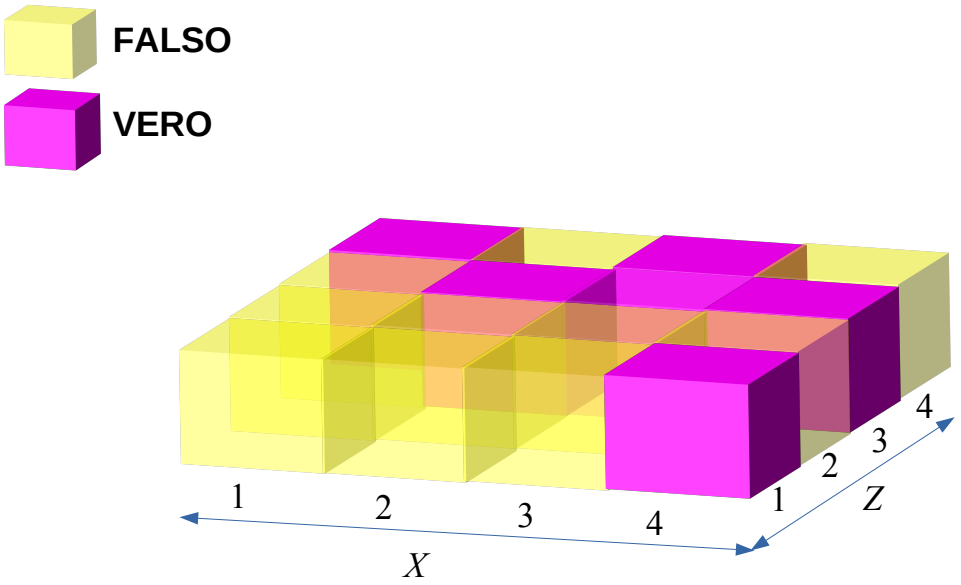
VINCOLI N-ari

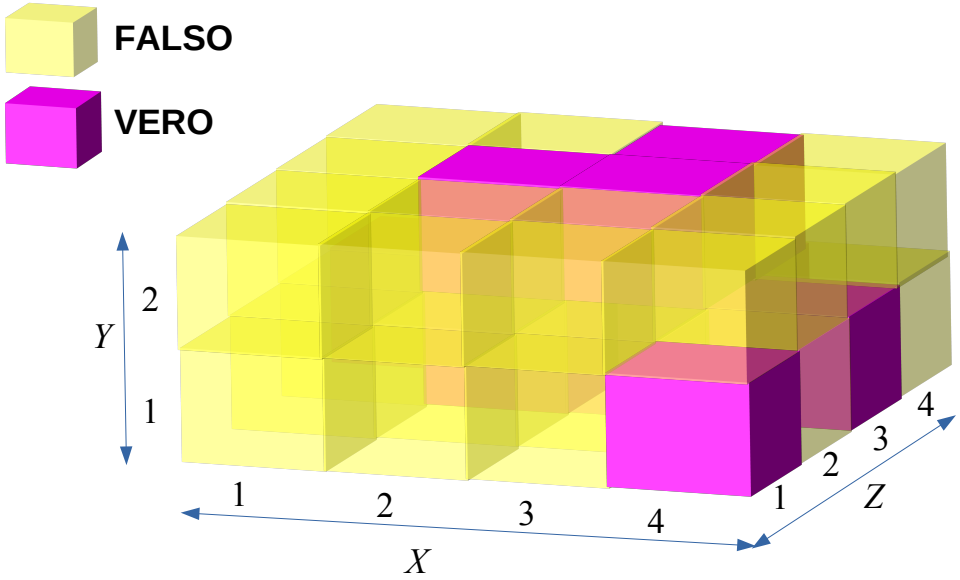
- Per vincoli *n*-ari, non c'è più l'interpretazione come grafo.
  Es: $X+Y=Z$
- *Generalized Arc Consistency (GAC)*
  (o Hyper Arc-Consistency o Domain-Consistency):
  Un vincolo $c(X_1,X_2,\ldots,X_n)$ è *arc consistent in senso generalizzato* se
  - Per ogni variabile X_i (i=1..n), per ogni valore $g \in dom(X_i)$
  - Esiste un assegnamento alle rimanenti n-1 variabili
    $$X_1 \rightarrow v_1, \ldots, X_{i-1} \rightarrow v_{i-1}, X_{i+1} \rightarrow v_{i+1}, \ldots, X_n \rightarrow v_n$$
  - *tale che* $c(v_1,\ldots,v_{i-1},g,v_{i+1},v_n)$ è vero (soddisfatto).

Domanda: Sapete definire Generalized Bound Consistency?

GAC: Graficamente

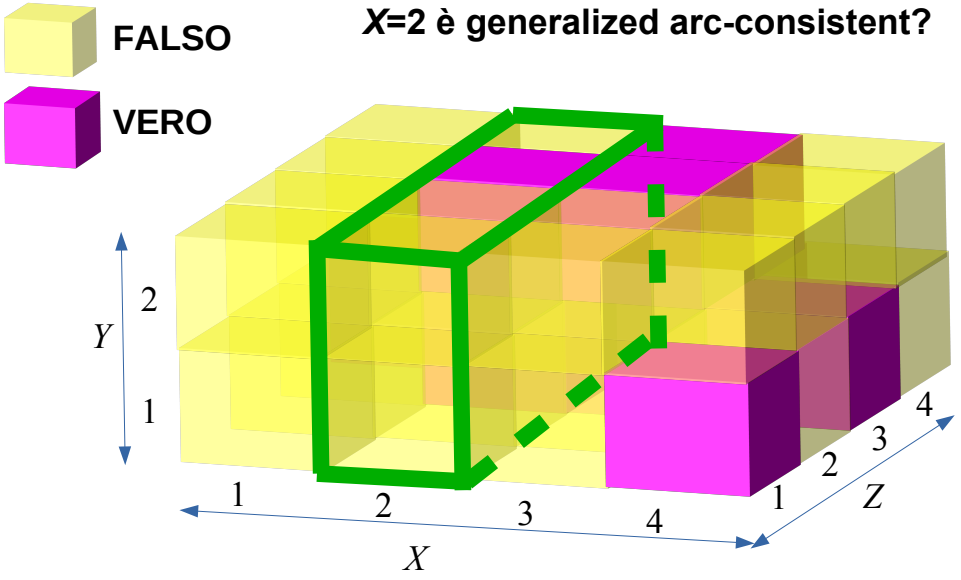


GAC: Graficamente

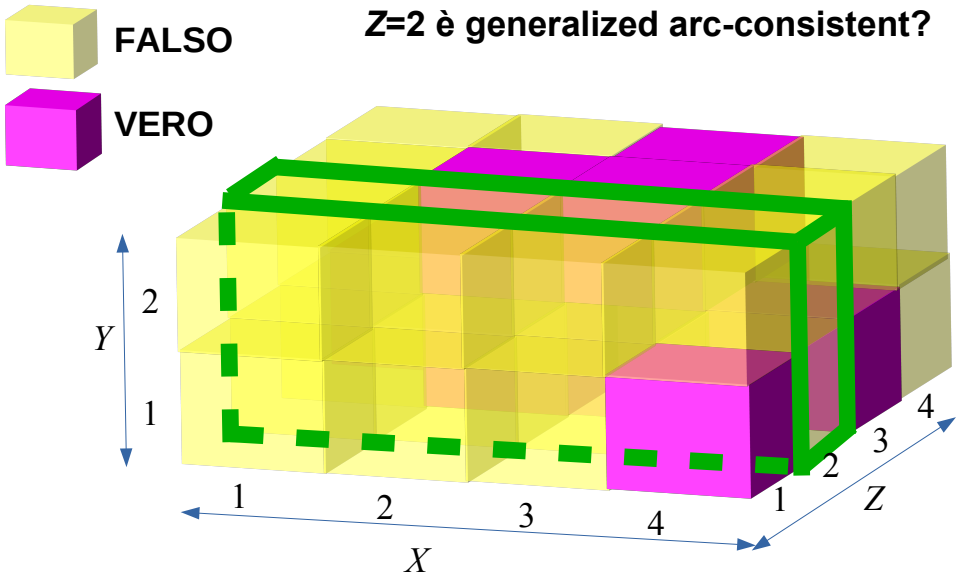


GAC: Graficamente

X=2 è generalized arc-consistent?



GAC: Graficamente



Vincoli N-ari: espressioni

- Si possono definire vincoli come

```
somma(A,B,C)
```

  vero se $A+B=C$, con propagazione GAC.
- Questi vincoli sono già definiti, con zucchero sintattico. Possiamo usare direttamente:

```
A+B #= C

A#< B*C+D,

...
```


Esercizio

- Si consideri il seguente CSP:

```
A::[-2..3], B :: [-1..4],
P::[2..7], A*B #= P.
```

- Si mostri la propagazione nei casi Generalized Arc-Consistency e Generalized Bound-Consistency.

PROPAGAZIONE DI VINCOLI

- Finora abbiamo visto propagazioni generali
- Vincoli Simbolici:
  - In generale, per ottenere la Generalized Arc Consistency sono noti solo algoritmi esponenziali nel numero di variabili
  - Per vincoli specifici, si può avere propagazione polinomiale
  - 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 o nella Ricerca Operativa
  - La propagazione termina quando la rete raggiunge uno stato di quiescenza; non si possono più cancellare valori
  - Propagazione incrementale

- 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])` $\leftrightarrow X_1 \neq X_2,\ X_1 \neq X_3, \ldots,\ X_{n-1} \neq X_n$

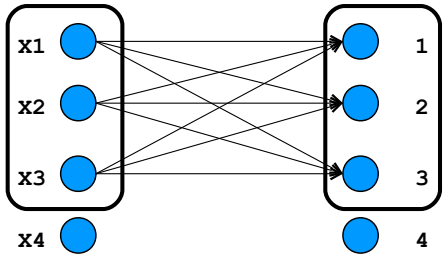
  Operazionalmente permette una propagazione più forte.

```
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

- Vincoli Simbolici: esempio 1

```
lib(fd_global).
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 cardinalità 3 che hanno medesimo dominio di cardinalità 3

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

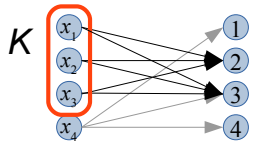
Hall's Marriage theorem (1935)

If a group of men and women marry only if they have been introduced to each other previously, then a complete set of marriages is possible if and only if every subset of men has collectively been introduced to at least as many women, and vice versa

Conseguenze teorema di Hall

- Quindi se trovo un sottoinsieme *K* delle variabili tale che

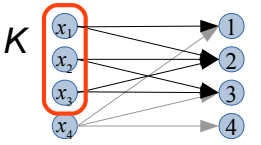
$$|K| > \left| \bigcup_{x_i \in K} D(x_i) \right|$$



Allora non c'è soluzione (fallisco)

- Se trovo un sottoinsieme *K* delle variabili tale che

$$|K| = \left| \bigcup_{x_i \in K} D(x_i) \right|$$



Allora i valori delle variabili che sono in *K* sono usati tutti dalle variabili in *K*, quindi posso rimuoverli dagli altri domini

Ricercare l'insieme K

- L'insieme *K* va cercato fra i sottoinsiemi dell'insieme delle variabili
- Con *n* variabili, quanti sono i sottoinsiemi?
- Se mi limito a Bound Consistency, posso considerare solo gli intervalli
  -
  -
  -
- Puget, usando un ordinamento intelligente dei domini, è riuscito in $O(n \log n)$

Esercizio

- Si consideri il vincolo **`alldifferent([A,B,C,D,E,F])`** che lavora sulle seguenti variabili

  `A::[1,3,5]`

  `B::[3,5,6]`

  `C::[1,2,4,6]`

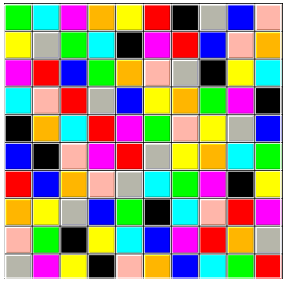
  `D::[3,5,6]`

  `E::[2,3,4,5]`

  `F::[3,5,6]`
- Si descriva la propagazione che il vincolo effettua sui domini delle variabili, supponendo che ottenga la generalized arc consistency

ALL-DIFFERENT

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



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

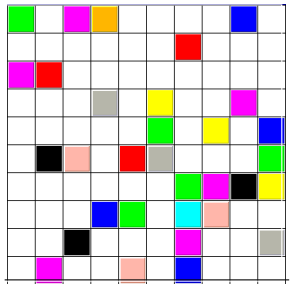
FACILE SE LA GRIGLIA E' VUOTA ma

non se PARZIALMENTE PIENA

35%-45% PERCENTUALE CRITICA

ALL-DIFFERENT NEL PARTIAL LATIN SQUARE

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



32% **preassignment**

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

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

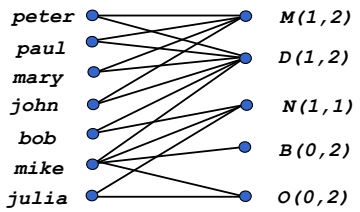
Nota sintattica

- In ECLiPSe il vincolo **`alldifferent`** è definito in diverse librerie, con implementazione diversa
  - **`fd:alldifferent`** impone $n(n\text{-}1)/2$ vincoli **`#\=`**
  - **`fd_global:alldifferent`** è il vincolo globale (in ECLiPSe fa bound consistency)
- Per distinguere due predicati con lo stesso nome definiti in librerie diverse (se entrambe le librerie sono state caricate), si usa la notazione

  `libreria:nome_predicato`
- Nelle versioni recenti di ECLiPSe è presente anche la libreria **`fd_global_gac`**, in cui **`alldifferent`** propaga con GAC, con un algoritmo $O(n^{2.5})$ [Règin '94]

GLOBAL CARDINALITY CONSTRAINT

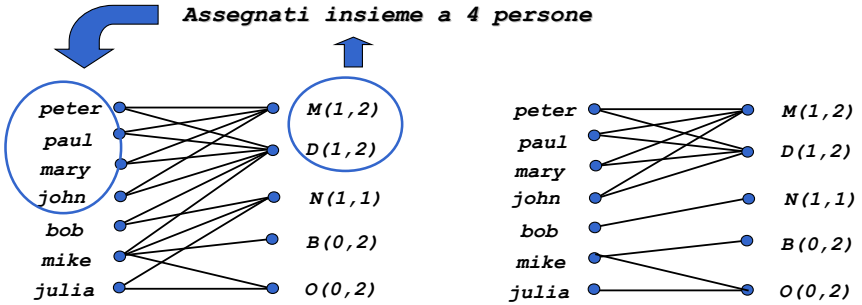
- Dei lavoratori (Peter, Paul, Mary, John, Bob, Mike e Julia) devono essere assegnati ai turni di una giornata.
- I turni sono:: ***M***orning, ***D***ay, ***N***ight, ***B***ackup, day-***O***ff
- Al turno ***M***orning possono essere assegnate 1 o 2 persone
- Al turno ***D***ay possono essere assegnate 1 o 2 persone



- Al turno ***N***ight deve essere assegnata 1 persona
- Al massimo 2 persone possono essere ***B***ackup
- Al massimo 2 persone possono essere ***O***ff

GLOBAL CARDINALITY CONSTRAINT

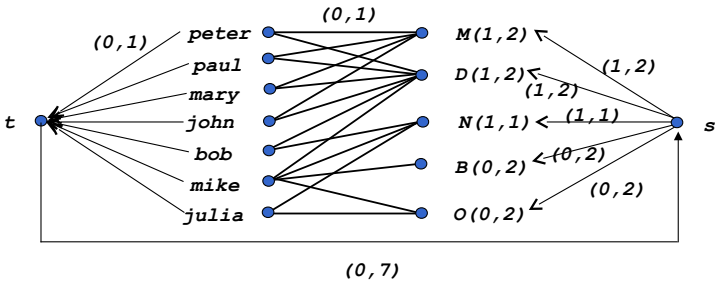
- **`gcc(Var,Val,LB,UB)`** *[Regin AAAI96]* **`Var`** sono variabili, **`Val`** valori **`LB`** e **`UB`** sono il minimo e massimo numero di occorrenze per ogni valore in **`Val`** assegnato a **`Var`**
- Esempio:



GLOBAL CARDINALITY CONSTRAINT

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

`gcc` su **`k`** variabili è consistente se
c'è un max flow da **`s`** a **`t`** di valore **`k`**



VINCOLO element

- Vincoli Simbolici: esempio
  - **element(N,[X_1,...X_m],Value)**
    l'**N**-esimo elemento della lista è uguale a Value
- propagazione da **N** a **Value** :
  - **N=i → X_i = Value**
- propagazione da **Value** a **N** e **X_i** :
  - **Value = x → N=1 and X_1=x or**
    N=2 and X_2=x or....
    N=m and X_m=x

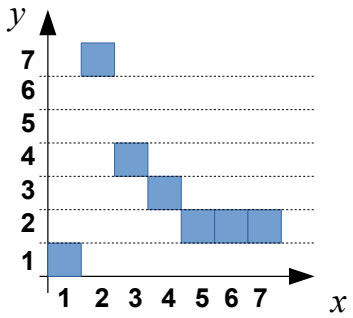
Esempio

- Si hanno 5 prodotti, identificati con i numeri da 1 a 5
- Ciascun prodotto ha un prezzo:
  - Prodotto 1: prezzo: 10
  - Prodotto 2: prezzo: 5
  - Prodotto 3: prezzo: 6
  - Prodotto 4: prezzo: 8
  - Prodotto 5: prezzo: 11
- Si desidera scegliere 2 prodotti il cui prezzo totale sia inferiore a 15
- **X:: 1..5, Y :: 1..5, X #\= Y,**
  element(X,[10,5,6,8,11],PrezzoX),
  element(Y,[10,5,6,8,11],PrezzoY),
  PrezzoX + PrezzoY #< 15.

Funzioni

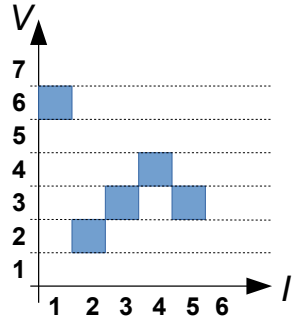
- Il vincolo **element** può essere usato per definire qualsiasi dipendenza funzionale di due variabili

element(X,[1,7,4,3,2,2,2],Y)



Vincolo element: propagazione

element(*I*,*L*,*V*)



- Se l'elemento $j \in D(I)$ allora l'elemento j-esimo di L (chiamiamolo L_j) non deve essere cancellato dal dominio di V
- Se l'elemento $k \in D(V)$ allora tutti gli indici j tali che $L_j = k$ non vanno eliminati dal dominio di I
- Tutti gli altri elementi (che non rispettano le regole precedenti) vanno eliminati

I::[1,3,4,5],V::[2,3,6],element(I,[6,2,3,4,3],V)

element(I,L,V): propagazione naive

Per tutti $j \in D(I)$

L_j è nel dominio di V?

Se no → elimina j da $D(I)$

Per tutti $k \in D(V)$

Trova i valori j tali che $L_j=k$

Se nessuno di questi è nel $D(I)$

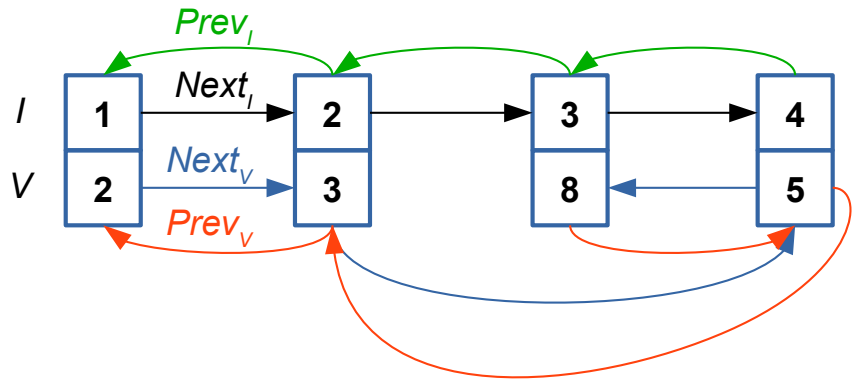
→ elimina k da $D(V)$

Complessità?

Supponiamo che $D(I)$, $D(V)$ e L abbiano n elementi

Element: propagazione

- Es: `element(I,[2,3,8,5],V)`



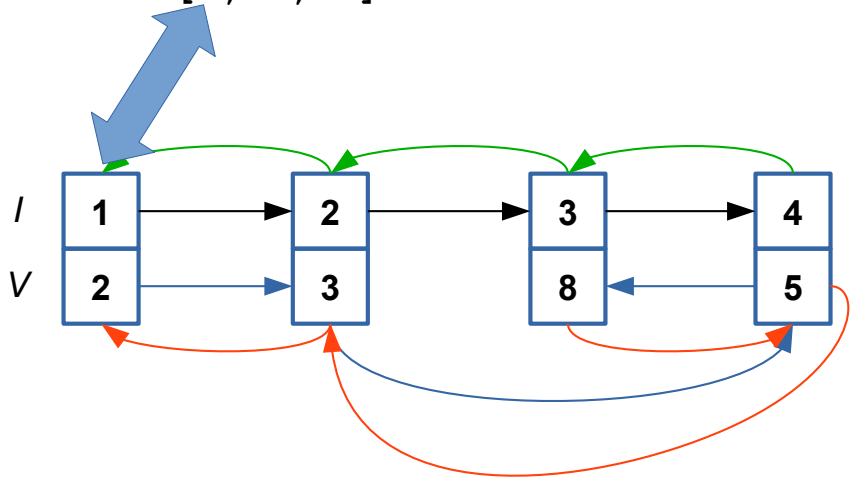
Element: propagazione

- Scorri contemporaneamente il dominio di I e la lista $Next_I$
  - Se c'è un valore nella lista e non nel dominio, elimina il valore dalla lista
- Lo stesso con $D(V)$ e la lista $Next_V$

Element: propagazione

- Es: `element(I,[2,3,8,5],V)`

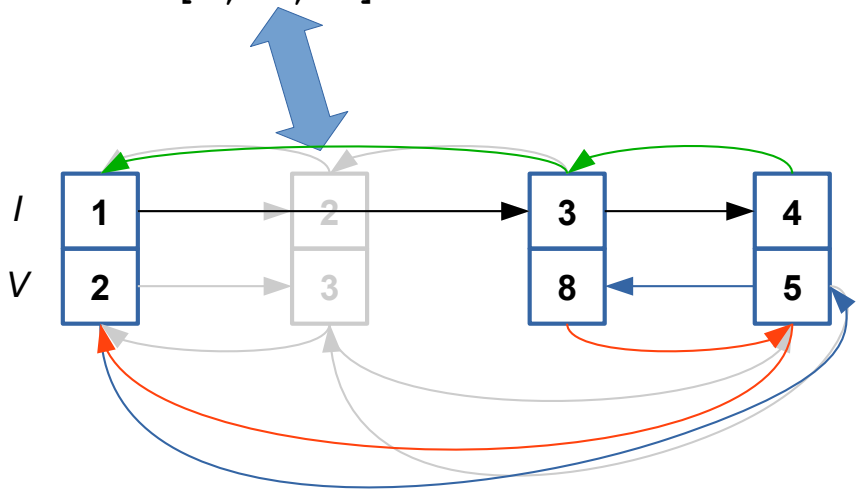
```
I :: [1, 3, 4]
```



Element: propagazione

- Es: `element(I,[2,3,8,5],V)`

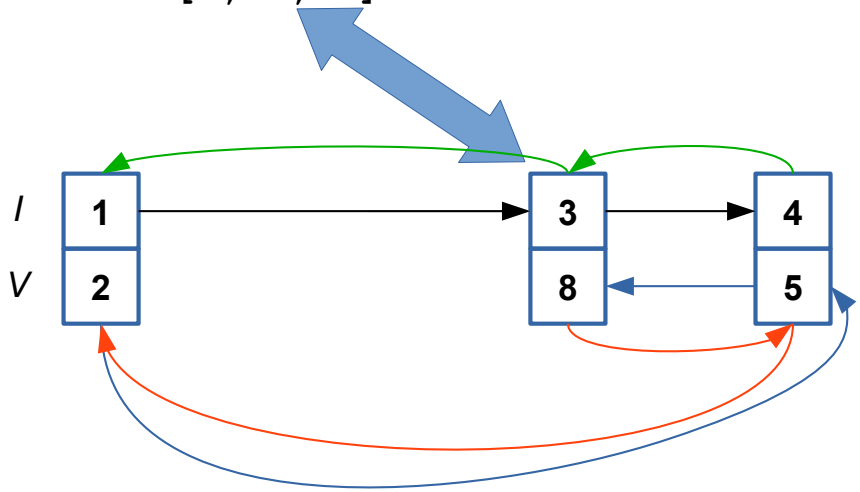
```
I :: [1, 3, 4]
```



Element: propagazione

- Es: `element(I,[2,3,8,5],V)`

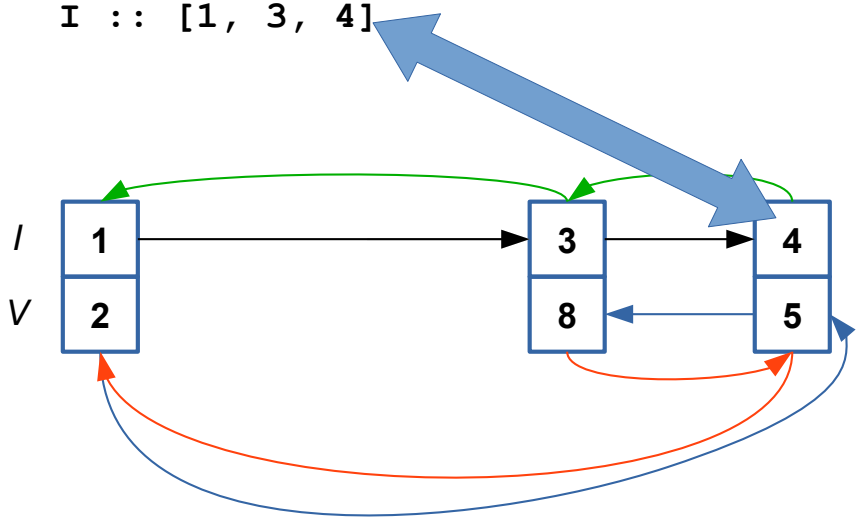
```
I :: [1, 3, 4]
```



Element: propagazione

- Es: `element(I,[2,3,8,5],V)`

```
I :: [1, 3, 4]
```

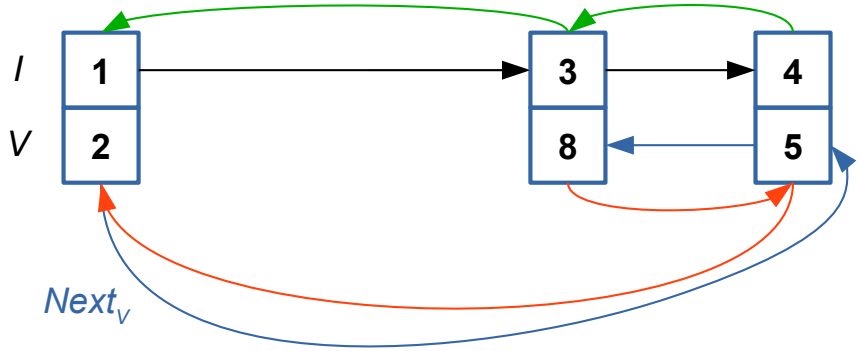


Element: propagazione

- Es: `element(I,[2,3,8,5],V)`

```
I :: [1, 3, 4]
```

A questo punto, seguendo la lista $Next_V$ si hanno in ordine i valori accettabili per V



Element: Complessità

Algoritmo	Complessità
• Scorri contemporaneamente $D(I)$ e la lista $Next_I$	n x
• Se c'è un valore nella lista e non nel dominio, elimina il valore dalla lista	1 +
• Scorri contemporaneamente $D(V)$ e $Next_V$	n x
• Se c'è un valore nella lista e non nel dominio, elimina il valore dalla lista	1 +
• Infine, scorrendo $Next_I$ si ha il dominio di I (in ordine)	n +
• Scorrendo $Next_V$ si ha il dominio di V (in ordine)	n =
	O(n)

Ciclo eseguito O(n) volte

O(1) con liste doppia-mente concate-nate

Esercizio: element

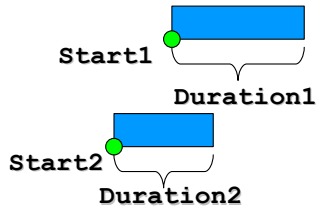
- Sia dato il seguente problema di soddisfacimento di vincoli:

```
I :: 2..5, V::1..5,
element(I,[1,4,2,3,6],V).
```

- Si mostri la propagazione effettuata.
- Cosa succede se si aggiunge il vincolo `I #> V` ?

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: `Start1` e `Start2` e la loro durata `Duration1` e `Duration2`.



  - Le due lezioni non possono sovrapporsi:

```
Start1 + Duration1 ≤ Start2
OR
Start2 + Duration2 ≤ Start1
```

  - Due problemi INDIPENDENTI uno per ogni parte della disgiunzione.

PROPAGAZIONE DI VINCOLI

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

DISGIUNZIONE COSTRUTTIVA

- ***P. Van Hentenryck, V. Saraswat, Y. Deville,***
  Design, Implementation and Evaluation of the Constraint Language cc(FD),
  Journal of Logic Programming, 1994.
- Sfrutta la disgiunzione per ridurre lo spazio di ricerca
- Idea: aggiungere al constraint store vincoli che sono implicati da tutte le parti della disgiunzione
- Esempio: `X::[5..10], Y::[7..11], Z::[1..20], (Z=X OR Z=Y)`
  - `Z=X` ridurrebbe il dominio di `Z` a `[5..10]`
  - `Z=Y` ridurrebbe il dominio di `Z` a `[7..11]`
  - risultato della disgiunzione costruttiva: `Z::[5..11]`
- In ECLiPSe non c'è, ma può essere implementato semplicemente (V. **`Propia`**)

OPERATORE DI CARDINALITÀ

- Symbolic Constraint: operatore di cardinalità
  - `#(l,[c1,…,cn],u)` ⇔ il numero `k` di vincoli c_i ($l \le i \le n$) soddisfatti non è minore di `l` e non maggiore di `u`
  - Con l'operatore di cardinalità modello i vincoli disgiuntivi nel modo seguente

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


Reified Constraints

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

`C` ⇔ `B`

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

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


Operatori per vincoli reificati

- Avendo i vincoli reificati, è possibile introdurre anche dei combinatori:
  - `#\+` *E* per not *E*
  - E_1 `#\/` E_2 per E_1 OR E_2
  - E_1 `#/\` E_2 per E_1 AND E_2
  - E_1 `#=>` E_2 per l'implicazione materiale $E_1 \rightarrow E_2$.
- Anche questi sono a loro volta vincoli reificati, quindi si possono scrivere espressioni!

```
X+Y #< Z #\/ (Z #< X #/\ Z #< Y)
```

- Può essere espanso in

```
X+Y #< Z #<=> B1,
  Z #< X #<=> B2,
  Z #< Y #<=> B3,
B2 #/\ B3 #<=> B4,
    B1 #\/ B4
```

Nota

- Le espressioni logiche (con gli operatori `#\/`, `#\+`, ecc.) possono essere create *solo* con vincoli reificati
- I vincoli reificati sono solo quelli "semplici": `#<`, `#\=`, `#=<`, ...
- Tipicamente NON sono reificati i vincoli globali
  - `alldifferent, gcc, element`, ...
- Quindi non posso scrivere

```
alldifferent([X,Y,Z]) #\/ element(X,[3,1,2],Y)
```

- In quanto dovrebbe essere espanso in

```
alldifferent([X,Y,Z],B1)
element(X,[3,1,2],Y,B2)
B1 #\/ B2
```

- Ma `alldifferent([X,Y,Z],B1)` e `element(X,[3,1,2],Y,B2)` non sono implementati!

VINCOLO CUMULATIVO

`cumulative([S1,...Sn], [D1,...Dn], [R1,...Rn], L)`

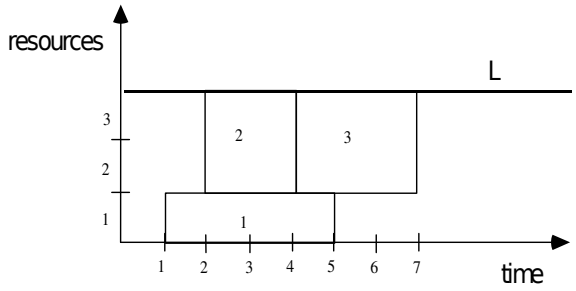
- `S1,...Sn` sono istanti di inizio di attività (variabili con dominio)
- `D1,...Dn` sono durate (variabili con dominio)
- `R1,...Rn` sono richieste di risorse (variabili con dominio)
- `L` limite di capacità delle risorse (fisso o variabile nel tempo)

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

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

VINCOLO CUMULATIVO

```
lib(edge_finder) o lib(edge_finder3).
cumulative([1,2,4],[4,2,3],[1,2,2],3)
```



Complessità di GAC di vincoli generali

Teorema [Bessière, Hebrard, Hnich, Walsh 2004]: Sia C un vincolo. Se esiste un algoritmo polinomiale per rendere GAC il vincolo C, allora esiste un algoritmo polinomiale per trovare una soluzione ad ogni CSP che contiene solo C.

Dimostrazione: Supponiamo che esista un algoritmo A che rende C Arc-Consistente in tempo polinomiale.
Allora si può trovare una soluzione al CSP che contiene C con questo algoritmo:

- Usa A per rendere GAC il vincolo. Siccome è GAC, il CSP ha soluzione sse tutti i domini sono non vuoti
- Ripeti finché ci sono variabili con almeno due valori nel dominio

  Scegli una variabile X con |dom(X)| > 1

  Siccome il vincolo è GAC, ogni valore nel dominio di X può essere esteso ad una soluzione. Assegna un valore ad X.

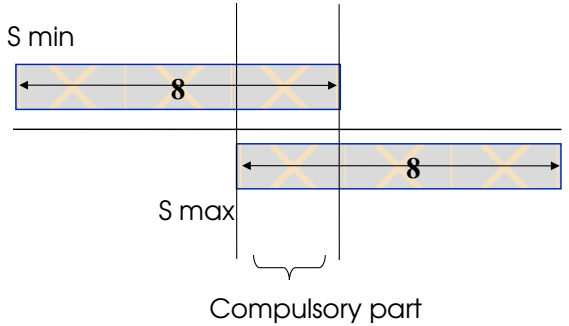
  Usa A per rendere GAC il vincolo

Complessità di GAC del `cumulative`

- Risolvere un problema in cui c'è solo il vincolo **`cumulative`** è NP-hard in generale (quindi non è noto alcun algoritmo di complessità meno che esponenziale)
- Quindi anche ottenere GAC è NP-hard
- Quindi esistono solo algoritmi esponenziali per calcolare GAC
- Ciò nonostante, ci possono essere algoritmi di filtering che cancellano alcuni valori inconsistenti (non tutti!) in tempo polinomiale

PROPAGAZIONE DI VINCOLI

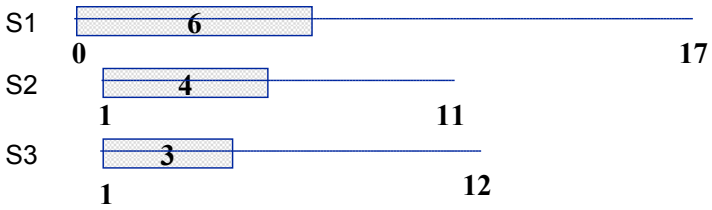
- Vincoli Simbolici: esempio 3
  - un esempio di propagazione usato nei vincoli sulle risorse è quello basato sulle *parti obbligatorie*



PROPAGAZIONE DI VINCOLI

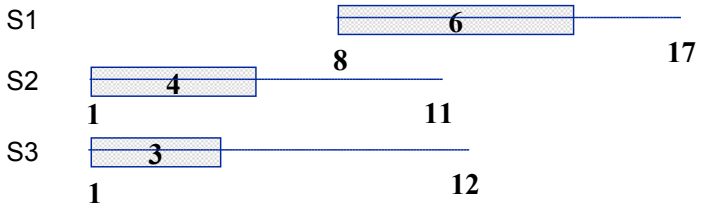
- Vincoli Simbolici: esempio 3
  - un'altra propagazione usata nel vincolo di capacità è quella basata sull' *edge finding* ***[Baptiste, Le Pape, Nuijten, IJCAI95]***

Consideriamo una risorsa unaria e tre attività



PROPAGAZIONE DI VINCOLI

- Vincoli Simbolici: esempio 3



Possiamo dedurre che minimo istante di inizio per S1 è 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 è almeno 13 (elemento non contenuto nel dominio di S2 e S3).

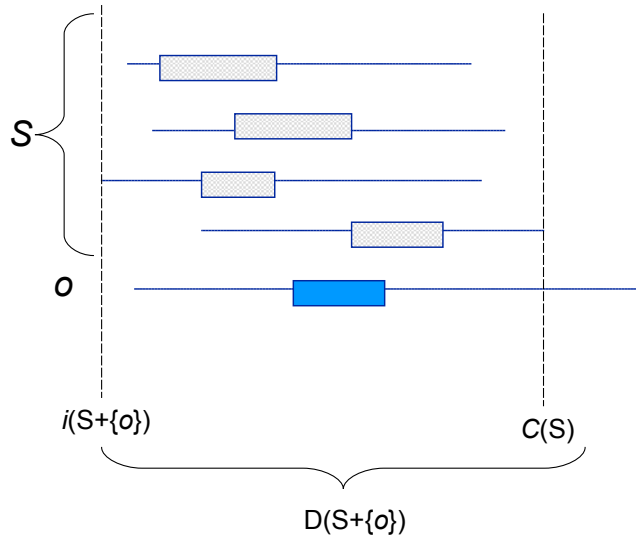
Teorema: [Carlier, Pinson, Man.Sci.95]

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

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

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

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



Esercizio

- Si considerino due attività che non si devono sovrapporre

Attività	Inizio	Durata
A1	Start1 :: [3..12]	7
A2	Start2 :: [1..9]	10

- Si mostri la propagazione effettuata dal filtering sulle parti obbligatorie

VINCOLI: CONSIDERAZIONI GENERALI

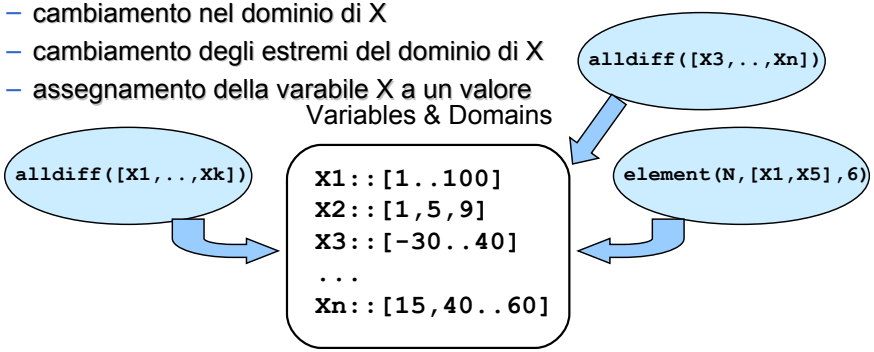
- Vincoli simbolici disponibili nella maggior parte dei linguaggi a vincoli
- Ragionamento locale vs. globale ⇒ propagazione potente
- Ragionamento locale vs. globale ⇒ costo computazionale

  } 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)

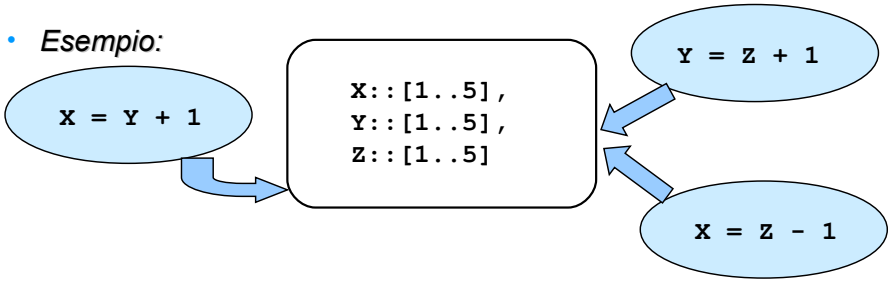
INTERAZIONE TRA VINCOLI

- 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 varabile X a un valore

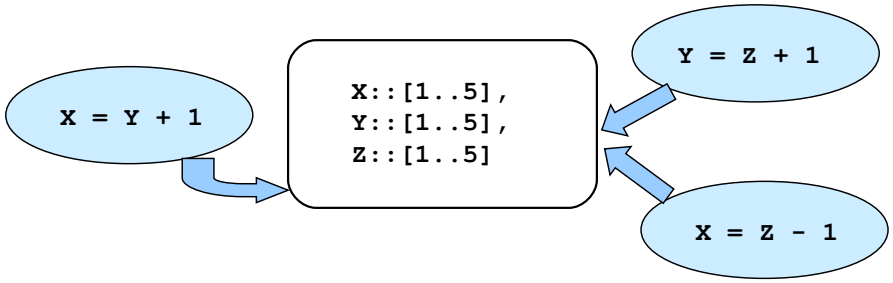


INTERAZIONE TRA VINCOLI

- In generale ogni variabile è coinvolta in molti vincoli. Di conseguenza, ogni cambiamento nel dominio della variabile come risultato di una propagazione può 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::[2..5]
Y::[1..4]
Z::[1..5]
```

`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 ***X = Z - 1*** porta a

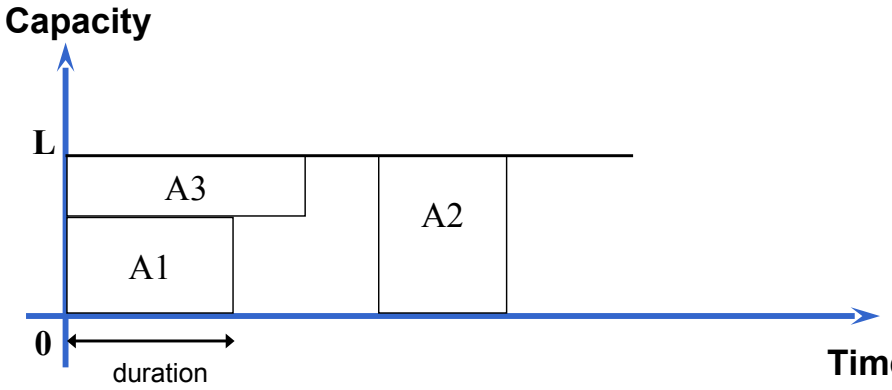
```
X::[]
Y::[2..4]
Z::[1..3]
```

FAIL

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

VINCOLI COME COMPONENTI SOFTWARE

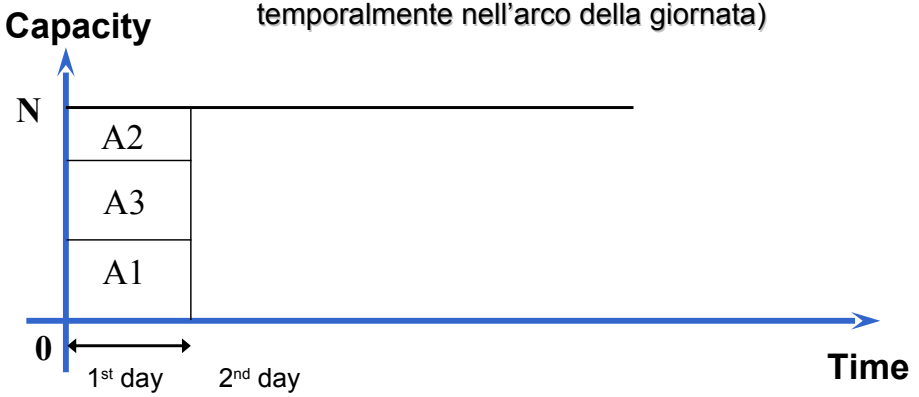
- I vincoli simbolici possono essere visti come componenti software utilizzabili in diverse situazioni. Ad esempio il vincolo cumulative può essere usato in vari modi
  - Scheduling (1): Attività A1, A2, A3 condividono la stessa risorsa con capacità limitata. Durate sull'asse X e uso delle risorse su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



VINCOLI COME COMPONENTI SOFTWARE

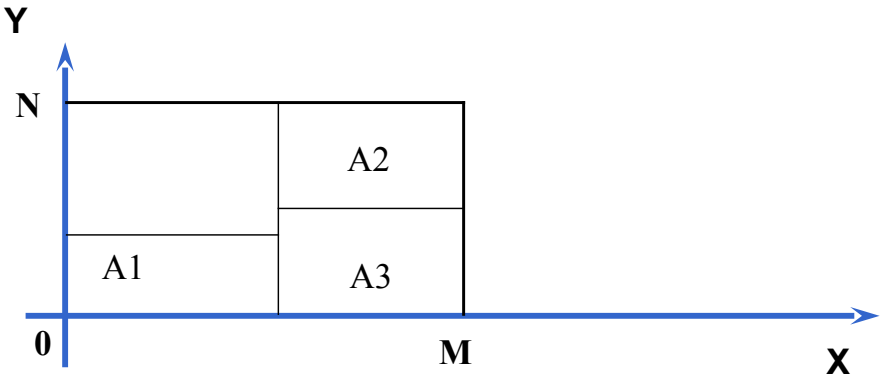
- Altro esempio di uso del vincolo cumulativo
  - Scheduling (2): *Numero limitato di risorse per giorno = N.* Rappresento i giorni sull'asse X e il numero di risorse usate su Y

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



VINCOLI COME COMPONENTI SOFTWARE

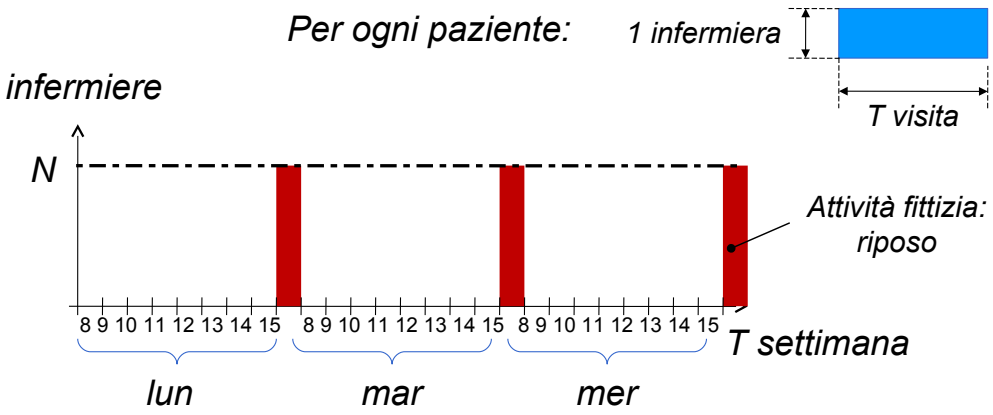
- Altro esempio di vincolo cumulativo
  - Packing: *Data una scatola di dimensioni M x N, è necessario collocare dei pezzi in modo che le dimensioni della scatola siano rispettate.*



Esempio

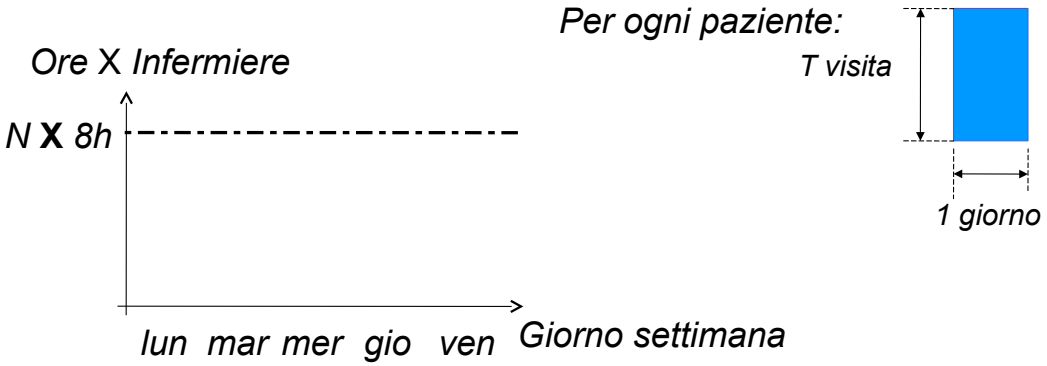
- *N* infermiere devono visitare *P* pazienti nell'arco di una settimana
- Per ogni paziente si sa quanto tempo è richiesto per la visita
  - In più, possono esserci vincoli sull'orario in cui viene visitato un paziente, alcuni pazienti vanno visitati più volte in una settimana, ecc.
- Ogni infermiera lavora 8 ore al giorno
- Trovare un assegnamento dei pazienti alle infermiere

Uso del vincolo cumulativo (I)



- Tempo = ore all'interno della settimana
- Risorse = infermiere
- Limite risorse = Numero infermiere

Uso del vincolo cumulativo (II)



- Tempo = giorno della settimana (da 1 a 7)
- Risorse = Tempo della visita
- Limite risorse = (Numero infermiere) **X** 8h

Alcuni vincoli utili

Vedere il manuale: librerie **`fd`** e **`fd_global`**

- atmost(+N, ?List, +V): At most N elements of the list List have the value V.
- element(?Index, +List, ?Value): Value is the Index'th element of the integer list List.
- alldifferent(+List,++Capacity): The list List contains at most Capacity elements of each value
- lexico_le(+List1, +List2): List1 is lexicographically less or equal to List2
- maxlist(+List, ?Max): Max is the maximum of the values in List
- minlist(+List, ?Min): Min is the minimum of the values in List
- occurrences(++Value, +List, ?N): The value Value occurs in List N times
- sorted(?List, ?Sorted): Sorted is a sorted permutation of List
- sumlist(+List, ?Sum): The sum of the list elements is Sum

Esercizio

Una colf deve svolgere i lavori di casa nell'arco di una mattinata. Le attività da svolgere sono:

- Lavare il bucato: ci vogliono 45 minuti, la lavatrice consuma 1,7kW.
- Asciugare il bucato: asciugatrice per 1 ora, 1kW.
- Stirare il bucato: 1 ora, il ferro da stiro consuma 2kW.
- Lavare i piatti: 40 minuti di lavastoviglie, la lavastoviglie consuma 1,8kW.
- Preparare una pizza: bisogna preparare l'impasto (15 minuti), lasciarlo lievitare da 1 a 2 ore, poi va cotta in forno per 15 minuti. Il forno consuma 2kW e va preriscaldato per 5 minuti (bisogna accenderlo 5 minuti prima di usarlo).
- Pulire la casa: 2 ore.

Calcolare la sequenza delle attività per far sì che la colf possa tornare a casa entro 3 ore e 20 minuti, sapendo che non si devono mai superare i 3kW di potenza consumata, altrimenti salta il contatore,

- gli elettrodomestici possono lavorare in autonomia senza intervento umano (a parte il ferro da stiro)
- e che, ovviamente, il bucato va prima lavato, poi asciugato e infine stirato.