APPLICAZIONI DI CP

- ***Outline:***
  - *Riassunto concetti principali su Constraint Programming*
    - *Modello – propagazione – search*
  - *Applicazioni*
    - *Scheduling*
    - *Routing*
    - *Riconoscimento oggetti*
  - *Linee di Ricerca aperte a Bologna*
    - *Allocazione e scheduling per MPSoCs*
    - *Dynamic Voltage scaling*

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([`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

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

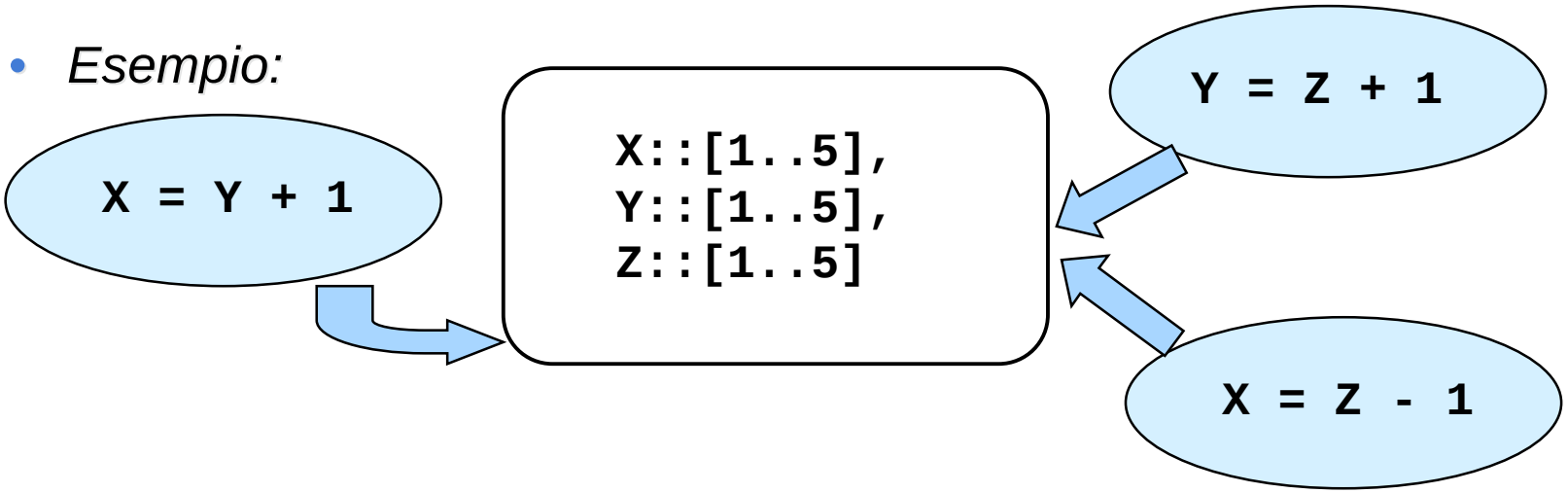
      `X::[6..10], Y::[5..9]` dopo la propagazione
  - Esempio 2
    - `X::[1..10], Y::[5..15], X=Y`
    - `X::[5..10], Y::[5..10]` dopo la propagazione
  - Esempio 3
    - `X::[1..10], Y::[5..15], X≠Y`
    - Nessuna propagazione

INTERAZIONE TRA VINCOLI

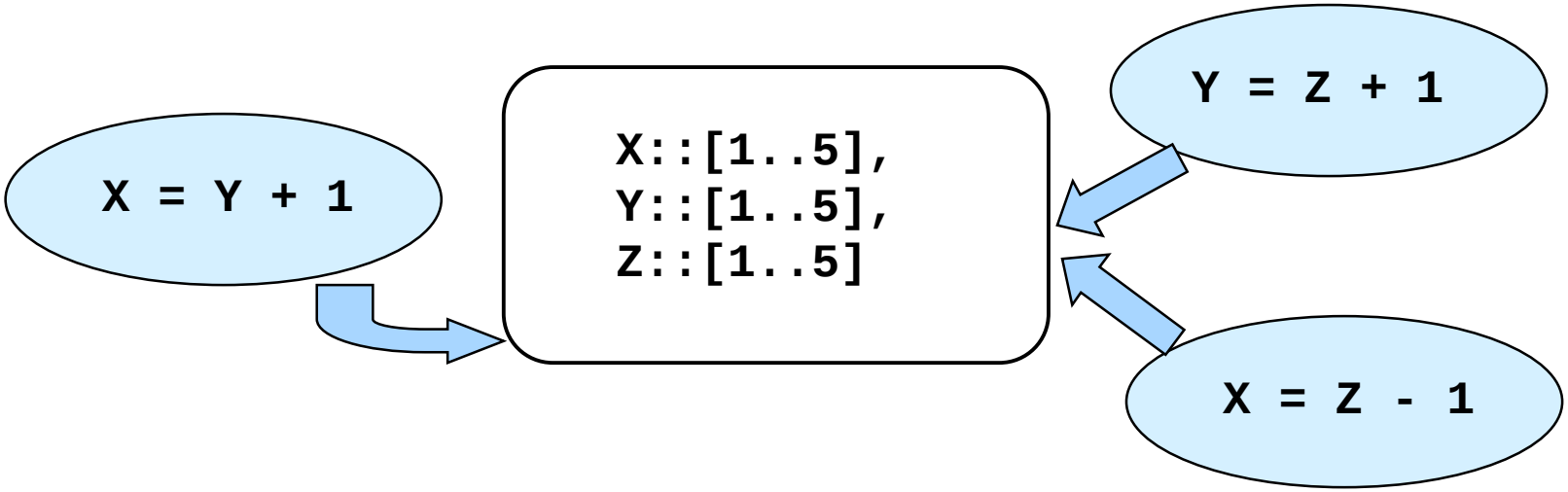
- In generale ogni variabile e' coinvolta in molti vincoli. Di conseguenza, ogni cambiamento nel dominio della variabile come risultato di una propagazione puo' causare la rimozione di altri valori dai domini delle altre variabili.

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

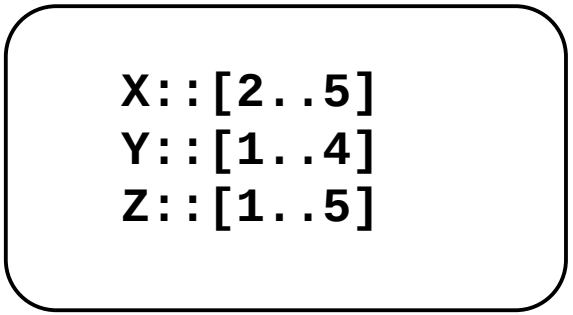
- *Esempio:*



INTERAZIONE TRA VINCOLI



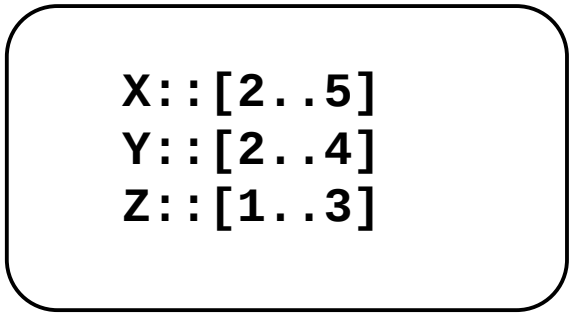
- Prima propagazione di ***X = Y + 1*** porta a



X = Y + 1
sospeso

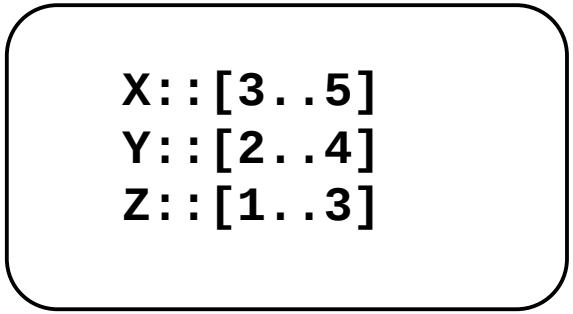
INTERAZIONE TRA VINCOLI

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



Y = Z + 1
suspeso

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



X = Y + 1
sospeso

INTERAZIONE TRA VINCOLI

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

```
X::[]
Y::[2..4]
Z::[1..3]
```

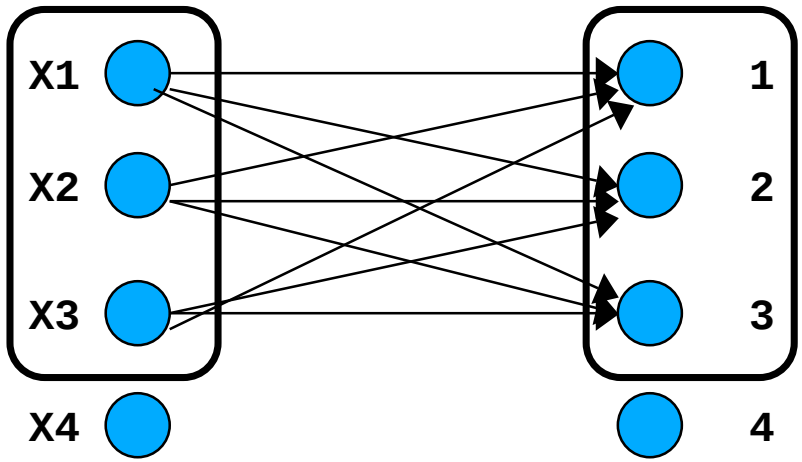
FAIL

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

PROPAGAZIONE DI VINCOLI

- 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]
```

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*

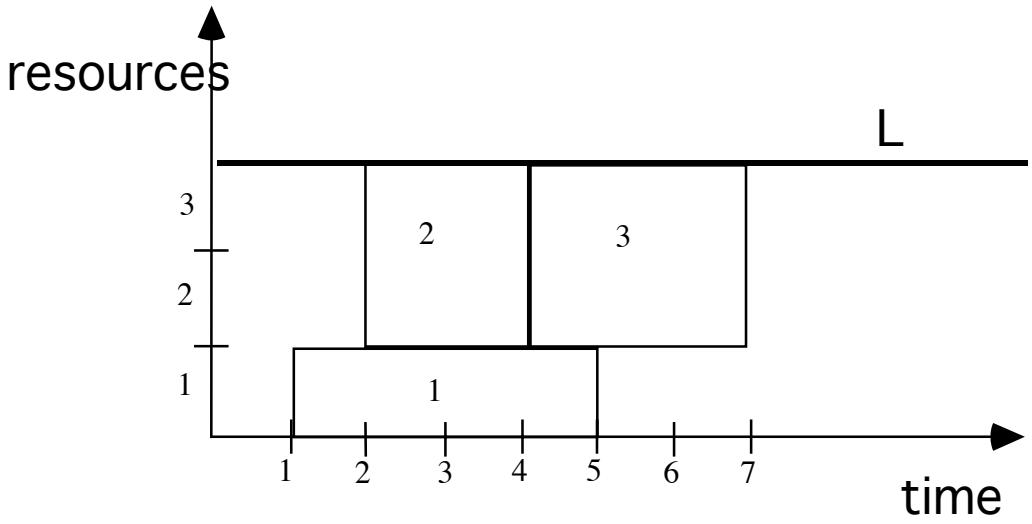
PROPAGAZIONE DI VINCOLI

- 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)
- Il vincolo cumulative assicura che

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

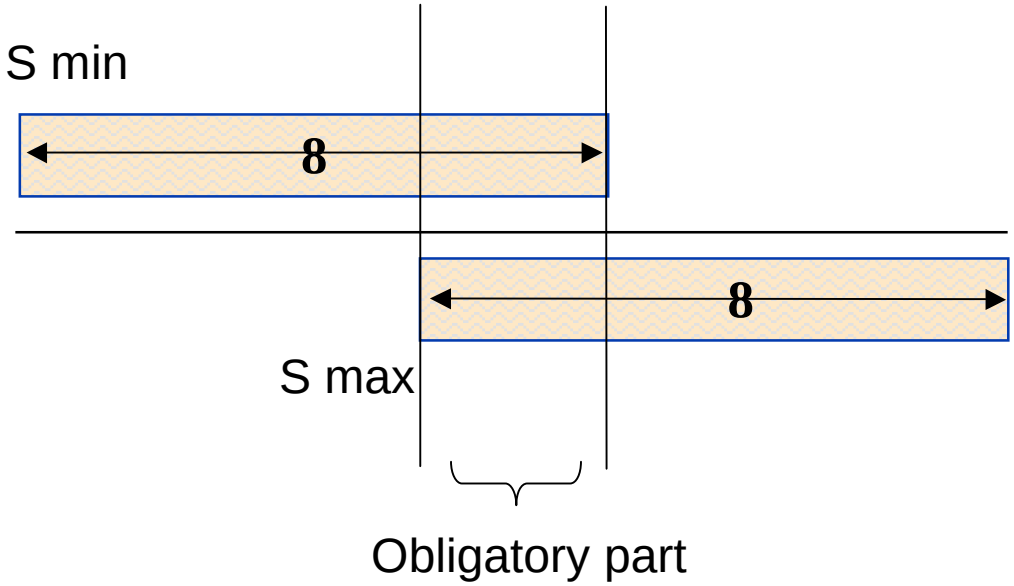
PROPAGAZIONE DI VINCOLI

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



PROPAGAZIONE DI VINCOLI

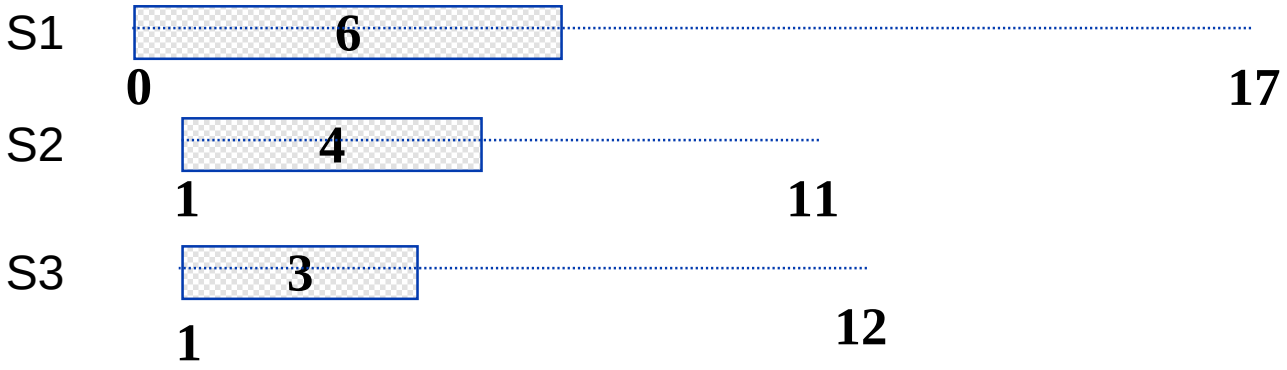
- Vincoli Simbolici:
- un esempio di propagazione usato nei vincoli sulle risorse e' quello basato sulle *parti obbligatorie*



PROPAGAZIONE DI VINCOLI

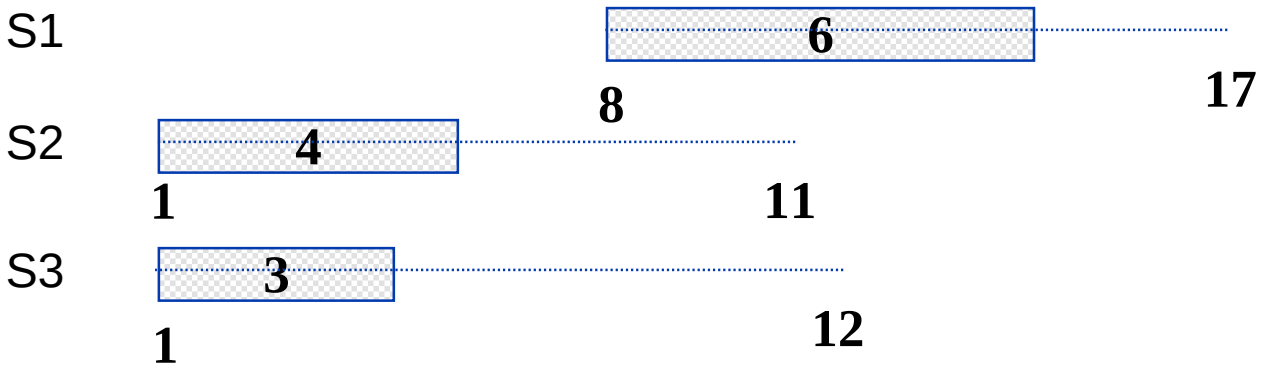
- Vincoli Simbolici:
  - un'altra propagazione usata nel vincolo di capacita' e' quella basata sul *edge finding* ***[Baptiste, Le Pape, Nuijten, IJCAI95]***

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).

PROPAGAZIONE DI VINCOLI

- Vincoli Simbolici:

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

Sia o un'attivita' e S un insieme di attivita' 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 e' possibile che o preceda alcuna operazione in S. Questo implica che il minimo istante iniziale per o puo' essere fissato a

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

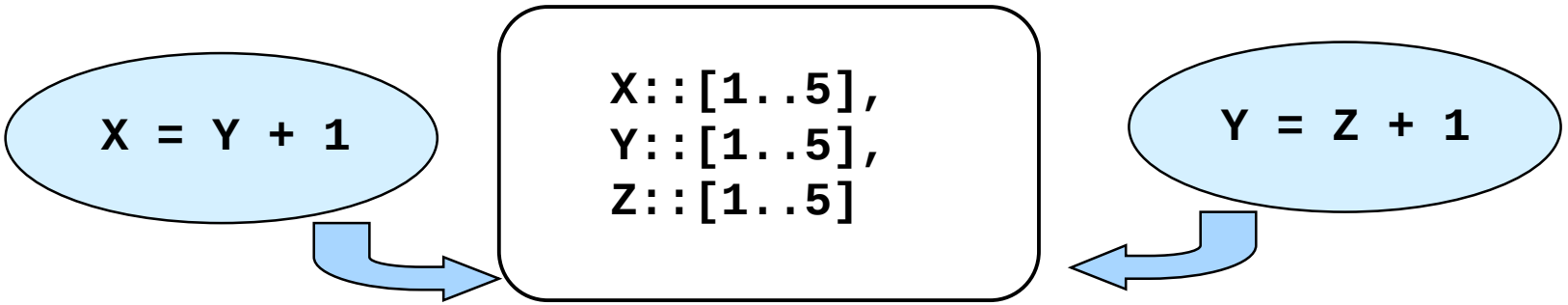
RICERCA

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

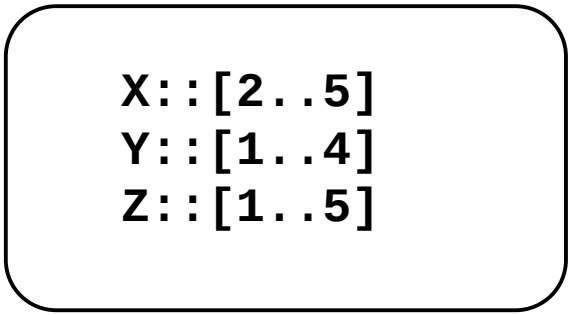
- Ricerca: idea
  - Divide il problema in sotto-problemi (BRANCHING) and risolve ciascuno di essi indipendentemente
  - I sottoproblemi devono essere una partizione del problema originale

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

Esempio



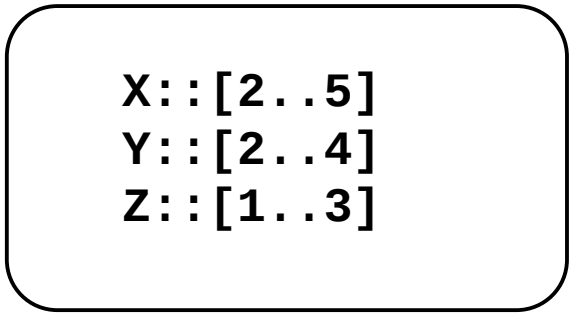
- Prima propagazione di ***X = Y + 1*** porta a



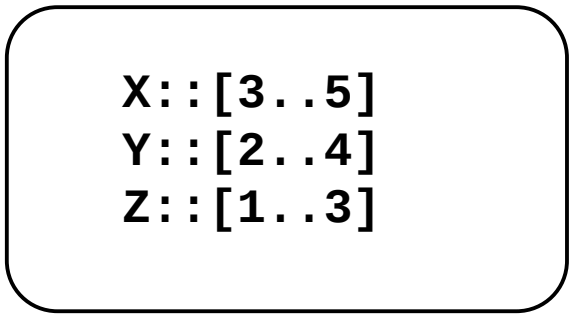
X = Y + 1
sospeso

INTERAZIONE TRA VINCOLI

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



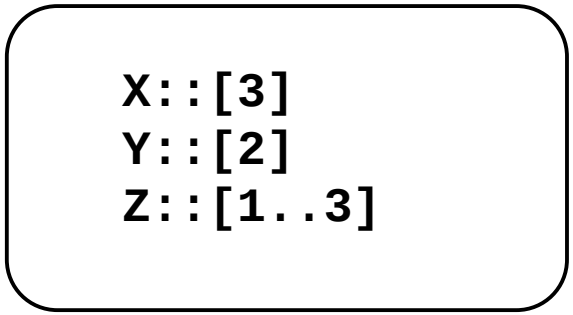
- Il dominio di ***Y*** e' cambiato ***X = Y + 1*** viene attivato



- NON ABBIAMO UNA SOLUZIONE E SIAMO A UN PUNTO FERMO

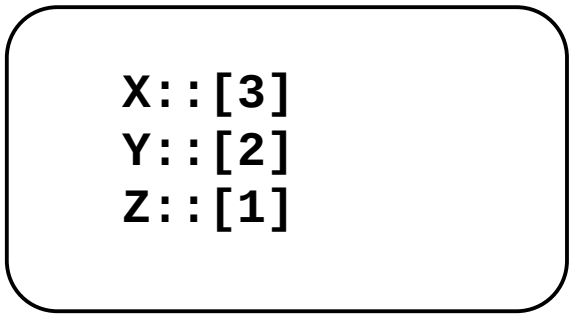
INTERAZIONE TRA VINCOLI

- Istanzio X a 3 (ho choice points): si sveglia X = Y+1 che istanzia Y a 2



X = Y + 1
RISOLTO

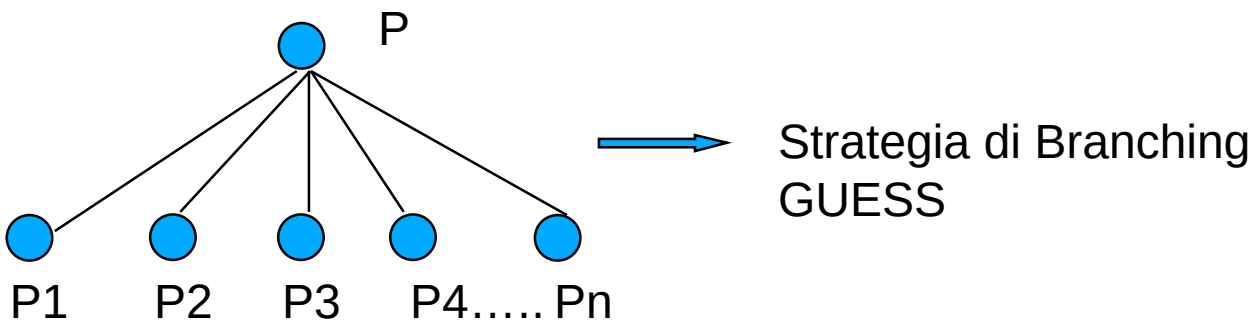
- Il dominio di ***Y*** e' cambiato ***Y = Z + 1*** viene attivato



Y = Z + 1
RISOLTO

- **IN BACKTRACKING ALTRE SOLUZIONI**

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 *labelling*

- LABELLING:
  - 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.

APPLICAZIONI

- Ci concentriamo su Programmazione a vincoli su domini finiti

- Applicazioni:
  - *Scheduling - Timetabling - Allocazione di risorse*
  - *Routing*
  - *Packing - Cutting*

  } ottimizzazione

  - *Riconoscimento di oggetti* ammissibilità

SCHEDULING - TIMETABLING - ALLOCAZIONE DI RISORSE

- Tre applicazioni con vincoli e caratteristiche comuni:
  - ci concentriamo sullo scheduling.

- Lo scheduling è forse l'applicazione che ha avuto risultati migliori usando la Programmazione a vincoli
  - flessibilità
  - generalità
  - codifica facile

- Problema NP-complete studiato nelle comunità di Ricerca Operativa e di Intelligenza Artificiale

SCHEDULING: definizione del problema

- Lo Scheduling riguarda l'assegnamento di risorse limitate (macchine, denaro, personale) alle attività (fasi di progetto, servizi, lezioni) sull'asse temporale
- Vincoli
  - restrizioni temporali
    - ordine tra attività
    - due dates - release dates
  - capacità delle risorse
    - tipi diversi di risorse
    - risorse consumabili e rinnovabili
- Criterio di ottimizzazione
    - makespan (lunghezza dello schedule)
    - bilanciamento delle risorse
    - ritardo sui tempi di consegna
    - costo dell'assegnamento delle risorse

SCHEDULING: Attività

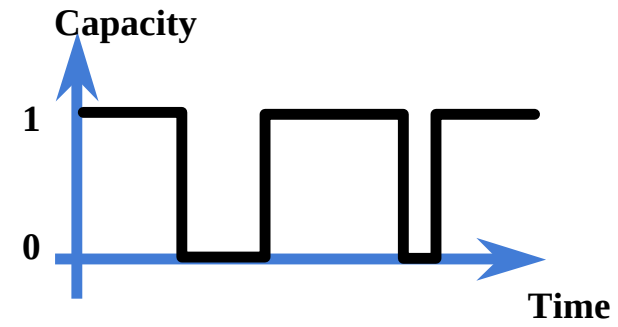
- Variabili decisioni :
  - Istante iniziale delle attività
  - Istante finale delle attività (oppure durata se variabile)
  - Risorse
  - Attività alternative (routing alternativi)

- Tipi di attività
  - interval activity: non può essere interrotta
  - breakable activity: può essere interrotta da breaks
  - preemptable activity: può essere interrotta da altre attività

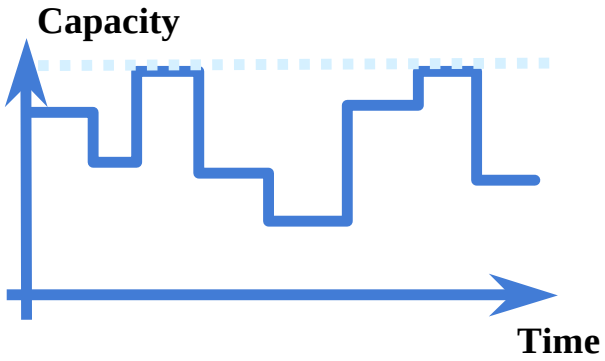
SCHEDULING: Risorse

- Tipi di risorse:
  - 1. Unarie
  - 2. Discrete
  - 3. Con stato
  - 4. Energia
  - 5. Stock

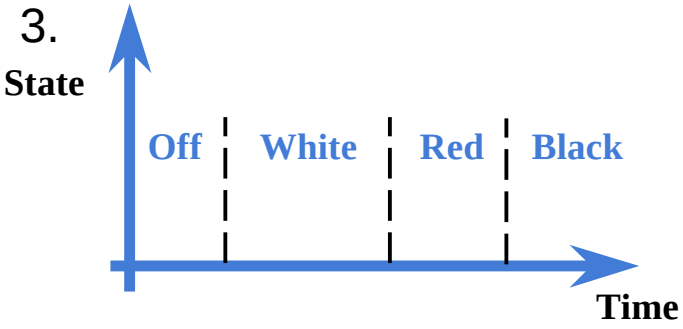
1.



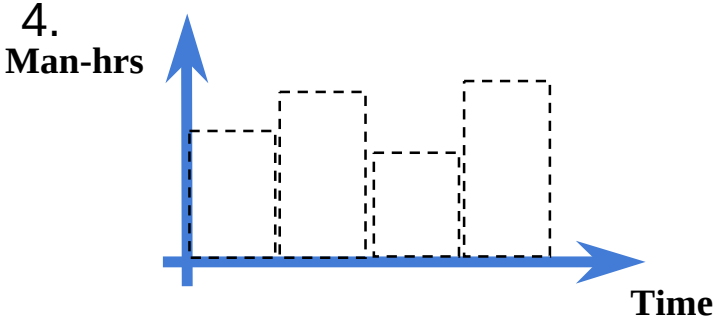
2.
5.



3.



4.



SCHEDULING: Esempio semplice

- 6 attività: ogni attività descritta da un predicato

  task(NAME,DURATION,LISTofPRECEDINGTASKS,MACHINE).

```
task(j1,3,[],m1).
task(j2,8,[],m1).
task(j3,8,[j4,j5],m1).
task(j4,6,[],m2).
task(j5,3,[j1],m2).
task(j6,4,[j1],m2).
```

- Le macchine m1 e m2 sono unarie (capacità 1).

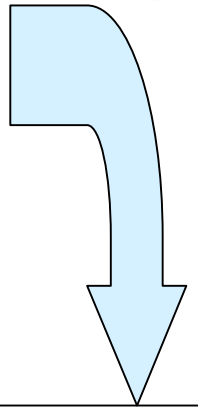
- Richiesto End: massimo tempo di fine schedule.

SCHEDULING: Esempio semplice

```
schedule(Data, End, TaksList):-
    makeTaskVariables(Data,End,TaskList),
    precedence(Data, TaskList),
    machines(Data, TaskList),
    minimize(labelTasks(TaskList),End).

makeTaskVariables([],_,[]).
makeTaskVariables([task(N,D,_,_)|T],End,[Start|Var]):-
    Start::[0..End-D],
    makeTaskVariables(T,End,Var).

precedence([task(N,_,Prec,_)|T], [Start|Var]):-
   select_preceding_tasks(Prec,T,Var,PrecVars,PrecDurations),
   impose_constraints(Start,PrecVars,PrecDurations),
   precedence(T,Var).

impose_constraints(_,[],[]).
impose_constraints(Start,[Var|Other],[Dur|OtherDur]):-
   Var + Dur <= Start
   impose_constraints(Start,Other,OtherDur).
```

SCHEDULING: Esempio semplice

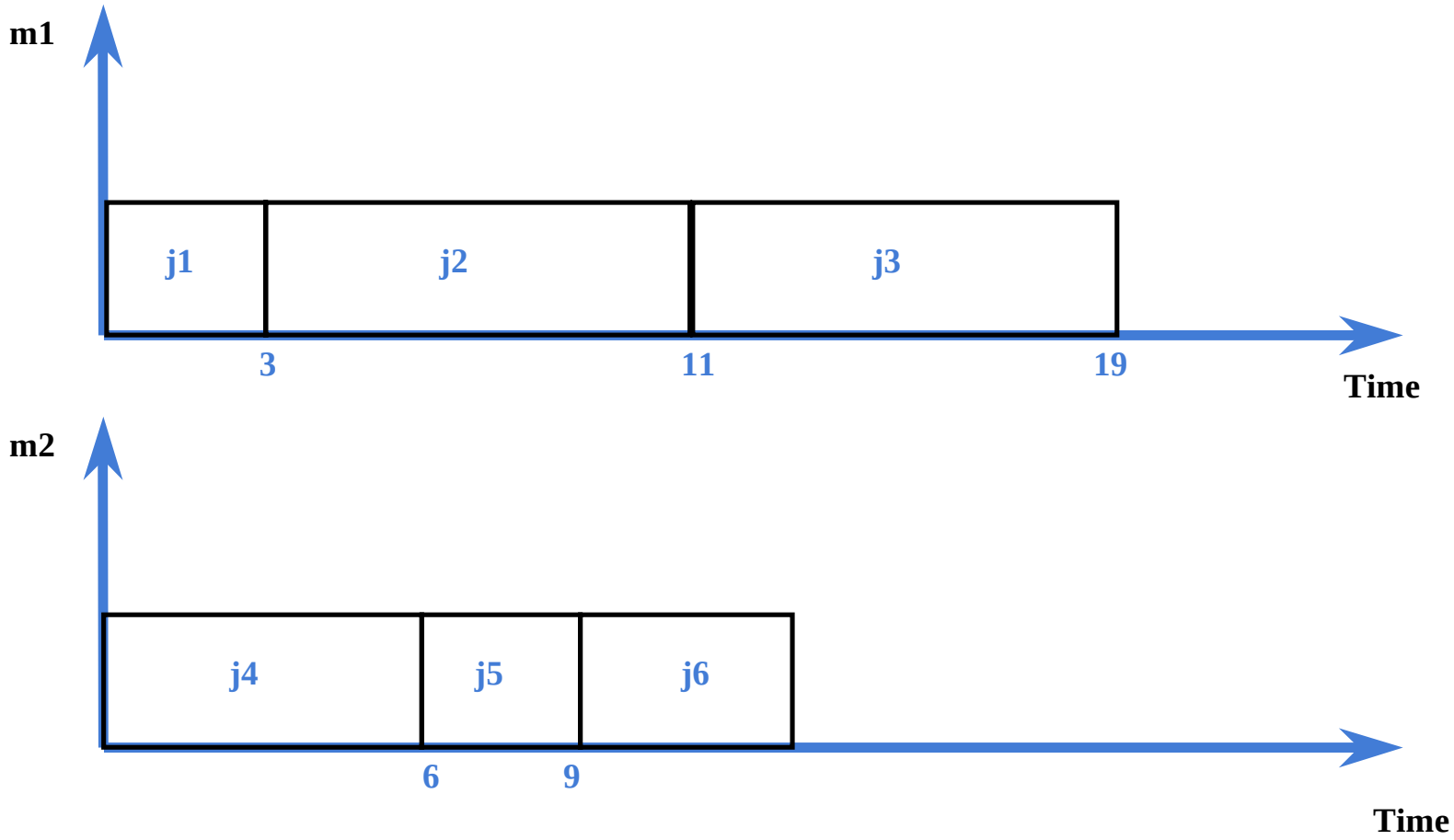
```
machines(Data, TaskList):-
   tasks_sharing_resource(Data,TaskList,SameResource,Durations),
   impose_cumulative(SameResource,Durations,Use).

impose_cumulative([],[],_).
impose_cumulative([ListSameRes|LSR],[Dur|D],[Use|U]):-
    cumulative(ListSameRes,Dur,Use,1),
     impose_cumulative(LSR,D,U).

labelTasks([]).
labelTasks([Task|Other]):-
    indomain(Task),
    labelTasks(Other).
```



Si specializza in

```
cumulative([Start1,Start2,Start3],[3,8,8],[1,1,1],1)
cumulative([Start4,Start5,Start6],[6,3,4],[1,1,1],1)
```

SCHEDULING: Soluzione ottima



SCHEDULING: Ottimalità

- **`minimize`**: predicato per trovare la soluzione ottima (Branch & Bound)

- Consideriamo la minimizzazione del makespan: uso una euristica che selezione sempre l'attività che può essere assegnata per prima. Come punto di scelta, la postpone.

```
labelTasks([]).
labelTasks(TaskList):-
    find_min_start(TaskList,First,MinStart,Others),
    label_earliest(TaskList,First,MinStart,Others,Postponed).

label_earliest(TaskList,First,Min,Others):- % schedule the task
    First = Min,
    labelTasks(Others).
label_earliest(TaskList,First,Min,Others):- % delay the task
    First ≠ Min,
    labelTasks(TaskList).
```

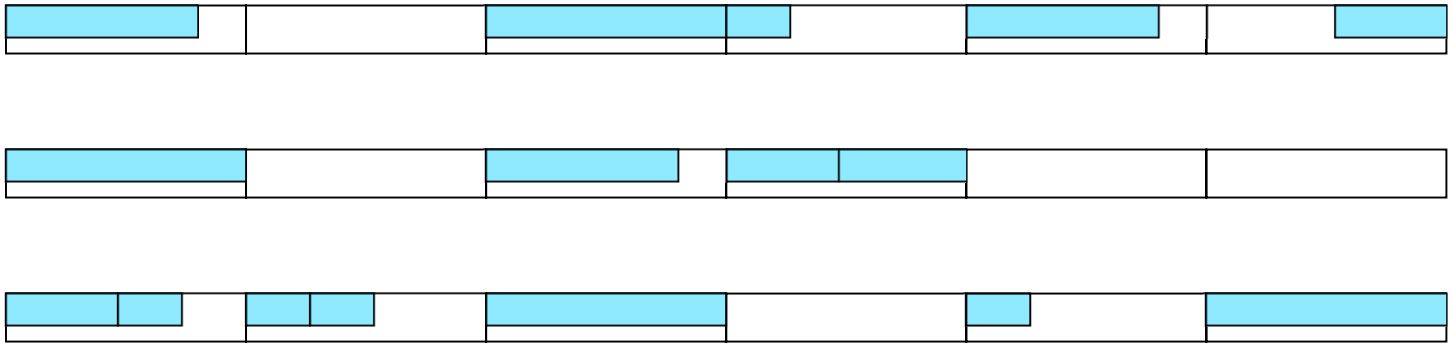
TIMETABLING: definizione del problema

- Timetabling riguarda la definizione di orari

- Vincoli
  - restrizioni temporali
    - ordine tra attività
    - due dates - release dates
  - capacità delle risorse
    - risorse discrete
- Criterio di ottimizzazione
    - costi/preferenze
    - bilanciamento risorse

TIMETABLING: esempio semplice

- Slot di 4 ore - Corsi che durano da 1 a 4 ore
- Due corsi non possono sovrapporsi
- Un corso deve essere contenuto in un singolo slot
- Preferenze su assegnamento Corso-Slot
  - Massimizzare la somma delle preferenze



TIMETABLING: codice con vincoli ridondanti

```
timetable(Data,Tasks,MaxTime,Costs):-
        define_variable_start(Tasks,MaxTime),
        define_variable_singleHours(Data,SingleHours),
        define_variable_courses3_4Hours(Data, Courses34Hours),
        impose_cumulative(Tasks),
        alldifferent(SingleHours),
        alldifferent(Courses34Hours),
        minimize(labeling(Tasks),Cost).
```

Vincoli ridondanti

- Variabili ridondanti
  - istanti iniziali
  - singole ore
  - corsi che durano 3 o 4 ore

Legate tra loro:

scambiano i risultati di propagazione

TIMETABLING: 0ttimalità e ricerca

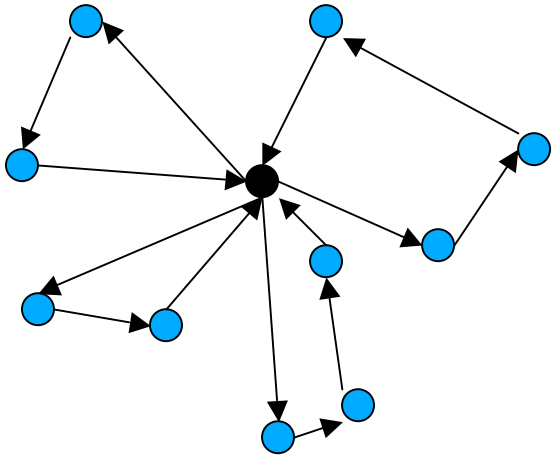
- Strategie di ricerca: quando si considerano problemi di ottimizzazione, possiamo sfruttare informazioni sui costi per definire una buona strategia di ricerca.

- Esempio:
  - Scegli la variabile che massimizza il regret
    - Regret: differenza tra la primo e il secondo miglior costo per ogni variabile.
    - Combinazione tra regret e first fail
  - Scegli il valore associato al minimo costo
- Esempio: euristica basata sulla soluzione di un rilassamento
  - Scegli la var che massimizza il regret
  - Come valore scegli la soluzione ottima del rilassamento

ROUTING: definizione del problema

- Il Routing riguarda il problema di trovare un insieme di percorsi coperti da un insieme di veicoli che partono e terminano allo stesso deposito.

- Vincoli
  - restrizioni temporali:
    - finestre temporali
    - durata massima di un viaggio
  - capacità del veicolo
  - domanda dei clienti
- Criteri di ottimizzazione
    - numero di veicoli
    - costo di viaggio
    - tempo di viaggio



ROUTING: definizione del problema

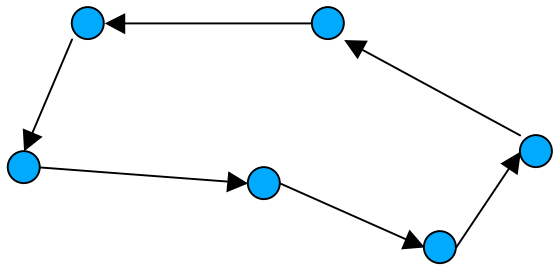
- Il routing è stato risolto con tecniche di ricerca operativa con tecniche di
  - Branch & Bound
  - Programmazione dinamica
  - Ricerca Locale
  - Branch & Cut
  - Column generation

- Il routing è stato risolto con Programmazione a vincoli
  - Branch & Bound
  - Ricerca Locale

- Componente di base: Travelling Salesman Problem (TSP) e varianti con finestre temporali.

TSP: definizione del problema

- TSP è il problema di cercare un cammino Hamiltoniano a costo minimo.



- Non sono ammessi sottocicli

- TSPTW: Le finestre temporali sono associate ad ogni nodo E' permesso l'arrivo prima del tempo, ma non dopo.

- Anche trovare un circuito Hamiltoniano (senza minimizzare il costo) è NP-completo

TSP: modello CP

- Variabili **Next** associate a ogni nodo. Il dominio di ogni variabile **Next** contiene i possibili nodi da visitare dopo **Next**

- N nodi ⟹ N + 1 variabili $\mathbf{Next_i}$ (il deposito viene duplicato)

- Per ogni **i**: $\mathbf{Next_i \neq i}$
- $\mathbf{nocycle([Next_0,\ldots Next_n])}$
- $\mathbf{alldifferent([Next_0,\ldots Next_n])}$
- Costo $\mathbf{c_{ij}}$ se $\mathbf{Next_i = j}$

- In alcuni modelli si possono usare variabili ridondanti **Prev** che indicano il predecessore del nodo.

TSP: codice

```
tsp(Data,Next,Costs):-
    remove_arcs_to_self(Next),
    nocycle(Next),
    alldifferent(Next),
    create_objective(Next,Costs,Z),
    minimize(labelling(Next),Z).
```

- **`nocycle`**: vincolo simbolico che assicura che non ci siano sottocicli in soluzione.
- **`create_objective`**: crea variabili **`Costs`**, e impone un vincolo **`element`** tra le variabili **`Next`** e **`Costs`**. Inoltre, crea una variabile **`Z`** che rappresenta la funzione obiettivo come somma di costi.

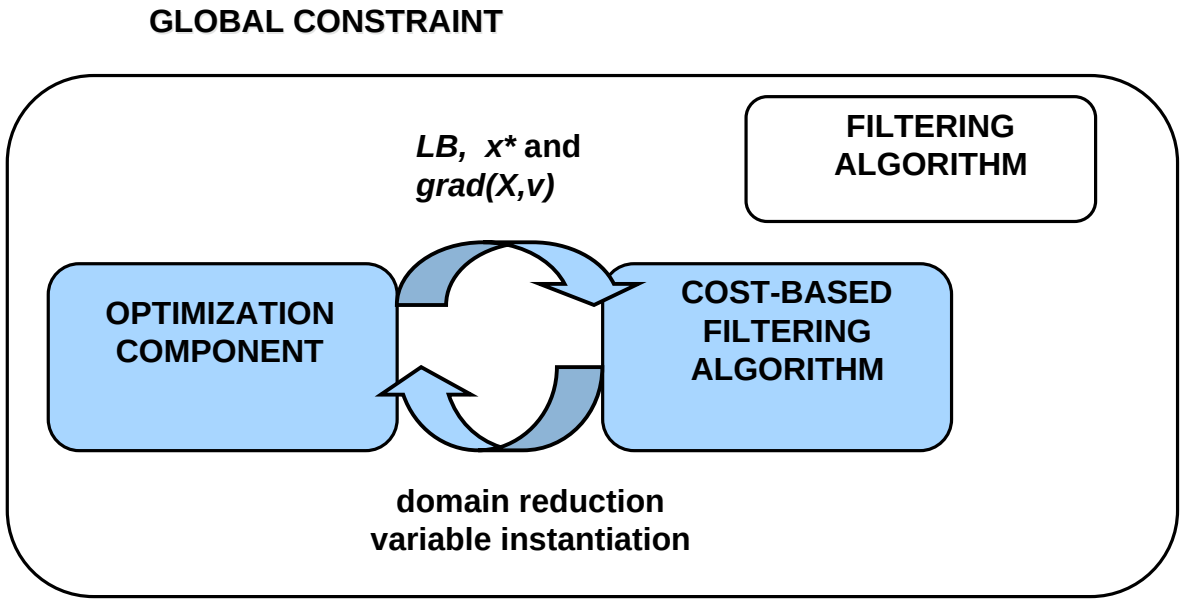
TSP: risultati

- Pura implementazione CP: risultati lontani dagli approcci di Ricerca Operativa.

- Integrazione di tecniche di Ricerca Operativa in CP: risultati migliori
  - ricerca locale
  - soluzioni ottime di rilassamenti
    - rilassamenti Lagrangiani
    - Problema dell'Assegnamento
    - Minimum Spanning Arborescence
  - strategie di ricerca basate su informazioni provenienti dai rilassamenti
    - eliminazione di sottocicli

- Aggiunta di finestre temporali in Ricerca Operativa richiede la riscrittura di molte parti di codice. In CP è immediata

VINCOLI GLOBALI DI OTTIMIZZAZIONE



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) ≥ Zmax* allora *Xi ≠ vj*

  classico *variable fixing* di ricerca operativa.

VINCOLI GLOBALI DI OTTIMIZZAZIONE

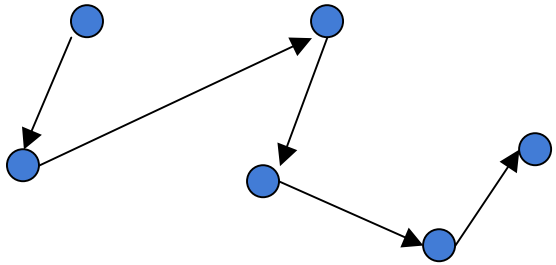
• L’esempio piu’ semplice di funzione gradiente è rappresentato dai *costi ridotti* calcolati dal rilassamento lineare o in alcuni bound combinatori.

• 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 sel'assegnamento di valori alle variabili $X_0, X_1, \ldots, X_k$ definisce un cammino semplice che coinvolge tutti i nodi ***0,...,k***.
Duplicando il primo nodo vincolo equivalente al nocycle
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

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

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

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

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

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.

RISULTATI SU TSPs

- Sebbene CP non sia competitiva con OR branch and cut (risolvono problemi fino a 14K nodi), l'aggiunta di vincoli di ottimizzazione permette di risolvere queste istanze in tempi ragionevoli. CP senza vincoli di ottimizzazione non risolve alcuna di queste istanze in un giorno. Le istanze random sono piu' semplici.

TSP and ATSP instances						
Instance	***pure AP***			***AP + cuts***		
	Opt	***Time***	***Fails***	***Opt***	***Time***	***Fails***
Gr17	2085	0.39	511	2085	0.49	30
Fri26	937	0.82	725	937	0.71	80
Bays29	2020	4.12	4185	2020	1.20	403
Dantzig42	699	>300	-	699	5.55	1081
RY48P	14854*	>300	-	14422	130.00	50K

RISULTATI SU TSPTW

• Non appena vengono aggiunti al modello time windows o vincoli di precedenza le cose cambiano

- Il TSPTW ha due componenti principali :
  - una componente di **routing** che considera l'*ottimizzazione*, ossia cerchiamo il cammino di costo minimo;
  - una componente di **scheduling** che considera l'ammissibilità ossia cerchiamo un cammino che soddisfi i vincoli imposti
    - *le visite nelle città sono viste come attività che possono essere svolte all'interno di finestre temporali. Tutte le attività sono svolte su una risorsa unaria in modo da non avere sovrapposizioni di visite.*

RISULTATI SU TSPTW

- Con i vincoli di ottimizzazione:

- TSPTW simmetrici:
  - migliorato lo stato dell'arte [Pesant et al. 1998]
  - Usare cutting planes nel rilassamento dell'assegnamento è molto efficace.

- TSPTW asimmetrici
  - risultati competitivi con metodi branch-and-cut (stato dell'arte) Ascheuer, et al. 2001
  - chiusi 4 problemi
  - Usare cutting planes nel rilassamento dell'assegnamento è inutile.

CUTTING & PACKING: definizione

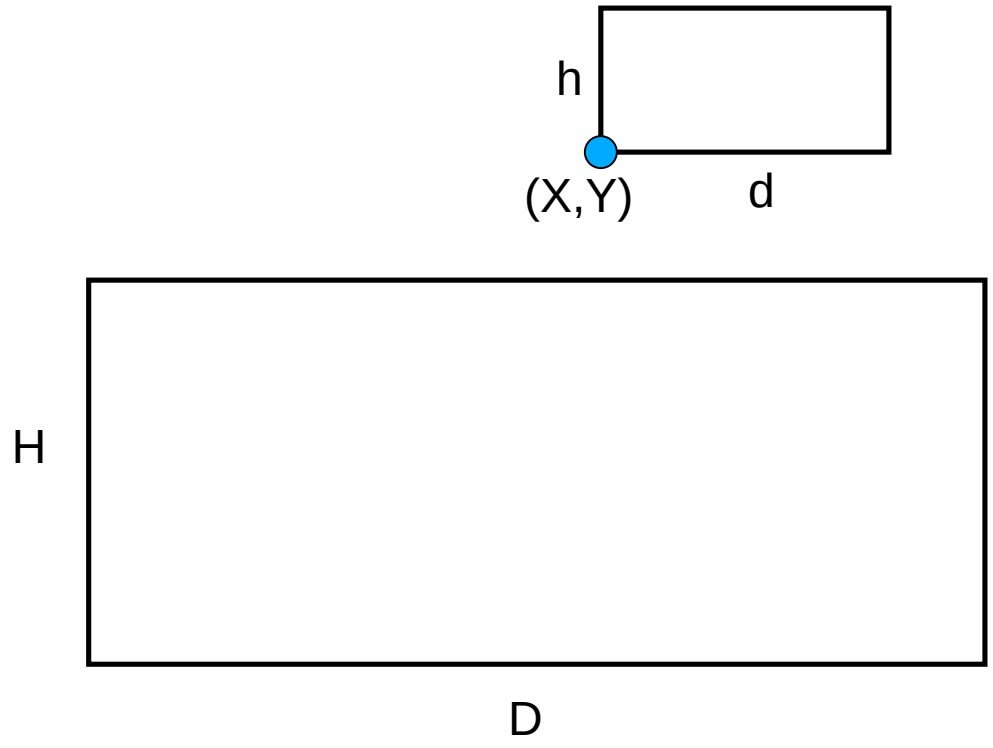
- Il Packing è il problema di piazzare un numero di pacchi (di dimensioni diverse) in uno o più contenitori in modo tale che non si sovrappongano e minimizzando lo spazio sprecato

- Il Cutting è il problema di tagliare un pezzo in modo di ottenere un dato numero di pezzi di dimensione fissa, minimizzando gli sprechi

- Molte varianti:
  - strip packing
  - guillottine cuts
  - rotazioni permesse o meno
  - 1 dimensione - 2 dimensioni - 3 dimensioni - 4 dimensioni

2-D PACKING: modello CP

- Per ogni pezzo da piazzare abbiamo una coppia di variabili che rappresentano le coordinate del punto in basso a sinistra



Pezzi:

`X::[0..D-d]`

`Y::[0..H-h]`

masterPiece o bin

2-D PACKING: modello CP

- Vincoli:
  - di non sovrapposizione: date due pezzi le cui coordinate sono (**X1**, **Y1**) and (**X2**, **Y2**) e dimensioni **D1**, **H1** e **D2**, **H2** rispettivamente

- **X1+D1 ≤ X2 OR Y1+H1 ≤ Y2 OR X2+D2 ≤ X1 OR Y2+H2 ≤ Y1**

- Forma di disgiunzione pesante: non c'e' propagazione anche dopo aver piazzato un pezzo

- Formulazione con vincoli simbolici:
- **cumulative(Xcoordinates,XDimension,Ydimension,H)**
  cumulative(Ycoordinates,YDimension,Xdimension,D)
- Esiste il vincolo diffn: alldifferent su piu' dimensioni

PACKING: codice

```
packing(Data,Xcoords,Ycoords,D,H):-
    create_variables(Data,Xcoords,Ycoords,D,H),
    state_disjunctive(Data,Xcoords,Ycoords),
    state_cumulatives(Data,Xcoords,Ycoords,D,H),
    create_objective(Xcoords,Ycoords,D,H,Z),
    minimize(label_squares(Xcoords,Ycoords),Z).
```

- **`create_objective`**: crea una variabile rappresentante lo spazio perso (o il numero di bin se sono più di uno)

- **`label_squares`** seleziona i pezzi più grandi per primi e assegna le coordinate per minimizzare lo spazio perso

RICONOSCIMENTO DI OGGETTI: definizione

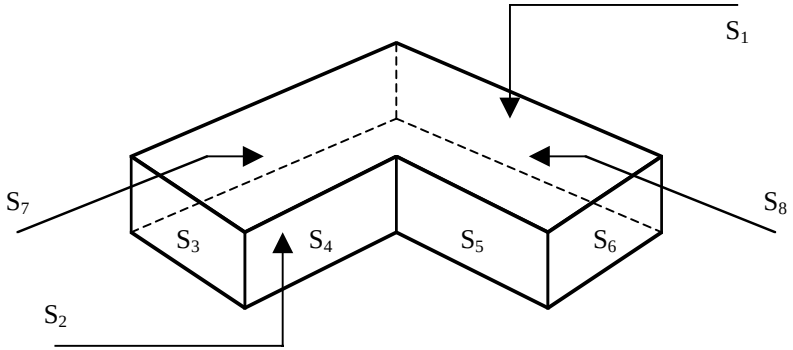
- Problema di riconoscere un oggetto in una scena

- Come descrivere il modello
- Come fare il mapping

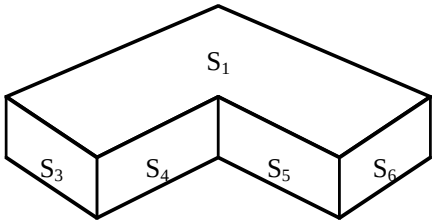
- MODELLO: Constraint graph
  - Nodi: parti dell'oggetto
  - Archi (vincoli): relazioni tra le parti
- RICONOSCIMENTO: soddisfacimento di vincoli

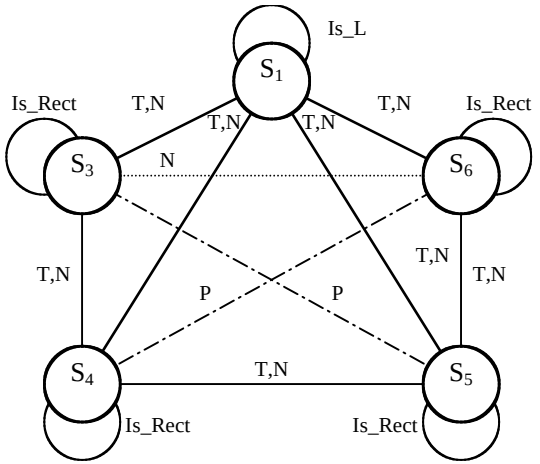
RICONOSCIMENTO DI OGGETTI

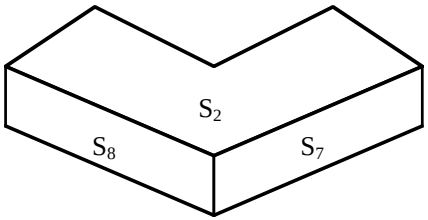
- OGGETTI 3-D

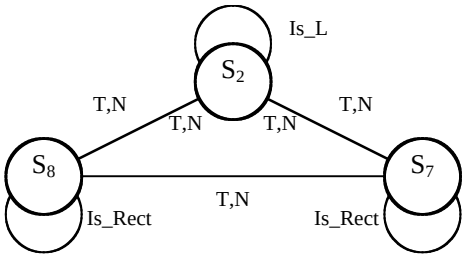


- Due viste e i corrispondenti modelli









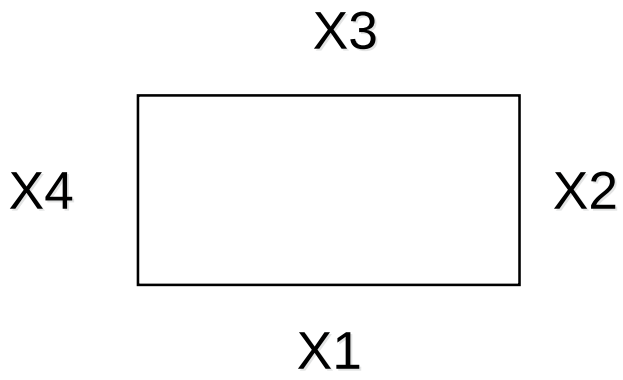
RICONOSCIMENTO DI OGGETTI

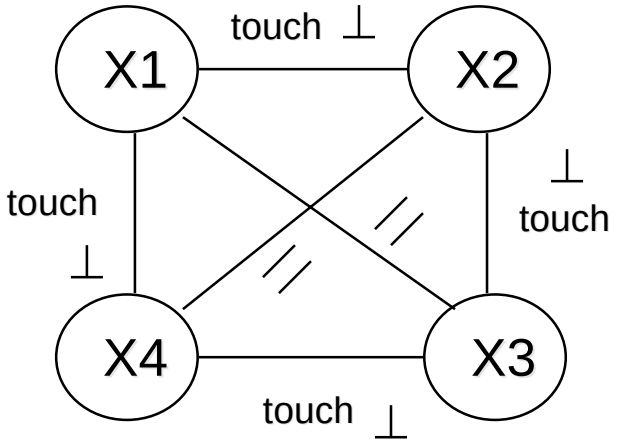
- Per riconoscere un oggetto, un sistema di visione di basso livello deve estrarre dall'immagine alcune caratteristiche visuali (superfici, segmenti)
- Possono essere applicate tecniche di soddisfacimento di vincoli per riconoscere un oggetto in una scena
- L'oggetto viene riconosciuto se le caratteristiche estratte soddisfano i vincoli contenuti nel modello
- I vincoli permettono di ridurre lo spazio di ricerca da esplorare

SEMPLICE ESEMPIO

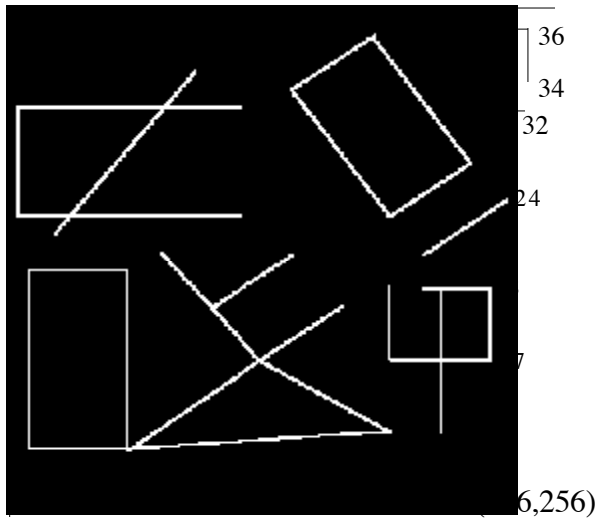
- Rettangolo

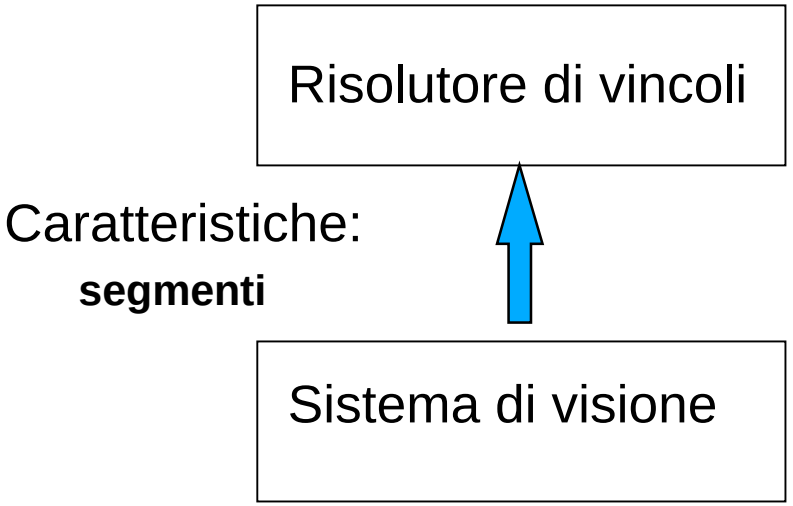
  Quattro lati (variabili) paralleli due a due che si toccano formando un angolo di 90...





ESEMPIO (2)





Rettangolo: modello CP

```
touch(X1,X2), touch(X2,X3), touch(X3,X4), touch(X1,X4),
perpend(X1,X2), perpend(X2,X3), perpend(X3,X4), perpend(X1,X4),
same_len(X2,X4), same_len(X1,X3), parallel(X2,X4), parallel(X1,X3)
```

Modello CP: VINCOLI USER DEFINED

Modello CP

```
recognize([X1,X2,X3,X4]):-
    X1,X2,X3,X4::[s1,s2,…,sn],
    touch(X1,X2), touch(X2,X3), touch(X3,X4), touch(X1,X4),
    perpend(X1,X2), perpend(X2,X3), perpend(X3,X4), perpend(X1,X4),
    same_len(X2,X4), same_len(X1,X3),
    parallel(X2,X4), parallel(X1,X3),
    labeling([X1,X2,X3,X4]).
```

- Dare la semantica dichiarativa e operazionale del vincolo: i segmenti sono descritti come fatti: **`segment(name,X1,Y1,X2,Y2)`**

- In tutti i linguaggi CP ci sono strumenti che permettono di definire nuovi vincoli.

- Esempio nella libreria CLP(FD) di ECL^{i}PS^e

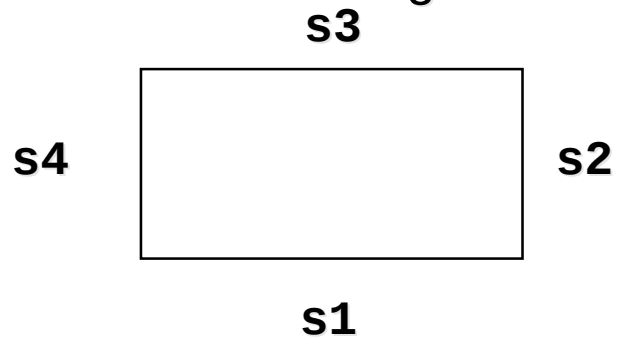
Modello CP: VINCOLI USER DEFINED

```
touch(X1,X2) :-
    dvar_domain(X1,D1),
    dvar_domain(X2,D2),
    arc_cons_1(D1,D2,D1new),  % user defined propagation
    (dom_compare(>,D1,D1new) -> dvar_update(X1,D1new); true),
     arc_cons_2(D1new,D2,D2new), % user defined propagation
    (dom_compare(>,D2,D2new) -> dvar_update(X2,D2new); true),
    (var(X1),var(X2) ->
            (make_suspension(touch(X1,X2),3,Susp),
            insert_suspension((X1,X2), Susp, any of fd, fd))
    ;       true),
    wake.
```

- Dopo la propagazione, se il vincolo non è risolto, viene sospeso e svegliato se si verifica un evento su una delle variabili coinvolte **`(X1,X2)`**.

Modello CP: SIMMETRIE

- Simmetrie: esistono quando abbiamo soluzioni equivalenti.
- Esempi: permulazioni, rotazioni ecc….
- Prendiamo quattro segmenti che formano un rettangolo
- Soluzione:
  - **X1 = s1**
  - **X2 = s2**
  - **X3 = s3**
  - **X4 = s4**



- Tre soluzioni identiche:

X1 = s2	**X1 = s3**	**X1 = s4**
X2 = s3	**X2 = s4**	**X2 = s1**
X3 = s4	**X3 = s1**	**X3 = s2**
X4 = s1	**X4 = s2**	**X4 = s3**

Modello CP: SIMMETRIE

- Problema: si perde tempo per esplorare stati equivalenti che non aggiungono alcuna informazione.

- Metodi per rimuovere le simmetrie:
  - aggiungere vincoli al modello (esempio ordinamenti tra variabili)
  - aggiungere vincoli dinamicamente nel corso della ricerca
  - modificare la strategia di ricerca per rimuovere le simmetrie.

- Argomento su cui vi è una ricerca molto attiva

ESEMPIO SEMPLICE

- Pigeonhole problem: Abbiamo n-1 gabbie in cui devono essere collocati n piccioni uno per gabbia.
- Problema chiaramente insolubile ma abbiamo simmetrie di permutazione

	Con simmetrie		*NO simmetrie*	
N piccioni	*Size Tree*	*Tempo (s)*	*Size Tree*	*Tempo (s)*
6	*119*	*0.213*	*15*	*0.042*
7	*719*	*1.031*	*31*	*0.064*
8	*5039*	*7.619*	*63*	*0.102*
9	*40319*	*61.902*	*127*	*0.228*

COME RIMUOVERE LE SIMMETRIE

- Abbiamo n variabili P_1, … P_n che rappresentano i piccioni e un dominio contenente le gabbie da 1 a n-1
- Usiamo tutti vincoli di diverso binari (per vedere l'impatto delle simmetrie)
- Simmetrie di permutazione n variabili n! stati simmetrici
- Si può imporre $P_1 < P_2$, $P_2 < P_3$ ….$P_{n-1} < P_n$ e si rimuovono tutte le simmetrie

SPORT SCHEDULING

- Dobbiamo allocare delle partite in diverse settimane
- Ci sono n squadre (nell'esempio da 0 a 7). Tutte le squadre devono giocare contro tutte le altre, quindi in totale ho n*(n-1)/2 partite. Devo allocare una partita per ogni cella in modo che in ogni settimana una squadra giochi una sola volta.

	Week 1	*Week 2*	*Week 3*	*Week 4*	*Week 5*	*Week 6*	*Week 7*
Game 1	*0 vs 1*	*0 vs 2*	*4 vs 7*	*3 vs 6*	*3 vs 7*	*1 vs 5*	*2 vs 4*
Game 2	*2 vs 3*	*1 vs 7*	*0 vs 3*	*5 vs 7*	*1 vs 4*	*0 vs 6*	*5 vs 6*
Game 3	*4 vs 5*	*3 vs 5*	*1 vs 6*	*0 vs 4*	*2 vs 6*	*2 vs 7*	*0 vs 7*
Game 4	*6 vs 7*	*4 vs 6*	*2 vs 5*	*1 vs 2*	*0 vs 5*	*3 vs 4*	*1 vs 3*

SPORT SCHEDULING: simmetrie

- *Le settimane sono simmetriche*
- *I Game sono simmetrici*

	Week 1	*Week 2*	*Week 3*	*Week 4*	*Week 5*	*Week 6*	*Week 7*
Game 1	*0 vs 1*	*0 vs 2*	*4 vs 7*	*3 vs 6*	*3 vs 7*	*1 vs 5*	*2 vs 4*
Game 2	*2 vs 3*	*1 vs 7*	*0 vs 3*	*5 vs 7*	*1 vs 4*	*0 vs 6*	*5 vs 6*
Game 3	*4 vs 5*	*3 vs 5*	*1 vs 6*	*0 vs 4*	*2 vs 6*	*2 vs 7*	*0 vs 7*
Game 4	*6 vs 7*	*4 vs 6*	*2 vs 5*	*1 vs 2*	*0 vs 5*	*3 vs 4*	*1 vs 3*

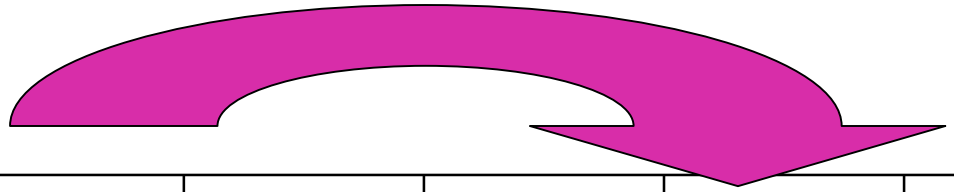
SPORT SCHEDULING: simmetrie

- *Le settimane sono simmetriche*
- *I Game sono simmetrici*

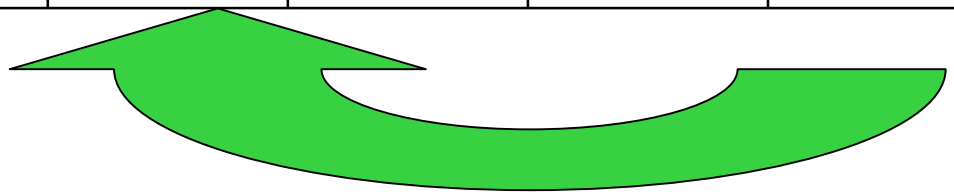
	Week 1	*Week 2*	*Week 3*	*Week 4*	*Week 5*	*Week 6*	*Week 7*
Game 1	*0 vs 1*	*0 vs 2*	*4 vs 7*	*3 vs 6*	*3 vs 7*	*1 vs 5*	*2 vs 4*
Game 2	*2 vs 3*	*1 vs 7*	*0 vs 3*	*5 vs 7*	*1 vs 4*	*0 vs 6*	*5 vs 6*
Game 3	*4 vs 5*	*3 vs 5*	*1 vs 6*	*0 vs 4*	*2 vs 6*	*2 vs 7*	*0 vs 7*
Game 4	*6 vs 7*	*4 vs 6*	*2 vs 5*	*1 vs 2*	*0 vs 5*	*3 vs 4*	*1 vs 3*

SPORT SCHEDULING: simmetrie

- *Cambiando l'ordine di due settimane trovo una soluzione equivalente*



	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Game 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4
Game 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6
Game 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7
Game 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3



SPORT SCHEDULING: simmetrie

- *Cambiando l'ordine di due settimane trovo una soluzione equivalente*

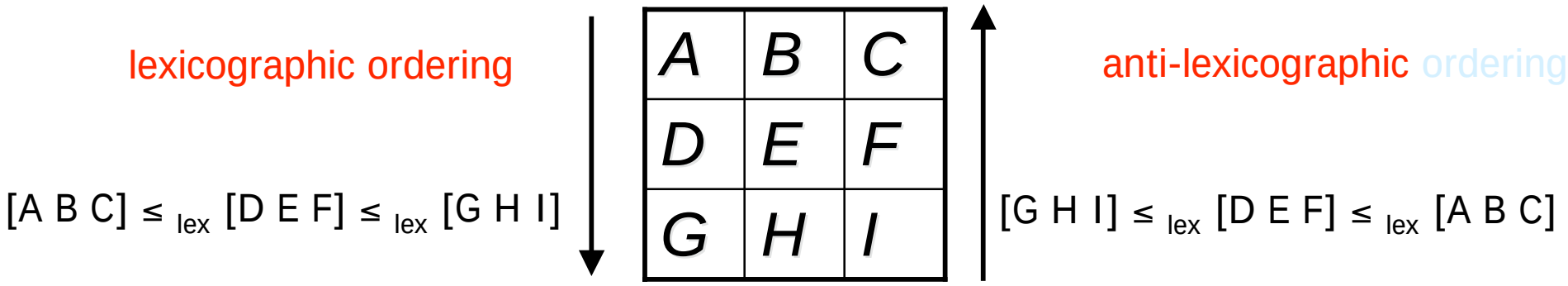
	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	0 vs 1	3 vs 7	4 vs 7	3 vs 6	0 vs 2	1 vs 5	2 vs 4
Period 2	2 vs 3	1 vs 4	0 vs 3	5 vs 7	1 vs 7	0 vs 6	5 vs 6
Period 3	4 vs 5	2 vs 6	1 vs 6	0 vs 4	3 vs 5	2 vs 7	0 vs 7
Period 4	6 vs 7	0 vs 5	2 vs 5	1 vs 2	4 vs 6	3 vs 4	1 vs 3

PERCHE' PREOCCUPARSENE

- Per una matrice **nxm** che ha simmetrie di riga e di colonna ci sono **n!*m!** simmetrie (super-esponenziale)
- Per ogni soluzione (totale o parziale) ce ne sono **n!*m!** simmetriche, ma anche per ogni fallimento
- Eliminare tutte le simmetrie non e' facile
- Ci si accontenta spesso di eliminarne alcune, in modo da ottenere un albero ridotto.

COME ELIMINARLE: VINCOLI AGGIUNTIVI NEL MODELLO

- Solo simmetrie di riga, posso eliminarle con un ordinamento totale sulle righe:
- Forzare le righe ad essere lessicograficamente ordinate rompe tutte le simmetrie di riga



SIMMETRIE DI RIGA E COLONNA

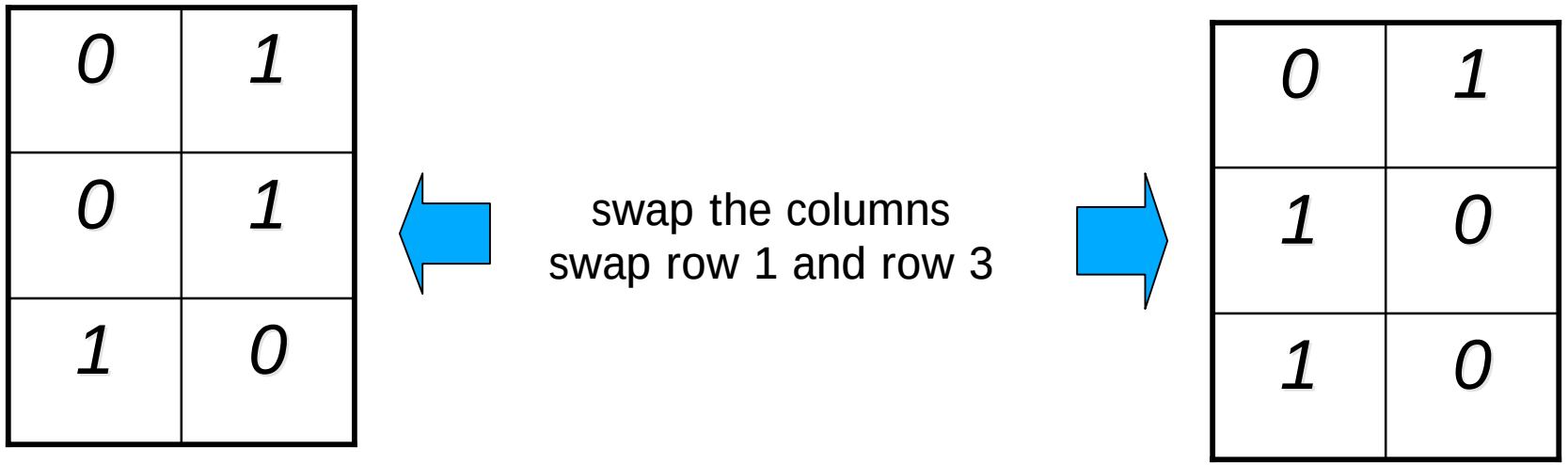
- Riusciamo a eliminare le simmetrie di riga e colonna singolarmente, ma se compaiono insieme, imporre gli ordinamenti su righe e colonne non elimina tutte le simmetrie.

Good News ☺

- Una simmetria definisce una classe di equivalenza
  - Due assegnamenti sono equivalenti se una simmetria mappa un assegnamento in un altro.
- Ogni simmetria ha ALMENO UN elemento in cui SIA le righe SIA le colonne sono ordinate in senso lessicografico
  - Puo' non esistere un elemento con le righe ordinate in senso lessicografico e le colonne in senso antilessicografico
- Per eliminare le simmetrie di riga e colonna possiamo ordinare lessicograficamente righe e colonne *(double-lex)*

Bad news ☹

- Una simmetria puo' avere piu' di un elemento con righe e colonne ordinate lessicograficamente
- Double-lex non elimina tutte le simmetrie



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
- 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

CP ESTENSIONI PER L'OTTIMIZZAZIONE
TESI DISPONIBILI

- Integrazione di tecniche di Ricerca Operativa in CP:
  - Solver MP: 2LP ***[McAloon, Tretkoof PPCP94],*** OPL ***[Van Hentenryck, 99],*** Planner ***[ILOG Planner Manual]***
    - Integrazione di CPLEX e XPRESS (simplesso) in solver FD
  - Integrazione di algoritmi specifici per:
    - calcolare bound
    - costi ridotti

    } ***[Caseau, Laburthe ICLP97 and CP97], [Focacci, Lodi, Milano ICLP99 and CP99],***
  - Miglioramenti di branch and bound
    - ***[Rodosek, Wallace, Hajian Annals OR 97], [Caseau, Laburthe ICLP94 and JICSLP96], [Beringer, DeBacker, LP Formal Method and Pract. Appl. 95]***
  - Integrazione di tecniche di local search
  - ***[DeBacker, Furnon,Shaw CPAIOR99], [Caseau, Laburthe CPAIOR99], [Gendreau, Pesant, Rousseau, Transp. Sci. 98]***
  - Integrazione di branch and cut in logica
    - ***[Bockmayr ICLP95], [Kasper PhD, 99]***

ALTRE ESTENSIONI

- Estensioni per problemi sovravincolati

  [A. Borning OOPSLA87], [A. Borning et al. ICLP89], [M.Jamper PhD, 96]

- Estensioni per cambiamenti dinamici

  [R.Dechter, A.Dechter, AAAI88], [Verfaillie, Schiex AAAI94], [Bessiere, AAAI91], [Lamma et al. IJCAI99]

- Estensioni per problemi multi-criteria
  - TESI DISPONIBILI

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`**
- *Mailing list:* **`CPWORLD@gmu.edu`**
- *Archivi:* **`http://www.cs.unh.edu/ccc/archive/`**