)*
  - *Razionali: CLP(Q)*
  - *Booleani: CLP(B)*
  - *Domini Finiti CLP(FD) .....*

- ***Completezza del test di consistenza***
  - *Se il test ha successo il set di vincoli è sicuramente soddisfacibile*
    - *CLP(FD) → risolutore non completo*
    - *CLP(R) → risolutore completo*

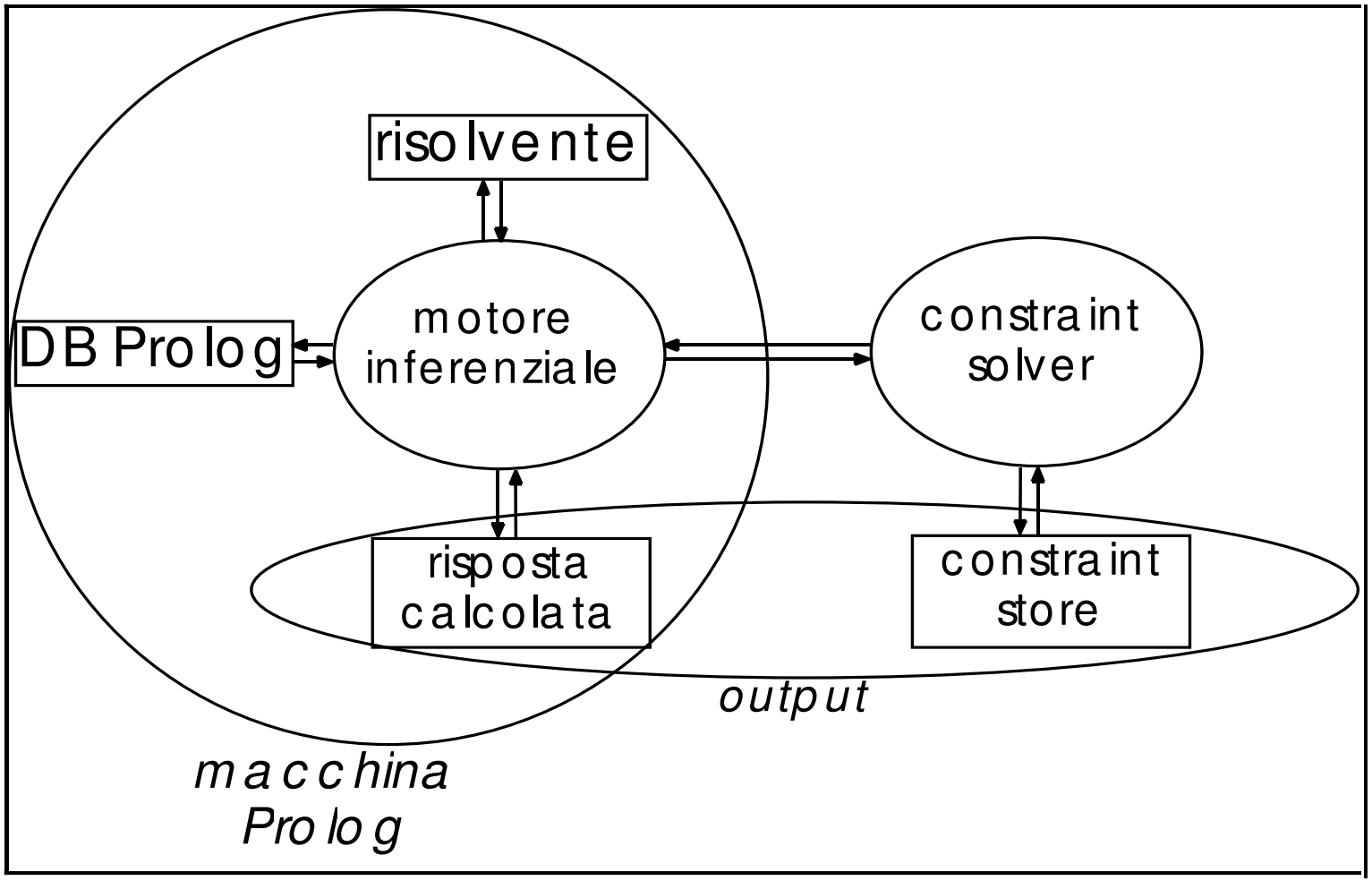
- ***Complessità del test di consistenza***
  - *Polinomiale vs. esponenziale*

ESTENDERE LA MACCHINA LOCIGA

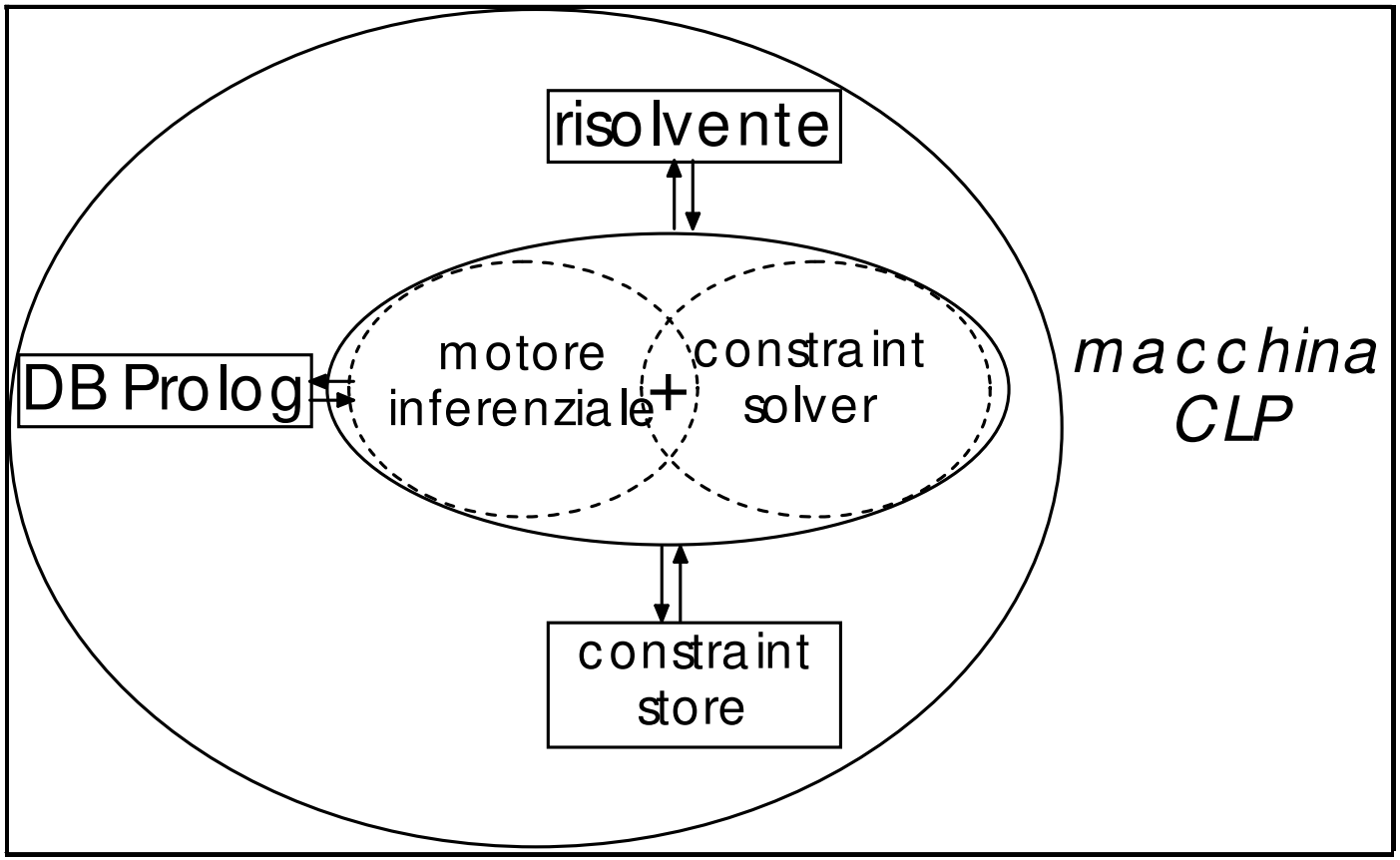
- ***Eventi rilevanti***
  - *selezione di un vincolo dal risolvente*
  - *binding di una variabile sul dominio*

- ***Estensioni***
  - *constraint solver*
  - *constraint store*
  - *definizione di predicati ad-hoc (vincoli) dotati di semantica*
  - estensione del meccanismo di unificazione

ESTENDERE LA MACCHINA LOCIGA



MACCHINA CLP



MACCHINA CLP: PASSI FONDAMENTALI

- ***Risoluzione r***
  - *selezione di un atomo dal risolvente*
  - *unificazione: aggiunta di vincoli al constraint store*
- ***Constraining c***
  - *selezione di un vincolo dal risolvente*
  - *aggiunta del vincolo al constraint store*
- ***Infer (propagazione) i***
  - *trasformazione del constraint store*
- ***Consistenza s***
  - *verifica della soddisfacibilità del constraint store*

MACCHINA CLP: PASSI FONDAMENTALI

```
r(X,Y):-
     p(X), X > Y, q(Y).

p(Z):- Z = 3.

q(K):- K > 5.
q(K):- K > 2.



?- r(A,B).
```

MACCHINA CLP: PASSI FONDAMENTALI

passo	***risolvente***	***constraint store***
risoluzione	***r(A,B)***	∅
(unif.)	*p(X'), X'>Y', q(Y')*	***A=X', B=Y'***
inferenza	*(idem)*	*A=X', B=Y'*
consistenza	*(idem)*	*(idem)*
risoluzione	***p(X')***, *X'>Y', q(Y')*	*A=X', B=Y'*
(unif.)	*Z'=3, X'>Y', q(Y')*	*A=X', B=Y',* ***X'=Z'***
inferenza	*(idem)*	*A=X'=Z', B=Y'*
consistenza	*(idem)*	*(idem)*
constraint	***Z'=3***, *X'>Y', q(Y')* *X'>Y', q(Y')*	*A=X'=Z', B=Y'* *A=X'=Z', B=Y',* ***Z'=3***
inferenza	*(idem)*	*A=X'=Z'=3, B=Y'*
consistenza	*(idem)*	*(idem)*

MACCHINA CLP: PASSI FONDAMENTALI

passo	***risolvente***	***constraint store***
constraint	***X'>Y'***, *q(Y')* *q(Y')*	*A=X'=Z'=3, B=Y'* *A=X'=Z'=3, B=Y',* ***X'>Y'***
inferenza	*(idem)*	*A=X'=Z'=3>Y', B=Y'*
consistenza	*(idem)*	*(idem)*
(unif.)	*K'>5*	*A=X'=Z'=3>Y', B=Y',* ***Y'=K'***
inferenza	*(idem)*	*A=X'=Z'=3>Y', B=Y'=K'*
consistenza	*(idem)*	*(idem)*
constraint	***K'>5*** $\varnothing$	*A=X'=Z'=3>Y', B=Y'=K'* *A=X'=Z'=3>Y', B=Y'=K',* ***K'>5***
inferenza	*(idem)*	*A=X'=Z'=3>Y', B=Y'= K'>5*
consistenza	***fallimento***	*(inconsistente)*

MACCHINA CLP: PASSI FONDAMENTALI

passo	***risolvente***	***constraint store***
risoluzione	***q(Y')***	*A=X'=Z'=3>Y', B=Y'*
(unif.)	*K'>2*	*A=X'=Z'=3>Y', B=Y', **Y'=K'***
inferenza	*(idem)*	*A=X'=Z'=3>Y', B=Y'=K'*
consistenza	*(idem)*	*(idem)*
constraint	***K'>2***	*A=X'=Z'=3>Y', B=Y'=K'*
	∅	*A=X'=Z'=3>Y', B=Y'=K', **K'>2***
inferenza	*(idem)*	*A=X'=Z'=3>Y', B=Y'= K'>2*
consistenza	*(idem)*	*(idem)*
***successo**!*		

CLP(FD)

- ***CLP(FD): Constraint Logic Programming su domini finiti***
  - particolarmente adatta a modellare e risolvere problemi a vincoli
- *Modello del problema CSP = <X,D,C>*
  - *Le VARIABILI rappresentano le entita' del problema*
  - *Definite su DOMINI FINITI di oggetti arbitrari (normalmente interi)*
  - *Legate da VINCOLI (relazioni tra variabili)*
    - *matematici*
    - *simbolici*
  - *Nei problemi di ottimizzazione si ha una FUNZIONE OBIETTIVO*
- *Risoluzione*
  - *Algoritmi di propagazione (incompleti) incapsulati nei vincoli*
  - *Strategie di ricerca*

CLP(FD) SINTASSI

- ***Esistono diversi linguaggi CLP(FD)***
  - CHIP
  - SICStus
  - ECL^{i}PS^{e}
  - B-Prolog

- La sintassi è simile ma non identica
- Ciascuno ha caratteristiche distintive

CLP(FD) SINTASSI

- ***In generale però si hanno***
  - Un metodo per definire il dominio delle variabili
    - ECLIPSE: `X:: [1..10, 13..15]`
    - SICStus: `X in (1..10) \/ (13..15)`
  - Una sintassi che contraddistingue i vincoli dagli altri predicati (ECLIPSE e SICStus usano il #)
  - `#>` `#>=` `#<` `#<=` `#=` `##` sono vincoli

VINCOLI

- Vincoli matematici: =, >, <, ≠, ≥, ≤
  - Propagazione: arc-consistency

- Vincoli Simbolici ***[Beldiceanu, Contejean, Math.Comp.Mod. 94]***
  - Incapsulano un metodo di propagazione globale ed efficiente
  - Formulazioni piu' concise
    - `alldifferent([`X_1`,...`X_m`])`
      tutte le variabili devono avere valori differenti
    - `element(N,[`X_1`,...`X_m`],Value)`
      l'ennesimo elemento della lista deve avere valore uguale a Value
    - `cumulative([`S_1`,...`S_m`], [`D_1`,...`D_n`], [`R_1`,...`R_n`], L)`
      usato per vincoli di capacita'
    - vincoli disgiuntivi

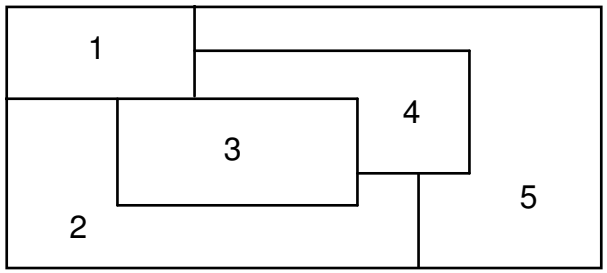
ESEMPIO DI MODELLO DI UN PROBLEMA in Eclipse

- Map Colouring

```
map_colouring([V1,V2,V3,V4,V5]):-
        V1::[red,green,yellow,blue,pink],
        V2::[red,green,yellow,blue,pink],
        V3::[red,green,yellow,blue,pink],
        V4::[red,green,yellow,blue,pink],
        V5::[red,green,yellow,blue,pink],
        V1##V2, V1##V3, V1##V4, V1##V5, V2##V3,
        V2##V4, V2##V5, V3##V4, V4##V5,
        .....
```

variabili & domini

vincoli



CP: RISOLUZIONE DI PROBLEMI

- Nozione di consistenza:
  - L'insieme dei vincoli e' consistente ?
  - Esiste una soluzione ?

- Propagazione di vincoli: meccanismo di inferenza
  - Rimuovere dai domini valori inconsistenti
  - Inferire nuovi vincoli

- Ricerca: strategie di branching
  - Selezione di una variabile
  - Selezione del valore

CP: CONSISTENZA

- Un insieme di vincoli e' CONSISTENTE (SODDISFACIBILE) se ammette almeno una soluzione
- Risolutori COMPLETI sono in grado di decidere se un insieme di vincoli e' soddisfacibile (Es. risolutori sui reali)
- Risolutori INCOMPLETI sono in grado di individuare alcune forme di inconsistenza, ma non di decidere se un insieme di vincoli e' soddisfacibile. (Es. Risolutori su domini finiti)
  - l'inconsistenza e' identificata quando il dominio di una variabile diventa vuoto.

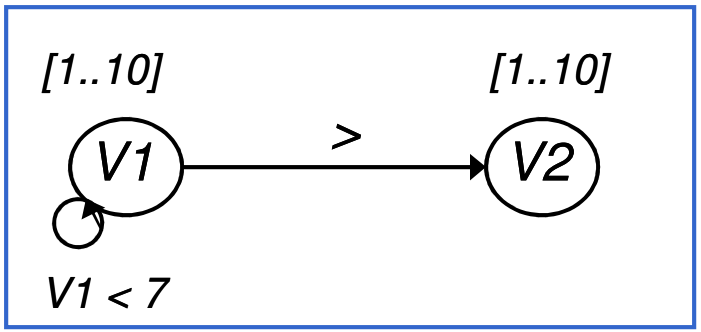
CP(FD): CONSISTENZA

- Inconsistenza: dominio di una variabile vuoto
  - non esistono valori che possono essere assegnati alla variabile
- Se ho una variabile con dominio vuoto sicuramente l'insieme di vincoli e' INSODDISFACIBILE
- Se l'insieme di vincoli e' INSODDISFACIBILE non e' detto che una variabile abbia dominio vuoto.
- Se riuscissimo a trovare TUTTI i valori inconsistenti e a rimuoverli dal dominio delle variabili, avremmo un risolutore completo. Il problema di trovare tutti i valori inconsistenti ha la stessa complessita' del problema originale.

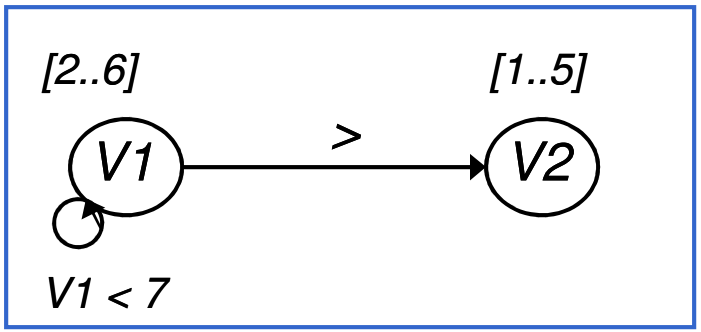
PROPRIETA' DI CONSISTENZA

- NODE CONSISTENCY
  - *una rete e' node consistent se in ogni dominio di ogni nodo ogni valore e' consistente con i vincoli unari che coinvolgono la variabile*

- ARC CONSISTENCY
  - *a rete e' arc consistent se per ogni arco (vincolo binario) che connette le variabili Vi eVj per ogni valore nel dominio di Vi esiste un valore nel dominio di Vj consistente con il vincolo*



Non Node consistent
Non Arc consistent



Node consistent
Arc consistent

COMPLESSITA'

A volte l'Arc-Consistency è troppo costosa

Quando risveglio un vincolo c(X,Y), per ogni valore in dom(X) cerco un valore consistente in dom(Y)

intuitivamente, circa d^2 confronti

ogni volta che risveglio (se d è la cardinalità dei domini)!

Mi accorgo di un fallimento quando un dominio è vuoto.

Potrei fermarmi quando ho trovato un valore consistente

Ragionare per intervalli, invece di verificare tutti gli elementi del dominio

BOUND CONSISTENCY

Un vincolo c(X,Y) è bound consistente se

$\forall d \in \{\min(X),\max(X)\}\ \exists g \in \text{dom}(Y)$ t.c. c(d,g) è vero (soddisfatto) e viceversa

✓ Comodo per vincoli come <, >, …

✓ Non ho bisogno di tenere una lista di elementi del dominio, ma bastano gli estremi

✓ Risveglio un vincolo c(X,Y) solo se elimino gli estremi del dom di X o Y

✘ Faccio meno pruning

ALGORITMI DI FILTERING

- Vincoli matematici:
  - Esempio 1
    - `X::[1..10], Y::[5..15], X>Y`

      Arc-consistency: per ogni valore v della variabile X, se non esiste un valore per Y compatibile con v, allora v viene cancellato dal dominio di X e viceversa

      `X::[6..10], Y::[5..9]` dopo il filtering
  - Esempio 2
    - `X::[1..10], Y::[5..15], X=Y`
    - `X::[5..10], Y::[5..10]` dopo il filtering
  - Esempio 3
    - `X::[1..10], Y::[5..15], X≠Y`
    - Nessun filtering

CP(FD): FILTERING e PROPAGAZIONE

- FILTERING e' quella forma di inferenza che permette di identificare e rimuovere dai domini i valori inconsistenti
- PROPAGAZIONE Meccanismo secondo cui l'aggiornamento (riduzione) dei domini permette ad altri vincoli di fare filtering
- X::[1..10], Y::[1..10], Z::[1..10], X #>Y, Y#>Z
- Singolarmente il vincolo X #>Y fa filtering X::[2..10], Y::[1..9]
- Singolarmente il vincolo Y #>Z fa filtering Y::[1..9], Z::[1..9]
- Per ridurre completamente il problema (eliminando tutti i valori inconsistenti) la propagazione avrebbe complessita' esponenziale.

CP(FD): FILTERING e PROPAGAZIONE

- FILTERING e' quella forma di inferenza che permette di identificare e rimuovere dai domini i valori inconsistenti

- PROPAGAZIONE Meccanismo secondo cui l'aggiornamento (riduzione) dei domini permette ad altri vincoli di fare filtering

- ESEMPIO X::[1..10], Y::[1..10], Z::[1..10], X #>Y, Y#>Z

- Singolarmente il vincolo X #>Y fa filtering X::[~~1~~**2**..10], Y::[1..~~10~~**9**]

- Singolarmente il vincolo Y #>Z fa filtering Y::[~~1~~**2**..9], Z::[1..~~10~~**8**]

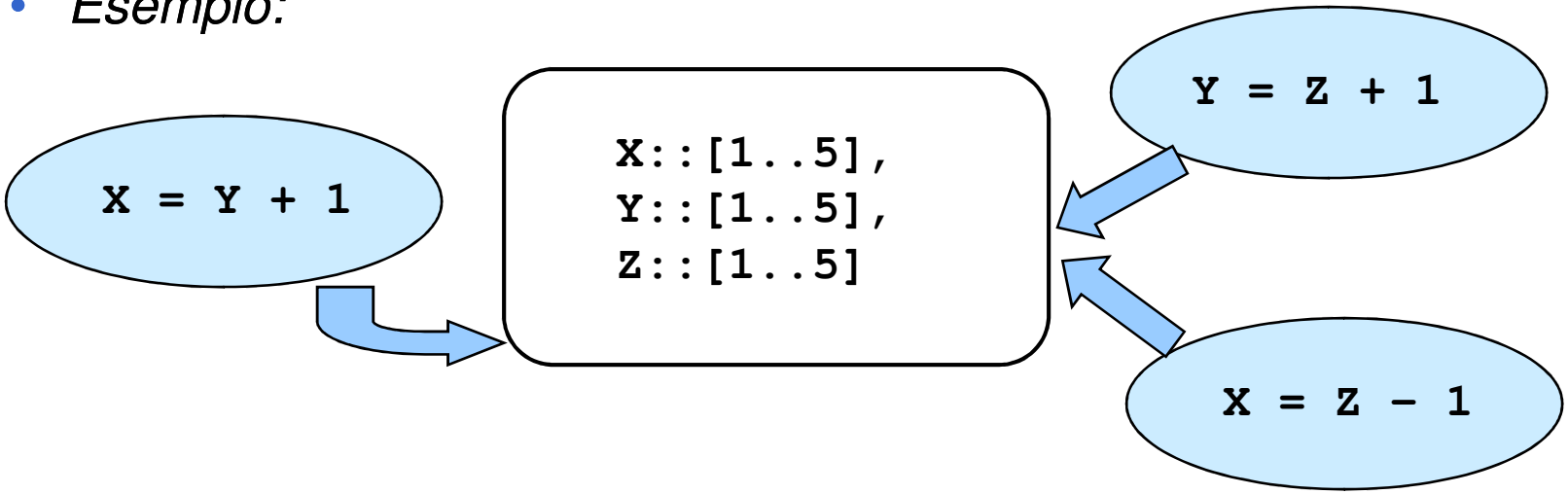
- Rimozione di 1 da Y attiva una propagazione X::[~~2~~**3**..10], Y::[2..9]

PROPAGAZIONE DI VINCOLI

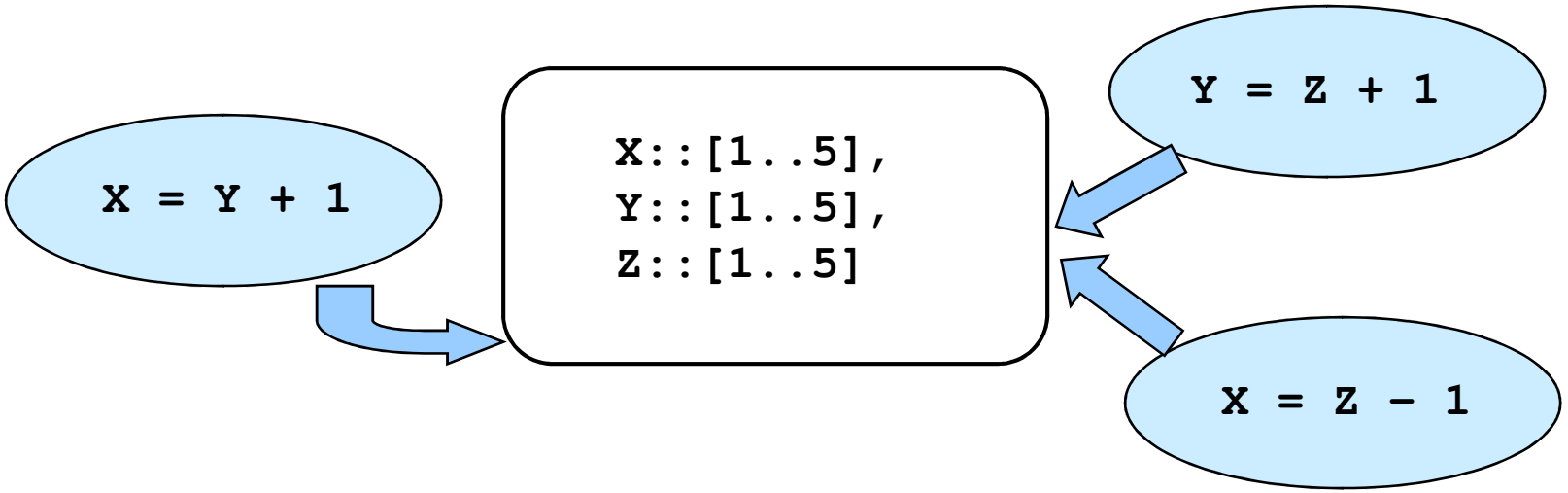
- In generale ogni variabile e' coinvolta in molti vincoli. Di conseguenza, ogni cambiamento nel dominio della variabile si può propagare ad altri valori dai domini delle altre variabili.

- Prospettiva ad agenti: durante il loro tempo di vita, i vincoli alternano il loro stato da sospesi e attivi (attivati da eventi)

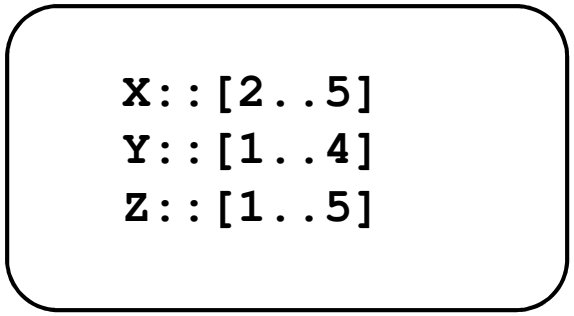
- *Esempio:*



INTERAZIONE TRA VINCOLI



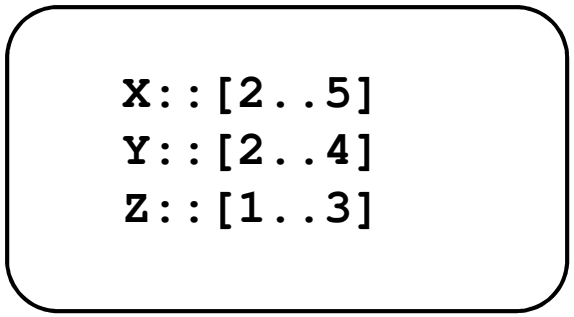
- Prima attivazione di ***X = Y + 1*** porta a



X = Y + 1
sospeso

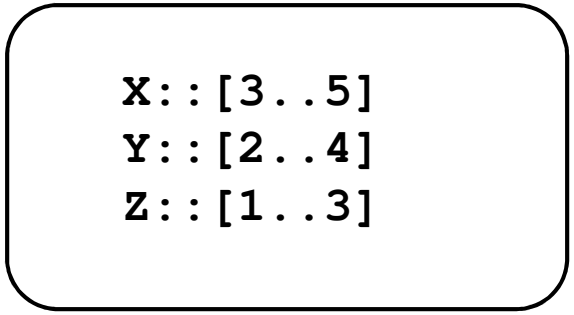
INTERAZIONE TRA VINCOLI

- Seconda attivazione di ***Y = Z + 1*** porta a



Y = Z + 1
sospeso

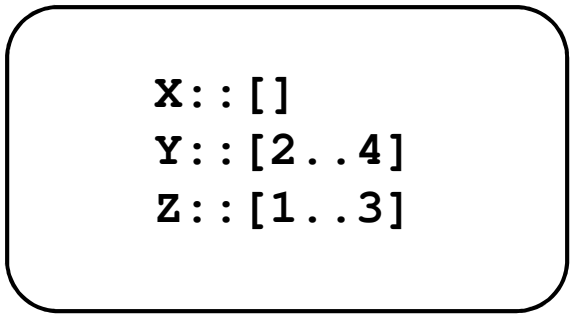
- Il dominio di ***Y*** e' cambiato ***X = Y + 1*** viene attivato



X = Y + 1
sospeso

INTERAZIONE TRA VINCOLI

- Terza attivazione di ***Z = X - 1*** porta a



FAIL

- L'ordine nel quali i vincoli sono considerati (sospesi/risvegliati) non influenza il risultato MA puo' influenzare le prestazioni computazionali

BIBLIOGRAFIA

- NODE CONSISTENCY: banale

- ARC CONSISTENCY
  - *Algoritmi proposti*
    - *AC1 - AC2 - AC3* ***[Mackworth AIJ (8), 77]*** ***[Montanari Inf.Sci (7), 74]***, *AC4* ***[Mohr, Henderson AIJ(28), 86]***, *AC5* ***[Van Hentenryck, Deville and Teng AIJ(58), 92]***, AC6 ***[Bessiere AIJ(65), 94]***, AC7 ***[Bessiere, Freuder, Regin AIJ(107), 99]***
    - *Varianti: DAC* ***[Detcher,Pearl IJCAI85]***
      MAC ***[Bessiere, Freuder, Regin, IJCAI95]***
    - Bound Consistency ***[Van Hentenryck, Saraswat, Deville TR Brown, CS-93-02, 93]***
    - Complessita': ***[Machworth, Freuder AIJ(25), 85]***, ***[Mohr, Henderson AIJ(28), 86]***, ***[Detcher,Pearl AIJ (34), 88]*** ***[Han, Lee AIJ(36), 88]***, ***[Cooper AIJ (41), 89]***

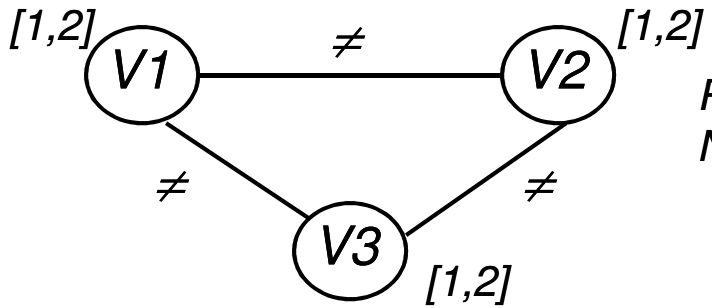
- PATH CONSISTENCY
  - *PC1 - PC2* ***[Mackworth AIJ (8), 77]***
  - *PC3* ***[Mohr, Henderson AIJ(28), 86]***
  - *PC4* ***[Han, Lee AIJ(36), 88]***

INCOMPLETEZZA ALGORITMI DI CONSISTENZA

- NODE, ARC CONSISTENCY in generale non sono completi
  - sono complete per particolari problemi che hanno strutture particolari
    *[**Freuder JACM (29), 82], [Freuder JACM (32), 85]***

- Algoritmo completo: N-CONSISTENCY per problemi di N variabili. Complessita' esponenziale
  *[**Freuder CACM (21), 78], [Cooper AIJ (41), 89]***

- Esempio:



Rete Node + Arc consistent
Non ammette soluzioni ammissibili

VINCOLI SIMBOLICI

- Vincoli Simbolici (Global Constraints)
- ***[Beldiceanu, Contejean, Math.Comp.Mod. 94]***
  - Incapsulano un metodo di propagazione globale ed efficiente
  - Formulazioni piu' concise
    - `alldifferent([`X_1`,...`X_m`])`
      tutte le variabili devono avere valori differenti
    - `element(N,[`X_1`,...`X_m`],Value)`
      l'ennesimo elemento della lista deve avere valore uguale a Value
    - `cumulative([`S_1`,...`S_m`], [`D_1`,...`D_n`], [`R_1`,...`R_n`], L)`
      usato per vincoli di capacita'
    - vincoli disgiuntivi

ESEMPIO DI MODELLO DI UN PROBLEMA in Eclipse

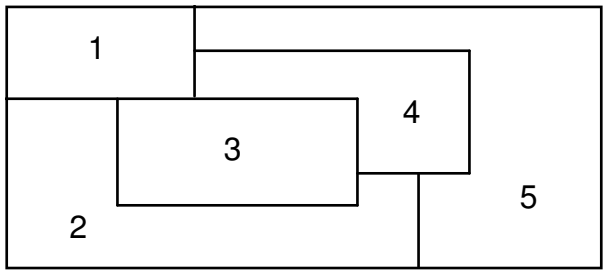
- Map Colouring

```
map_colouring([V1,V2,V3,V4,V5]):-
        V1::[red,green,yellow,blue,pink],
        V2::[red,green,yellow,blue,pink],
        V3::[red,green,yellow,blue,pink],
        V4::[red,green,yellow,blue,pink],
        V5::[red,green,yellow,blue,pink],
        alldifferent([V1,V2,V3,V4]),
        alldifferent([V1,V2,V4,V5]),
.....
```

variabili & domini

vincoli

MODELLO PIU' COMPATTO



VINCOLI N-ari: CONSISTENZA

- Per vincoli N-ari, non c'è più l'interpretazione come grafo. Es: X+Y=Z ?

- Generalized Arc Consistency (GAC) (o Hyper Arc-Consistency): Un vincolo $c(X_1,X_2,\ldots,X_n)$ è arc consistente in senso generalizzato se
  - presa una variabile X_i (i=1..n)
  - per ogni assegnamento delle rimanenti n-1 variabili

$$X_1 \rightarrow v_1,\ldots,X_{i-1} \rightarrow v_{i-1},X_{i+1} \rightarrow v_{i+1}, \ldots,X_n \rightarrow v_n$$

$\exists g \in dom(X_i)$ t.c. $c(v_1,\ldots,v_{i-1},g,v_{i+1},v_n)$ è vero (soddisfatto).

A volte raggiungere la GAC e' troppo costoso; per vincoli n-ari, conviene SFRUTTARE la STRUTTURA e la SEMANTICA dei vincoli stessi

VINCOLI GLOBALI

- Non solo zucchero sintattico
- Vincoli Simbolici (Global Constraints)

  Ogni vincolo ha associato un algoritmo di *FILTERING*
  - Algoritmo di filtering implementa tecniche complesse derivanti da
    - Intelligenza Artificiale
    - Ricerca Operativa
    - Teoria dei Grafi
    - Teoria dei linguaggi di programmazione (automi)

  - La propagazione termina quando la rete raggiunge uno stato di quiescenza non si possono cancellare piu' valori

  - Propagazione incrementale lungo un ramo dell'albero di ricerca

ALL-DIFFERENT

- Vincoli Simbolici: esempio 1
  - **alldifferent([X_1,...X_n])**

    vero se tutte le variabili assumono valori diversi

  Equivalente da un punto di vista dichiarativo all'insieme di vincoli binari

  alldifferent([X_1,...X_n]) $\leftrightarrow$ $X_1 \neq X_2$, $X_1 \neq X_3$,…, $X_{n-1} \neq X_n$

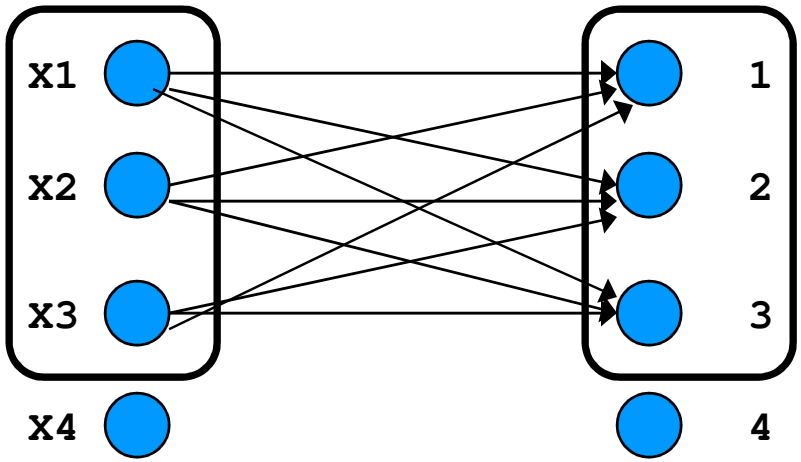
  Operazionalmente permette una propagazione piu' forte.

- **X1::[1,2,3],X2::[1,2,3],X3::[1,2,3],X4::[1,2,3,4],**
- *Arc consistency: non rimuove alcun valore !!*
- *Algoritmo di filtering **[Regin AAAI94]**: rimuove da* **X4** *i valori 1,2,3*
  L'algoritmo si basa sul flusso in un grafo

ALL-DIFFERENT

- Vincoli Simbolici: esempio 1

```
X1::[1,2,3],X2::[1,2,3],X3::[1,2,3],X4::[1,2,3,4],
alldifferent([X1,X2,X3,X4])
```



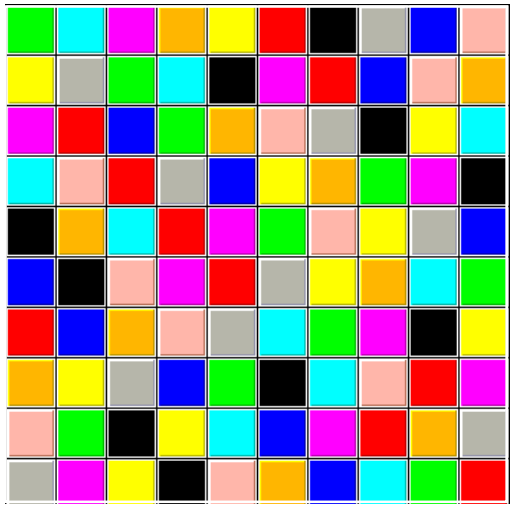
Insieme di variabili di cardinalita' 3 che hanno medesimo dominio i cardinalita' 3

```
X4::[1,2,3,4]
```

ALL-DIFFERENT

- Vincoli Simbolici: L'alldifferent si usa in tantissime applicationi

- Esempio: Partial Latin Square



Colorare ogni riga e colonna con 10 colori in modo che su ogni riga e colonna ci siano tutti colori diversi

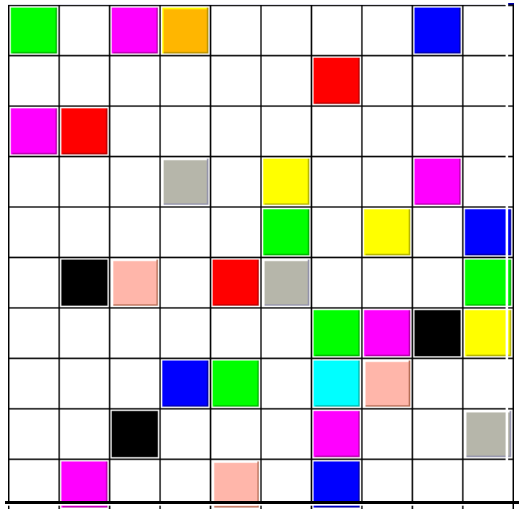
FACILE SE LA GRIGLIA E' VUOTA ma

non se PARZIALMENTE PIENA

35%-45% PERCENTUALE CRITICA

ALL-DIFFERENT NEL PARTIAL LATIN SQUARE

- Problema la cui struttura si trova in molte applicazioni (scheduling e timetabling, routing in fibre ottiche, ecc…)
- Modello del problema: alcune variabili già assegnate, altre hanno come dominio tutti i colori



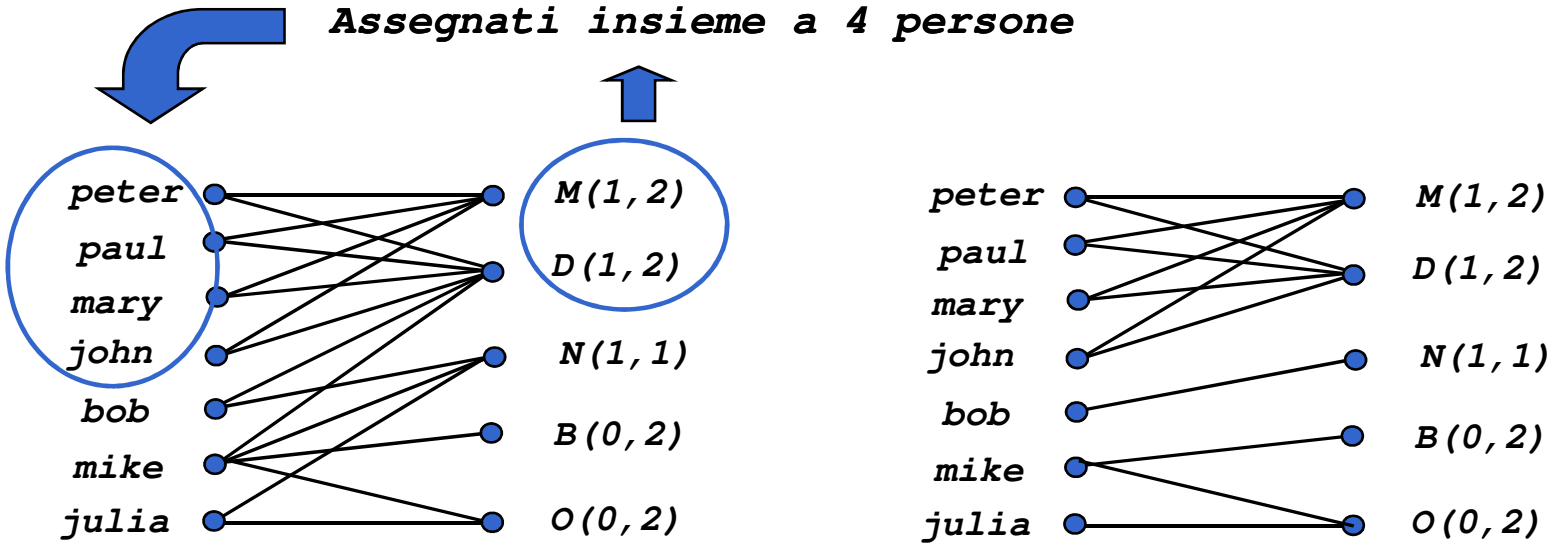
32% **preassignment**

per ogni riga i=1..n
alldifferent([X_{i1},X_{i2},….X_{in}])
per ogni colonna j=1..n
alldifferent([X_{1j},X_{2j},….X_{nj}])

SI VEDA
http://www.cs.cornell.edu/gomes

GLOBAL CARDINALITY CONSTRAINT

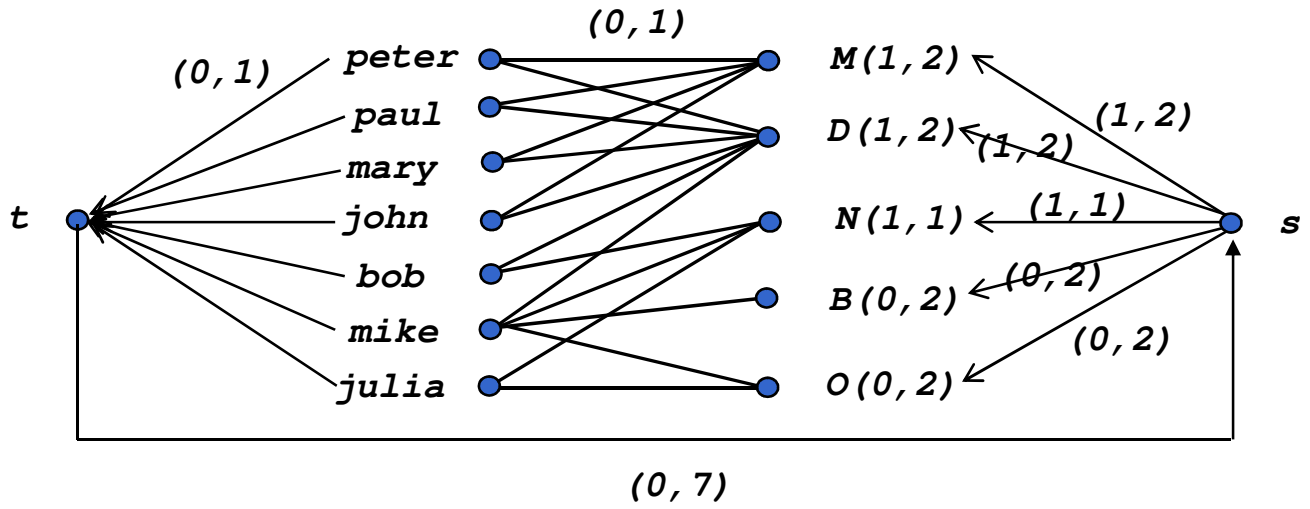
- **`gcc(Var,Val,LB,UB)`** *[Regin AAAI96]* **`Var`** sono variabili, **`Val`** valori **`LB`** e **`UB`** sono il minimo e massimo numero di occorrenze per ogni valore in **`Val`** assegnato a **`Var`**
- Esempio:



GLOBAL CARDINALITY CONSTRAINT

- La nozione di consistenza si basa su un algoritmo di *network maximum flow* sulla value network N(C)

- **gcc su k variabili e' consistente**
- **c'e' un max flow da s a t di valore k**



PROPAGAZIONE DI VINCOLI

- Vincoli Simbolici: vincoli disgiuntivi
  - Supponiamo di avere due lezioni che devono essere tenute dallo stesso dovente. Abbiamo gli istanti di inizio delle lezioni: **`L1Start`** e **`L2Start`** e la loro durata **`Duration1`** e **`Duration2`**.
  - Le due lezioni non possono sovrapporsi:

$$\texttt{L1Start} + \texttt{Duration1} \leq \texttt{L2Start}$$

$$\texttt{OR}$$

$$\texttt{L2Start} + \texttt{Duration2} \leq \texttt{L1Start}$$

  - Due problemi INDEPENDENTI uno per ogni parte della disgiunzione.

PROPAGAZIONE DI VINCOLI

- Vincoli Simbolici: vincoli disgiuntivi

  – Due problemi INDEPENDENTI, uno per ogni parte della disgiunzione: una scelta non ha effetto sull'altra ⇒ Trashing

  – Numero esponenziale di problemi:
    - N disgiunzioni ⇒ 2^N Problemi
    - Fonte primaria di complessita' in problemi reali

  – Soluzioni proposte:
    - *disgiunzione costruttiva*
    - *operatore di cardinalita'*
    - *meta-constraint*
    - *cumulative*

PROPAGAZIONE DI VINCOLI

- Vincoli Simbolici:
- **cumulative([S_1,...S_m], [D_1,...D_n], [R_1,...R_n], L)**

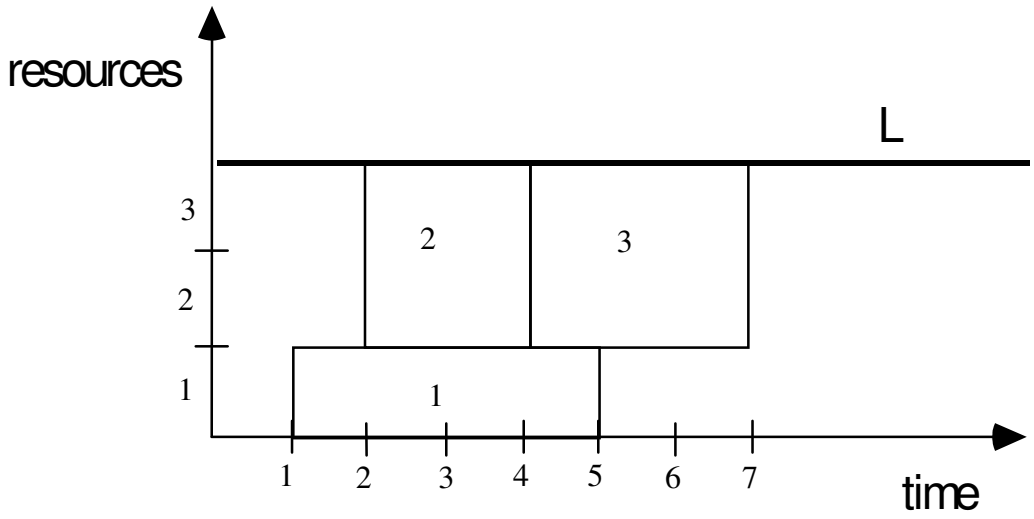
  - **S_1,...S_m** sono istanti di inizio di attivita' (variabili con dominio)
  - **D_1,...D_n** sono durate (variabili con dominio)
  - **R_1,...R_n** sono richieste di risorse (variabili con dominio)
  - **L** limite di capacita' delle risorse (fisso o variabile nel tempo)

- Dato l'intervallo [min,max] dove min = min_i {S_i}, max = max{S_i+D_i} - 1, il vincolo cumulative assicura che

$$\max\{\sum_{j \mid S_j \le i \le S_j+D_j} R_i\} \le L$$

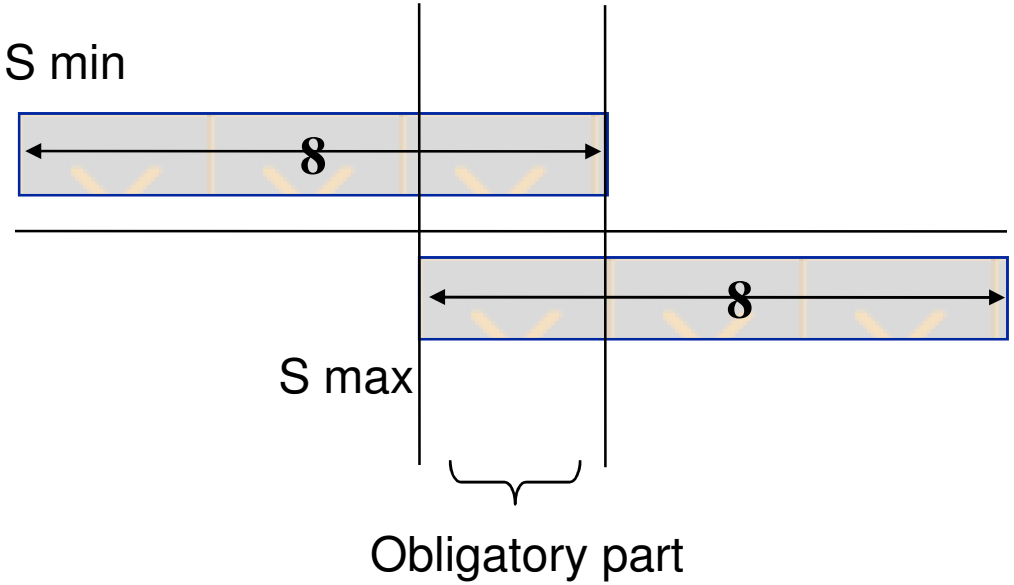
PROPAGAZIONE DI VINCOLI

- Vincoli Simbolici:
  `cumulative([1,2,4],[4,2,3],[1,2,2],3)`



PROPAGAZIONE DI VINCOLI

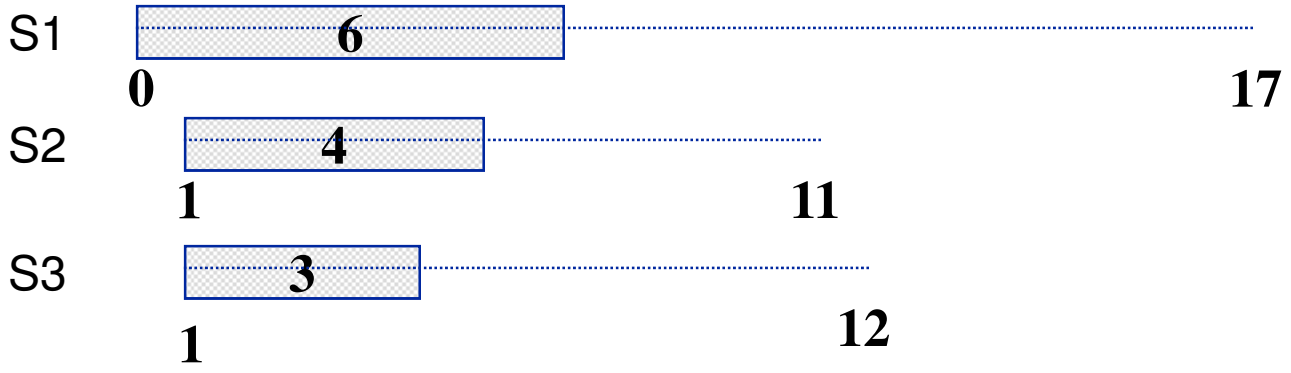
- Vincoli Simbolici:
- un esempio di propagazione usato nei vincoli sulle risorse e' quello basato sulle *parti obbligatorie*



PROPAGAZIONE DI VINCOLI

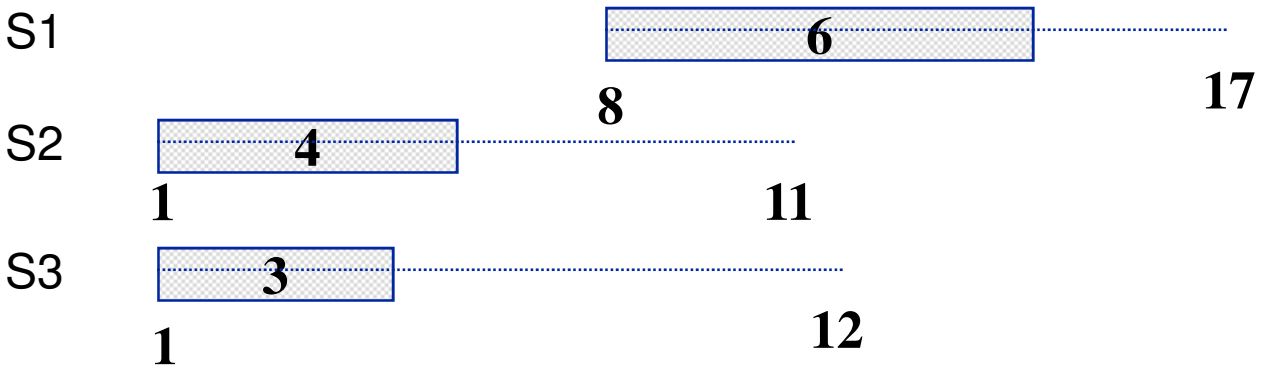
- Vincoli Simbolici:
  - un'altra propagazione usata nel vincolo di capacita' e' quella basata sul *edge finding* ***[Baptiste, Le Pape, Nuijten, IJCAI95]***

Consideriamo una risorsa unaria e tre attivita'



PROPAGAZIONE DI VINCOLI

- Vincoli Simbolici:



Possiamo dedurre che minimo istante di inizio per S1 e' 8.

Considerazione basata sul fatto che S1 deve essere eseguito dopo S2 e S3.

Ragionamento globale: supponiamo che S2 o S3 siano eseguiti dopo S1. Allora l'istante di fine di S2 e S3 e' almeno 13 (elemento non contenuto nel dominio di S2 e S3).

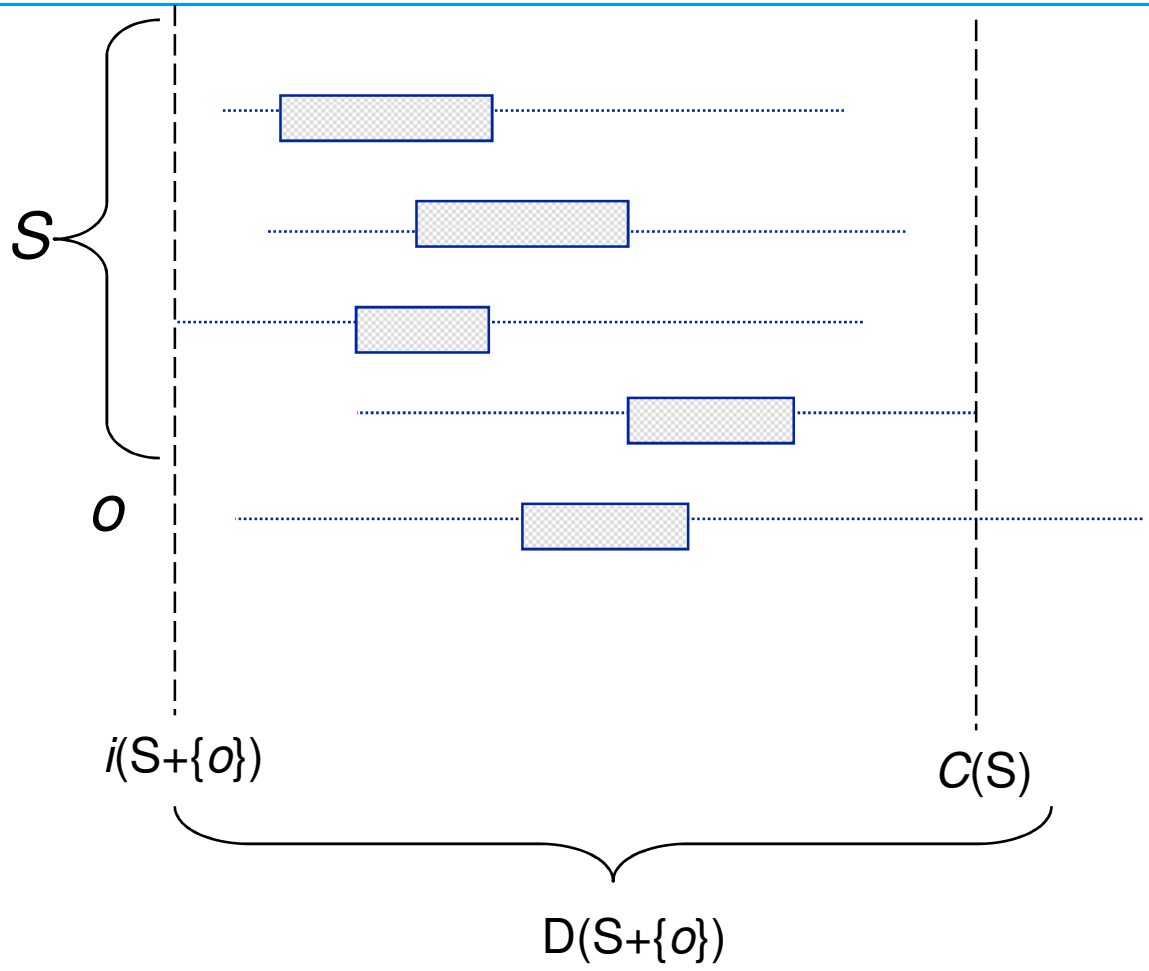
Teorema: [Carlier, Pinson, Man.Sci.95]

Sia *o* un'attività e *S* un insieme di attività che utilizzano la stessa risorsa unaria (*o* non contenuta in *S*). Il minimo istante di inizio è *i*, la somma delle durate è *D* e il massimo istante di terminazione è *C*. Se

$$i(S+\{o\}) + D(S+\{o\}) > C(S)$$

Allora non è possibile che *o* preceda alcuna operazione in *S*. Questo implica che il minimo istante iniziale per *o* può essere fissato a

$$\max_{(S' \subseteq S)} \{i(S') + D(S')\}.$$



META VINCOLI

- Vincoli Simbolici: riprendiamo i vincoli disgiuntivi
  - Supponiamo di avere due lezioni che devono essere tenute dallo stesso docente. Abbiamo gli istanti di inizio delle lezioni: **`L1Start`** e **`L2Start`** e la loro durata **`Duration1`** e **`Duration2`**.
  - Le due lezioni non possono sovrapporsi:

$$\texttt{L1Start} + \texttt{Duration1} \leq \texttt{L2Start}$$

$$\texttt{OR}$$

$$\texttt{L2Start} + \texttt{Duration2} \leq \texttt{L1Start}$$

  - Due problemi INDEPENDENTI uno per ogni parte della disgiunzione.

PROPAGAZIONE DI VINCOLI

- Vincoli Simbolici: vincoli disgiuntivi
  - Due problemi INDIPENDENTI, uno per ogni parte della disgiunzione: una scelta non ha effetto sull'altra ⟹ Trashing
  - Numero esponenziale di problemi:
    - N disgiunzioni ⟹ 2^N Problemi
    - Fonte primaria di complessità in problemi reali
  - Soluzioni proposte:
    - *disgiunzione costruttiva*
    - *operatore di cardinalità*
    - *meta-constraint*

DISGIUNZIONE COSTRUTTIVA

- ***P. Van Hentenryck, V. Saraswat, Y. Deville,***
  Design, Implementation and Evaluation of the Constraint Language cc(FD),
  Tech. Rep. Brown University, CS-93-02, 1993.

- Sfrutta la disgiunzione per ridurre lo spazio di ricerca

- Idea: aggiungere al constraint store vincoli che sono implicati da tutte le parti della disgiunzione

- Esempio: `X::[5..10], Y::[7..11], Z::[1..20], (Z=X OR Z=Y)`
  - `Z=X` ridurrebbe il dominio di `Z` a `[5..10]`
  - `Z=Y` ridurrebbe il dominio di `Z` a `[7..11]`
  - risultato della disgiunzione costruttiva: `Z::[5..11]`
- In ECLiPSe non c'è, ma può essere implementato semplicemente (V. **`Propia`**)

OPERATORE DI CARDINALITA'

- Symbolic Constraint: operatore di cardinalità

  - **#(l,[c_1,…,c_n],u)** ⇔ il numero **k** di vincoli **c_i** (**1≤i≤n**) soddisfatti non è minore di **l** e non maggiore di **u**

  - Con l'operatore di cardinalità modello i vincoli disgiuntivi nel modo seguente

#(1,[L1Start+Duration1 ≤ L2Start,
L2Start+Duration2 ≤ L1Start],1)

META-CONSTRAINTS

- Vincoli Simbolici: meta-constraints o Vincoli Reificati
  - A ogni vincolo (matematico, del tipo #>, #<, #<=, #\=, …) viene associata una variabile booleana `B`.
    - Se `B=1` il vincolo è verificato e viceversa,
    - se `B=0` il vincolo non è verificato e viceversa.

$$\mathtt{C} \Leftrightarrow \mathtt{B}$$

  - Con i meta-constraints posso modellare la disgiunzione nel modo seguente:

```
B1 ::[0,1], B2::[0,1],
L1Start+Duration1 #<= L2Start #<=> B1,
L2Start+Duration2 #<= L1Start #<=> B2,
B1 + B2 #= 1.
```

CLASSIFICAZIONE VINCOLI

- Sulla base della struttura del grafo: global constraint catalog Nicolas Beldiceanu

  http://www.emn.fr/x-info/sdemasse/gccat/

- Sulla base della semantica: Jean-Charles Règin (articolo su web)
  - Solution based (TABLE constraint)
  - Counting constraints (alldifferent, permutation, gcc)
  - Balancing constraints (balance, deviation, spread)
  - Combination based constraints (MAX-SAT, and or)
  - Sequencing constraints (among, sequence)
  - Distance constraints (inter-distance, sum-inequality)
  - Geometric constraints (diff-n)
  - Summation based constraints (subset-sum, knapsack)
  - Packing constraints (nvalue, bin-packing)
  - Graph based constraints (cycle, path, tree, wst)
  - Order based constraints (lexicolex, sort)

VINCOLI GLOBALI

- References:
  - Global constraints in CHIP ***[Beldiceanu Contejean, Math.Comp.Mod.94]***
  - Task Intervals ***[Caseau Laburthe ICLP94] [Caseau Laburthe LNCS1120] [Caseau Laburthe JICSLP96] [Caseau Laburthe LNCS1120]***
  - alldifferent ***[Regin AAAI94]*** Symmetric alldifferent ***[Regin IJCAI99]***
  - Edge Finder ***[Baptiste Le Pape Nuijten, IJCAI95], [Nuijten Le Pape JoH98] [Nuijten Aarts Eur.J.OR96]***
  - Sequencing ***[Regin Puget CP97]***
  - Cardinality ***[Regin AAAI96]***
  - Regular ***[Pesant, CP2004]***

VINCOLI: CONSIDERAZIONI GENERALI

- Vincoli simbolici disponibili nella maggior parte dei linguaggi a vincoli

- Ragionamento locale vs. globale ⇒ propagazione potente
- Ragionamento locale vs. globale ⇒ costo computazionale

} Tradeoff

- Generalizzazione di vincoli che appaiono frequentemente in applicazioni reali

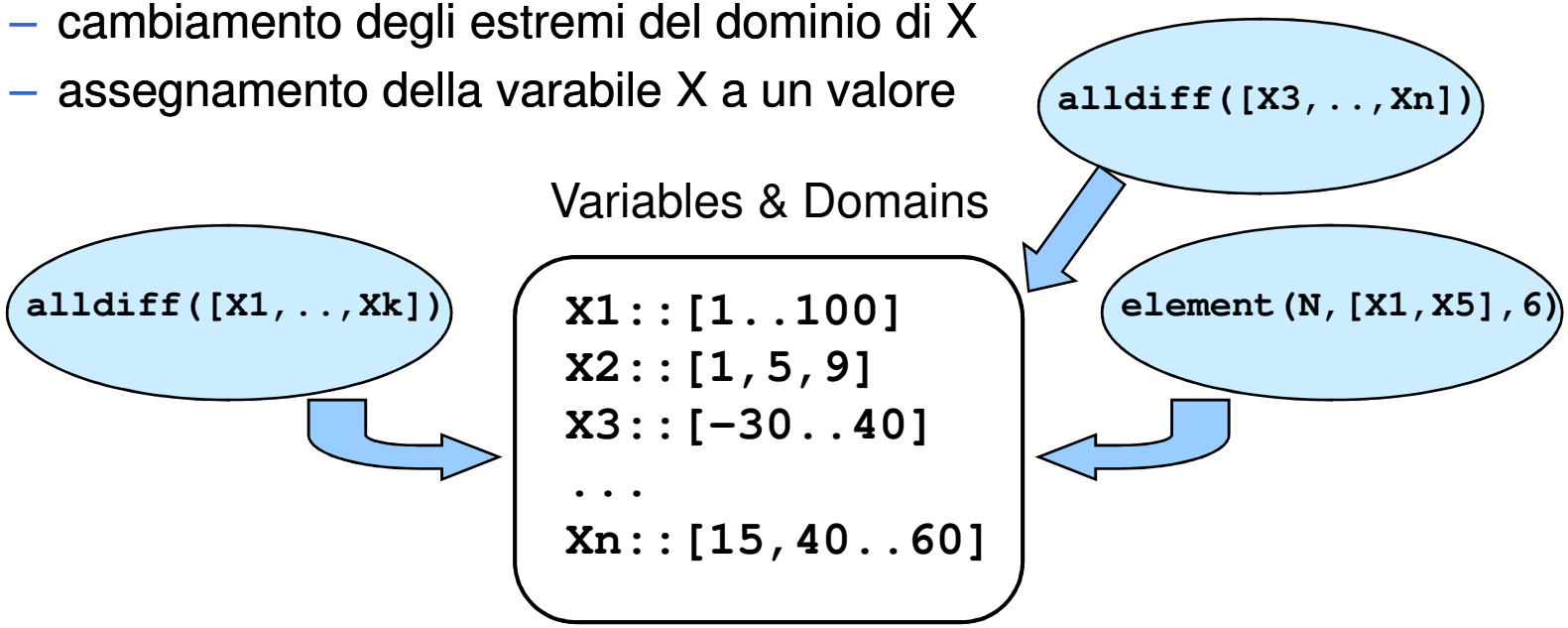
- Codice conciso e semplice da capire e scrivere

- Vincoli simbolici rappresentano sottoproblemi indipandenti (rilassamenti del problema originale)

INTERAZIONE TRA VINCOLI

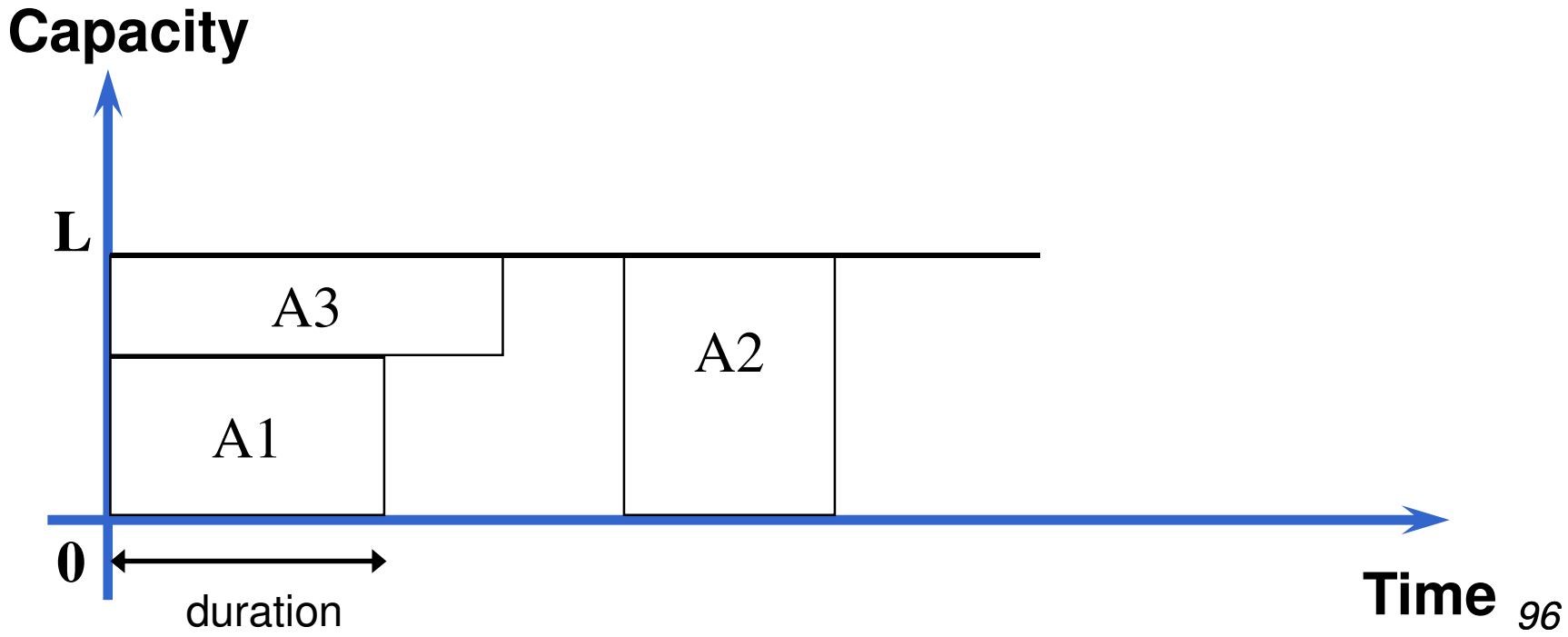
- I vincoli interagiscono attraverso variabili condivise nel constraint store

- La propagazione di un vincolo attivata quando sulla variabile X coinvolta nel vincolo si scatena un evento
  - cambiamento nel dominio di X
  - cambiamento degli estremi del dominio di X
  - assegnamento della varabile X a un valore



VINCOLI COME COMPONENTI SOFTWARE

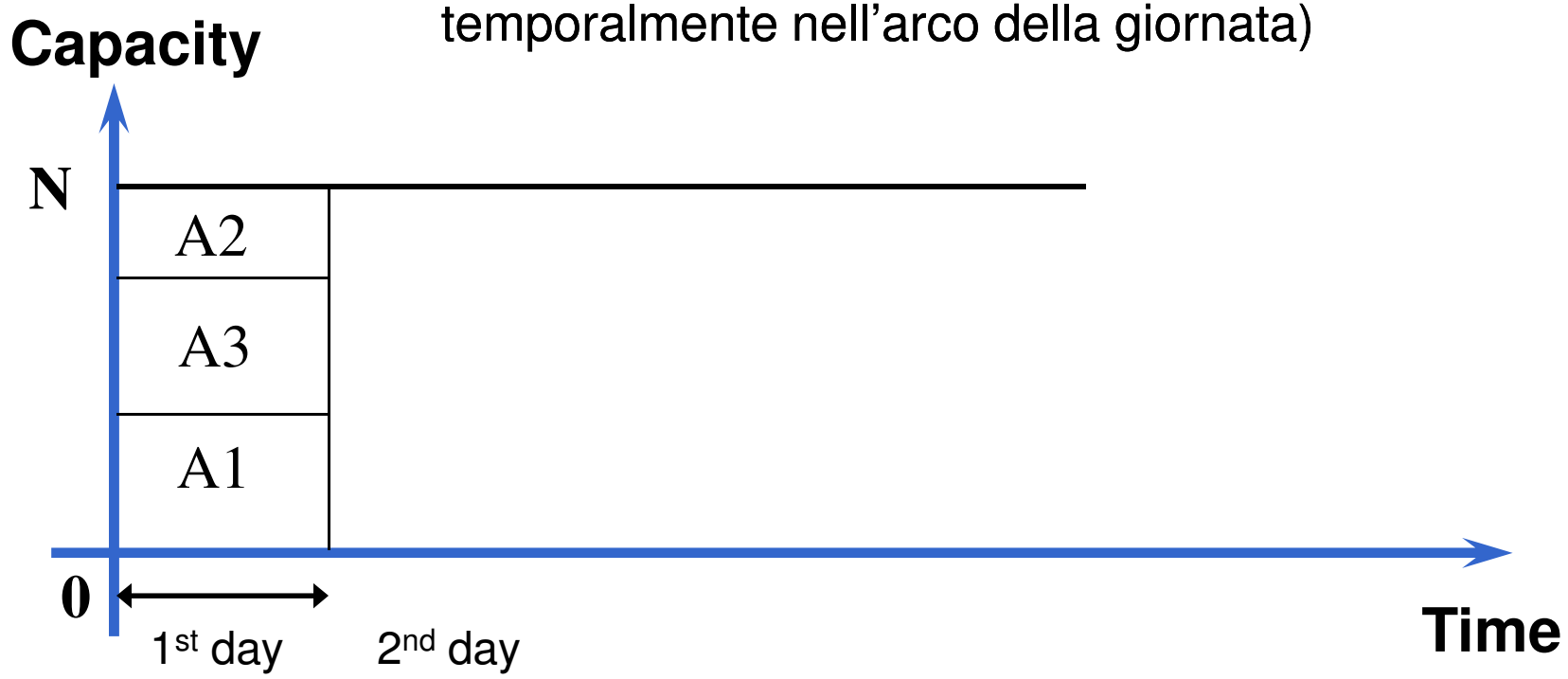
- I vincoli simbolici possono essere visti come componenti software utilizzabili in diverse situazioni. Ad esempio il vincolo cumulative puo' essere usato in vari modi
  - Scheduling (1): Attivita' A1, A2, A3 condividono la stessa risorsa con capacita' limitata. Durate sull'asse X e uso delle risorse su Y



VINCOLI COME COMPONENTI SOFTWARE

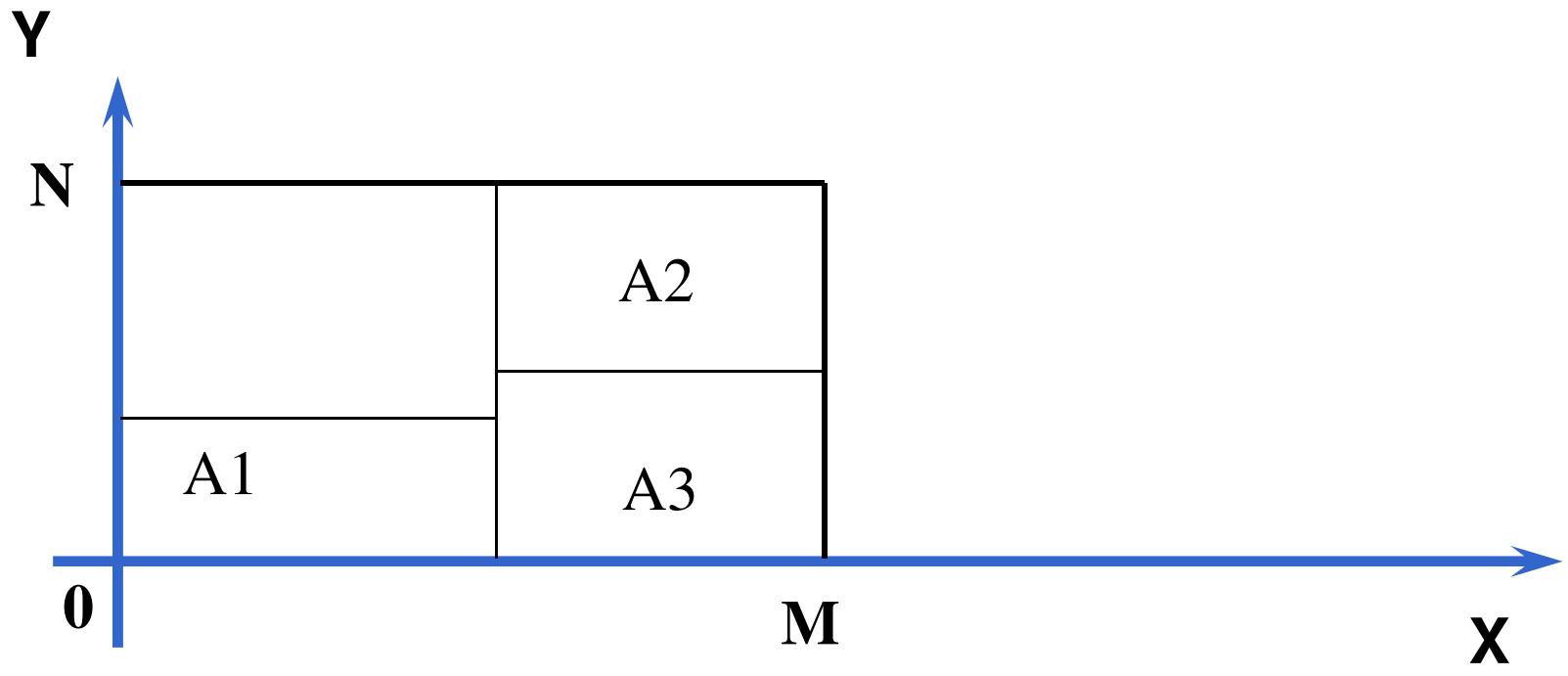
- Altro esempio di uso del vincolo cumulativo
  - Scheduling (2): *Numero limitato di risorse per giorno = N.* Rappresento i giorni sull'asse X e il numero di risorse usate su Y

(Non interessa dove le attivita' sono collocate temporalmente nell'arco della giornata)



VINCOLI COME COMPONENTI SOFTWARE

- Altro esempio di vincolo cumulativo
  - Packing: *Data una scatola di dimensioni M x N, e' necessario collocare dei pezzi in modo che le dimensioni della scatola siano rispettate.*

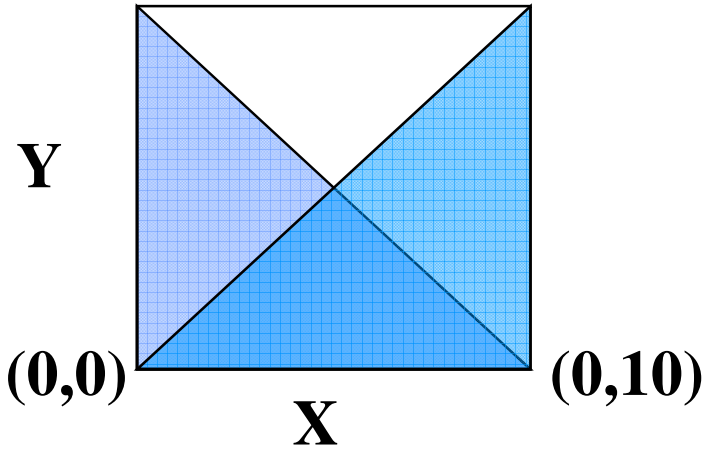


VINCOLI RIDONDANTI

- La propagazione in generale non e' completa: alcuni valori inconsistenti possono essere lasciati nei domini
- I vincoli ridondanti possono essere utili per ridurre lo spazio di ricerca anche se introducono un overhead (treadeoff).
- Un vincolo ridondante C e' un vincolo che e' implicato da altri vincoli {C1…Ck}, ma il risolutore non identifica questa implicazione a causa della sua incompletezza

VINCOLI LOGICAMENTE RIDONDANTI

```
[X,Y]::0..10,      X>=Y,
     Y <= 10-X
```

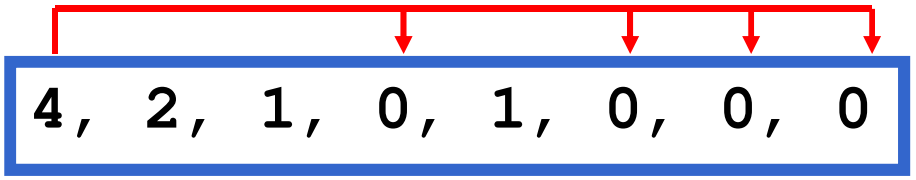


- Nessuna propagazione (entrambi i vincoli già AC).
- Se aggiungo il vincolo (logicamente ridondante) Y<=5 ho propagazione

VINCOLI RIDONDANTI

VINCOLI RIDONDANTI

- Esempio: Sequenza magica
- Dato un insieme di n+1 variabili $X_0,\ldots,X_n$. Ogni X_i deve rispettare i seguenti vincoli :
  - 0 compare X_0 volte in soluzione
  - 1 compare X_1 volte
  - …
  - n appare X_n volte.



- Vincolo occurrences(V,L,N) (fd_global) impone che V compaia N volte nella lista L
- **magic_sequence([X_0,…,X_n]):-**

```
X0,…,Xn ::[0..n],
occurrences(0,[X0,…,Xn], X0),
occurrences(1,[X0,…,Xn], X1),
...,
occurrences(n,[X0,…,Xn], Xn),
...
```

VINCOLI RIDONDANTI

- Vincolo ridondante: si noti che la somma di tutte le variabili moltiplicate per il loro valore e' uguale al numero di celle nella sequenza. Quindi, le variabili soddisfano il vincolo :
  - $X_1 + 2*X_2 + \ldots + N*X_n = N + 1$

- `magic_sequence([`$X_0,\ldots,X_n$`]) :-`

```
X0,…,Xn ::[0..n],
occurrences(0,[X0,…,Xn], X0),
occurrences(1,[X0,…,Xn], X1),
...,
occurrences(n,[X0,…,Xn], Xn),
X1+ 2*X2 +…+ N*Xn = N + 1,
```

Con P4 2Ghz, N=23

•senza vincolo ridondante

5.88s

•con vincolo ridondante

0.55s

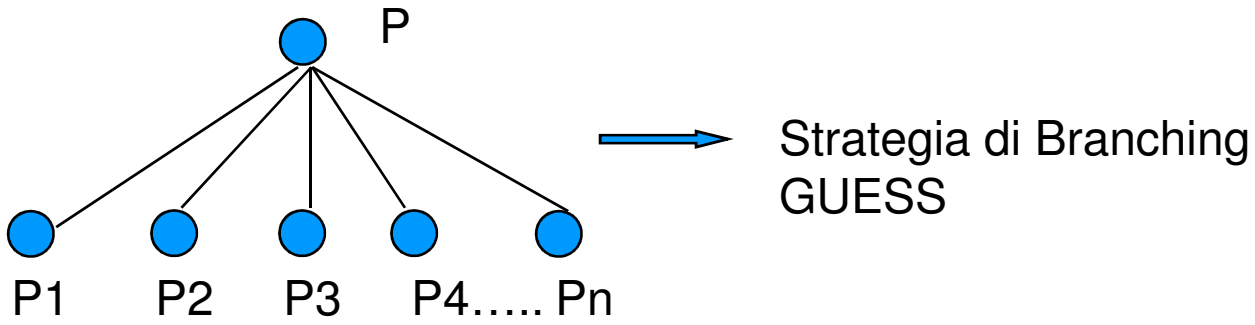
RICERCA

- La propagazione non e' in generale completa. Dopo la propagazione:
  - Trovo una soluzione ⟶ stop
  - Fallimento ⟶ backtracking
  - I domini contengono alcuni valori ⟶ SEARCH

- Ricerca: idea
  - Divide il problema in sotto-problemi (BRANCHING) and risolve ciascuno di essi indipendentemente
  - I sottoproblemi devono essere una partizione del problema originale

- Scopo: mantenere lo spazio di ricerca il piu' piccolo possibile
  - per convenzione, i rami di sinistra vengono esplorati per primi.

RICERCA



- Strategie di Branching definiscono il modo di partizionare il problema P in sottoproblemi piu' facili P1, P2, …, Pn.

- Per ogni sotto problema si applica di nuovo la propagazione. Possono essere rimossi nuovi rami grazie alle nuove informazioni derivate dal branching

RICERCA

- In Programmazione logica a vincoli la tecnica piu' popolare di branching e' detta *labeling*

- LABELING:
  - Seleziona una VARIABILE
  - Seleziona un VALORE nel suo dominio
  - Assegna il VALORE alla VARIABILE

  } ***EURISTICHE***

- L'ordine in cui le variabili e i valori vengono scelti (la search strategy) non influenza la completezza dell'algoritmo ma ne influenza penantemente l'efficienza.

- Attivita' di ricerca volta a trovare buone strategie.

ESEMPIO COMPLETO

- Map Coloring:

```
map_colouring([V1,V2,V3,V4,V5]):-
        V1::[red,green,yellow,blue,pink],
        V2::[red,green,yellow,blue,pink],
        V3::[red,green,yellow,blue,pink],
        V4::[red,green,yellow,blue,pink],
        V5::[red,green,yellow,blue,pink],
        V1≠V2, V1≠V3, V1≠V4, V1≠V5, V2≠V3,
        V2≠V4, V2≠V5, V3≠V4, V4≠V5,
        labeling([V1,V2,V3,V4,V5]).
```

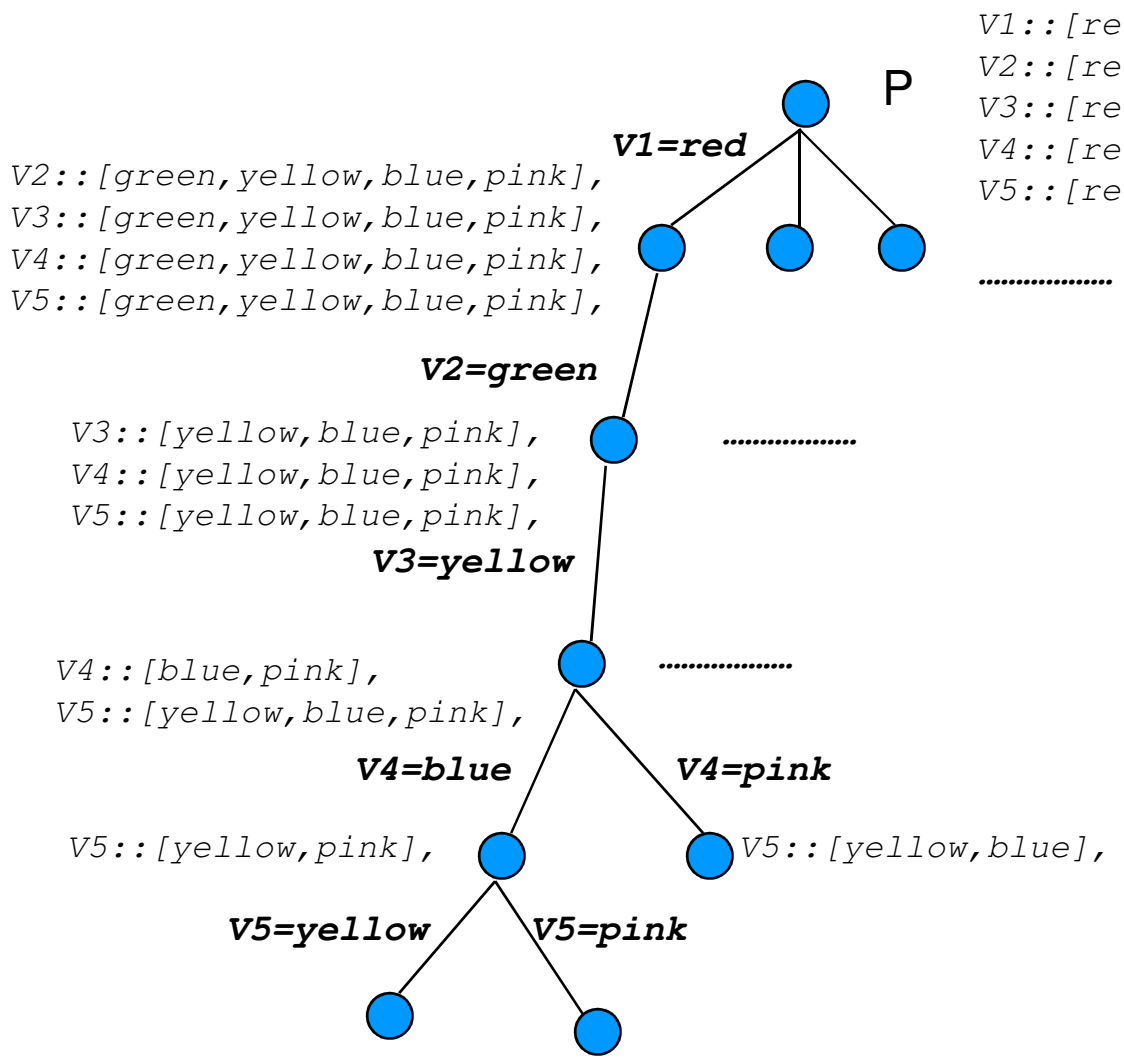
Variabili & domini

vincoli

ricerca

- Separazione tra
  - Modello del problema
  - Strategia per risolverlo

SPAZIO DI RICERCA



STRATEGIE DI RICERCA: CRITERI GENERALI

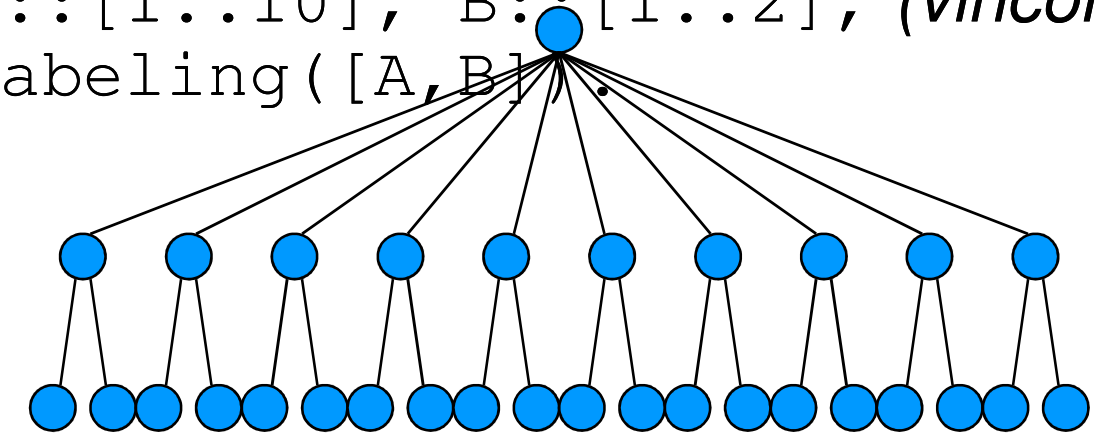
- Scelta della Variabile: prima istanzio le variabili piu' difficili
  - FIRST FAIL: seleziono prima la variabile con dominio piu' piccolo
  - MOST CONSTRAINING PRINCIPLE: seleziono prima la variabile coinvolta nel maggior numero di vincoli
  - APPROCCI IBRIDI: combinazioni dei due

- Scelta del valore: valori piu' promettenti prima
  - LEAST CONSTRAINING PRINCIPLE.

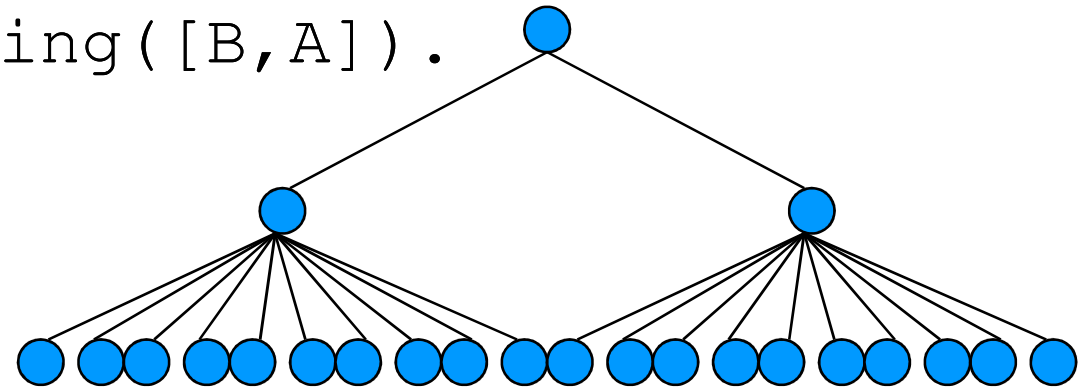
- Sono poi state definite numerose strategie dipendenti dal problema.

First Fail

- `A::[1..10], B::[1..2],` *(vincoli ...),* `labeling([A,B]).`



- … `labeling([B,A]).`



COME SCEGLIERE UNA STRATEGIA

- Non esistono regole definite per scegliere la migliore strategia di ricerca. Dipende dal problema che dobbiamo risolvere e tipicamente la scelta viene effettuata dopo prove computazionali con diverse strategie

- CRITERIO: una strategia e' piu' efficiente se rimuove prima rami dello spazio delle soluzioni.

- PARAMETRI da considerare
  - tempo computazione
  - numero di fallimenti.

STRATEGIA CRONOLOGICA PER SCHEDULING

- Nei problemi di scheduling, i domini delle variabili contengono possibili istanti di inizio delle attività.
- Due problemi
  - I domini possono contenere un numero di valori grandissimo.
  - Se uno schedule che assegna una attività a un certo istante di tempo T è inconsistente, è poco probabile che uno schedule che sposta l'attività di un'unità di tempo (T+1) sia consistente

- SOLUZIONE: si usa una strategia che
  - seleziona l'attività che ha earliest start time più basso
  - nel ramo di sinistra la assegna a tale valore
  - nel ramo di destra la postpone

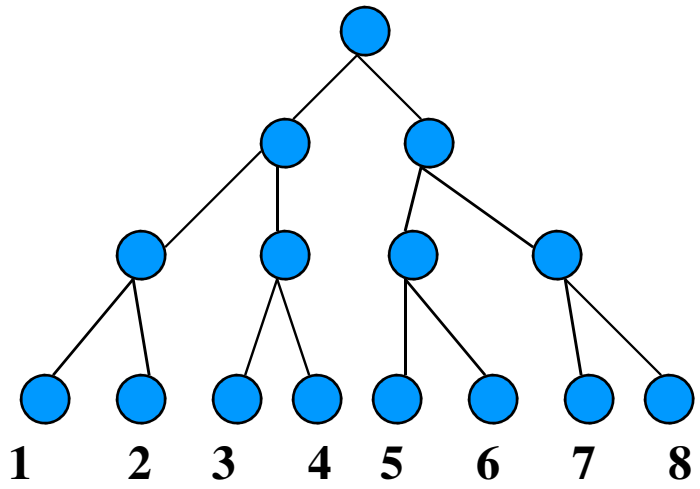
- Molti sforzi per definire strategie ottimali per classi di problemi

VARIANTI ALLA DEPTH FIRST

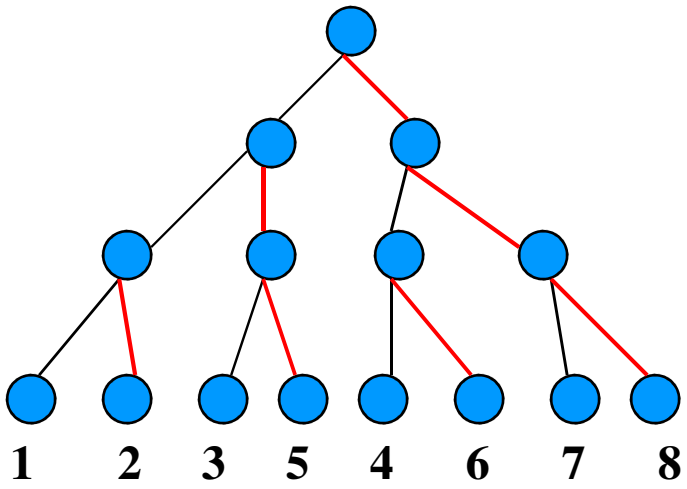
- Una strategia molto usata recentemente in programmazione a vincoli è la LIMITED DISCREPANCY SEARCH

- Ipotesi: l'euristica per la scelta del valore sbaglia raramente

DEPTH FIRST

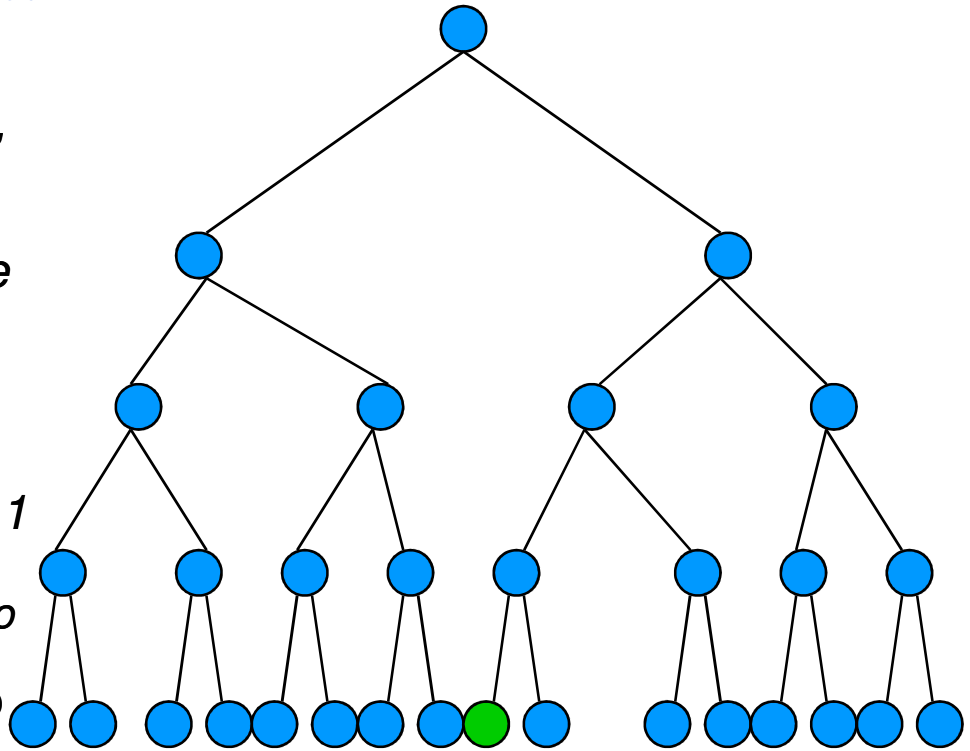


LIMITED DISCREPANCY



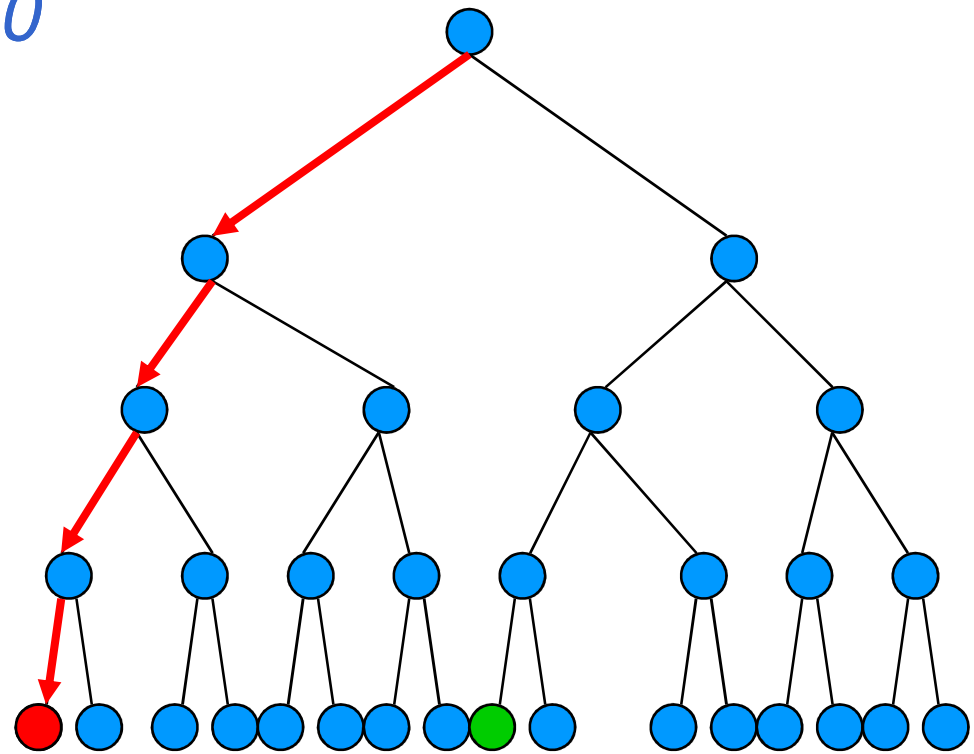
VARIANTI ALLA DEPTH FIRST

- *Esplorazione depth-first: in profondità lungo il ramo di sinistra, poi torna indietro 1 passo, ecc.*
- *esplora le foglie di sinistra prima di quelle di destra.*
- *Supponiamo che l'euristica sia molto buona: in tutto il percorso compie solo 1 errore.*
  - *errore all'ultimo nodo => ☺*
  - *errore al primo => ☹*



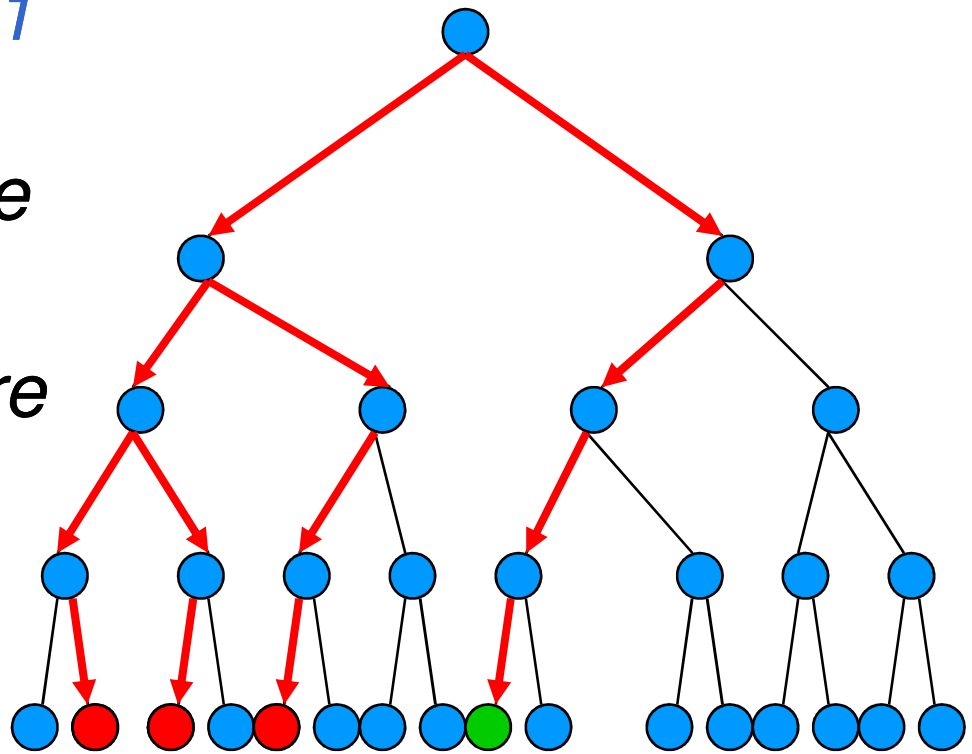
LIMITED DISCREPANCY SEARCH

- *Discrepanza=0*
  seguo l'euristica



Limited Discrepancy Search

- *Discrepanza=1*
  Permetto che ci sia un errore nell'euristica.
- *Prendo sempre il ramo di sinistra e prendo solo una volta il ramo di destra*



OTTIMIZZAZIONE

- In molte applicazioni non siamo interessati a soluzioni ammissibili, ma nella soluzione ottima rispetto a un certo criterio.
- ENUMERAZIONE ⟹ inefficiente
  - trova tutte le soluzioni ammissibili
  - scegli la migliore

- I linguaggi di Programmazione a Vincoli in generale incapsulano una forma di Branch and Bound
  - ogni volta che viene trovata una soluzione di costo C*, viene imposto un vincolo sulla parte rimanente dello spazio di ricerca. Il vincolo afferma che soluzioni successive (il cui costo e' C) devono essere migliori della migliore soluzione trovata fin'ora.

$$C < C^*$$

BRANCH AND BOUND

- Il vincolo aggiunto fa normalmente poca propagazione.
- Si usa una variabile Z che rappresenta la funzione obiettivo e la si lega alle variabili decisionali, esempio somma di costi
- Se il legame tra Z e tali variabili e' forte, la propagazione elimina molti rami dall'albero, altrimenti, cercare una soluzione ottima è molto difficile
- Esempi
  - Per lo scheduling funziona bene (min la lunghezza dello schedule)
  - Per le somme di costi meno bene
  - Per la minimizzazione dei tempi di setup, non funziona affatto.

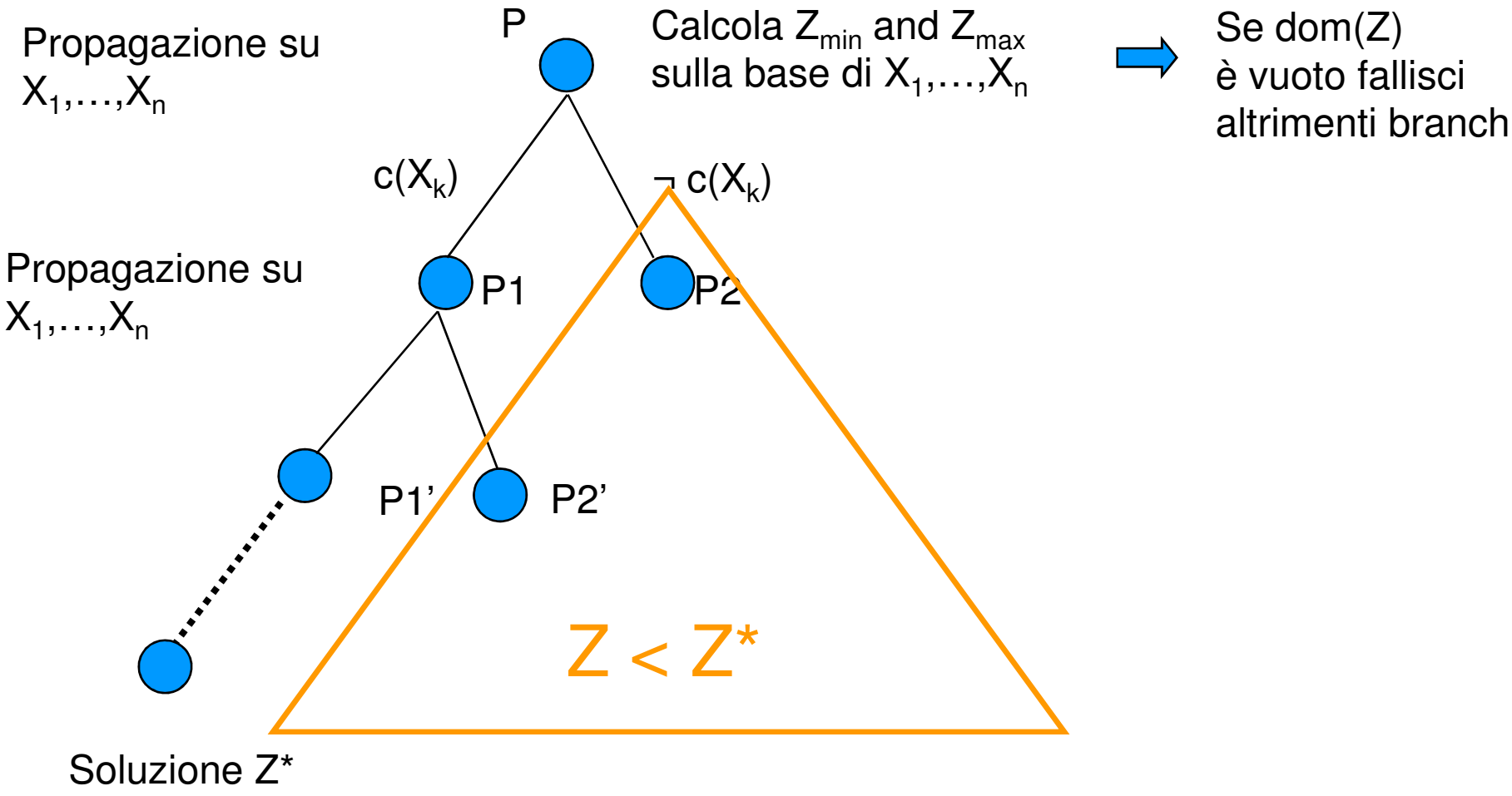
OTTIMIZZAZIONE

- La Ricerca Operativa ha una lunga tradizione nella soluzione di problemi di ottimizzazione

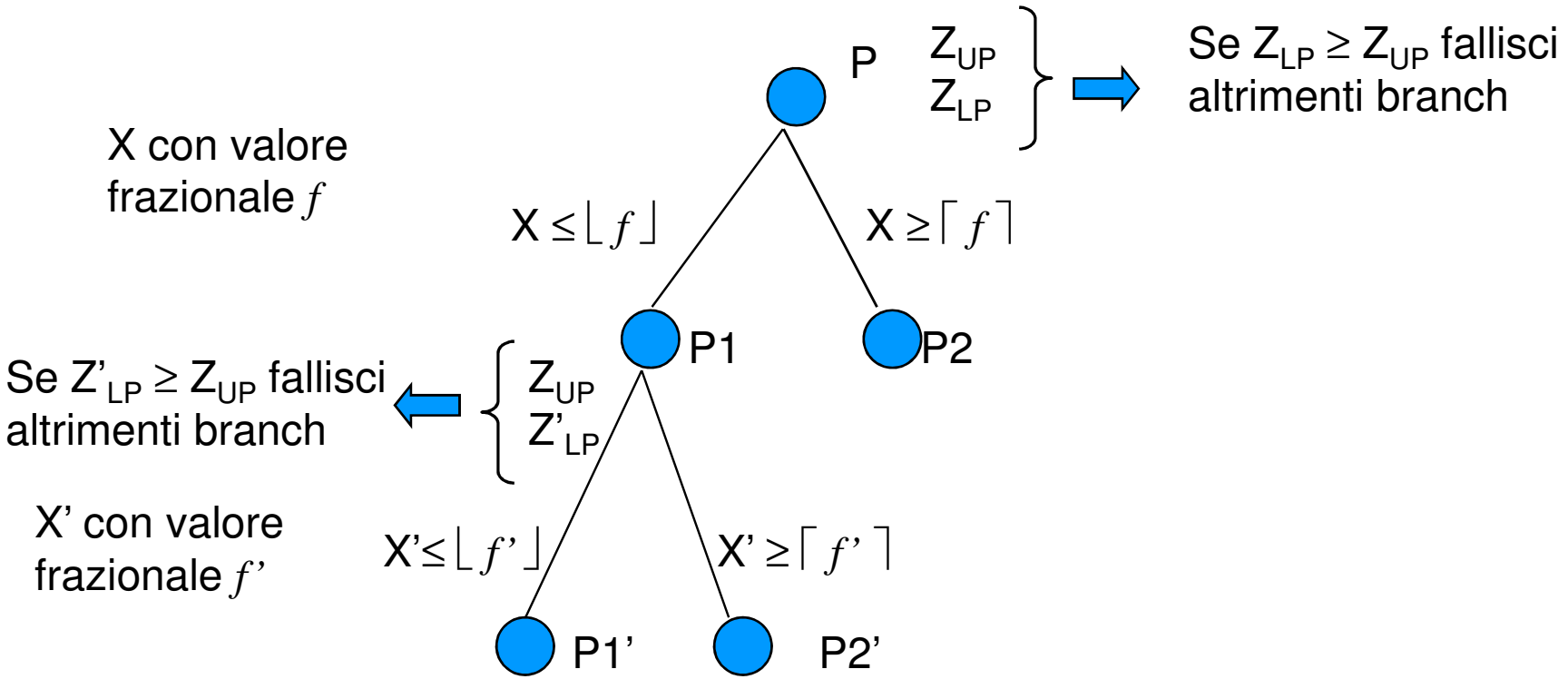
- I metodi Branch & Bound si basano sulla soluzione ottima di un rilassamento del problema che produce un BOUND, ossia una valutazione ottimistica del problema
  - Rilassamento: problema in cui alcuni vincoli vengono rilassati o tolti

- Linea di ricerca particolarmente promettente: integrazione di tecniche di Ricerca Operativa nella programmazione a vincoli al fine di migliorarne l'efficienza nella risoluzione di problemi di ottimizzazione

BRANCH & BOUND in PROGRAMMAZIONE A VINCOLI



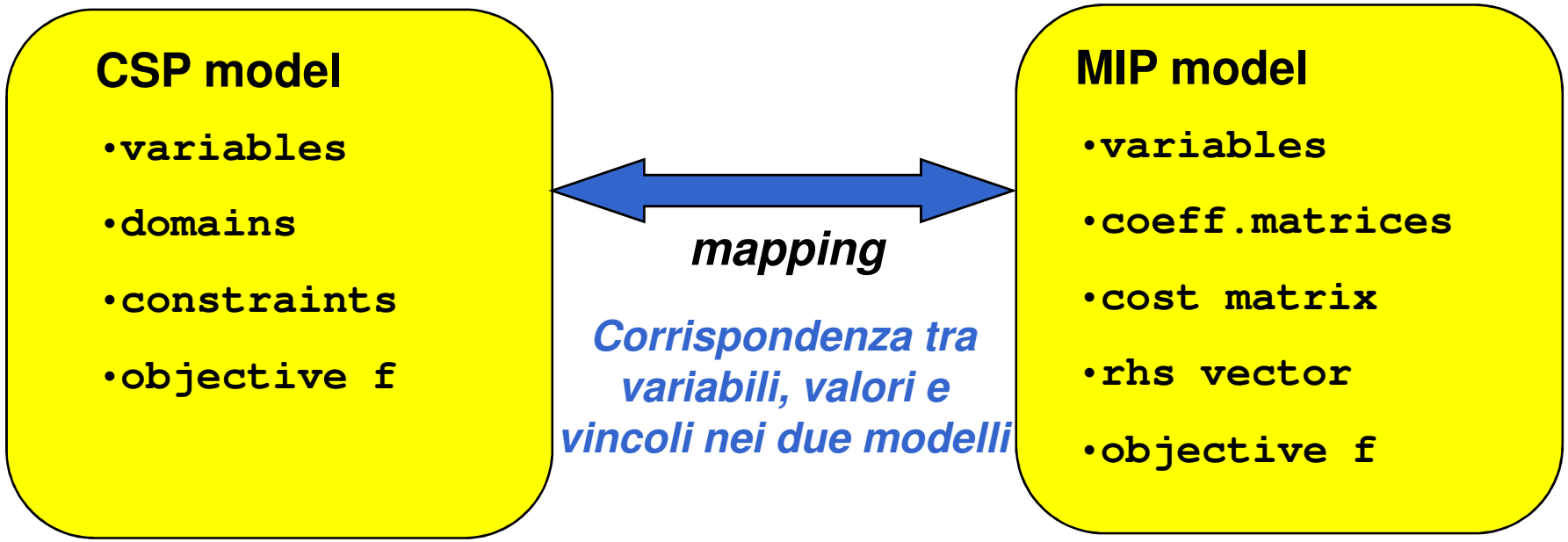
BRANCH & BOUND
in PROGRAMMAZIONE LINEARE INTERA



INTEGRAZIONE CP - OR

- Motivazioni per approcci ibridi CP e OR
  - Combinare i vantaggi dei due approcci
    - CP: modellazione, interazioni tra vincoli, pruning basato su ammissibilità
    - MIP: ragionamento globale (rilassamento lineare), metodi risolutivi specifici per classi di problemi, pruning basato su ottimalità
  - Superare le limitazioni dei due
    - CP: ragionamento su objective function scarso
    - MIP: modelli poco flessibili (in presenza di cutting planes), no vincoli globali

INTEGRAZIONE: MODELLING



INTEGRATION: MODELLING

- Modelli CP e MIP devono coesistere, cooperare o unirsi in un singolo linguaggio.

- Diversi livelli di integrazione:
  - Approcci che forniscono all'utente il linguaggio CP e nascondono un modello OR trasparente all'utente *[Rodosek, Wallace,Hajian 98] [Focacci,Lodi,Milano CP99, Regin CP99] [Refalo CP2000]*
  - Approcci che forniscono all'utente entrambi I modelli *[Beringer, De Backer 95] [Heipcke PhD99]*
  - Approcci che uniscono i due modelli in un linguaggio unificato *[Hooker et al. AAAI99]*
  - *Approcci basati sulla decomposizione [Hooker CP2005]*

VINCOLI GLOBALI DI OTTIMIZZAZIONE

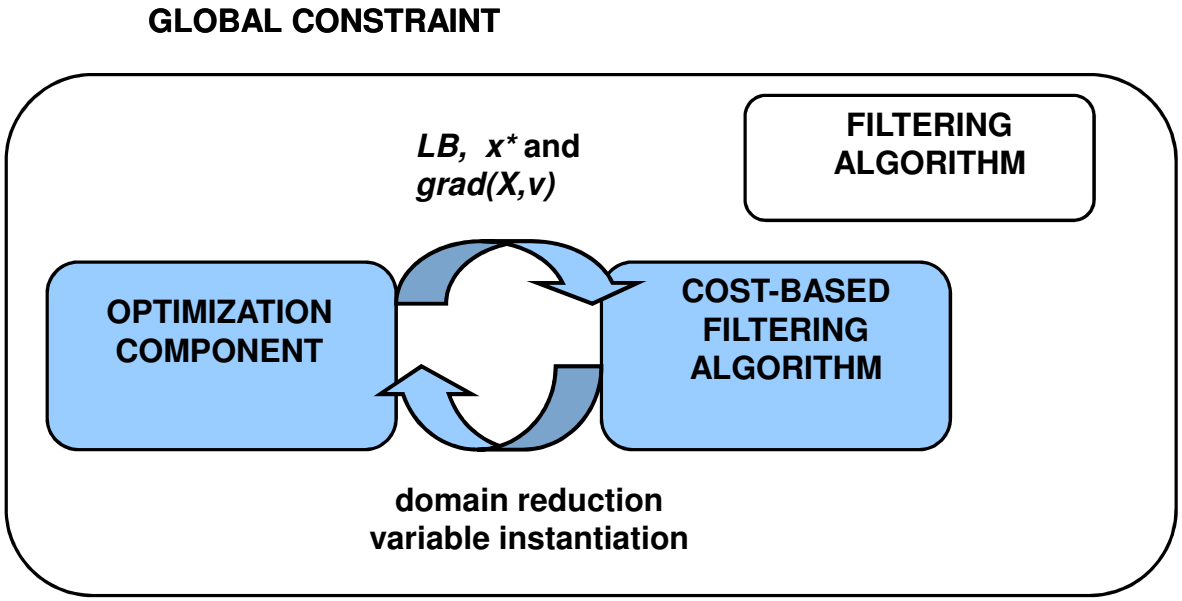
- Molti vincoli globali che rappresentano problemi risolubili in tempo polinomiale possono essere estesi con una variabile costo

`cost_alldiff(X, Cost, H)` vero se e solo se le variabili in X sono tutte diverse e la somma dei loro costi e' minore di H. Consistenza e filtering basati su un algoritmo di flusso

`cost_gcc(X,Val,LB,UB,Cost, H)` vero se e solo se le il valore Val compare X un numero di volte compreso tra LB e UB e la somma dei costi delle var in X e' minore di H. Consistenza e filtering basati su un algoritmo di flusso

E se il problema non e' polinomiale???

VINCOLI GLOBALI DI OTTIMIZZAZIONE



- Si aggiunge all'algoritmo di filtering tradizionale un componente di ottimizzazione che incapsula un rilassamento del vincolo e fornisce:
  - Lower bound (soluzione ottima del rilassamento)
  - funzione gradiente che calcola il costo var = valore

VINCOLI GLOBALI DI OTTIMIZZAZIONE

- Propagazione basata su Lower Bound:
  dal *LB* verso la funzione obiettivo *Z::[Zmin..Zmax]:*
  LB < Zmax

- cost-based filtering:
  Dalla funzione gradiende verso le variabili decisionali:

  Per ogni *Xi::[v1,v2,…,vm]* e *vj* esiste una funzione gradiente *grad(Xi,vj)* che misura il costo addizionale se *Xi = vj*

  Se $LB + grad(Xi,vj) \geq Zmax$ allora $Xi \neq vj$

  classico *variable fixing* di ricerca operativa.

VINCOLI GLOBALI DI OTTIMIZZAZIONE

• L'esempio piu' semplice di funzione gradiente è rappresentato dai *costi ridotti* calcolati dal rilassamento lineare o in alcuni bound combinatori.

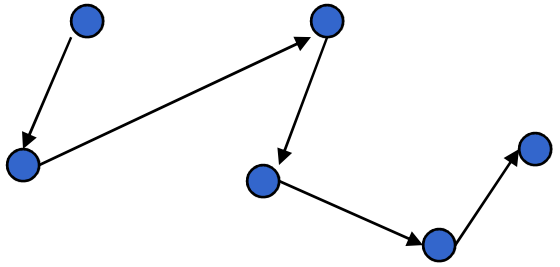
• Esempio: path constraint

PATH CONSTRAINT

Dato un grafo diretto *G=(V,A)* with *|V| = n*, associamo ad ogni nodi ***i*** una variabile X_i il cui dominio contiene i possibili next nel cammino, il **path constraint**

$X_0::D_0, X_1::D_1, \ldots, X_k::D_k$

$path([X_0, X_1, \ldots, X_k])$



È vero se e solo se l'assegnamento di valori alle variabili $x_0, x_1, \ldots, x_k$ definisce un cammino semplice che coinvolge tutti i nodi ***0,…,k***.

In generale il vincolo path e' un multi-path

$path([X_0, X_1, \ldots, X_k], NumberOfPaths)$

MAPPING

CP-Model:

$X_i :: [i_1, i_2, \ldots, i_m]$ i=0..n

path($[X_0, X_1, \ldots, X_n]$)

$C_i :: [c_{i1}, c_{i2}, \ldots, c_{im}]$ i=0..n

$C_0 + \ldots + C_n = Z$

minimize(Z)

Mapping

ILP-Model

$$\min z = \sum_{i \in V} \sum_{j \in V} c_{ij}\, x_{ij}$$

$$\sum_{i \in V} x_{ij} = 1 \quad j \in V \quad (A)$$

$$\sum_{j \in V} x_{ij} = 1 \quad i \in V \quad (B)$$

$$\sum_{i \in S} \sum_{j \in V \setminus S} x_{ij} \geq 1 \quad S \subset V \quad S \neq \emptyset$$

x_{ij} integer

$X_i = i_j$	⟷	$x_{ij}=1$
path $([X_0, X_1, \ldots, X_n])$	⟷ *relaxed in*	(A) + (B)
$C_i = c_{ij}$	⟷	c_{ij} in z

PATH CONSTRAINT

- Il modello ILP precedente può fornire un bound usando il simplesso oppure algoritmi ad-hoc. Ad esempio con l'Hungarian algorithm otteniamo un bound Z_{AP}, una soluzione **intera** x^*, e i costi ridotti. Inoltre, tale algoritmo è incrementale: ($O(n^3)$ la prima soluzione, $O(n^2)$ ogni ri-computazione).

- Questo bound puo' essere molto scarso (soprattutto per TSP simmetrici). Lo si puo' migliorare con la generazione di **cutting planes**.

- I piu' semplici cutting planes sono I Subtour Elimination Constraints (SECs) la cui separazione è polinomiale.

PROBLEMI SOVRA-VINCOLATI

- Alcuni problemi nella vita reale non hanno soluzione: sono sovra-vincolati
- CLP in questo caso dà come risposta semplicemente ‘no’
- Questo in applicazioni reali non è sufficiente
- Come gestire questo problema?

SOFT CONSTRAINTS

- Uno dei metodi è quello di usare vincoli soft (che possono essere rilassati)
  - cerchiamo di massimizzare i vincoli soft che sono soddisfatti
- I vincoli reificati possono essere usati come vincoli soft: massimizzo il numero di vincoli soddisfatti
- Es. Orario delle lezioni:
  - *cerco di soddisfare le richieste degli insegnanti*

```
Mat #> 10, Fisica #< 14, …
```

  - *Con vincoli Soft:*

```
Mat #> 10 #<=>B1, Fisica#< 14 #<=> B2,
sumlist([B1,B2,…],Sum),
F #= -Sum,
minimize(labeling(…),F),
```

SOFT CONSTRAINTS

- Esistono vincoli globali soft:
  - **`softalldiff`**
  - **`softgcc`**
  - **`softregular`**

- Incapsulano un livello di violazione
  - Variable-based: minimo numero di variabili che devono cambiare valore affinchè il vincolo sia soddisfatto
  - Decomposition-based: se un vincolo globale C si può decomporre in una serie di vincoli binari C_{DEC} allora la violazione e' il numero di vincoli binari violati

CONSTRAINT PROGRAMMING TOOLS

	Logic-based	*a oggetti*
Open source	Eclipse   	 *Gecode*
Commerciali	***CHIP*** 	*ILOG* 

… e molti altri

CONSTRAINT PROGRAMMING TOOLS

- CLP(R) ***[Jaffar et al. Trans. Progr. Lang and Sys. 92],***
- Prolog III ***[Colmerauer CACM(33) 90]***,
- CHIP ***[Dincbas et al., JICSLP88],***
- CLP(PB) ***[Bockmayer ICLP95]***,
- Ecl^{i}ps^{e} ***[Wallace et al.97],*** Conjunto ***[Gervet, Constraints(1), 97]***
- *Oz* ***[Smolka JLP 91]***,
- *AKL* ***[Carlson, S.Haridi, S.Janson ILPS94],***
- *CIAO* ***[Hermenegildo et al. PPCP94]***
- *ILOG* ***[Puget SPICIS94], [Puget, Leconte ILPS95]***
- *CHARME* ***[Bull Corporation 90]***
- ***SICStus***

VANTAGGI CP

- Modellazione di problemi semplice
- Regole di propagazione combinate attraverso vincoli
- Flessibilità nel trattamento di varianti di problemi:
  - semplice introduzione di nuovi vincoli
  - interazione trasparente con nuovi vincoli
- Controllo semplice della ricerca

LIMITAZIONI

- Lato ottimizzazione non molto efficace: integrazione con tecniche di Programmazione Lineare Intera

- Problemi sovravincolati:
  - non c'e' modo di rilassare i vincoli
  - vincoli e preferenze

- Cambiamenti dinamici:
  - cancellazione/aggiunta di variabili
  - cancellazione/aggiunta di valori nel dominio
  - cancellazione/aggiunta di vincoli

PER SAPERNE DI PIU' …..

- *Conferenze:*
  - *International Conference on Principles and Practice of Constraint Programming CP*
  - *International Conference on Practical Applications of Constraint Technology PACT (PACLP)*
  - *Logic programming (ILPS - ICLP - JICSLP)*
  - *AI (ECAI - AAAI - IJCAI)*
  - *OR (INFORMS - IFORS)*
  - *Conferenza (CP-AI-OR)*

- *Libro: K. Marriott and P. Stuckey*

  Programming with constraints: An Introduction
  - *MIT Press*

PER SAPERNE DI PIU' …..

- *Riviste:*
  - *Constraint - An International Journal*
  - *Riviste di AI - LP - OR*
- *Applicazioni industriali:*
  - *COSYTEC, ILOG, ECRC, SIEMENS, BULL*

- *News group:* **`comp.constraints`**
- *Association of Constraint Programming: ACP*
  `http://4c110.ucc.ie/acp/a4cp/`