

PROGRAMMAZIONE LOGICA A VINCOLI

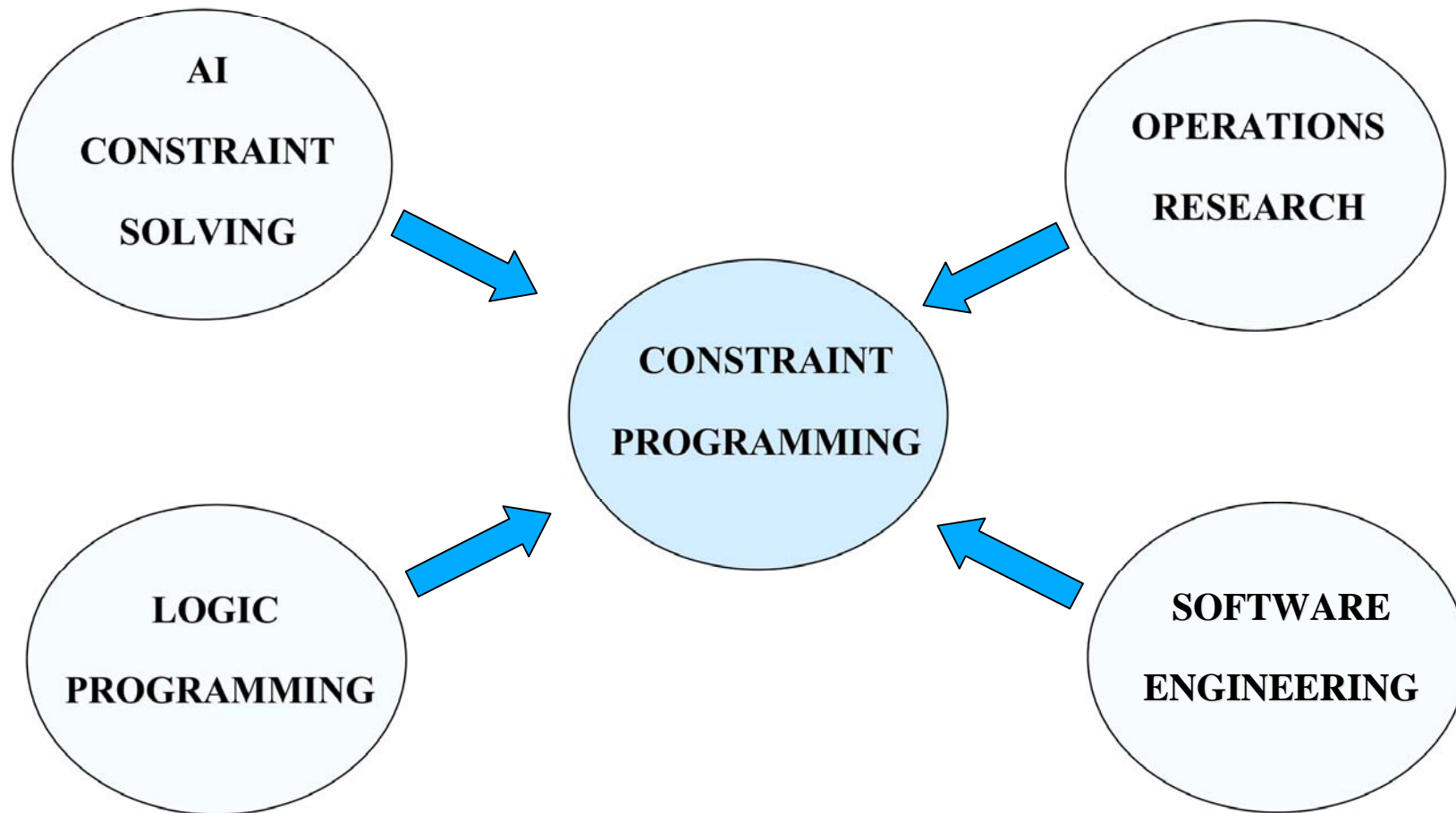
Michela Milano

DEIS - Università di Bologna

`mmilano@deis.unibo.it`

`http://www-lia.deis.unibo.it/Staff/MichelaMilano/`

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

PROBLEMI DI SODDISFACIMENTO DI VINCOLI

- Un problema di soddisfacimento di vincoli è definito da:
 - un insieme di variabili ($X_1, X_2 \dots X_n$)
 - un dominio discreto per ogni variabile ($D_1, D_2 \dots D_n$)
 - un insieme di vincoli su queste variabili:

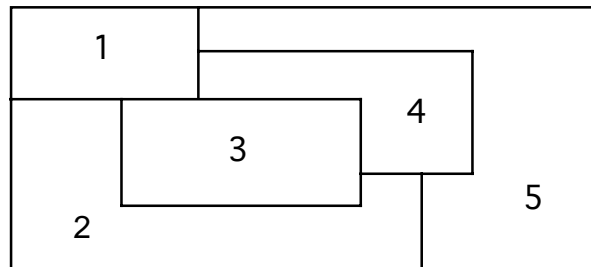
vincolo: una relazione tra variabili che definisce un sottoinsieme del prodotto cartesiano dei domini

$$D_1 \times D_2 \times \dots \times D_n$$

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

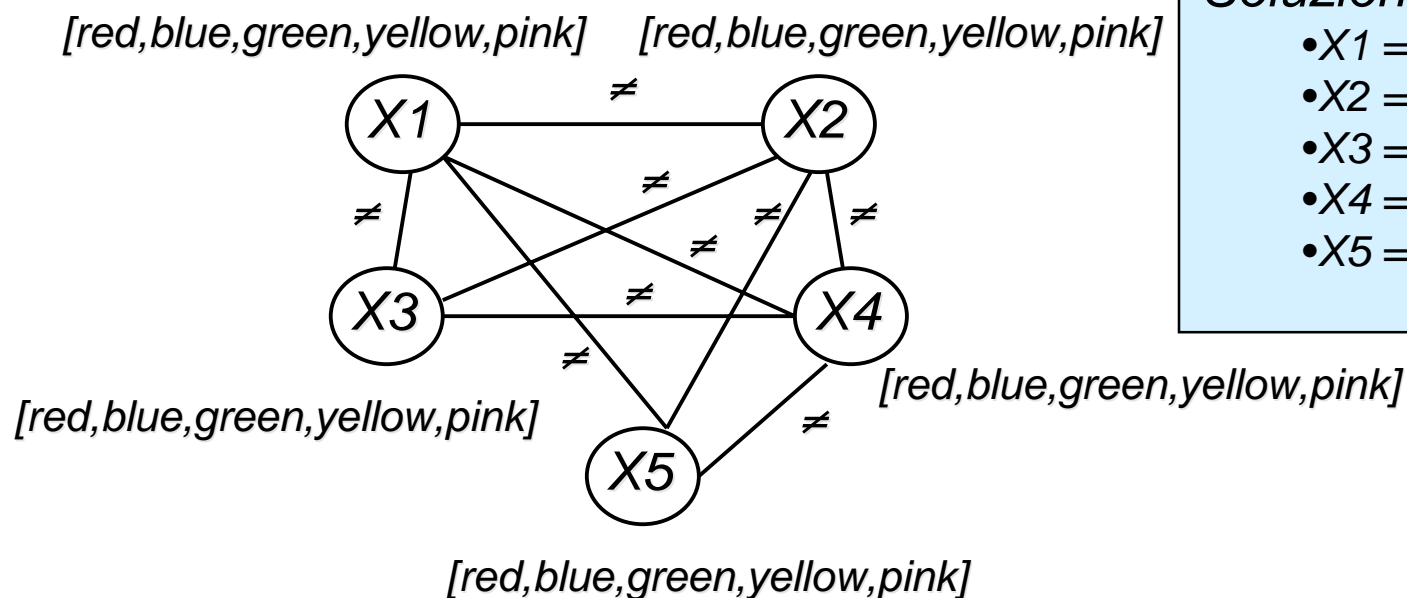


- Trovare un assegnamento di colori alle variabili consistente con i vincoli
 - variabili X_1, X_2, X_3, X_4, X_5 : zone
 - domini D_1, D_2, D_3, D_4, D_5 : [red, blue, green, yellow, pink]
 - vincoli : $near(X_i, X_j) \Rightarrow X_i \neq X_j$

CONSTRAINT GRAPHS

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

- *variabili* \longleftrightarrow *nodi*
- *vincoli* \longleftrightarrow *(iper)-archi*



• Soluzione ammissibile:

- X1 = red
- X2 = green
- X3 = blue
- X4 = yellow
- X5 = pink

PROBLEMI DI OTTIMIZZAZIONE

Un problema di ottimizzazione è definito da:

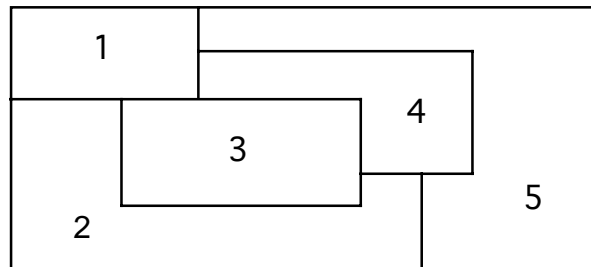
- *un insieme di variabili* ($X_1, X_2 \dots X_n$)
- *un dominio discreto per ogni variabile* ($D_1, D_2 \dots D_n$)
- *un insieme di vincoli su queste variabili:*

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

- *una funzione obiettivo* $f(X_1, X_2 \dots X_n)$

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

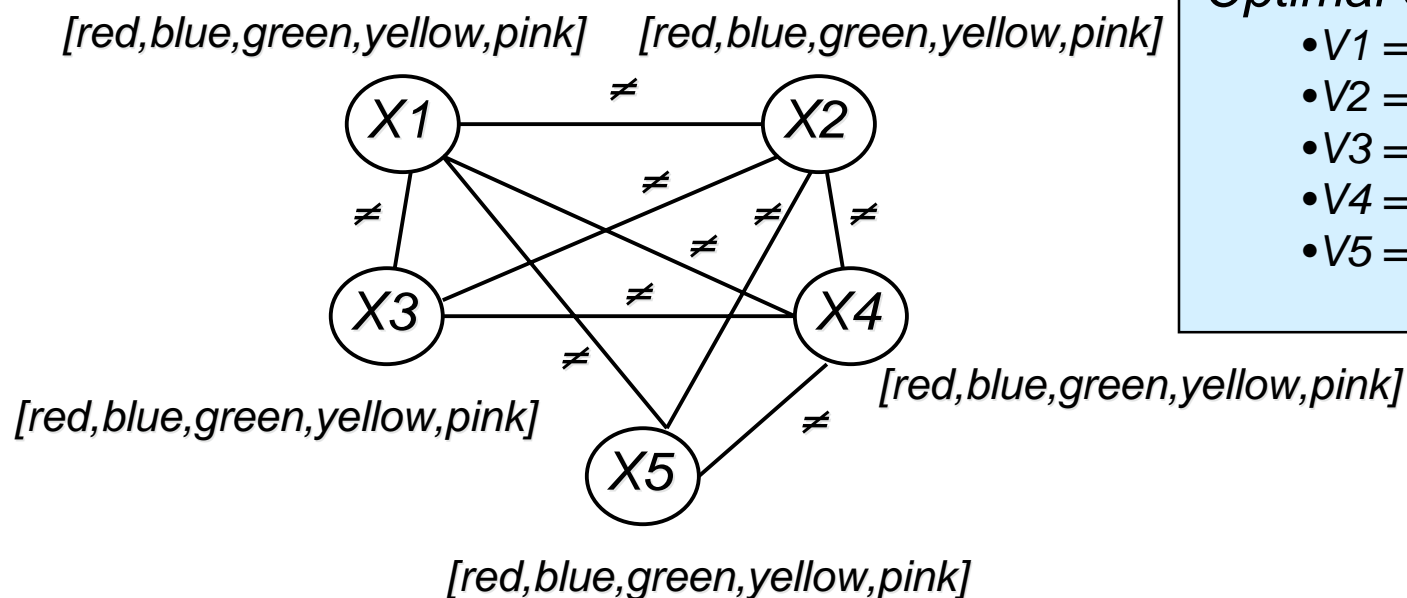


- Trovare un assegnamento di colori alle zone tale che due zone adiacenti sono colorate con colori diversi, e **MINIMIZZANDO** il numero di colori usati
 - variabili V_1, V_2, V_3, V_4, V_5 : zone
 - domini D_1, D_2, D_3, D_4, D_5 : [red, blue, green, yellow, pink]
 - vincoli: $\text{near}(X_i, V_j) \Rightarrow X_i \neq X_j$

CONSTRAINT GRAPHS

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

- *variabili* \longleftrightarrow *nodi*
- *vincoli* \longleftrightarrow *(iper)-archi*

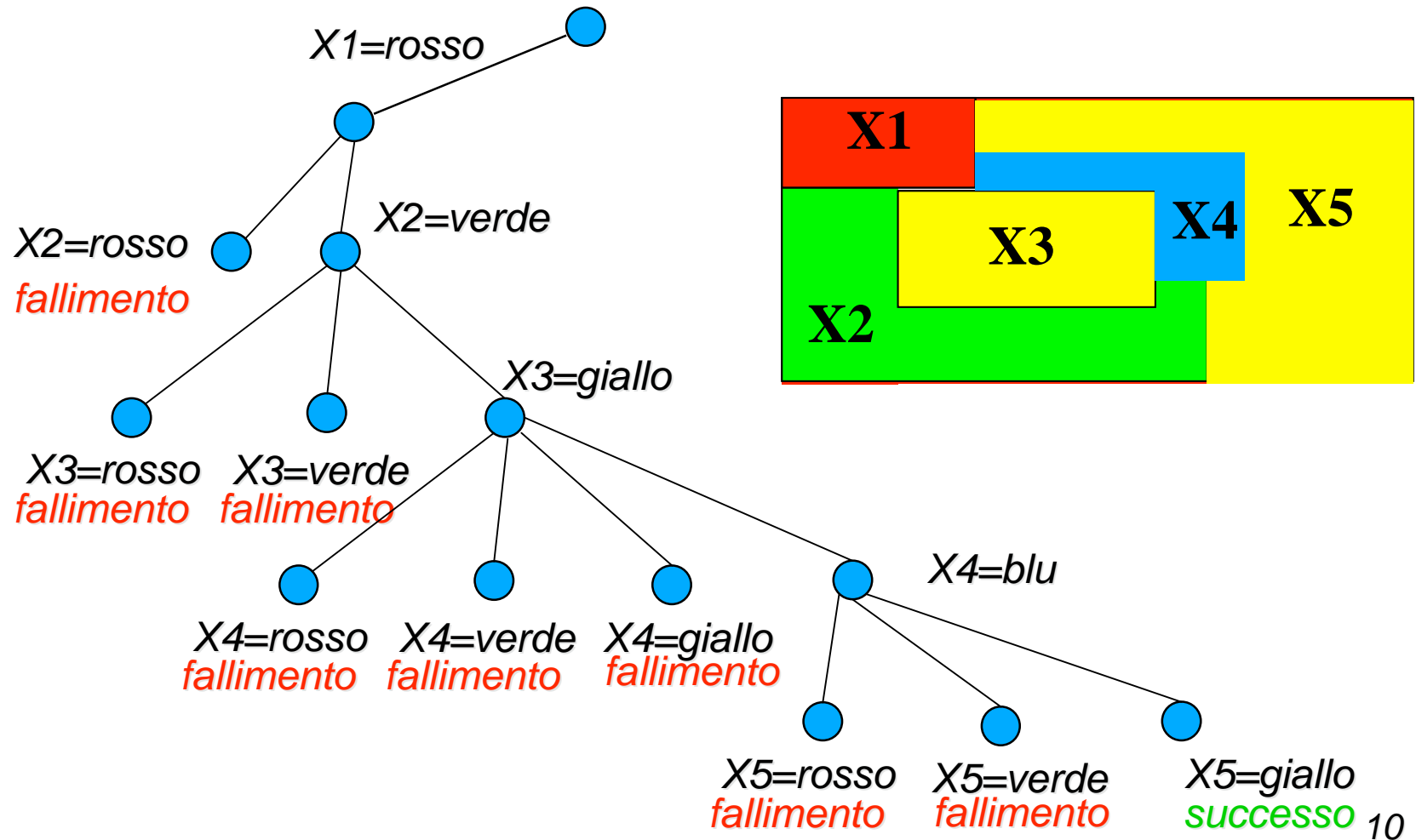


• Optimal Solution:

- V1 = red
- V2 = green
- V3 = yellow
- V4 = blue
- V5 = yellow

ESEMPIO: Map Coloring

- Algoritmo semplice (ma a volte inefficiente)

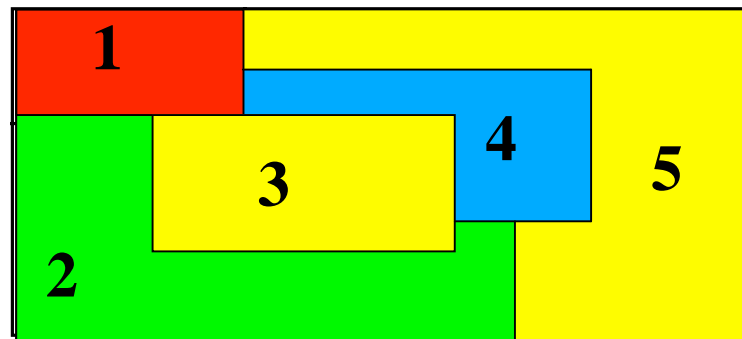


CONCETTI GENERALI

- **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: $\forall i,j \in [1,5], \text{near}(i, j) \rightarrow X_i \neq X_j$
- Soddisfacibilità
 - *Una possibile soluzione*

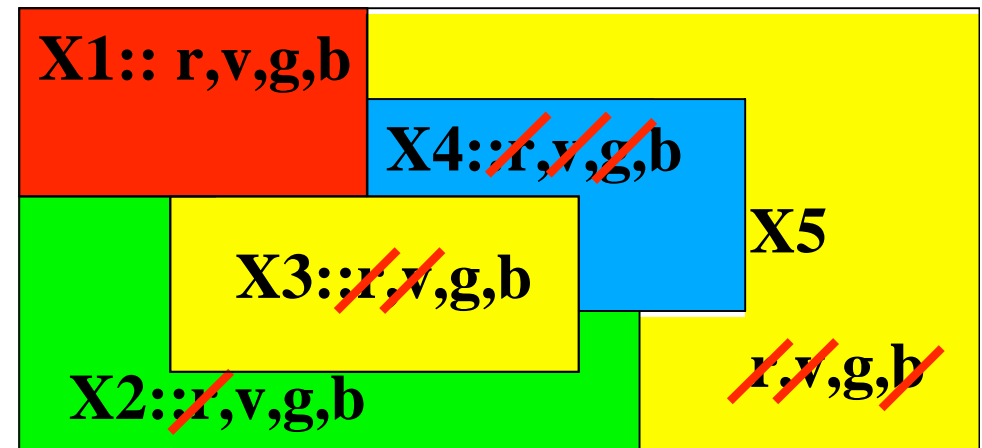
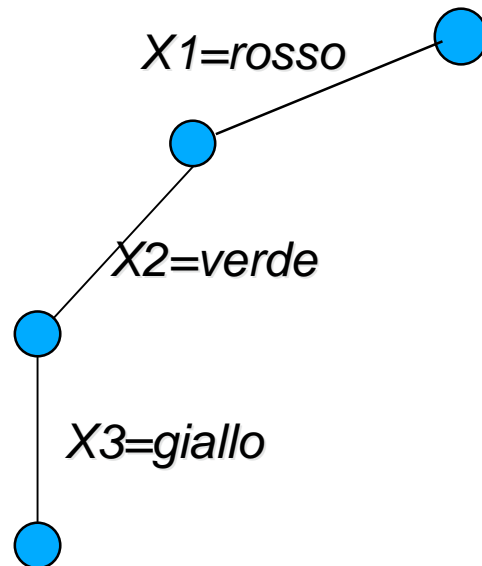


ESEMPIO

- 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)*
 $X_1 = r \wedge X_1 \neq X_2 \wedge X_2 \in \{r,g,b,y\}$
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. $p(X) :- q(f(X))$.
 - $p(a)$.
 - $p(b)$.
 - $q(f(c))$.
 - $H = \{a, f(a), f(f(a)), f(f(\dots f(a))) \dots, b, f(b), \dots, c, f(c), f(f(c))\}$

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*
- $:- p(Z, Y), \dots$
 - *unificazione del goal con la testa della clausola*
 - *vincoli risultanti: $Z = X, Y = X$, consistenti*
 - *propagazione: $Z = Y$*
- $:- \dots, Z = 3, \dots$
 - *nuovo vincolo $Z = 3$*
 - *test di consistenza ok*
 - *propagazione: $X = Y = 3$*
- $:- \dots, Y = 4.$
 - *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 $2+5 \Rightarrow X = 7$ fallisce*
 - *Soluzione parziale: predicato is*
 - *X is $2 + 5 \Rightarrow X = 7$ 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

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.*

ESEMPIO

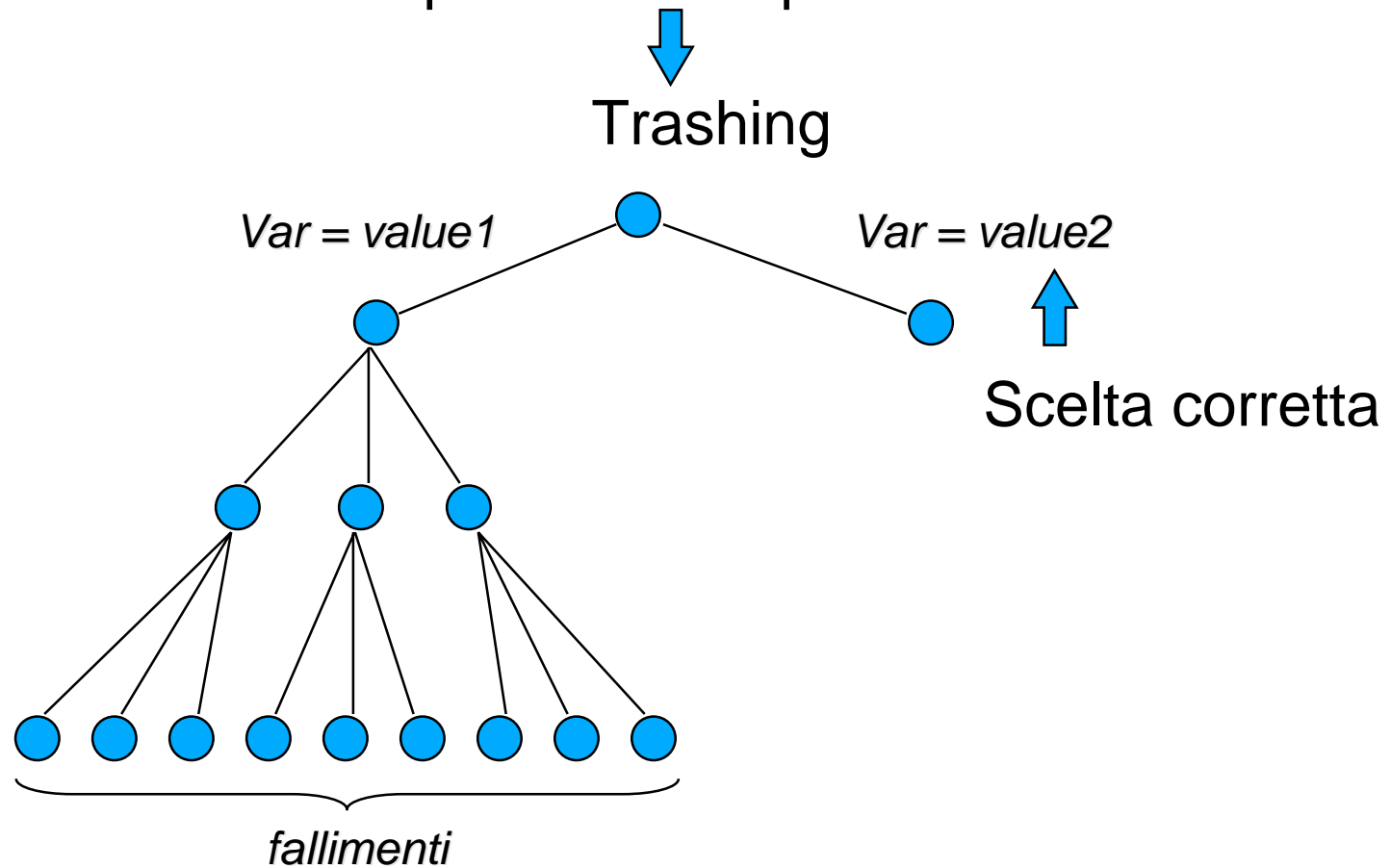
- **Esempio: Inefficienza dell'esplorazione dello spazio delle soluzioni**
 - In PL si genera una soluzione poi si esegue il test
 - $X_1 = r, X_1 \neq X_2, X_2 \in \{r, g, b, y\}$
 - $X_2 = r$ fail (backtracking)
 - $X_2 = g$ consistente (proseguo)
- **Cosa ci serve?**
 - Propagazione dei vincoli a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

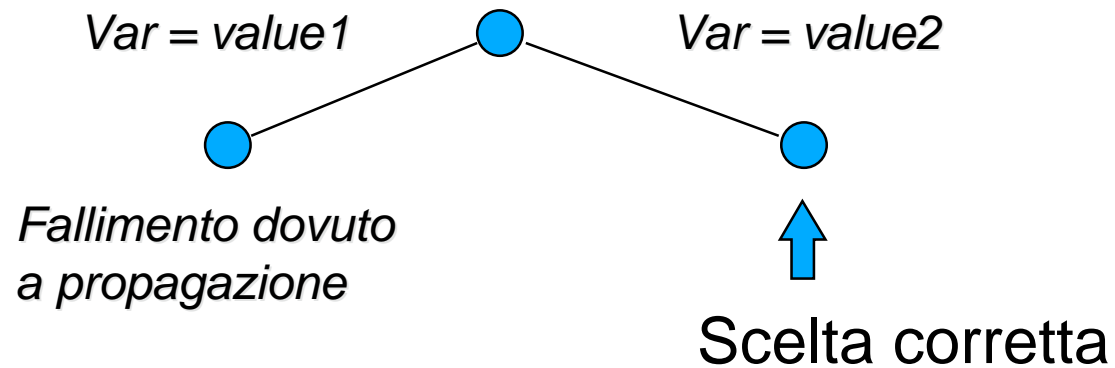
- *Enumerazione: algoritmi basati su backtracking*
 - *Assegna un valore di tentativo e controlla i vincoli*
 - *Inefficienza dovuta alle dimensioni dello spazio di ricerca*
 - *Trashing*
- *Propagazione di Vincoli*
 - *scopo: ridurre lo spazio di ricerca*
 - *la propagazione di vincoli rimuove **A PRIORI** 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



PROPAGAZIONE DI VINCOLI



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

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)

| | |
|-------------------------|--------------------|
| <code>p(X,X).</code> | DB Prolog |
| <code>?- p(Z,Y).</code> | Goal |
| <code>yes Z = Y</code> | Risposta calcolata |
 - non funziona con un predicato come >

| | |
|----------------------------------|-----------|
| <code>p(X,Y) :- X > Y.</code> | DB Prolog |
| <code>?- p(Z,Y).</code> | Goal |
| <code>fail</code> | |
- **Cosa ci serve ?**
 - Risposta: constraint store al termine della computazione

| | |
|------------------------------|--------------------|
| <code>yes if Z > Y</code> | Risposta calcolata |
|------------------------------|--------------------|

PROGRAMMAZIONE LOGICA A VINCOLI

- **Constraint Logic Programming (CLP)**
 - nata nel 1987 da studi condotti da Jaffar e Lassez
- **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 operativa) 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 $CLP(FD)$ E' IL PIU' USATO !!!
- **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 LOGICA

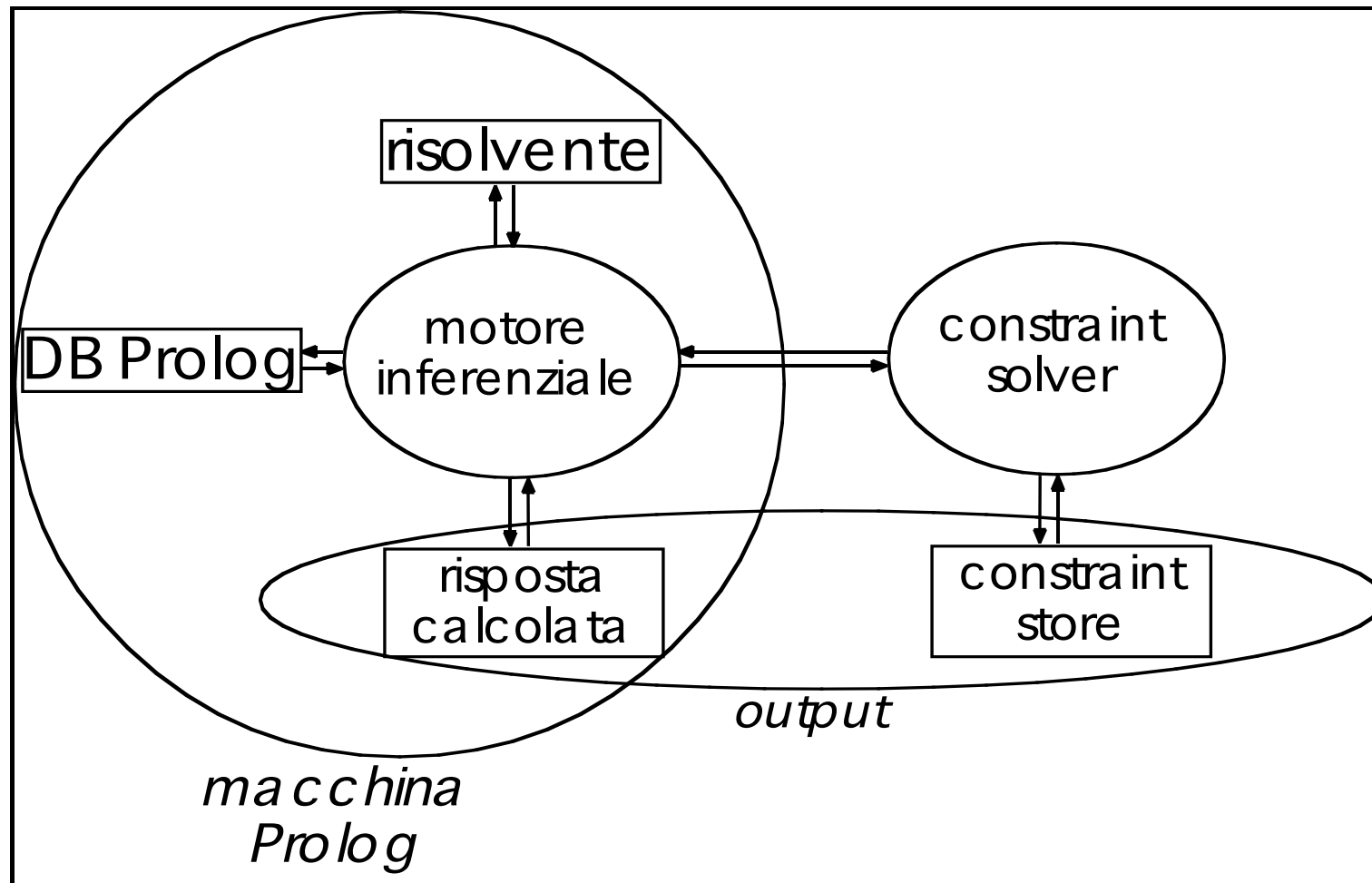
- **Eventi rilevanti**

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

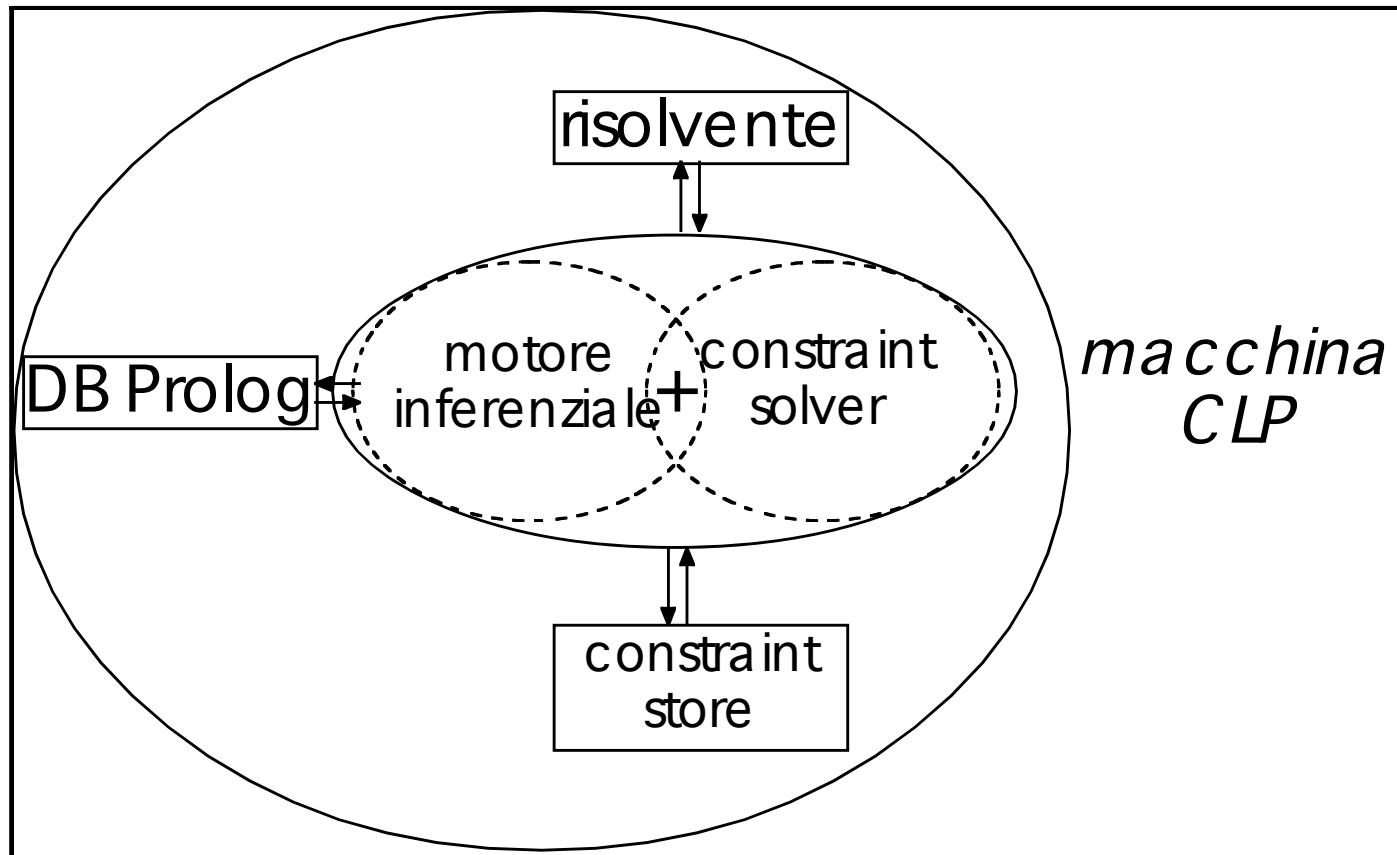
- **Estensioni**

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

ESTENDERE LA MACCHINA LOGICA



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 |
|--------------------|---|---|
| <i>risoluzione</i> | $r(A,B)$ | \emptyset |
| <i>(unif.)</i> | $p(X'), X' > Y', q(Y')$ | $A=X', B=Y'$ |
| <i>inferenza</i> | <i>(idem)</i> | $A=X', B=Y'$ |
| <i>consistenza</i> | <i>(idem)</i> | <i>(idem)</i> |
| <i>risoluzione</i> | $p(X'), X' > Y', q(Y')$ | $A=X', B=Y'$ |
| <i>(unif.)</i> | $Z'=3, X' > Y', q(Y')$ | $A=X', B=Y', \mathbf{X'=Z'}$ |
| <i>inferenza</i> | <i>(idem)</i> | $A=X'=Z', B=Y'$ |
| <i>consistenza</i> | <i>(idem)</i> | <i>(idem)</i> |
| <i>constraint</i> | $Z'=3, X' > Y', q(Y')$ $X' > Y', q(Y')$ | $A=X'=Z', B=Y'$ $A=X'=Z', B=Y', \mathbf{Z'=3}$ |
| <i>inferenza</i> | <i>(idem)</i> | $A=X'=Z'=3, B=Y'$ |
| <i>consistenza</i> | <i>(idem)</i> | <i>(idem)</i> |

MACCHINA CLP: PASSI FONDAMENTALI

| passo | risolvente | constraint store |
|--------------------|-----------------------------|--|
| <i>constraint</i> | $X' > Y', q(Y')$ $q(Y')$ | $A=X'=Z'=3, B=Y'$ $A=X'=Z'=3, B=Y', X' > Y'$ |
| <i>inferenza</i> | (idem) | $A=X'=Z'=3 > Y', B=Y'$ |
| <i>consistenza</i> | (idem) | (idem) |
| (unif.) | $K' > 5$ | $A=X'=Z'=3 > Y', B=Y', Y'=K'$ |
| <i>inferenza</i> | (idem) | $A=X'=Z'=3 > Y', B=Y'=K'$ |
| <i>consistenza</i> | (idem) | (idem) |
| <i>constraint</i> | $K' > 5$ \emptyset | $A=X'=Z'=3 > Y', B=Y'=K'$ $A=X'=Z'=3 > Y', B=Y'=K', K' > 5$ |
| <i>inferenza</i> | (idem) | $A=X'=Z'=3 > Y', B=Y'=K' > 5$ |
| <i>consistenza</i> | fallimento | (inconsistente) |

MACCHINA CLP: PASSI FONDAMENTALI

| passo | risolvente | constraint store |
|--|-------------------------|--|
| <i>risoluzione</i> (<i>unif.</i>) | $q(Y')$ $K' > 2$ | $A = X' = Z' = 3 > Y', B = Y'$ $A = X' = Z' = 3 > Y', B = Y', Y' = K'$ |
| <i>inferenza</i> | (<i>idem</i>) | $A = X' = Z' = 3 > Y', B = Y' = K'$ |
| <i>consistenza</i> | (<i>idem</i>) | (<i>idem</i>) |
| <i>constraint</i> | $K' > 2$ \emptyset | $A = X' = Z' = 3 > Y', B = Y' = K'$ $A = X' = Z' = 3 > Y', B = Y' = K', K' > 2$ |
| <i>inferenza</i> | (<i>idem</i>) | $A = X' = Z' = 3 > Y', B = Y' = K' > 2$ |
| <i>consistenza</i> successo! | (<i>idem</i>) | (<i>idem</i>) |

CLP(FD)

- **CLP(FD): Constraint Logic Programming su domini finiti**
 - particolarmente adatta a modellare e risolvere problemi a vincoli
- *Modello del problema*
 - 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

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([X1, ..., Xm])`
tutte le variabili devono avere valori differenti
 - `element(N, [X1, ..., Xm], Value)`
l'ennesimo elemento della lista deve avere valore uguale a Value
 - `cumulative([S1, ..., Sm], [D1, ..., Dn], [R1, ..., Rn], L)`
usato per vincoli di capacita'
 - vincoli disgiuntivi

ESEMPIO DI MODELLO DI UN PROBLEMA

- 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],`} **variabili & domini**
 - `V1≠V2, V1≠V3, V1≠V4, V1≠V5, V2≠V3,`
 - `V2≠V4, V2≠V5, V3≠V4, V4≠V5,`} **vincoli**
 - `.....`

Alternativamente...

```
alldifferent([V1,V2,V3,V4]),
alldifferent([V1,V2,V4,V5]).
```


MODELLO DI UN PROBLEMA CODICE ECLIPSE

```
:-lib(fd).
```

```
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,
```

```
    .....
```

```
    MANCA LA RICERCA
```

} 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.

CP(FD): PROPAGAZIONE

- La propagazione di vincoli e' quella forma di inferenza che permette di identificare e rimuovere dai domini i valori inconsistenti
- Riduzione dello spazio di ricerca in cui ogni nodo corrisponde all'istanziamento di una variabile del problema a un valore contenuto nel suo dominio.
- Per ridurre completamente il problema (eliminando tutti i valori inconsistenti) la propagazione avrebbe complessita' esponenziale.
- Allora e' necessario trovare un compromesso tra costo computazionale della propagazione e riduzione dello spazio di ricerca.

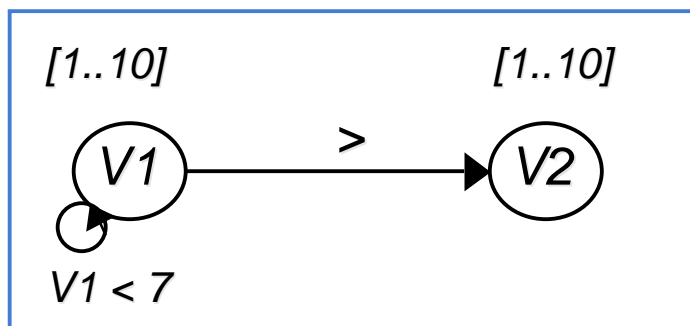
PROPRIETA' DI CONSISTENZA

- **NODE CONSISTENCY**

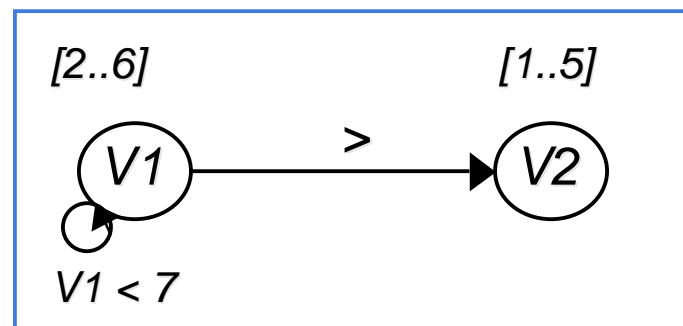
- *una rete e' node consistent se in ogni dominio di ogni nodo ogni valore e' consistente con i vincoli unari che coinvolgono la variabile*

- **ARC CONSISTENCY**

- *a rete e' arc consistent se per ogni arco (vincolo binario) che connette le variabili V_i e V_j per ogni valore nel dominio di V_i esiste un valore nel dominio di V_j 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 $\text{dom}(X)$
cerco un valore consistente in $\text{dom}(Y)$

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

PROPAGAZIONE DI VINCOLI

- **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 la propagazione

- **Esempio 2**

- $X::[1..10], Y::[5..15], X=Y$

- $X::[5..10], Y::[5..10]$ dopo la propagazione

- **Esempio 3**

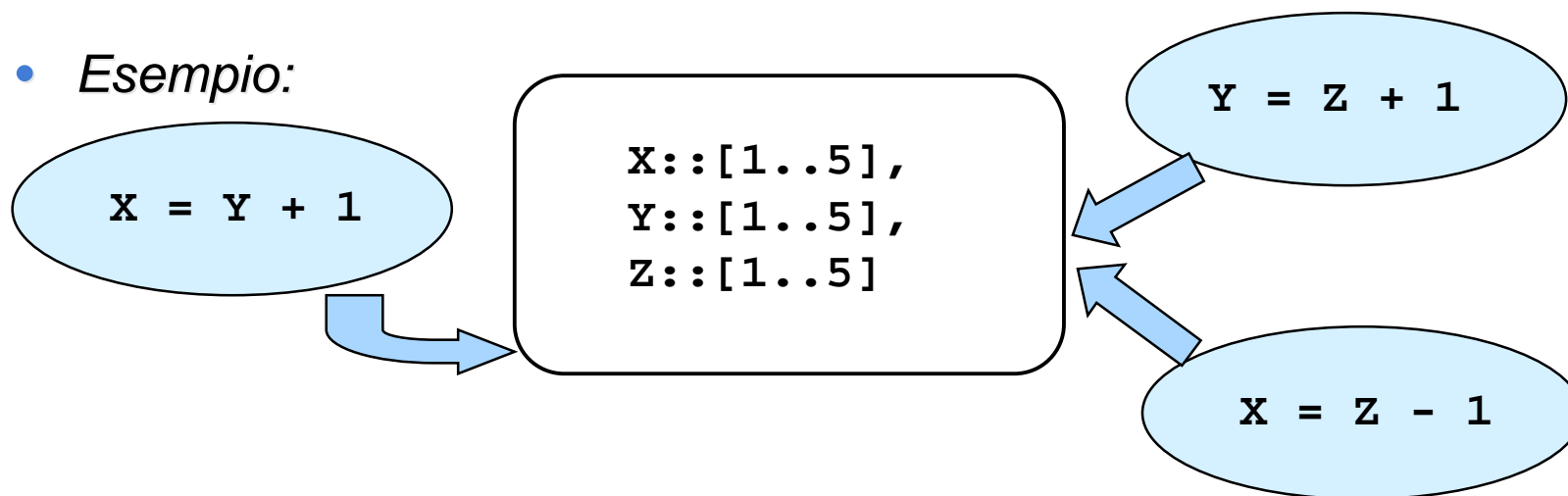
- $X::[1..10], Y::[5..15], X\neq Y$

- Nessuna propagazione

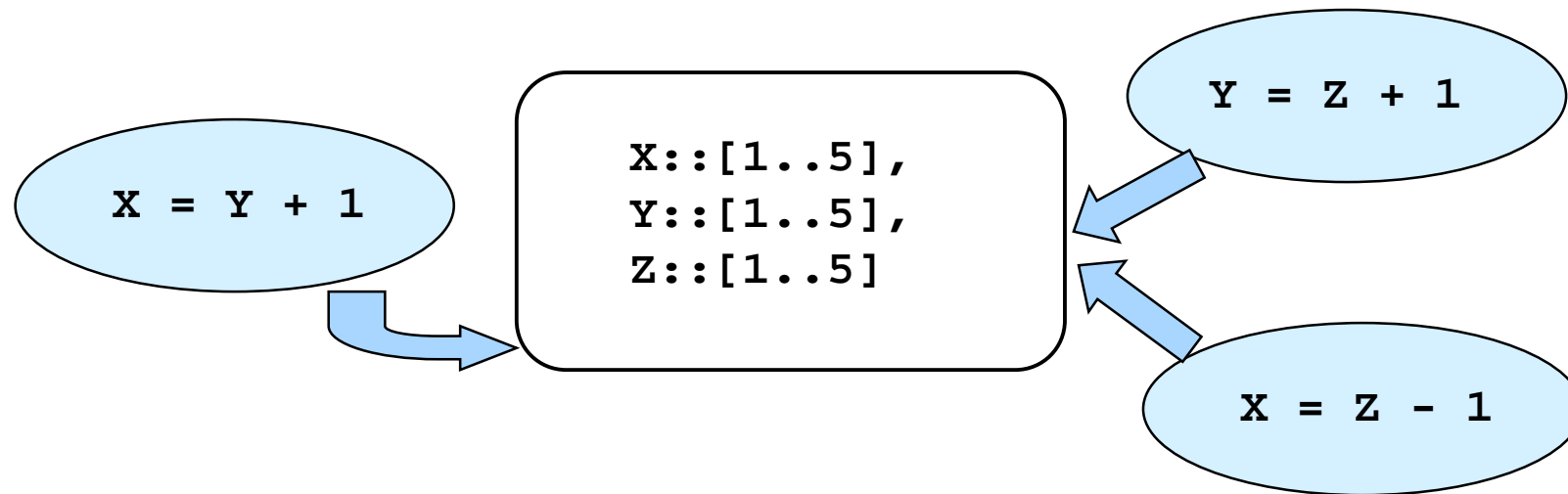
INTERAZIONE TRA VINCOLI

- In generale ogni variabile e' coinvolta in molti vincoli. Di conseguenza, ogni cambiamento nel dominio della variabile come risultato di una propagazione puo' causare la rimozione di 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)

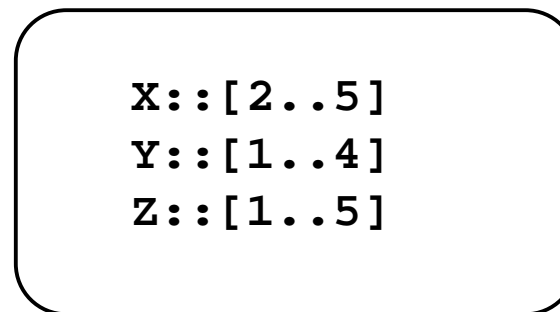
- *Esempio:*



INTERAZIONE TRA VINCOLI



- Prima propagazione di $x = y + 1$ porta a



$x = y + 1$
sospeso

INTERAZIONE TRA VINCOLI

- Seconda propagazione di $Y = Z + 1$ porta a

$X :: [2..5]$

$Y :: [2..4]$

$Z :: [1..3]$

$Y = Z + 1$

sospeso

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

$X :: [3..5]$

$Y :: [2..4]$

$Z :: [1..3]$

$X = Y + 1$

sospeso

INTERAZIONE TRA VINCOLI

- Terza propagazione di $z = x - 1$ porta a

X::[]

Y::[2..4]

Z::[1..3]

FAIL

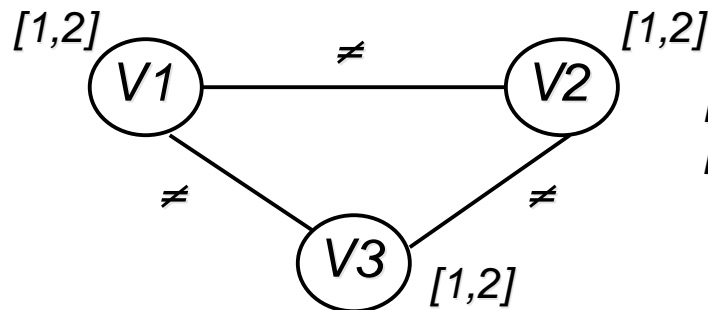
- L'ordine nel quali i vincoli sono considerati (sospesi/risvegliati) non influenza il risultato MA puo' influenzare le prestazioni computazionali

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*

PROPAGAZIONE DI VINCOLI

- Finora abbiamo visto propagazioni generali
- **Vincoli Simbolici:**
 - Ogni vincolo ha associato un algoritmo di *PROPAGAZIONE* o di *FILTERING*
 - Algoritmo di filtering implementa tecniche complesse di propagazione che derivano da studi effettuati nell'Intelligenza Artificiale e Ricerca Operativa
 - La propagazione termina quando la rete raggiunge uno stato di *quiescenza* non so possono cancellare piu' valori
 - Propagazione incrementale

VINCOLI N-ari

- 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 $c(X_1, X_2, \dots, X_n)$ è arc consistente in senso generalizzato se
 - presa una variabile X_i ($i=1..n$)
 - per ogni assegnamento delle rimanenti $n-1$ variabili
$$X_1 \rightarrow v_1, \dots, X_{i-1} \rightarrow v_{i-1}, X_{i+1} \rightarrow v_{i+1}, \dots, X_n \rightarrow v_n$$
 $\exists g \in \text{dom}(X_i) \text{ t.c. } c(v_1, \dots, v_{i-1}, g, v_{i+1}, v_n) \text{ è vero (soddisfatto).}$

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

ALL-DIFFERENT

- **Vincoli Simbolici: esempio 1**

- `alldifferent([X1, ..., Xn])`

- vero se tutte le variabili assumono valori diversi

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

`alldifferent([X1, ..., Xn])` \leftrightarrow $X_1 \neq X_2, X_1 \neq X_3, \dots, X_{n-1} \neq X_n$

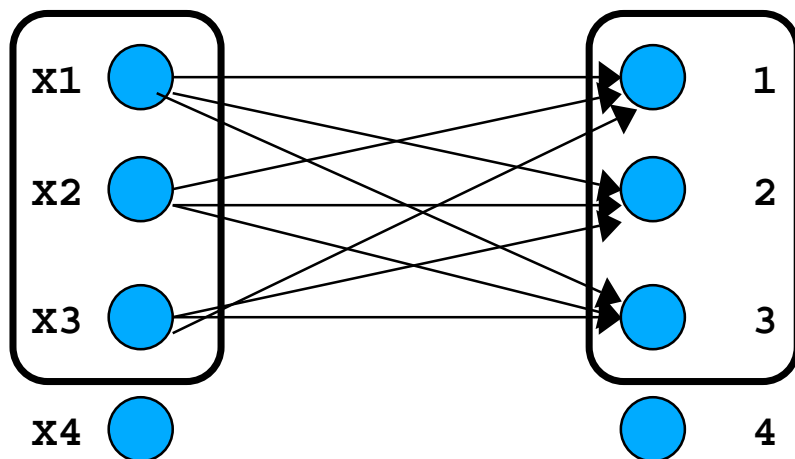
Operazionalmente permette una propagazione piu' forte.

- `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

$x1 :: [1, 2, 3], x2 :: [1, 2, 3], x3 :: [1, 2, 3], x4 :: [1, 2, 3, 4],$
 $alldifferent([x1, x2, x3, x4])$



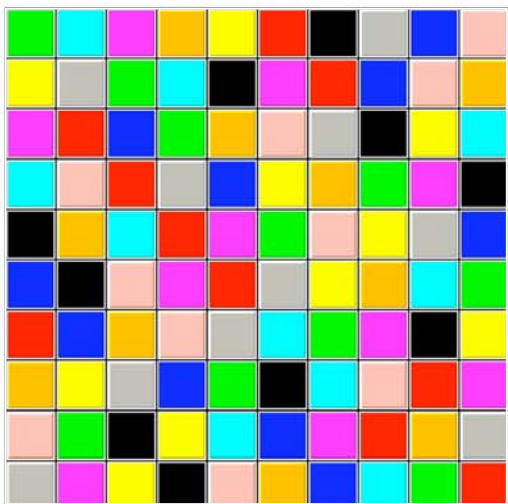
Insieme di variabili di cardinalita' 3 che hanno medesimo dominio i cardinalita' 3



$x4 :: [\cancel{1}, \cancel{2}, \cancel{3}, 4]$

ALL-DIFFERENT

- **Vincoli Simbolici:** L'alldifferent si usa in tantissime applicazioni
- **Esempio: Partial Latin Square**

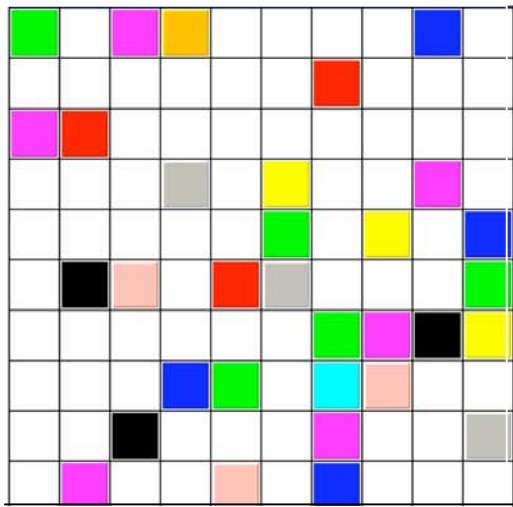


*Colorare ogni riga e colonna
con 8 colori in modo che su ogni
riga e colonna ci siano tutti colori
diversi*

*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 ($[X_{i1}, X_{i2}, \dots, X_{in}]$)

per ogni colonna $j=1..n$

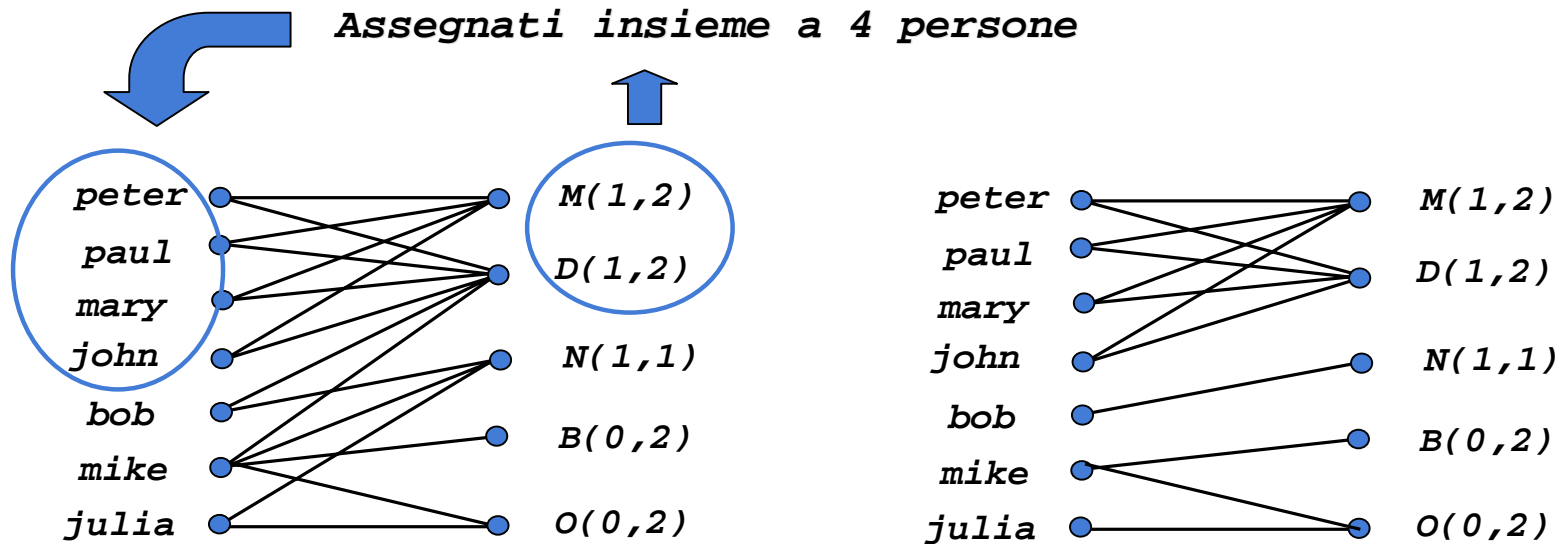
alldifferent ($[X_{1j}, X_{2j}, \dots, X_{nj}]$)

SI VEDA

<http://www.cs.cornell.edu/gomes>

GLOBAL CARDINALITY CONSTRAINT

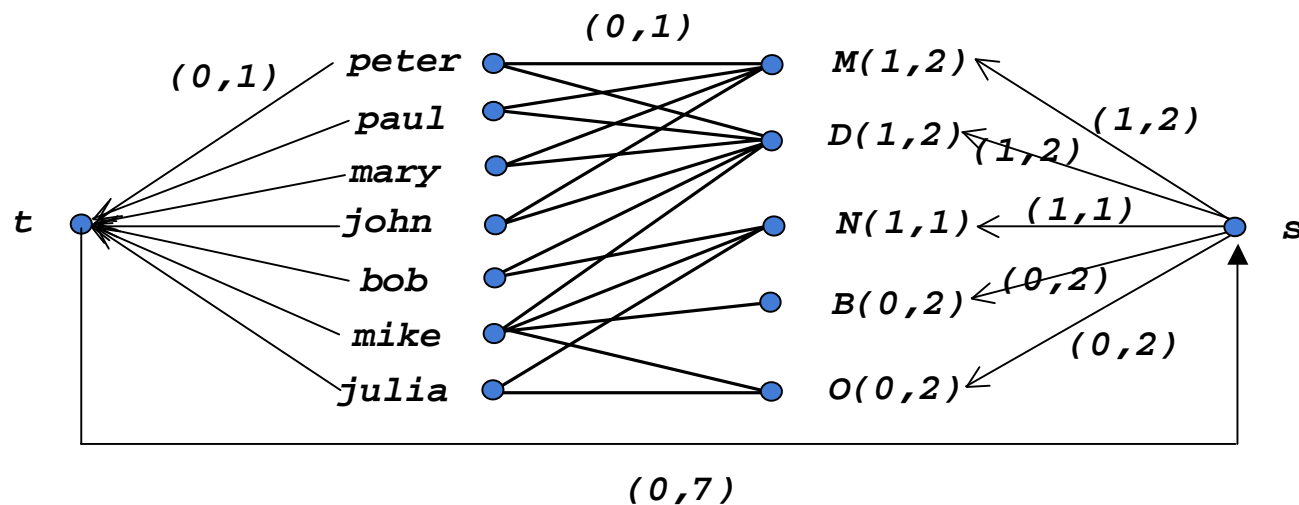
- $gcc(Var, Val, LB, UB)$ [Regin AAA/96] var sono variabili, val valori LB e UB sono il minimo e massimo numero di occorrenza per ogni valore in val assegnato a var
- Esempio:



GLOBAL CARDINALITY CONSTRAINT

- La nozione di consistenza si basa su un algoritmo si *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 docente. Abbiamo gli istanti di inizio delle lezioni: $L1start$ e $L2start$ e la loro durata $Duration1$ e $Duration2$.
 - Le due lezioni non possono sovrapporsi:
$$L1start + Duration1 \leq L2start$$

OR

$$L2start + Duration2 \leq L1start$$
 - 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**($[S_1, \dots, S_n]$, $[D_1, \dots, D_n]$, $[R_1, \dots, R_n]$, L)

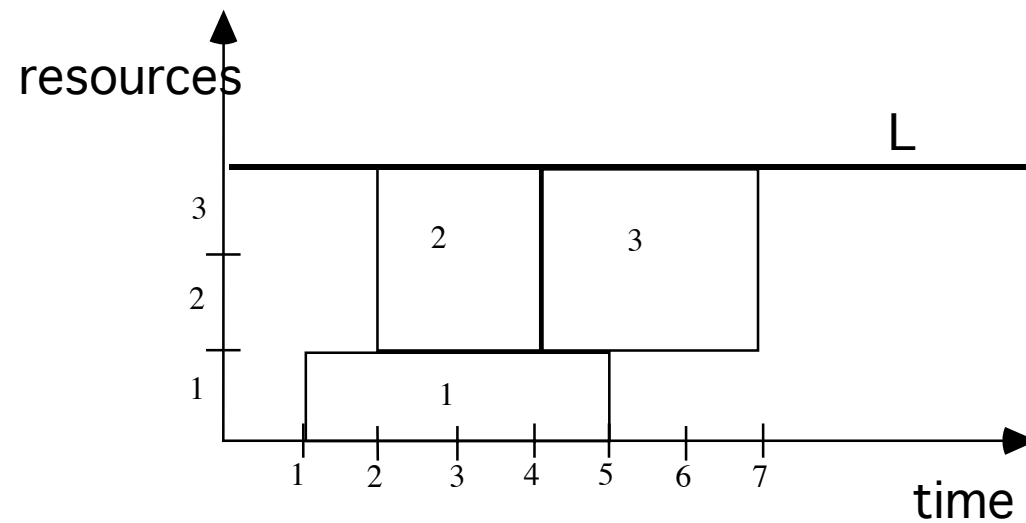
- S_1, \dots, S_n sono istanti di inizio di attivita' (variabili con dominio)
- D_1, \dots, D_n sono durate (variabili con dominio)
- R_1, \dots, R_n sono richieste di risorse (variabili con dominio)
- L limite di capacita' delle risorse (fisso o variabile nel tempo)

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

$$\max \left\{ \sum_{j | S_j \leq i \leq S_j + D_j} R_i \right\} \leq L$$

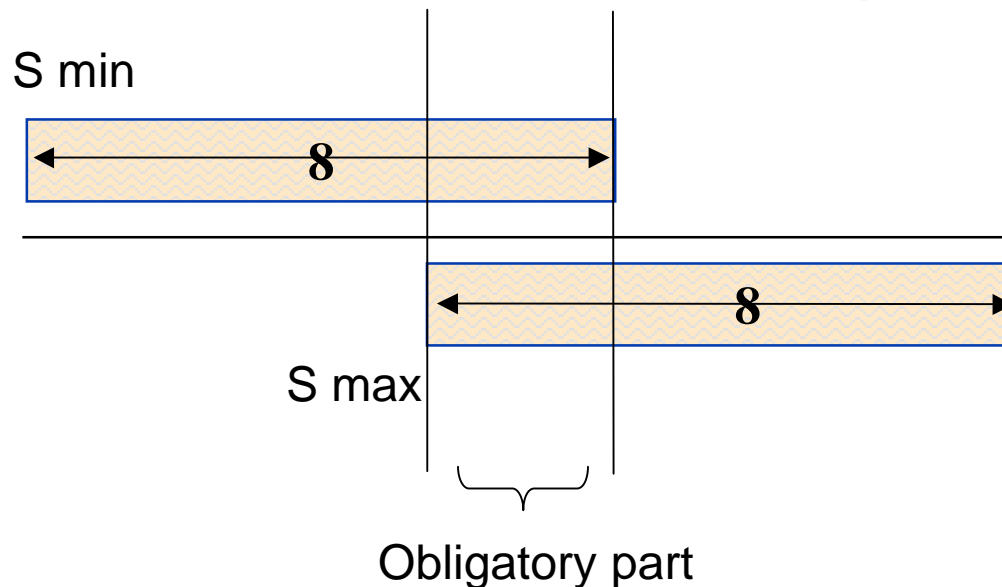
PROPAGAZIONE DI VINCOLI

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



PROPAGAZIONE DI VINCOLI

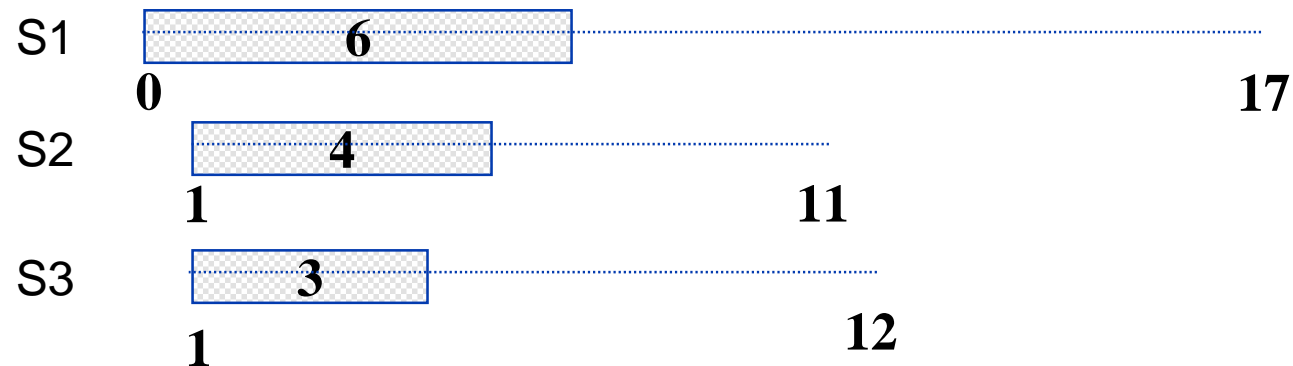
- Vincoli Simbolici:
- un esempio di propagazione usato nei vincoli sulle risorse e' quello basato sulle *parti obbligatorie*



PROPAGAZIONE DI VINCOLI

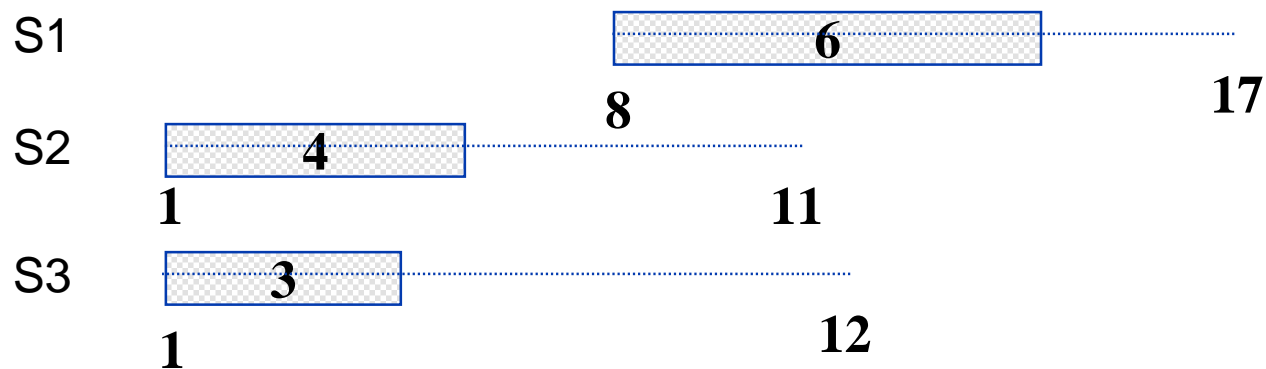
- **Vincoli Simbolici:**
 - un'altra propagazione usata nel vincolo di capacità e' quella basata sul *edge finding* [**Baptiste, Le Pape, Nuijten, IJCAI95**]

Consideriamo una risorsa unaria e tre attività'



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).

PROPAGAZIONE DI VINCOLI

- Vincoli Simbolici:

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 e' e, la somma delle durate e' D e il massimo istante di terminazione e' C . Se

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

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

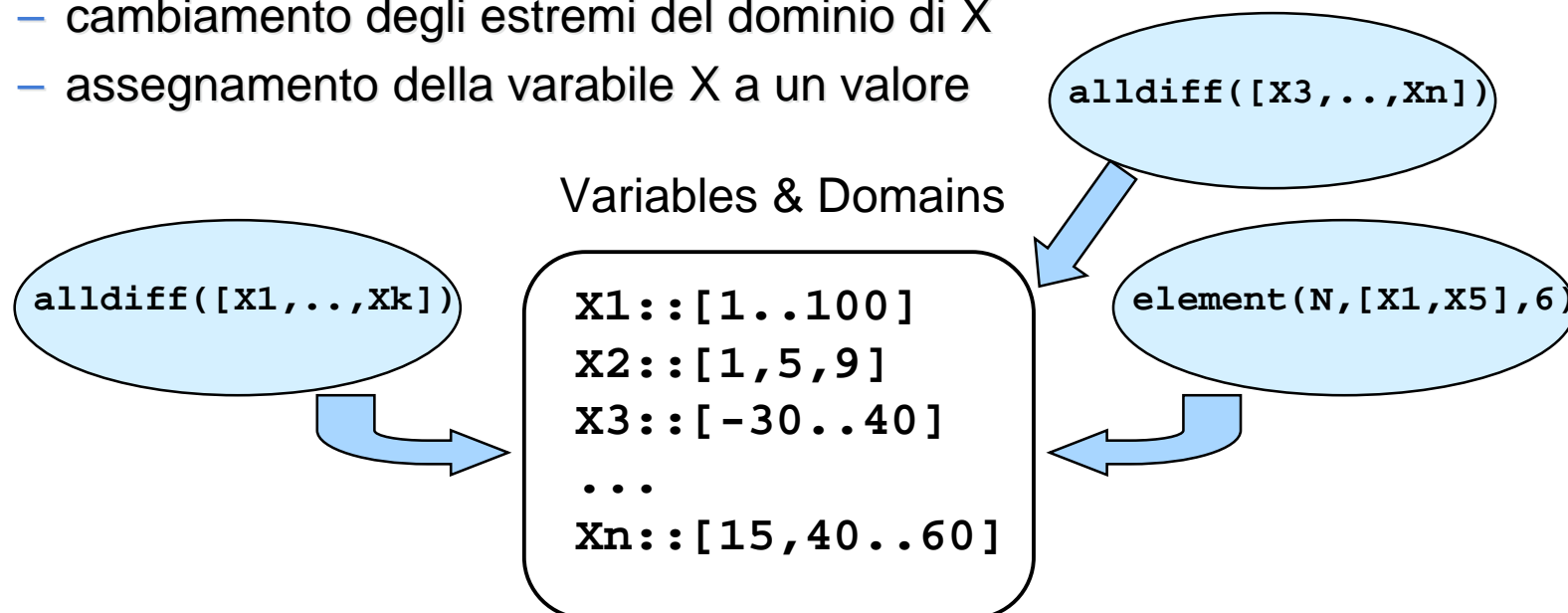
$$\max_{(S' \subseteq S)} \{e(S') + D(S')\}.$$

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 indipendenti (rilassamenti del problema originale)

INTERAZIONE TRA VINCOLI

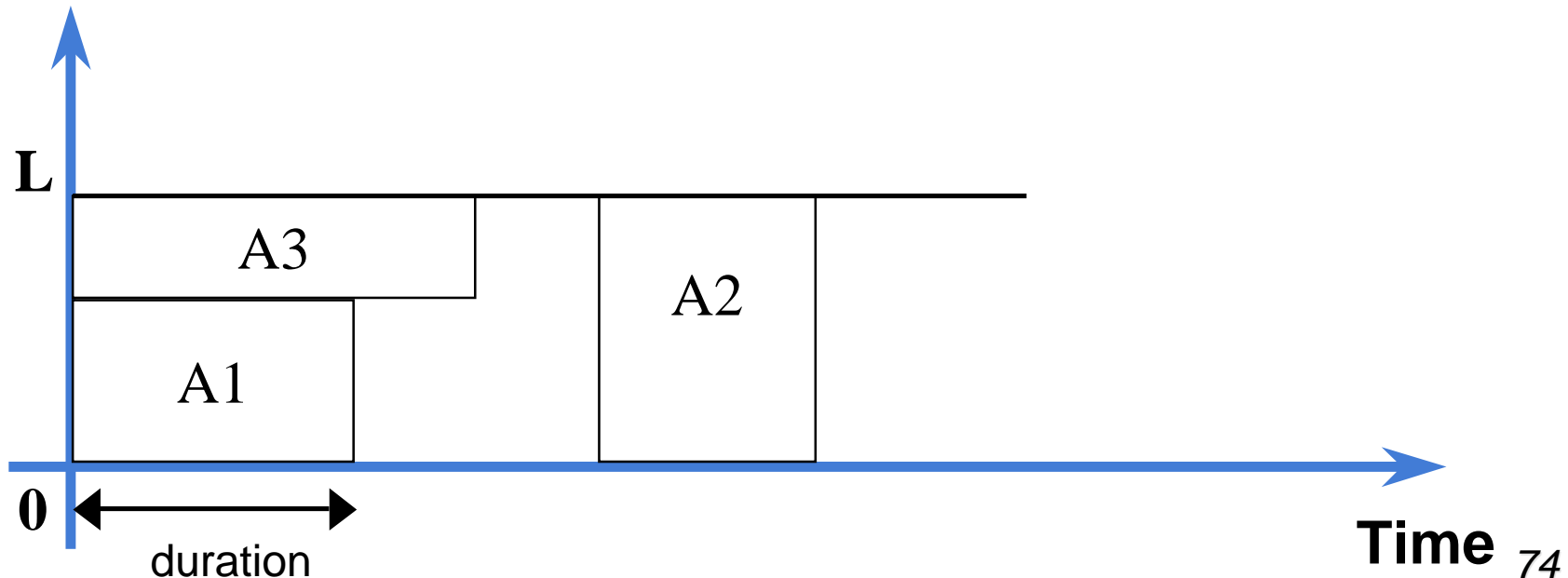
- I vincoli interagiscono attraverso variabili condivise nel constraint store
- La propagazione di un vincolo attivata quando sulla variabile X coinvolta nel vincolo si scatena un **evento**
 - cambiamento nel dominio di X
 - cambiamento degli estremi del dominio di X
 - assegnamento della variabile X a un valore



VINCOLI COME COMPONENTI SOFTWARE

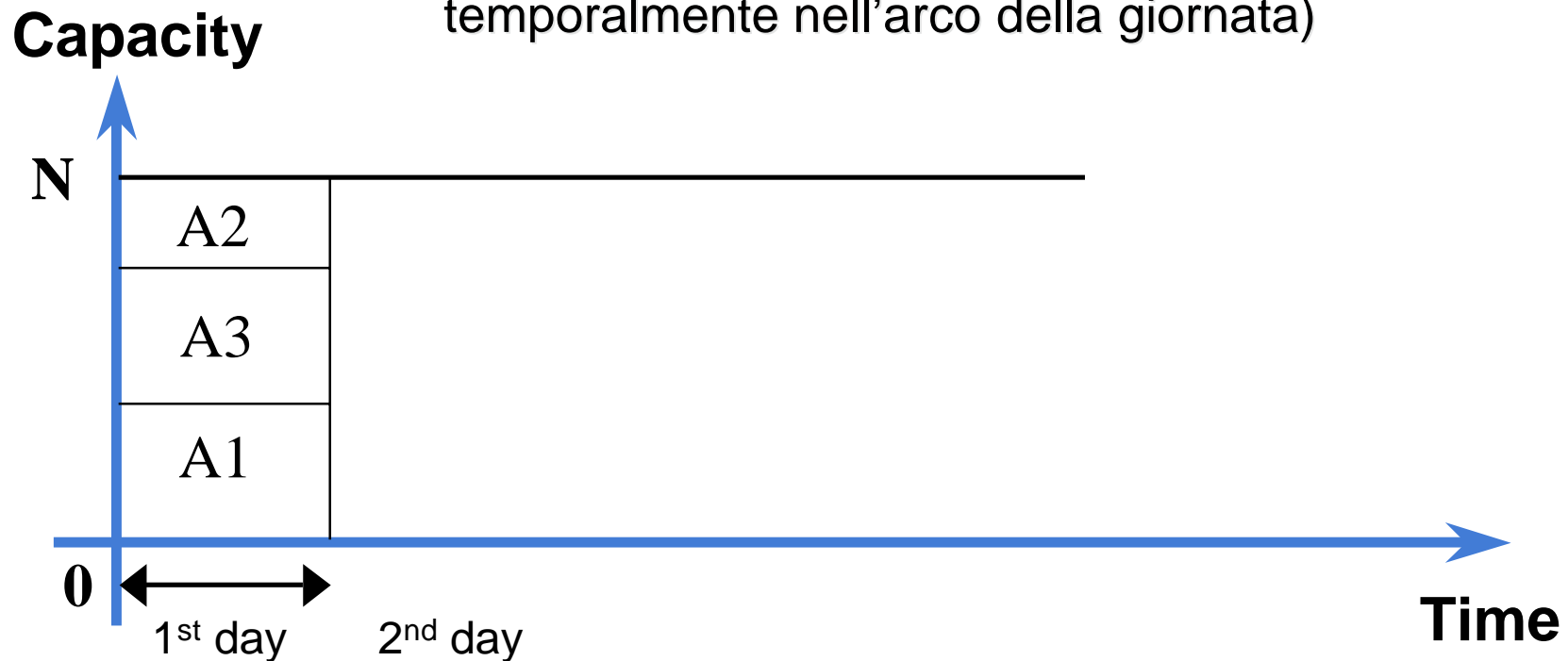
- I vincoli simbolici possono essere visti come componenti software utilizzabili in diverse situazioni. Ad esempio il vincolo cumulative puo' essere usato in vari modi
 - Scheduling (1): Attivita' A1, A2, A3 condividono la stessa risorsa con capacita' limitata. Durate sull'asse X e uso delle risorse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

Capacity



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 attività sono collocate temporalmente nell'arco della giornata)

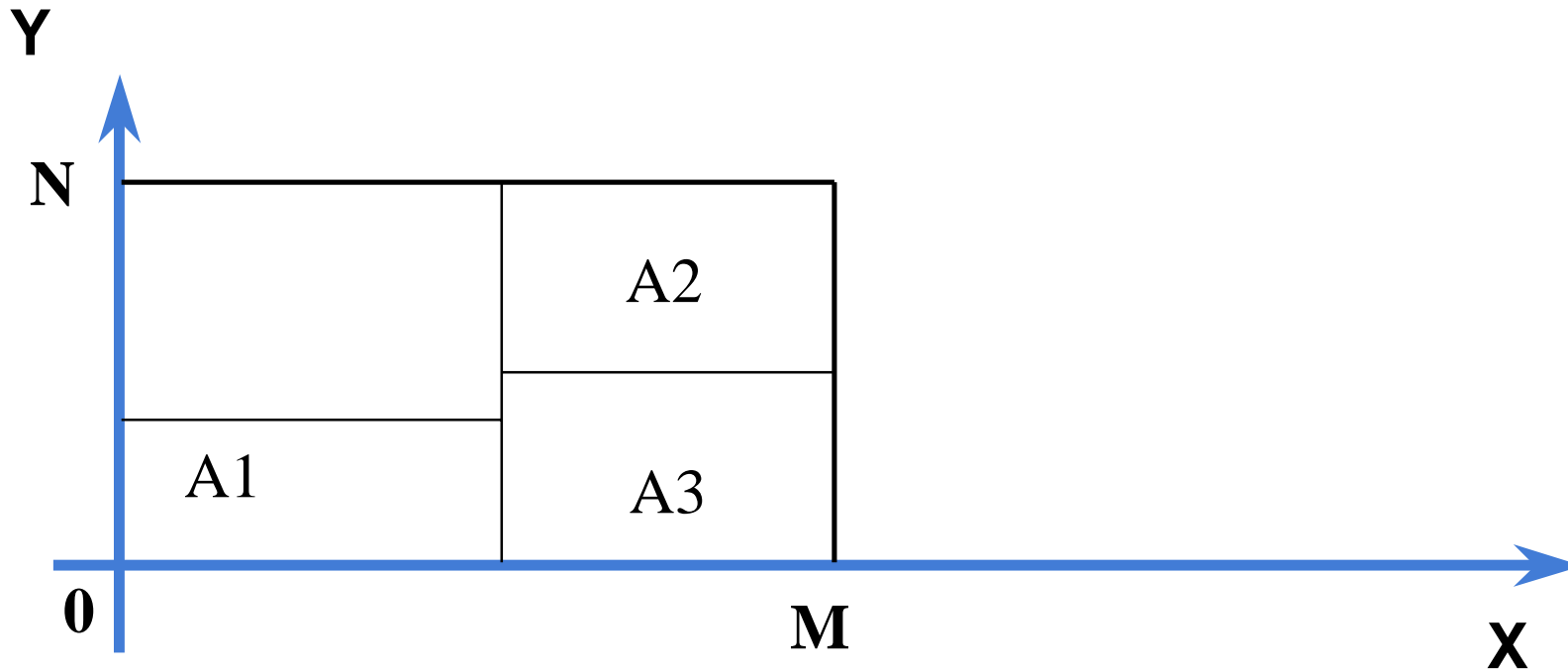


VINCOLI GLOBALI

- 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 COME COMPONENTI SOFTWARE

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

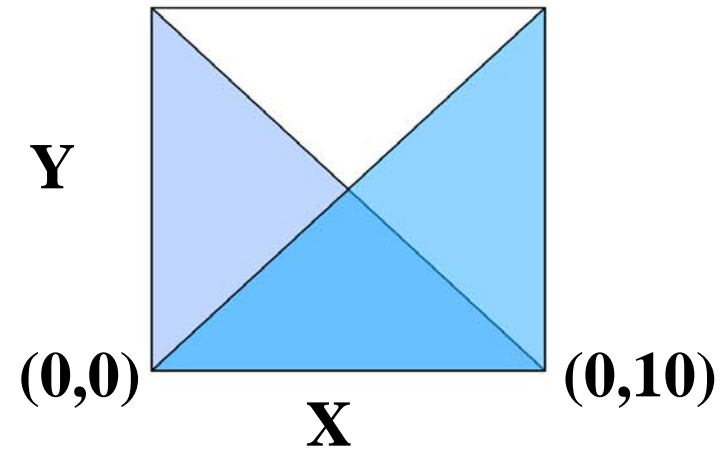


VINCOLI RIDONDANTI

- 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 (tradeoff).
- Un vincolo ridondante C e' un vincolo che e' implicato da altri vincoli $\{C_1 \dots C_k\}$, ma il risolutore non identifica questa implicazione a causa della sua incompletezza

VINCOLI LOGICAMENTE RIDONDANTI

$$[X, Y] :: 0..10, \quad X \geq Y,$$
$$Y \leq 10 - X$$



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

VINCOLI RIDONDANTI

- Esempio: Sequenza magica
- Dato un insieme di $n+1$ variabili X_0, \dots, 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.

- `magic_sequence([X0, ..., Xn]) :-`
 `X0, ..., Xn :: [0..n],`
 `occurrences(0, [X0, ..., Xn], X0),`
 `occurrences(1, [X0, ..., Xn], X1),`
 `...,`
 `occurrences(n, [X0, ..., Xn], Xn),`
 `labeling([X0, ..., Xn]).`

Esempi:

Con 4 variabili X₀, ..., X₄

soluzione 1,2,1,0

Con 7 variabili x₀, ..., x₇

soluzione 3,2,1,1,0,0,0

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 :

- $X_1 + 2 * X_2 + \dots + N * X_n = N + 1$

- `magic_sequence ([X0, ..., Xn]) :-`

`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,`


`labeling ([X0, ..., Xn]).`

Con P4 2Ghz, N=23

• **senza vincolo ridondante**




 **5.88s**

• **con vincolo ridondante**

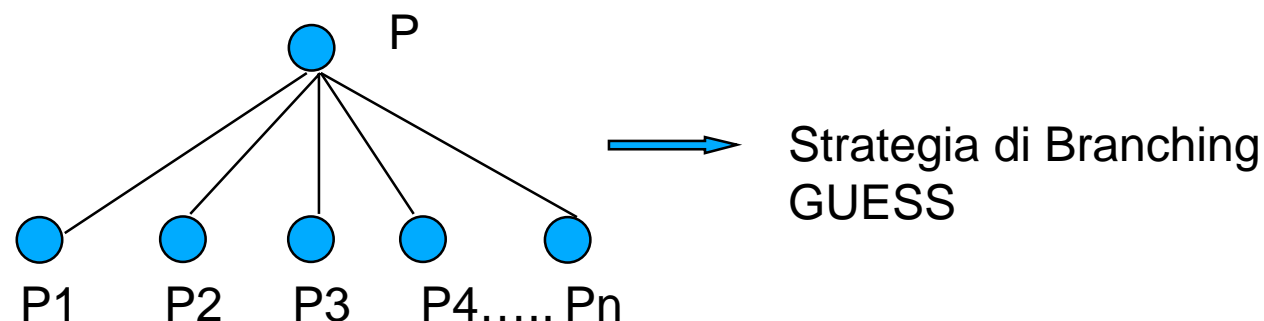
 **0.55s**

• **CODICE GENERALE SU WEB**

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 $P1, P2, \dots, Pn$.
- 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.

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.

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.

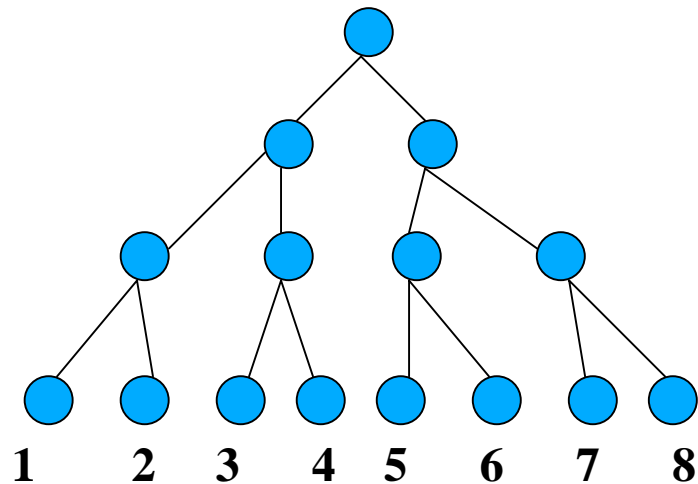
STRATEGIA CRONOLOGICA PER SCHEDULING

- 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

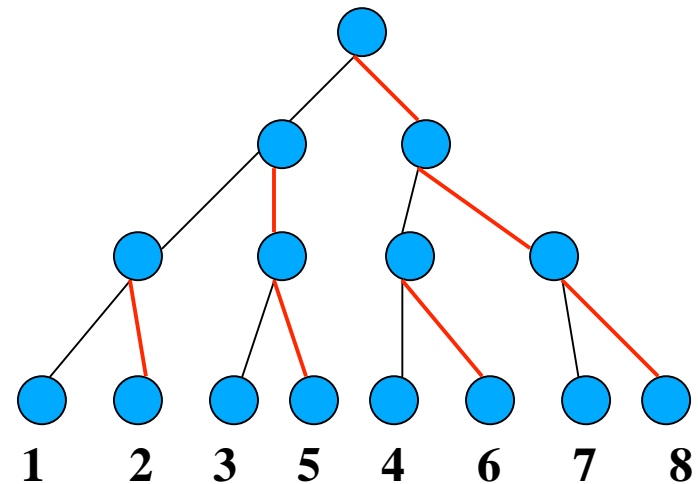
VARIANTI ALLA DEPTH FIRST

- 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



ESEMPIO COMPLETO

```
:-lib(fd).
```

```
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

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

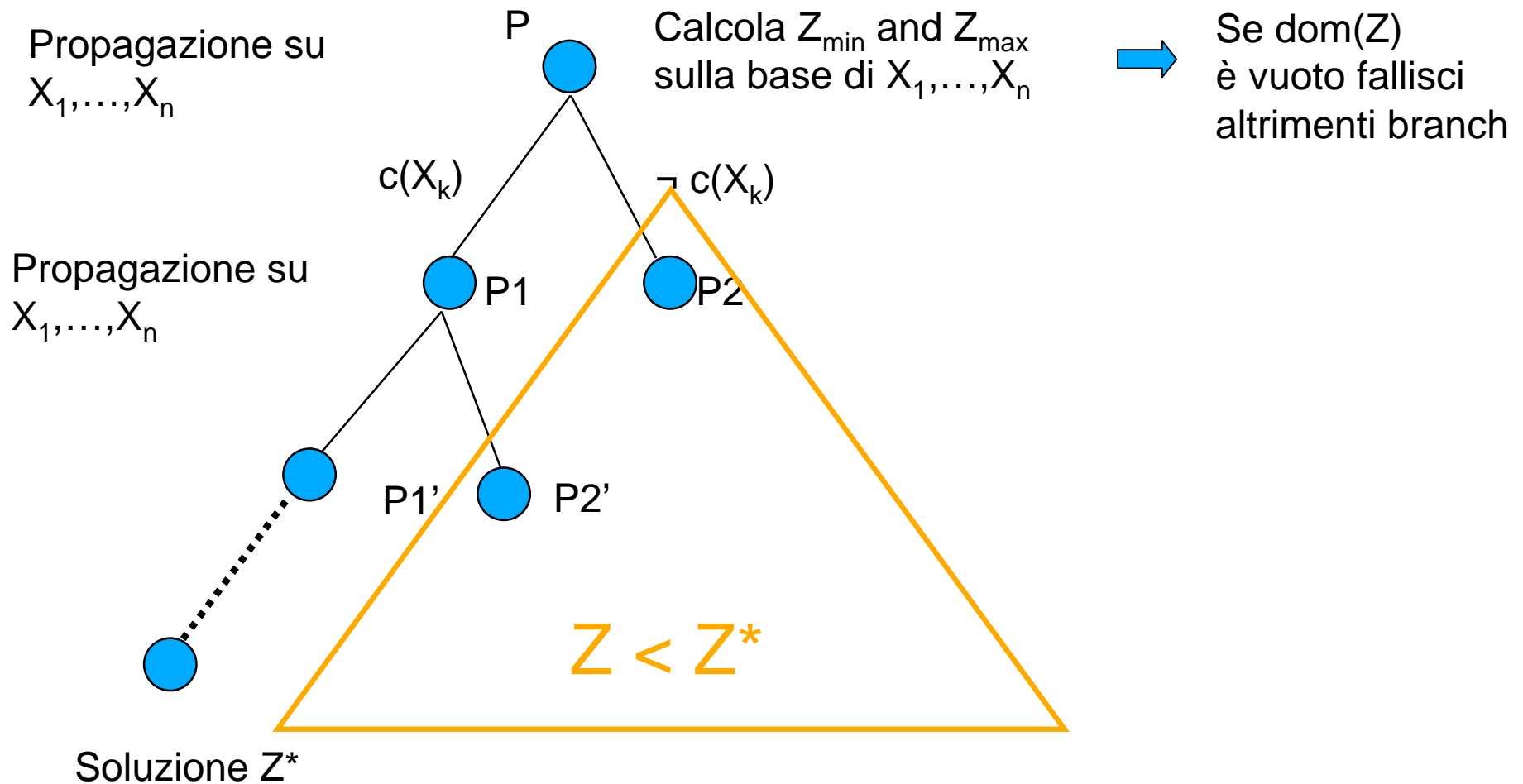
- Il vincolo aggiunto fa normalmente poca propagazione.
- Si usa una variabile Z che rappresenta la funzione obiettivo e la si lega alle variabili decisionali, esempio somma di costi
- Se il legame tra Z e tali variabili e' forte, la propagazione elimina molti rami dall'albero, altrimenti, cercare una 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

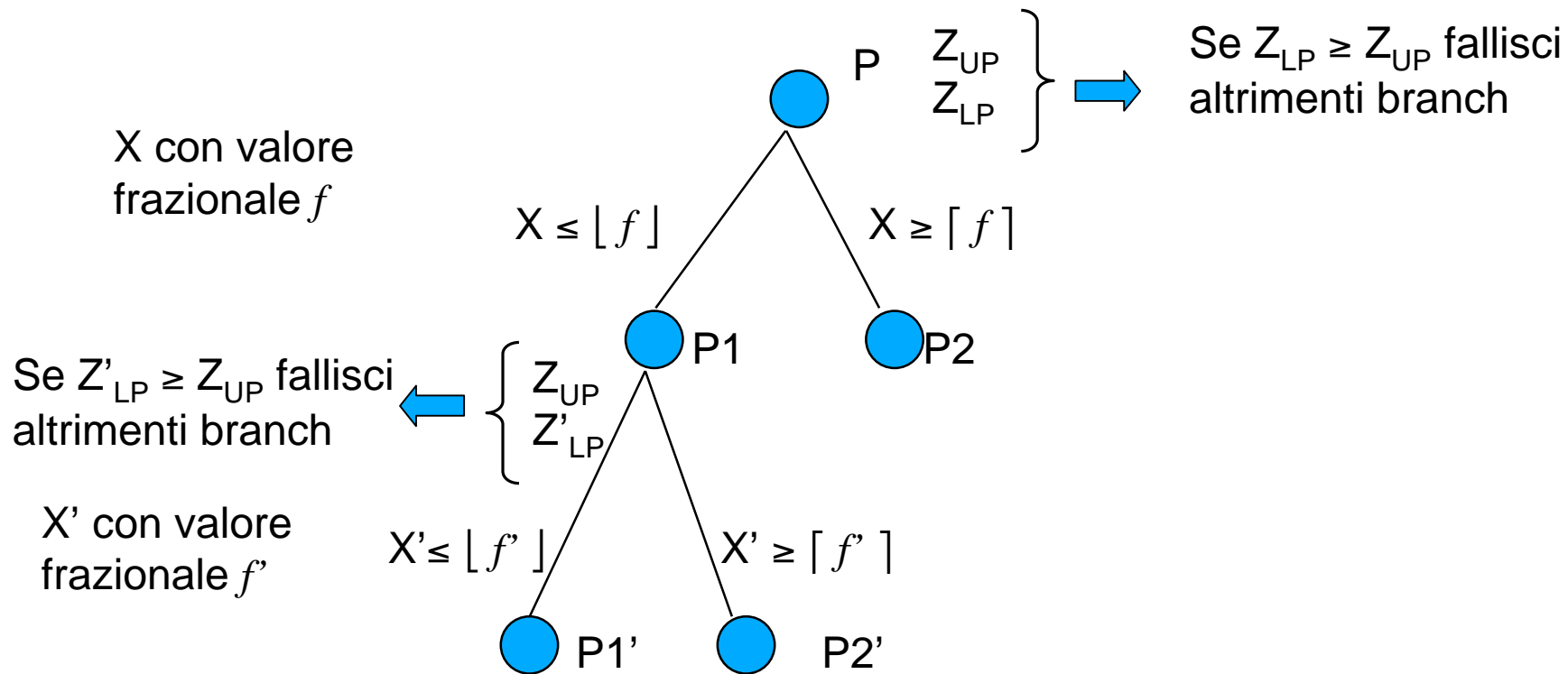
BRANCH & BOUND

in PROGRAMMAZIONE A VINCOLI



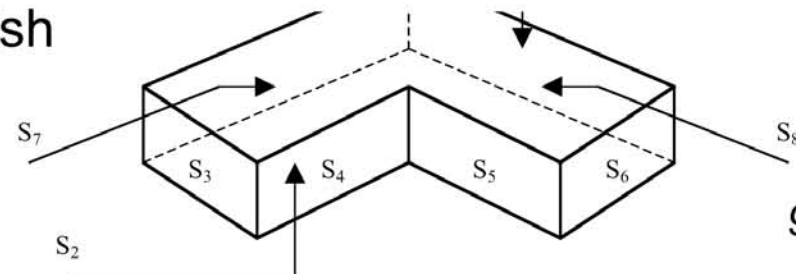
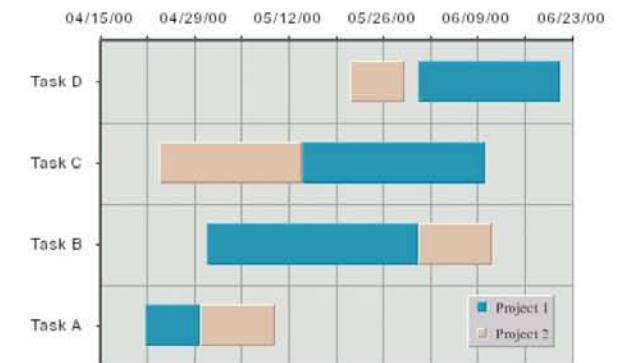
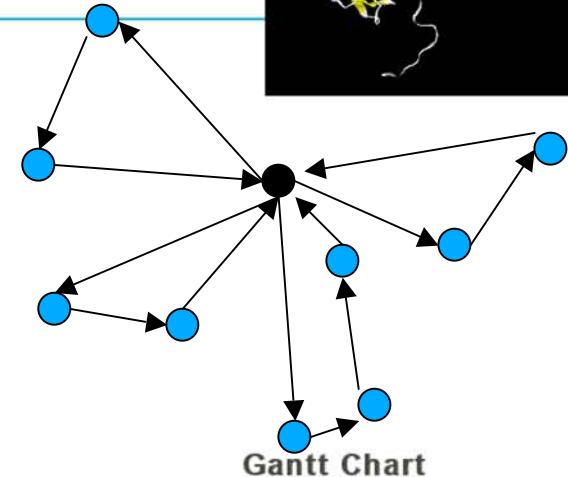
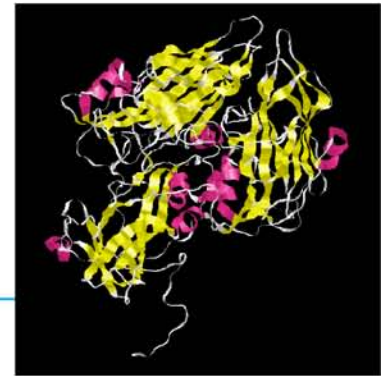
BRANCH & BOUND

in PROGRAMMAZIONE LINEARE INTERA



APPLICAZIONI RECENTI

- Assegnazione di frequenze nei cellulari
 - 2 cellulari nella stessa cellula non possono avere stessa frequenza → stesso modello del graph coloring!
- Pianificazione di orario, scheduling
- Routing: cammino minimo, traveling salesman, ...
- Problemi di packing
- Bioinformatica
 - calcolo dello sviluppo spaziale di proteine
- Object recognition
- Ottimizzazione embedded system design
- Tecnologia usata da Airfrance, British Airways, Cisco Systems, ...



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'è 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 (PACLIP)*
 - *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://slash.math.unipd.it/acp/>
- **Tools**
 - *SICStus prolog based*
 - *Eclipse prolog based: <http://www.icparc.ic.ac.uk/eclipse/>*
 - *Choco java based open source <http://choco.sourceforge.net>*