APPLICAZIONI

- Ci concentriamo su Programmazione a vincoli su domini finiti

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

  } ottimizzazione

  - *Grafica - Visione*
  - *Le aste elettroniche*

  } 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

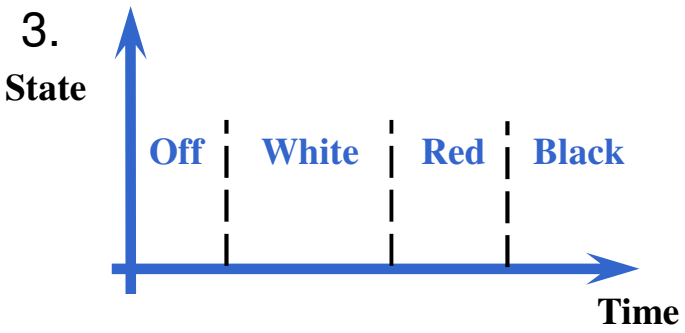
SCHEDULING: Attività

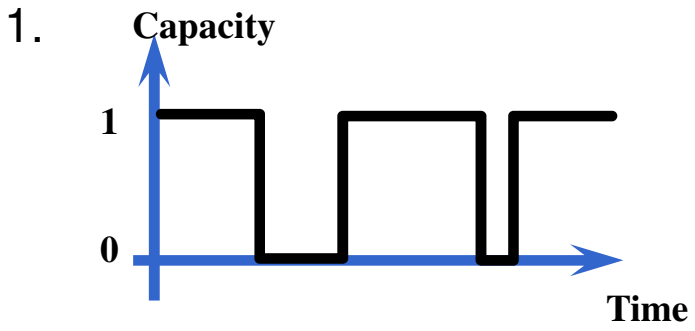
- Variabili decisioni :
  - Istante iniziale delle attività
  - Istante finale delle attività (oppure durata se variabile)
  - Risorse
  - Attività alternative (routing alternativi)

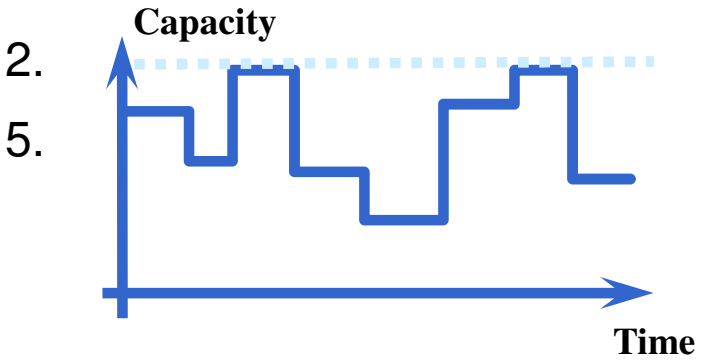
- Tipi di attività
  - interval activity: non può essere interrotta
  - breakable activity: può essere interrotta da breaks
  - preemptable activity: può essere interrotta da altre attività

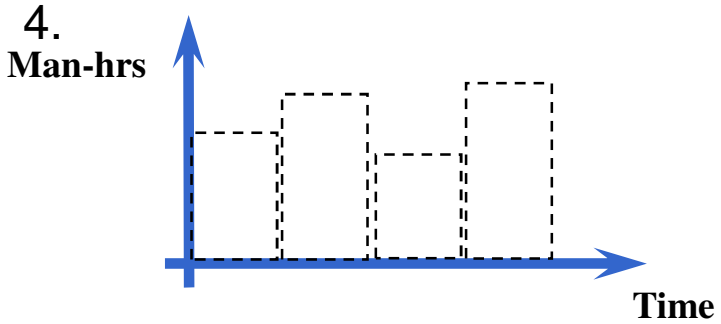
SCHEDULING: Risorse

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









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).
```

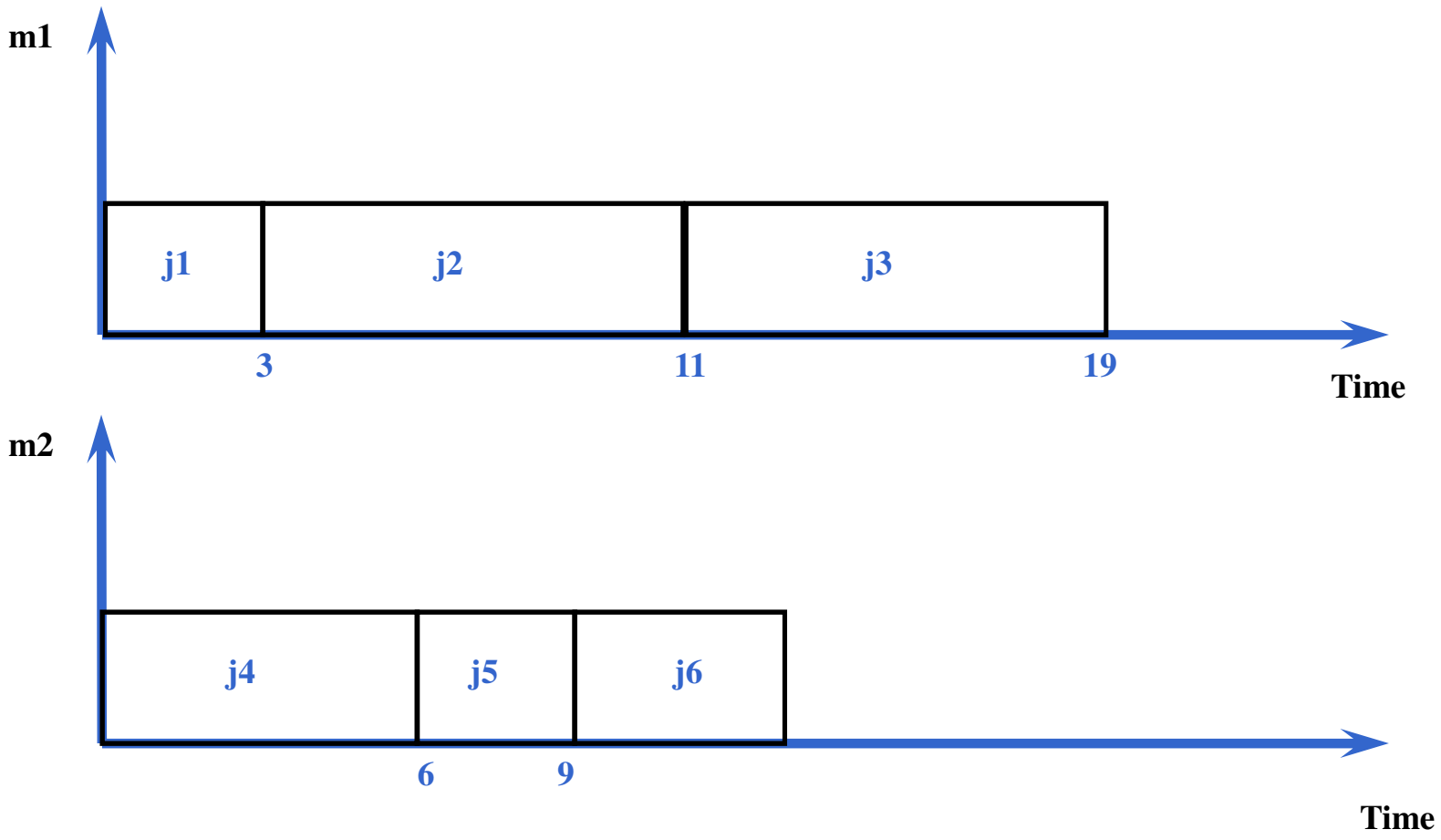
SCHEDULING: Esempio semplice

```
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).
```

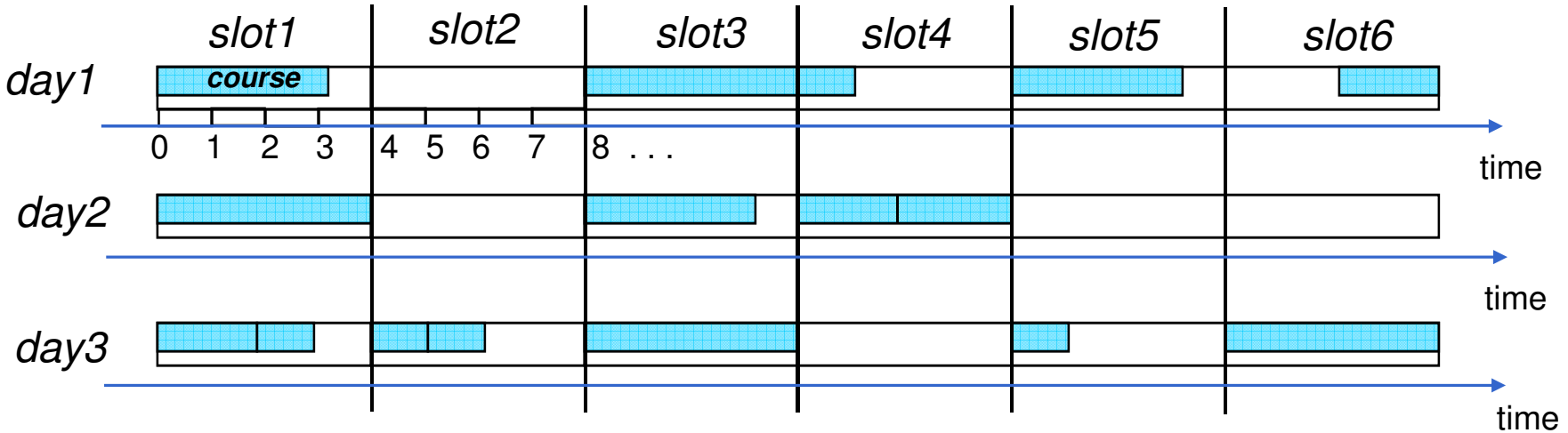
TIMETABLING: definizione del problema

- 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

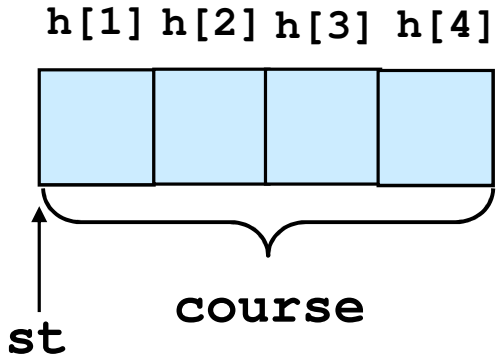
TIMETABLING: esempio semplice

- *Esempio preso da [Caseau Laburthe CP97]*
- 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



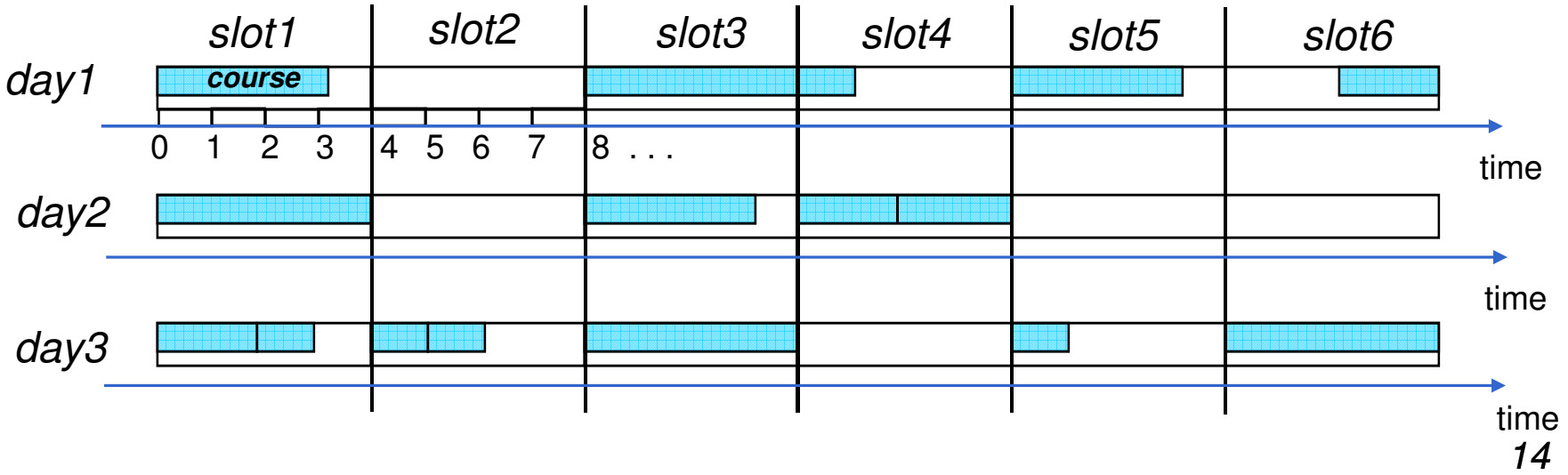
MODELLO

- Variabili associate ai corsi. Durata corso **`Nh`** ore
  - Start time **`st`**: il dominio contiene possibili start time del corso
  - Single hours **`h[i] i=1..Nh`**: il dominio contiene le ore
  - Corsi **`course`**: il dominio contiene gli slot
- Esistono vincoli che legano le diverse variabili



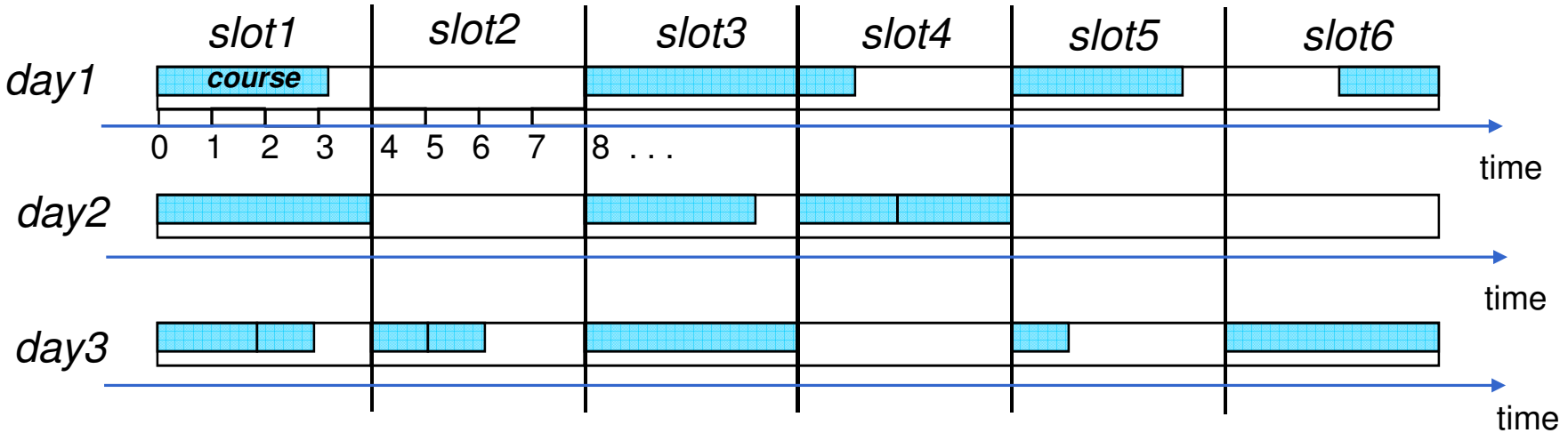
MODELLO: Esempio

- Ho un corso che dura **Nh=3** ore
  - **st::[0,1,4,5,8,9,…]**
  - **h[1]::[0,1,4,5,8,9,…] h[2]::[1,2,3,4,9,10,…]**
  - **h[3]::[2,3,6,7,10,11,…]**
  - **course::[day1slot1,day1slot2,…,day2slot1,…,daynslotm]**



EXAMPLE

- Constraints
  - `cumulative(startVars, durations, [1..1], 1);`
  - `AllDiff(singleHours);`
  - `// subproblem defined by 3 and 4 hour courses`
  - `AllDiff(courses34Hours);`



pseudo-codice con vincoli ridondanti

```
timetable(Data,Start,MaxTime,Costs):-
        define_variable_start(Start,MaxTime),
        define_variable_singleHours(Data,SingleHours),
        define_variable_courses3_4Hours(Data, Courses34Hours),
        impose_cumulative(Start),
        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

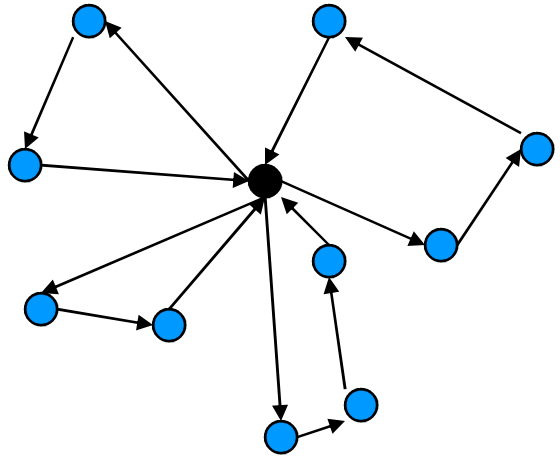
- Strategie di ricerca: quando si considerano problemi di ottimizzazione, possiamo sfruttare informazioni sui costi per definire una buona strategia di ricerca.

- 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

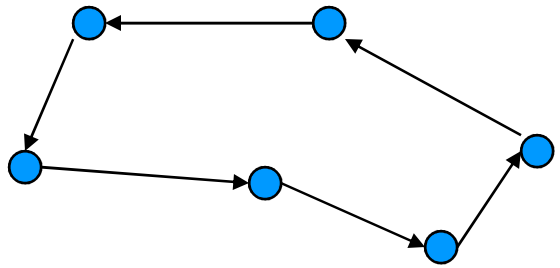
- Il routing è stato risolto con tecniche di ricerca operativa con tecniche di
  - Branch & Bound
  - Programmazione dinamica
  - Ricerca Locale
  - Branch & Cut
  - Column generation

- 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**: $\mathbf{Next_i \neq i}$

path([Next$_0$,…Next$_n$])

alldifferent([Next$_0$,…Next$_n$]) *ridondante*

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),
    path(Next),
    alldifferent(Next),
    create_objective(Next,Costs,Z),
    minimize(labelling(Next),Z).
```

- **`path`**: 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.

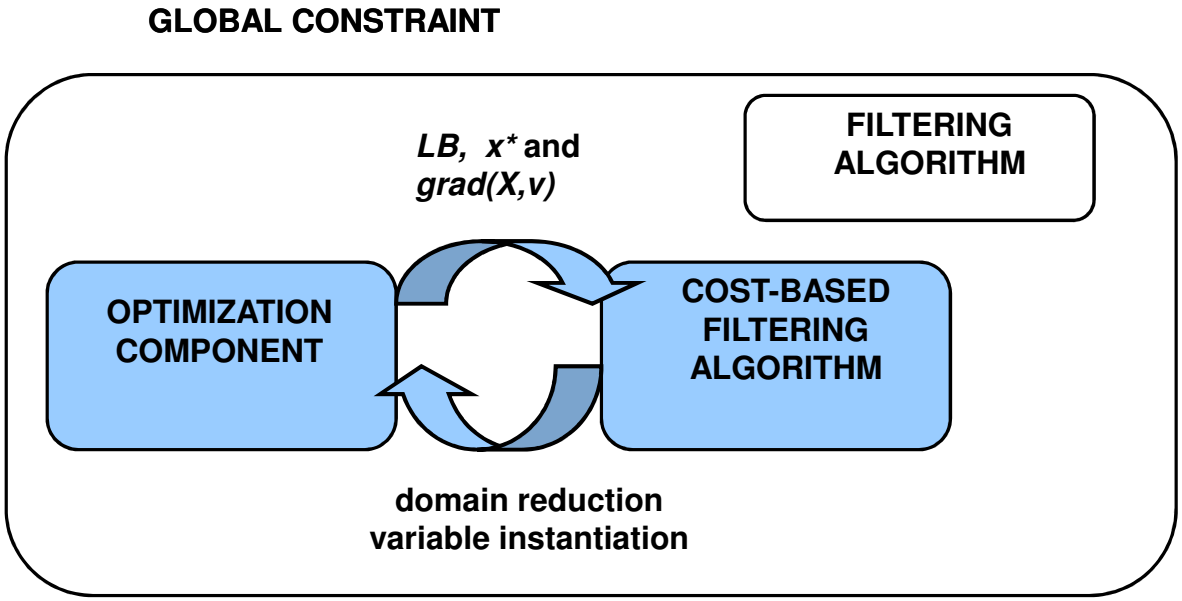
TSP: risultati

- 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



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

VINCOLI GLOBALI DI OTTIMIZZAZIONE

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

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

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

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

  classico *variable fixing* di ricerca operativa.

VINCOLI GLOBALI DI OTTIMIZZAZIONE

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

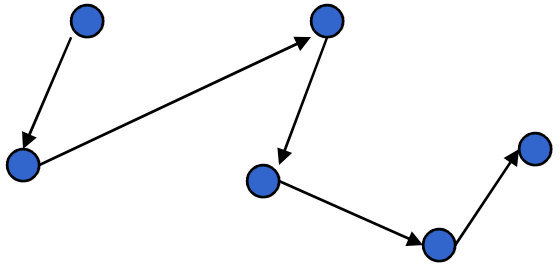
• Esempio: path constraint

PATH CONSTRAINT

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

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

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



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

In generale il vincolo path e' un multi-path

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

MAPPING

CP-Model:

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

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

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

$C_0 + \ldots + C_n = Z$

minimize(Z)

Mapping

ILP-Model

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

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

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

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

x_{ij} integer

$X_i = i_j$ ⟷ $x_{ij}=1$

path $([X_0, X_1, \ldots, X_n])$ ⟷ *relaxed in* ⟷ (A) + (B)

$C_i = c_{ij}$ ⟷ c_{ij} in z

PATH CONSTRAINT

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

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

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

RISULTATI SU TSPs

- Sebbene CP non sia competitiva con OR branch and cut (risolvono problemi fino a 24K 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: [Focacci Lodi Milano 2002]

- 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.

RICONOSCIMENTO DI OGGETTI: definizione

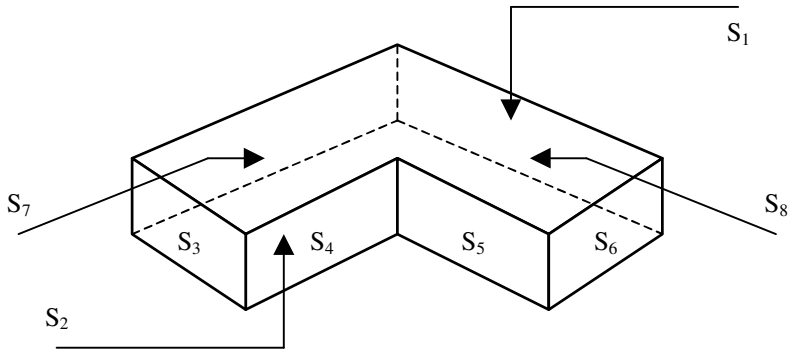
- Problema di riconoscere un oggetto in una scena

- Come descrivere il modello
- Come fare il mapping

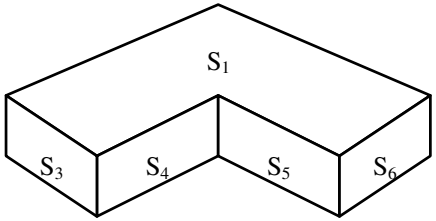
- MODELLO: Constraint graph
  - Nodi: parti dell'oggetto
  - Archi (vincoli): relazioni tra le parti
- RICONOSCIMENTO: soddisfacimento di vincoli

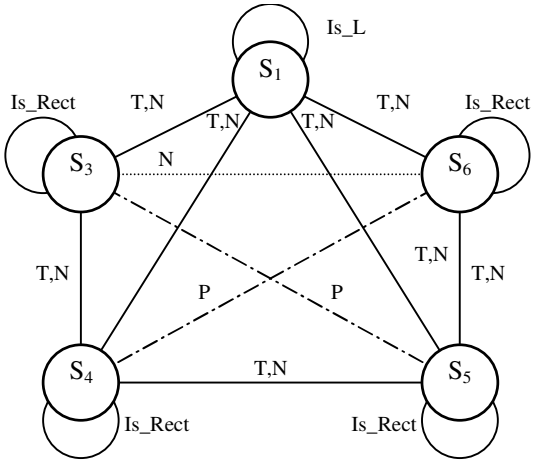
RICONOSCIMENTO DI OGGETTI

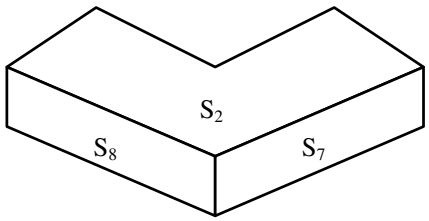
- OGGETTI 3-D

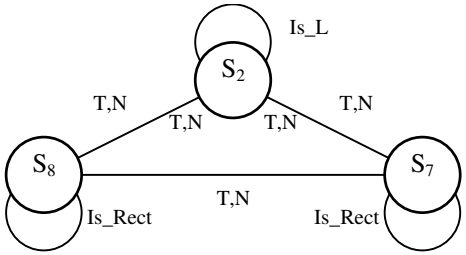


- Due viste e i corrispondenti modelli









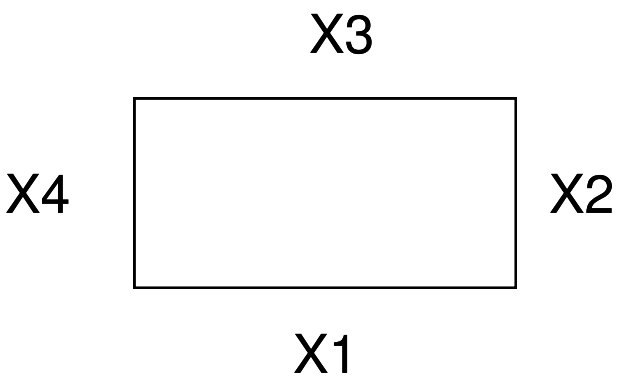
RICONOSCIMENTO DI OGGETTI

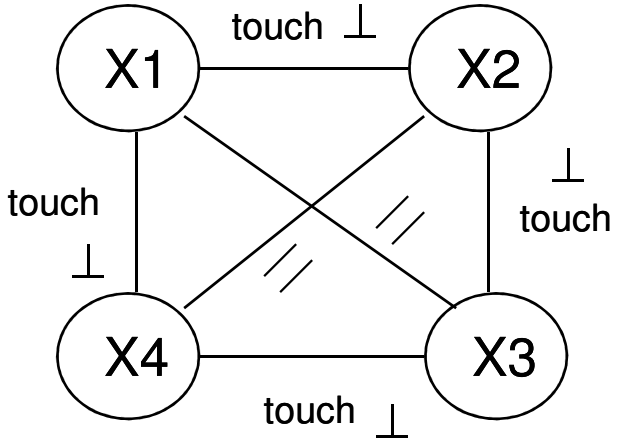
- Per riconoscere un oggetto, un sistema di visione di basso livello deve estrarre dall'immagine alcune caratteristiche visuali (superfici, segmenti)
- Possono essere applicate tecniche di soddisfacimento di vincoli per riconoscere un oggetto in una scena
- L'oggetto viene riconosciuto se le caratteristiche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

SEMPLICE ESEMPIO

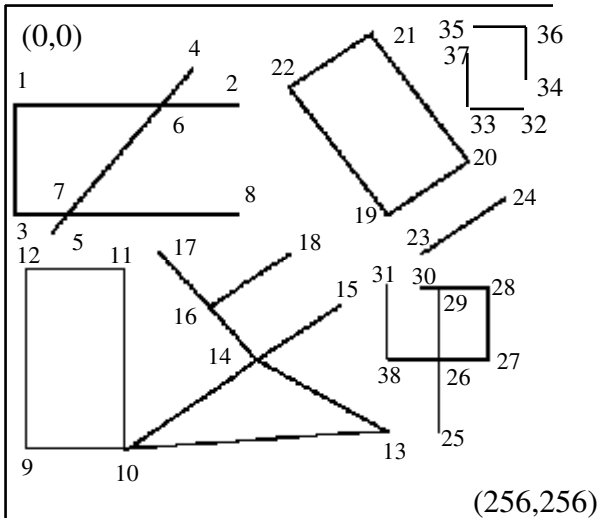
- Rettangolo

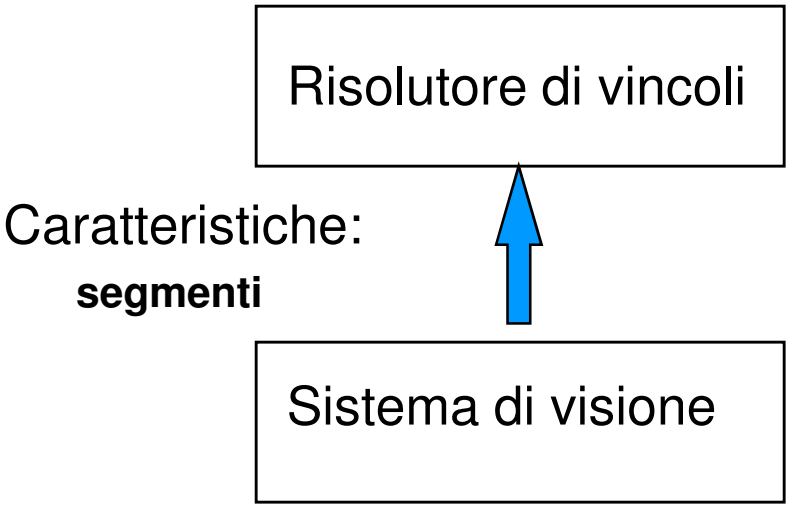
  Quattro lati (variabili) paralleli due a due che si toccano formando un angolo di 90...





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*

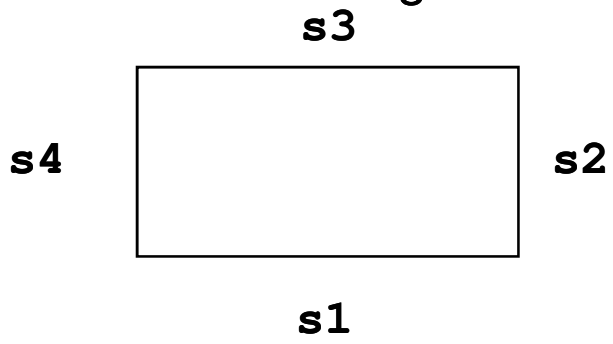
Modello CP: VINCOLI USER DEFINED

```
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*

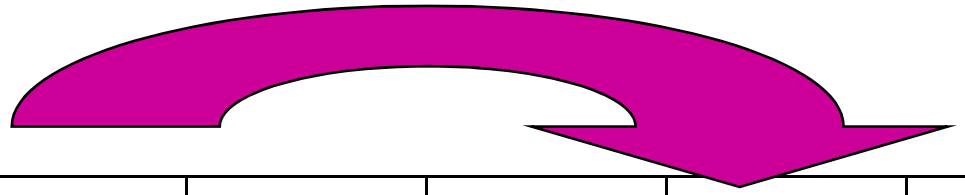
SPORT SCHEDULING: simmetrie

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

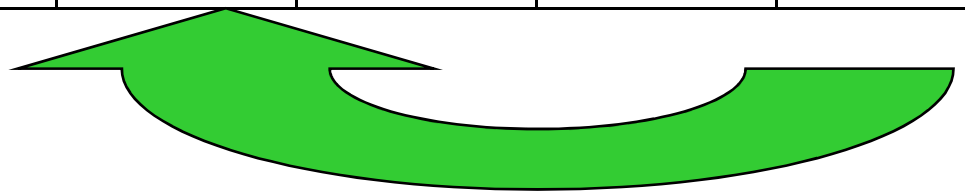
	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Game 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4
Game 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6
Game 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7
Game 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3

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

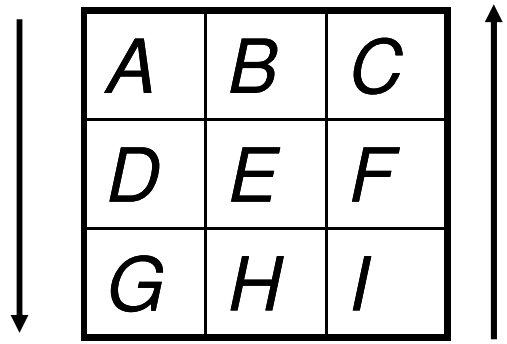
- Per una matrice **nxm** che ha simmetrie di riga e di colonna ci sono **n!*m!** simmetrie (super-esponenziale)
- Per ogni soluzione (totale o parziale) ce ne sono **n!*m!** simmetriche, ma anche per ogni fallimento
- Eliminare tutte le simmetrie non e' facile
- Ci si accontenta spesso di eliminarne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

lexicographic ordering

$[A\ B\ C] \leq_{lex} [D\ E\ F] \leq_{lex} [G\ H\ I]$



anti-lexicographic ordering

$[G\ H\ I] \leq_{lex} [D\ E\ F] \leq_{lex} [A\ B\ C]$

SIMMETRIE DI RIGA E COLONNA

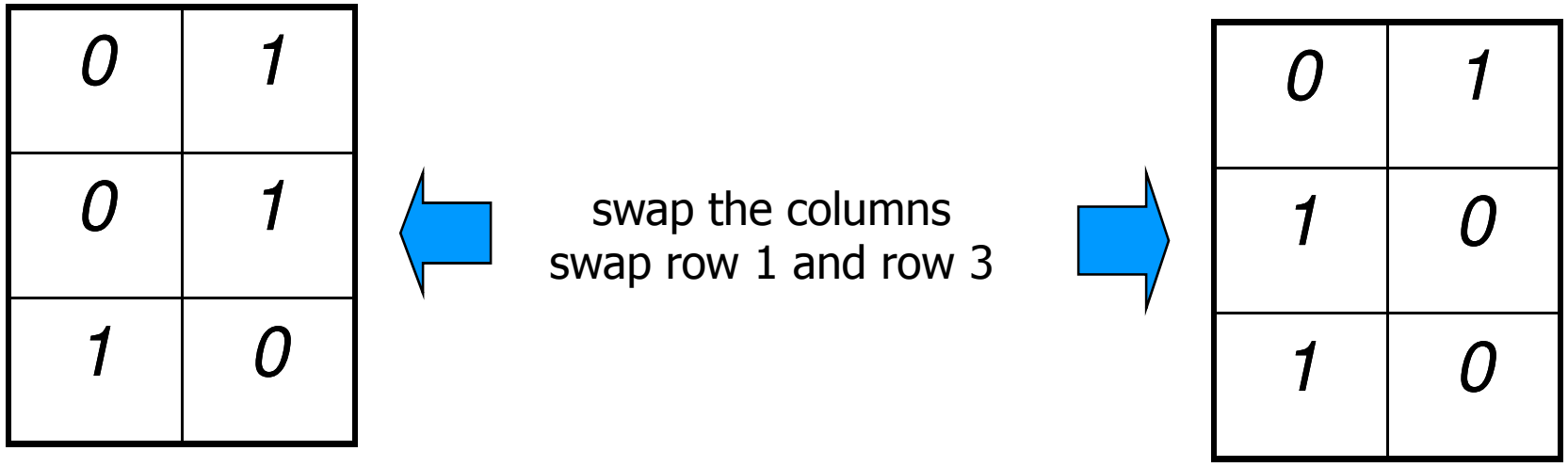
- Riusciamo a eliminare le simmetrie di riga e colonna singolarmente, ma se compaiono insieme, imporre gli ordinamenti su righe e colonne non elimina tutte le simmetrie.

Good News ☺

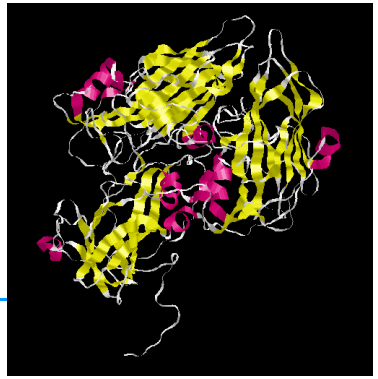
- Una simmetria definisce una classe di equivalenza
  - Due assegnamenti sono equivalenti se una simmetria mappa un assegnamento in un altro.
- Ogni simmetria ha ALMENO UN elemento in cui SIA le righe SIA le colonne sono ordinate in senso lessicografico
  - Puo' non esistere un elemento con le righe 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



APPLICAZIONI RECENTI



- Assegnazione di frequenze nei cellulari
  - 2 cellulari nella stessa cellula non possono avere stessa frequenza ➔ stesso modello del graph coloring!
- Bioinformatica
  - calcolo dello sviluppo spaziale di proteine
- Object recognition
- Computational Sustainability
- Embedded System Design
- Tecnologia usata da Airfrance, British Airways, Cisco Systems, …