OTTIMIZZAZIONE

- In molte applicazioni non siamo interessati a soluzioni ammissibili, ma alla soluzione ottima rispetto a un certo criterio.
- ENUMERAZIONE ⟹ inefficiente
  - trova tutte le soluzioni ammissibili
  - scegli la migliore
- I linguaggi di Programmazione a Vincoli in generale incapsulano una forma di Branch and Bound
  - ogni volta che viene trovata una soluzione di costo C^*, viene imposto un vincolo sulla parte rimanente dello spazio di ricerca. Il vincolo afferma che soluzioni successive (il cui costo è C) devono essere migliori della migliore soluzione trovata finora.

$$C < C^*$$

Ottimizzazione in ECLiPSe

`minimize(Goal,Cost)`

`min_max(Goal,Cost)`

- *invocano Goal; fra le varie soluzioni ne forniscono una per cui Cost è minimo*
- *Usano algoritmi diversi, entrambi basati su branch & bound*

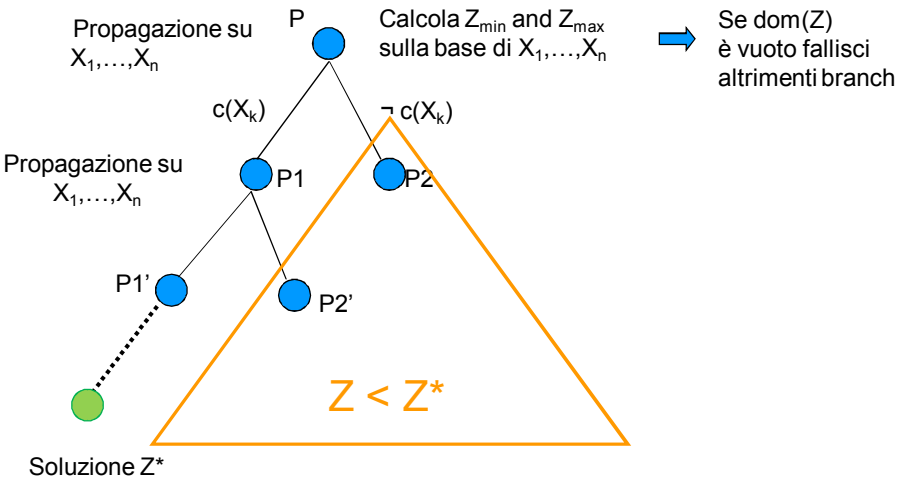
Ottimizzazione: Esempio (min_max)

```
[A,B]::0..5, A+B#<5,A*B#= -C, min_max(labeling([A,B]),C).
```

1. *Impone i vincoli e propaga:*
   A{[0..4]}, B{[0..4]}, C{[-16..0]}
2. *esegue labeling e trova una soluzione:*
   A=0, B=0, C=0
3. *backtracking: torna alla situazione al passo 1*
4. *impone nuovo vincolo* C #< 0 *e propaga:*
   A{[1..3]}, B{[1..3]}, C{[-9 .. -1]}
5. *esegue labeling e trova una soluzione:*
   A=1, B=1, C=-1
6. *backtracking: torna alla situazione al passo 1*
7. *impone nuovo vincolo C #< -1 e propaga:*
   A{[1..3]}, B{[1..3]}, C{[-9 .. -2]}

- *… etc.*
- *Quando non trova più soluzioni, fornisce l'ultimo risultato ottenuto come soluzione*

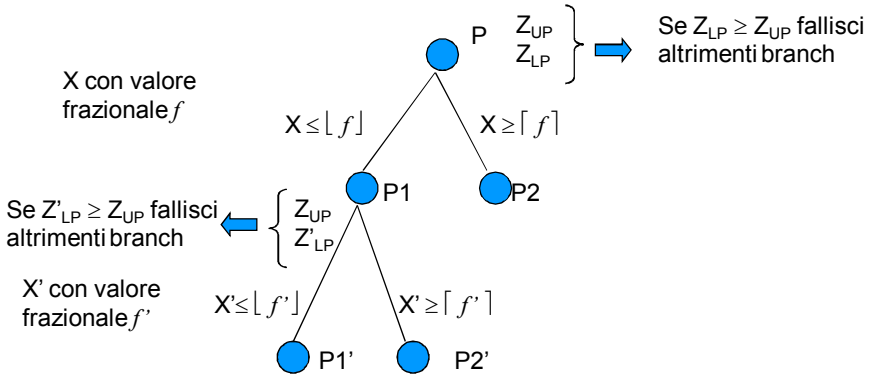
BRANCH & BOUND (minimize)



BRANCH AND BOUND

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

BRANCH & BOUND in PROGRAMMAZIONE LINEARE INTERA

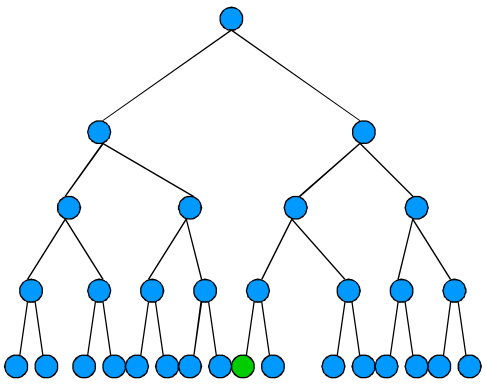


OTTIMIZZAZIONE

- La Ricerca Operativa ha una lunga tradizione nella soluzione di problemi di ottimizzazione
- I metodi Branch & Bound si basano sulla soluzione ottima di un rilassamento del problema che produce un BOUND, ossia una valutazione ottimistica della funzione obiettivo
  - relaxation: same problem with some constraints relaxed
- Linea di ricerca particolarmente promettente: integrazione di tecniche di Ricerca Operativa nella programmazione a vincoli al fine di migliorarne l'efficienza nella risoluzione di problemi di ottimizzazione

Altri algoritmi di Search

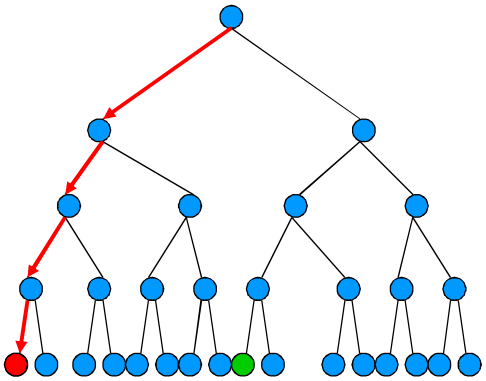
- *Esplorazione depth-1st: in profondità lungo il ramo di sinistra, poi torna indietro 1 passo, ecc.*
- *esplora le foglie di sinistra prima di quelle di destra.*
- *Supponiamo che l'euristica sia molto buona: in tutto il percorso compie solo 1 errore.*
  - *errore all'ultimo nodo =>* □
  - *errore al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



Soluzione

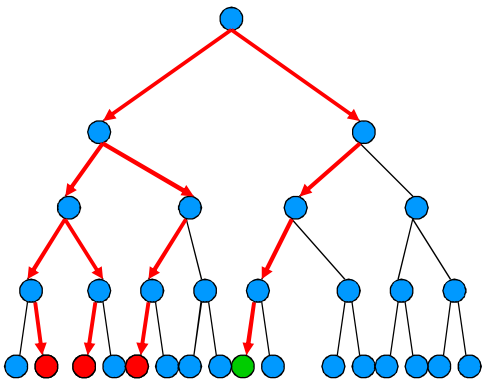
Limited Discrepancy Search

- *Discrepanza=0*
  seguo l'euristica



Limited Discrepancy Search

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



fd_search

- *Libreria* fd_search*:*

  search(L, Arg, Select, Choice, Method, Opt)

- *L: Lista di variabili.*
- *Arg = 0, Opt=[] (per i nostri usi, V manuale per approfondimenti)*
- *Select: euristica di selezione della variabile.*
  - *input_order, first_fail, smallest, largest, occurrence, most_constrained, max_regret, anti_first_fail*
- *Choice: euristica di selezione del valore*
  - *indomain, indomain_min, indomain_max, indomain_middle, indomain_median, indomain_split, indomain_random, indomain_interval*
- *Method: Algoritmo di search*
  - *complete, bbs(Steps), lds(Disc), credit(Credit, bbs(Steps) or lds(Disc)), dbs(Level, bbs(Steps) or lds(Disc)), sbds*

LDS in ECLiPSe

```
risolvi(L):-
   crea_domini(L),
   imponi_vincoli(L),
   search(L,0,most_constrained, indomain, lds(5),[]).
ottimizza(L):-
   crea_domini(L),
   imponi_vincoli(L),
   crea_obiettivo(L,F),
   minimize(search(L,0,most_constrained, indomain,
   lds(5),[]), F).
```

Domanda: è completo così? Come posso renderlo completo?

Problemi sovravincolati

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

Vincoli soft

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

    `Mat #> 10, Fisica #< 14, …`
  - *Con vincoli Soft:*

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

Ancora sui vincoli reificati

- *I vincoli reificati possono servire per collegare due o più vincoli:*
  - **`X#>Y #<=> B1, X#>Z #<=> B2, X+Y#>0`**
    - *Ci sono altri connettori* **`#\/, #/\, #=>`**, *…*
    - *Si può scrivere anche* **`X#>Y #\/ X#>Z`**
- *Poi posso rilassarli insieme:*
  - **`Elettr#<10 #\/ Elettr#>12 #<=> B1, sumlist([B1, …],Sum), F #= -Sum, minimize(labeling(…),Sum).`**

Creazione di nuovi vincoli

- *In certi casi può essere utile creare nuovi vincoli*
  - *se il linguaggio non è abbastanza espressivo*
  - *se ho trovato un algoritmo efficiente per implementare un vincolo globale*
- *Metodi per implementare nuovi vincoli*
  - *implementazione sospensioni*
  - *vincolo* **`element/3`**
  - *libreria* **`propia`**
  - *Constraint Handling Rules*

Sospensioni

- Vincoli gestiti internamente tramite il concetto di Sospensione. Una sospensione è un goal che può essere addormentato in attesa che avvenga un certo evento.
- Ricordate l'algoritmo AC3?
  - Due liste di vincoli: Lista dei vincoli attivi e lista di vincoli addormentati
  - Operazioni di: inserimento in lista, estrazione dalla lista, spostare vincoli da una lista all'altra (addormentare vincoli e riscegliare vincoli)
- ECLiPSe gestisce automaticamente le liste tramite uno scheduler.
- Noi dobbiamo solo addormentare un vincolo, in attesa di un evento. Quando l'evento si verifica, ECLiPSe mette il vincolo (goal) nel risolvente.
- Gli eventi nella libreria FD sono
  - min cancellazione del minimo nel dominio
  - max cancellazione del massimo
  - any cancellazione di un elemento qualsiasi del dominio
  - Ci sono altri eventi in altre librerie. Ad esempio, inst nella libreria suspend corrisponde all'istanziazione della variabile

AC3 (Mackworth)

La (List of active constraints) = lista di tutti i vincoli;

Ls (List of sleeping constraints) = ∅;

while La ≠ ∅ do

  prendi un vincolo c(X,Y) ∈ La e toglilo da La

  se ci sono elementi inconsistenti in dom(X)

    allora eliminali (se dom(X) = ∅, fallisci)

      metti in La tutti i vincoli in Ls che coinvolgono X

  (stesso ragionamento per Y)

  se c(X,Y) non è completamente risolto

    allora mettilo in Ls

Propagazione e sospensioni

- *per implementare un nuovo vincolo (a basso livello) si*
  - *definisce un predicato*
  - *il predicato elabora i domini delle variabili coinvolte*
  - *se il vincolo non è completamente risolto, si sospende*
- *In pratica,*
  - *ECLiPSe implementa l'algoritmo AC3, con la lista dei vincoli sospesi (addormentati).*
  - *Noi possiamo implementare nuovi vincoli implementando la parte di propagazione (eliminazione di valori inconsistenti).*

Primitive per gestire i domini

- *dvar_domain(X,D) fornisce il dominio (dato astratto) della variabile X*

  Es X::1..10, dvar_domain(X,D).

  yes, D=1..10
- *dom_to_list(D,L) trasforma il dominio in lista*

  Es X::[1..3, 100..102], dvar_domain(X, D),

  dom_to_list(D,L).

  yes, D = [1..3, 100..102]

  List = [1, 2, 3, 100, 101, 102]

  Passa da una rappresentazione compatta ad una estesa: spesso sconsigliato

Primitive per gestire i domini

- *dom_check_in(+Element, +Dom)*
  - *Verifica che l'intero Element sia nel dominio Dom.*
- *dom_compare(?Res, +Dom1, +Dom2)*
  - *Confronta due domini. Res viene unificato con*
    - *= sse Dom1 = Dom2,*
    - *< sse Dom1 ⊂ Dom2,*
    - *> sse Dom2 ⊂ Dom1.*
  - *Fallisce se nessuno è sottoinsieme dell'altro*
- *dom_member(?Element, +Dom)*
  - *istanzia nondeterministicamente Element ad uno dei valori in Dom*
- *dom_range(+Dom, ?Min, ?Max)*
  - *Fornisce il minimo ed il massimo del dominio*
- *dom_size(+Dom, ?Size)*
  - *Fornisce il numero di elementi nel dominio*

Primitive per gestire i domini

- *dvar_remove_element(+DVar, +El)*
  - *Elimina El dal dominio della variabile DVar*
- *dvar_remove_smaller(+DVar, +El)*
  - *Elimina dal dominio di DVar gli elementi < El*
- *dvar_remove_greater(+DVar, +El)*
  - *Elimina dal dominio di DVar gli elementi > El*

(meta)predicato Suspend

suspend(+Goal, +Prio, +CondList)

- *Sospende il Goal in attesa che si verifichi una delle condizioni nella CondList*
- *CondList è una lista che contiene elementi del tipo*

  Variabili -> libreria:evento
- *Prio è una priorità da 1 a 12: vengono risvegliati prima i vincoli con priorità bassa*
- *Es:* suspend(c(A,B),3,[A->fd:min,B->suspend:inst])

  sospende il goal c(A,B) e lo risveglia quando o viene eliminato il minimo nel dominio di A o viene istanziato B.

Esempio: implementazione di un vincolo

```
minimo(A,B,C):-
 dvar_domain(A,DomA),
 dom_range(DomA,MinA,MaxA),
 dvar_domain(B,DomB),
 dom_range(DomB,MinB,MaxB),
 min_int(MinA,MinB,MinMin),
 min_int(MaxA,MaxB,MaxMin),
 dvar_remove_smaller(C,MinMin),
 dvar_remove_greater(C,MaxMin),
 ( nonvar(A), nonvar(B), nonvar(C)
   ->true
   ; suspend(minimo(A,B,C),3,[A->fd:min,
        A->fd:max,B->fd:min,B->fd:max])
 ), wake.

min_int(A,B,B):- A>=B,!.
min_int(A,B,A):- A<B.
```

Estraggo i domini

Calcolo i nuovi domini

Elimino i val inconsistenti

Se vincolo risolto -> fine

Altrimenti sospendo

Risveglio altri vincoli

Esempio

- A::3..5,B::2..6, C::0..9, minimo(A,B,C).

  A = A{[3..5]}
  B = B{[2..6]}
  C = C{[2..5]}

  Delayed goals:
  minimo(A{[3..5]}, B{[2..6]}, C{[2..5]})

Miglioramenti?

- *Non faccio propagazione da C verso A e B*

  A :: 3..5, B :: 2..6, C :: 1..2, minimo(A, B, C).
  A = A{[3..5]},B = B{[2..6]},C = 2
  Delayed goals:
  minimo(A{[3..5]}, B{[2..6]}, 2)

- *Risveglio il vincolo troppo spesso*
  - *Se MinA<MinB non c'è bisogno di svegliarsi su B->fd:min*
- *Se MaxA<MinB, so già che A=C:*
  - *potrei evitare di sospendere il vincolo minore(A,B,C) ed imporre il vincolo A#=C.*

Domanda: che tipo di consistenza ottengo? GAC? GBC?

Vincolo element/3

- Un vincolo è una relazione, quindi può essere scritto come tabella in cui elenco le coppie consistenti e quelle inconsistenti

X	Y	c(X,Y)
0	0	✓
0	1	×
0	2	✓
1	0	✓
1	1	×
1	2	✓

Vincolo element/3

- Oppure con una tabella in cui elenco solo le consistenti
- In questo caso, il vincolo è soddisfatto solo se viene selezionata una riga

X	Y
0	0
0	2
1	0
1	2

```
c(X,Y):-
  element(I,[0,0,1,1],X),
  element(I,[0,2,0,2],Y).
```

PROPIA

- *Propia è una libreria per definire vincoli a partire da predicati (o trasformare predicati in vincoli)*
- *Considera le soluzioni possibili ed effettua la minima generalizzazione dei domini*

```
c(0,0).
c(0,2).
c(1,0).
c(1,2).
C(2,4).
```

```
?- c(X,Y) infers most.
X = X{[0..2]}
Y = Y{[0, 2, 4]}
Delayed goals:
     c(X{[0..2]}, Y{[0, 2, 4]}) infers most
```


PROPIA

- *Propia fornisce il metapredicato infers, che trasforma un predicato in vincolo. Dopo infers si possono usare varie parole chiave, che indicano il tipo di propagazione richiesto (AC è infers most o infers fd)*
- *Per effettuare la generalizzazione, utilizza il dominio delle variabili di un certo solver.*
  - *Si possono usare diversi solver (FD, Herbrand, …)*
  - *Bisogna aver caricato il solver corrispondente prima*

```
lib(fd).
lib(propia).
```

PROPIA

- *Si può usare anche con predicati non ground o ricorsivi*

```
no_overlap(Start1,Dur1,Start2,Dur2):-
   Start1 #>= Start2+Dur2.
no_overlap(Start1,Dur1,Start2,Dur2):-
   Start2 #>= Start1+Dur1.

?- S::1..10, no_overlap(S,2,5,3) infers fd.
    S = S{[1..3, 8..10]}
    yes
```

Simile a disgiunzione costruttiva.

Compito 13 set 2007

- *Un commesso viaggiatore deve passare per un insieme di città e poi tornare alla città iniziale, percorrendo meno chilometri possibile.*
- *Per ogni coppia di città collegate da una strada, è riportata la distanza in un insieme di fatti dista/3*

```
dista(bologna,ferrara,50).
dista(ferrara,bologna,50).
dista(bologna,ravenna,84).
...
```

Modello semplice

- *Lista di variabili:*
  - *La prima var è la prima città visitata*
  - *La seconda var è la seconda città visitata*
  - *…*
- *Domini: le varie città:*
  - **`[A,B,C,D…] :: [ferrara,ravenna,bologna…]`**
- *Vincoli: ci deve essere un arco fra una città e la successiva. Il vincolo è proprio il predicato* **`dista`***: devo trasformare il predicato in vincolo*
- *Funzione obiettivo: min somma delle distanze*

Consideriamo questo problema:

```
?- X :: 1..10000000,
   Y :: 1..10000000, X#>Y, Y#>X.
```

Soluzione?

- *Noi ci accorgiamo immediatamente del fallimento, perché 'vediamo' i vincoli dall'esterno, non dall'interno*
- *Sappiamo che quando ci sono alcune combinazioni di vincoli, non ci possono essere soluzioni*

Ordinamenti

- *In matematica, i vincoli* **<**, ≤, **>**, *… sono associati agli ordinamenti*
- *Un ordinamento (parziale) è una relazione binaria che gode delle proprietà:*
  - *Riflessività:* $a \leq a$.
  - *Antisimmetria:* $a \leq b, b \leq a \rightarrow a=b$
  - *Transitività:* $a \leq b, b \leq c \rightarrow a \leq c$.
- *Quindi il vincolo in matematica è definito in base alle sue proprietà. Possiamo definire anche noi un vincolo basandoci sulle proprietà?*

Constraint Handling Rules (CHR) 驰

- *Un modo per definire la propagazione di nuovi vincoli.*
- *I vincoli sono definiti tramite delle regole, che possono essere di tre tipi: simplification, propagation e simpagation (caso particolare di simplification)*

Propagation rules

- *Una Propagation rule ha la sintassi:*

  `c1, c2, … ==> guardia | body.`

  e significa che se i vincoli **`c1, c2,`** *... sono nel constraint store e la* **`guardia`** *è vera, allora anche il* **`body`** *deve essere vero.*
- *La* **`guardia`** *è opzionale.*
- *Nella testa (antecedente) possono comparire solo vincoli. Questi sono i nuovi vincoli che vogliamo definire (non possono comparire vincoli predefiniti, come #<, alldifferent, …)*
- *Operazionalmente, se i vincoli* **`c1, c2,`** *... sono nello store, verifica se la* **`guardia`** *è vera, poi esegue il* **`body`**.

Simplification rules

- *Una Simplification rule ha la sintassi:*

  `c1, c2, ... <=> guardia | body.`

  e significa che se la **`guardia`** *è vera, allora i vincoli* **`c1, c2, ...`** *sono equivalenti al* **`body`**.
- *Nella testa (antecedente) possono comparire solo vincoli. Questi sono i nuovi vincoli che vogliamo definire (non possono comparire vincoli predefiniti, come #<, alldifferent, …)*
- *Operazionalmente, se i vincoli* **`c1, c2, ...`** *sono nello store, verifica se la* **`guardia`** *è vera, poi toglie dal constraint store* **`c1, c2, ...`** *ed esegue il* **`body`**.

Esempio: vincolo leq (less or equal)

```
reflexivity@   leq(X,X) <=> true.
antisymmetry@  leq(X,Y), leq(Y,X) <=> X=Y.
transitivity@  leq(X,Y), leq(Y,Z) ==> leq(X,Z).
```

```
leq(A,B), leq(B,C), leq(C,A)

leq(A,B), leq(B,C), leq(C,A), leq(A,C)

leq(A,B), leq(B,A), A=C

A=B, A=C
```

Note

- *CHR non utilizza l'unificazione, ma il pattern-matching (fa l'unificazione solo in una direzione). Ad es,* `leq(A,B)` *non attiva la regola*

```
leq(X,X) <=> true.
```

(la testa della regola deve essere più specifica)

- *La guardia deve essere un semplice test (ad es, var, ground, …) non deve unificare variabili che compaiono nella testa. Se si effettuano unificazioni, la guardia fallisce. Ad es:*

```
leq(X,Y) <=> X=Y|true.
```

è equivalente a `leq(X,X) <=> true.`

- *Non posso usare nella testa delle regole dei vincoli predefiniti, ma solo vincoli definiti con CHR. Se voglio, posso scrivere delle regole che "trasformano" un vincolo CHR in un altro vincolo, tipo:*

```
leq(X,Y) ==> X#=<Y
```

oppure

```
leq(X,Y) <=> nonvar(X), nonvar(Y) | X =< Y.
```


CHR in ECLiPSe

- *Per usare CHR, si deve caricare la libreria chr, oppure la nuova implementazione ech (V. sul manuale le differenze).*

CLP(R) in ECLiPSe

- *ECLiPSe ha una libreria per l'uso di CLP(R), chiamata* **`eplex`**
- *Utilizzo di un risolutore esterno (Cplex, Xpress-MP oppure solver free), basato sull'algoritmo del simplesso*
- *Creazione di un'istanza del solver, in cui si stabilisce una funzione obiettivo*

```
eplex:eplex_solver_setup(min(X))
eplex:eplex_solver_setup(max(X))
```

- *Definizione di domini:*

```
$::  oppure  eplex:(L :: Dom)
```

- *Vincoli lineari:*

```
$>=, $=<, $=
```

- *risoluzione*

```
eplex_solve(Cost)
```


Esempio

```
:- lib(eplex).
lp_example(X,Y,Cost) :-
     eplex_solver_setup(min(X)),
     [X,Y] $:: 0..10,
     X+Y $>= 3,
     X-Y $= 0,
     eplex_solve(Cost).
```

- *Risultato:*

```
:- lp_example(X,Y,C).
X = X{0 .. 10 @ 1.5}
Y = Y{0 .. 10 @ 1.5}
C = 1.5
```

Soluzione ottima

Valore della soluzione ottima

Integrazione dei due solver

- *In molte applicazioni, fare cooperare i due solver porta a soluzioni più velocemente di ciascuno dei due*
- *Nell'esempio precedente, la propagazione Arc-Consistency non restringe i domini:*

```
[X,Y] #:: 0..10,X+Y #>= 3,X-Y #= 0.
X = X{[0..10]}
Y = X{[0..10]}
Delayed goals:  X{[0..10]} + X#>=3
```

- *eppure il valore ottimo del rilassamento lineare è X=1.5 (quindi non ci sono soluzioni con X=0, X=1)*
- *Posso calcolare il valore ottimo del rilassamento lineare e imporre X #>= C.*
- *E` anche possibile creare un vincolo che esegue il simplesso*

Insiemi

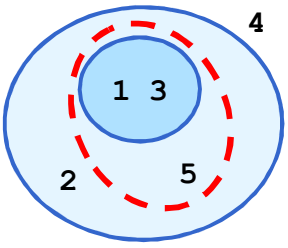
- *In alcune applicazioni e` utile usare gli insiemi*
- *ECLiPSe ha una libreria* **`fd_sets`**
- *Per definire una variabile di tipo set, si usa la sintassi*

```
SetVar :: LowerBound..UpperBound
```

- *Dove* **`LowerBound`** *e* **`UpperBound`** *sono due liste di interi*
- *Ad esempio*

```
S :: [1,3]..[1,2,3,5]
```

- *significa che la variabile* **`S`**
  - *contiene sicuramente gli elementi 1 e 3*
  - *può contenere anche gli elementi 2 e 5*
  - *(ma non può contenere, ad esempio, il valore 4)*



Vincoli sugli insiemi

`X in S` ($X \in S$)

- `S::[]..[1,2,3,4,5,6], 3 in S`

  `yes, S = S{[3] \/ ([] .. [1, 2, 4, 5, 6])`

`X notin S` ($X \notin S$)

- `… 5 notin S`

  `S = S{[3] \/ ([] .. [1, 2, 4, 6])`

Vincoli sugli insiemi

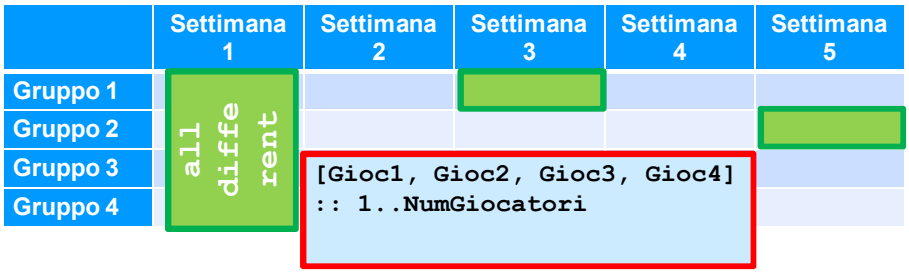
- `intersection(S1,S2,S3)` ⟺ $S1 \cap S2 = S3$
- `union(S1,S2,S3)` ⟺ $S1 \cup S2 = S3$
- `S1 disjoint S2` ⟺ $S1 \cap S2 = \varnothing$
- `S1 subset S2` ⟺ $S1 \subseteq S2$
- `difference(S1,S2,S3)` ⟺ $S1 \setminus S2 = S3$
- `#(S,C)` ⟺ $|S| = C$

cardinalità dell'insieme. Si noti che `C` *è una variabile con dominio finito, su cui posso imporre vincoli della libreria* `fd`.

Esempio: Social Golfer

- *32 giocatori di golf ogni settimana giocano in gruppi di 4 persone, per 16 settimane*
- *Vogliono far si` che ogni settimana ogni giocatore sia in un gruppo in cui non ha mai incontrato nessuno*

Social Golfer in CLP(FD)

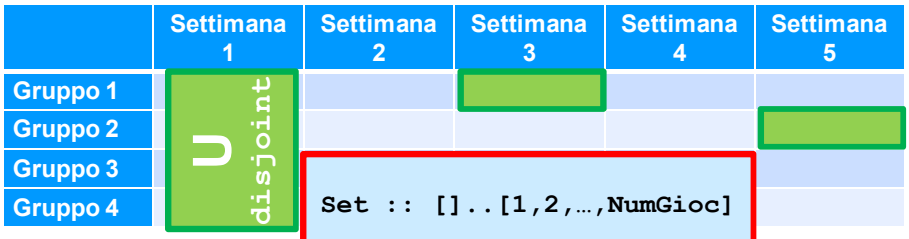


Vincoli:

- *Per ogni colonna:* `alldifferent(Lgiocatori)`
- *Per ogni coppia di celle in settimane diverse:*

  se due giocatori erano insieme nella prima settimana, allora non possono essere insieme nella seconda

Social Golfer con Set



Vincoli:

- *In ogni cella:* `#(Set,4)`
- *Per ogni colonna:* $\bigcup_i Set_i = [1,2,3\ldots,NumGioc]$
  - *Per ogni coppia di celle nella colonna:* `Set1 disjoint Set2`
- *Per ogni coppia di celle in settimane diverse:*

```
intersection(Set1,Set2,Int), #(Int,Card), Card::0..1
```


Vincolo weight

```
weight(?Set, ++Weights, ?Tot)
```

- *dove*
  - `Set` *è una variabile di tipo set*
  - `Weights` *è un termine ground, sintassi* `w(ElencoValori)`
  - `Tot` *è una variabile con dominio finito*
- *Impone che il peso totale degli elementi nell'insieme* `Set` *sia pari a* `Tot`
- *Esempio:*

1 2 3 4 5 6

```
S::[2]..[2,3,5],weight(S,w(1,4,3,7,5,2),T)
Yes, T:: 4..12
```

lista <->termine

- *Il vincolo* **`weight`** *richiede un termine* **`w(ElencoPesi)`**
- *Se ho i pesi in una lista, come li trasformo in un termine?*
- *Per trasformare liste in termini e viceversa, si può usare l'operatore*

```
Termine =.. Lista
```

- *La* **`Lista`** *conterrà il funtore come primo elemento, poi a seguire gli argomenti*
- *Es:*

```
p(a,2,X) =.. L.
yes, L = [p, a, 2, X]
```

- *Es:*

```
L =.. [w,1,3,4].
yes, T = w(1,3,4)
```


Search con gli insiemi

`insetdomain(Set,CardSel,ElemSel,Order)`

- *dove*
  - **`Set`** *è una variabile* **`fd_set`**
  - *Gli altri parametri decidono l'ordine in cui vengono provati gli assegnamenti. Se si vuole lasciare il default, lasciarli variabili.*
- *Es:* **`S::[]..[1,2,3], insetdomain(S,_,_,_).`**

```
yes, S=[1,2,3]      ;
yes, S=[1,2]        ;
yes, S=[1,3]        ;
yes, S=[1]          ;
yes, S=[2,3]
```

CONSTRAINT PROGRAMMING TOOLS

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

CONSTRAINT PROGRAMMING TOOLS

	Declarative	*Object-Oriented*
Open source	ECLiPSe ***[Wallace et al.97]*** ***[Smolka 91]*** mozart ***[Hermenegildo et al. 94]*** GNU Prolog	Gecode Choco
Commercial	***CHIP*** SICStus BinProlog	ILOG An IBM Company ***Comet*** dynadec

… and many more

AI APPLICATIONS: MODELLING and SOLVING

- Ci focalizzeremo su CLP(FD)
- Per ogni applicazione, mostriamo la descrizione del problema, il modello CLP e la risoluzione.

- Applicazioni discusse:
  - *Scheduling - Timetabling - Resource Allocation*
  - *Routing*
  - *Packing - Cutting*

  } optimization

  - *Graphics - Vision*
  - *Planning*

  } feasibility

SCHEDULING - TIMETABLING - RESOURCE ALLOCATION

- Three applications with analogous features/constraints:
  - we will focus on scheduling. Same considerations for timetabling and resource allocation (easier problem)

- Scheduling is probably one of the most successful applications of CP to date
  - flexibility
  - generality
  - easy code

- NP-complete problem studied by the AI community since 80s

SCHEDULING: problem definition

- Scheduling concerns the assignment of limited resources (machines, money, personnel) to activities (project phases, manufacturing, lessons) over time
- Constraints
  - temporal restrictions
    - ordering among activities
    - due dates - release dates
  - resource capacity
    - different types of resources
    - consumable/renewable resources
- Optimisation Criteria
    - makespan
    - resource balance
    - lateness on due dates
    - resource assignment cost

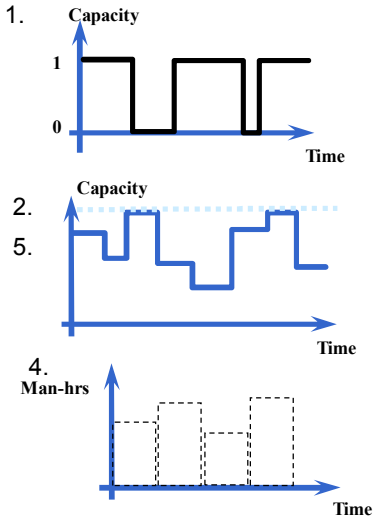
SCHEDULING: Activities

- Decision variables:
  - Activity start times
  - Activity end times
  - Activity resource assignments
  - Alternative activities (alternative routing)

- Activity types:
  - interval activity: cannot be interrupted
  - breakable activity: can be interrupted by breaks
  - preemptable activity: can be interrupted by other activities

SCHEDULING: Resources

- Resource types:
  - 1. Unary resources
  - 2. Discrete resources
  - 3. State resources
  - 4. Energy resources
  - 5. Stock



SCHEDULING: Simple Example

- 6 activities: each activity described by a predicate

```
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).
```

- Machines are unary resources.
- Minimising the maximum ending time End is required.

SCHEDULING: Simple Example

```
schedule(Data, End, TaskList):-
    makeTaskVariables(Data,End,TaskList),
    precedence(Data, TaskList),
    machines(Data, TaskList),
    minimize(labeling(TaskList),End).
makeTaskVariables([],_,[]).
makeTaskVariables([task(N,D,_,_)|T],End,[Start|Var]):-
    Start #<= End-D, Start #>=0,
    makeTaskVariables(T,End,Var).
precedence([],[]).
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: Simple Example

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

```
makeTaskVariables([],_,[]).
makeTaskVariables([task(N,D,_,_)|T],End,[Start|Var]):-
    Start #<= End-D, Start #>=0,
    makeTaskVariables(T,End,Var).
```

```
End :: 8..10000
```

```
[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)]
```

```
[Start1 :: 0..+10000,
 Start2 :: 0..+10000,
 Start3 :: 0..+10000,
 Start4 :: 0..+10000,
 Start5 :: 0..+10000,
 Start6 :: 0..+10000]
```

SCHEDULING: Simple Example

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

```
precedence([],[]).
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: Simple Example

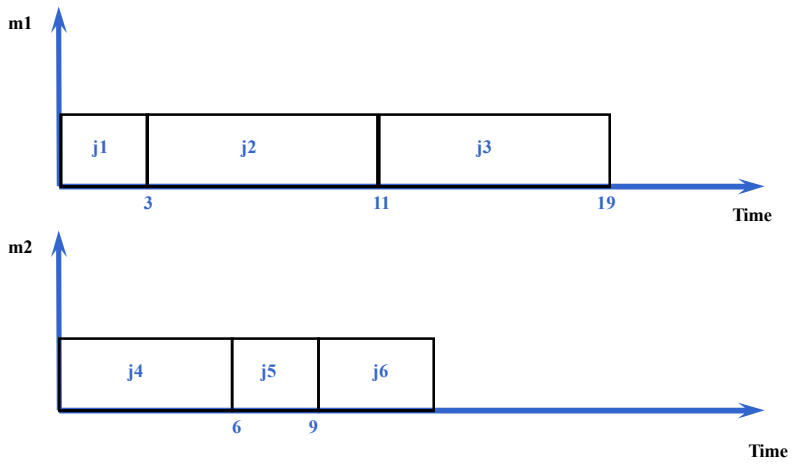
```
machines(Data, TaskList):-
 tasks_sharing_resource(Data,TaskList,SameResource,Durations),
 impose_cumulative(SameResource,Durations,Use).

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

results in

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

SCHEDULING: Optimal Solution



SCHEDULING: Optimality

- **`minimize`**: finds the optimal solution (simple Branch & Bound)
- Minimization of the makespan: a heuristic which always selects the task which can be assigned first and assigns to the task the minimal bound is in general a good heuristics. As a choice point, delay the task.

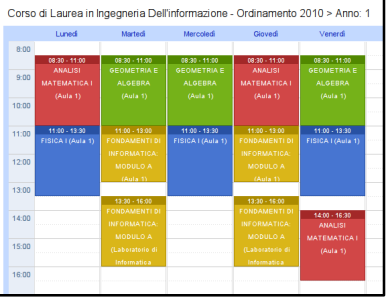
```
labelTasks([]).
labelTasks(TaskList):-
    find_min_start(TaskList,First,MinStart,Others),
    label_earliest(TaskList,First,MinStart,Others).

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: problem definition

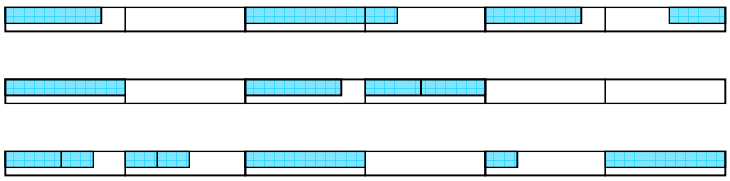
- Timetabling concerns the definitions of agenda (similar to scheduling)

- Constraints
  - temporal restrictions
    - ordering among activities
    - due dates - release dates
  - resource capacity
    - discrete resources
- Optimization Criteria
  - cost/preferences
  - resource balance



TIMETABLING: simple example

- 4-Hours Slots - 1 to 4 Hours Courses
- Two courses cannot overlap
- A course must be contained in a single slot
- Preferences are associated with
  - Course-Slot assignments
  - Maximize Sum of preferences



TIMETABLING: code with redundant constraints

```
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).
```

redundant constraints

- Redundant variables:
  - start times
  - single hours
  - courses lasting 3 or 4 hours

linked each other:

exchange propagation results

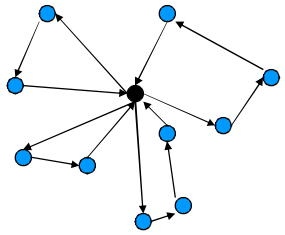
TIMETABLING: optimality & search

- Search strategy: when coping with objective functions, we can exploit information on costs for defining a good search strategy.
- Example:
  - Choose the variable with max value of regret
    - Regret: difference between the first and the second best on each row of the cost matrix.
    - Combination of regret and first-fail
  - Choose the value associated with the minimum cost
- Example: based on the optimal solution of a relaxation
  - Choose the variable with max value of regret
  - Choose the solution of the relaxation

ROUTING: problem definition

- Routing concerns the problem of finding a set of routes to be covered by a set of vehicles visiting a set of cities/customers once starting and ending at one (n) depot(s).

- Constraints
  - temporal restrictions:
    - time windows
    - maximal duration of a travel
  - vehicle capacity
  - customer demands
- Optimization Criteria
  - number of vehicles
  - travel cost
  - travel time

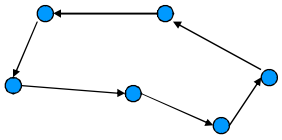


ROUTING: problem definition

- Routing has been solved within OR community by using
  - Branch & Bound approaches
  - Dynamic Programming
  - Local Search techniques
  - Branch & Cut
  - Column generation
- Routing has been solved within CP community by using
  - Branch & Bound approaches
  - Local Search techniques embedded in CP
- Basic component: Travelling Salesman Problem (TSP) and its time constrained variant.

TSP: problem definition

- TSP is the problem of finding a minimum cost tour covering a set of nodes once.



- No subtours are allowed
- TSPTW: Time windows are associated to each node. Early arrival is allowed. Late arrival is not permitted
- Even finding an Hamiltonian Circuit (no costs) is NP-complete (if the graph is not complete)

TSP: CP model

- Variable associated to each node. The domain of each variable contains possible next nodes to be visited
- N nodes ⟹ N + 1 variables $\mathtt{Next}_i$ (duplicate the depot)
- For all `i`: $\mathtt{Next}_i \neq \mathtt{i}$
- `nocycle(`$[\mathtt{Next}_0, \ldots \mathtt{Next}_n]$`)`
- `alldifferent(`$[\mathtt{Next}_0, \ldots \mathtt{Next}_n]$`)`
- Cost c_{ij} if $\mathtt{Next}_i = \mathtt{j}$
- In some models, we can find the redundant variables `Prev` indicating a node predecessor.

TSP: code

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

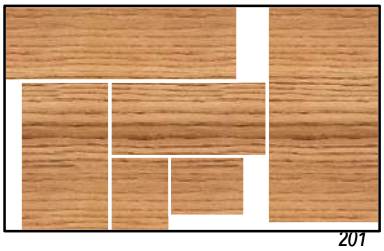
- `nocycle`: symbolic constraint that ensures that no subtour is present in the solution.
- `create_objective`: creates `Costs` variables, imposes an `element` constraint between the set of `Next` variables and `Costs` variables, and creates a variable `z` representing the objective function summing costs corresponding to assignments

TSP: results

- Pure CP implementations: still far from the state of the art OR approaches.
- Integration of OR techniques in CP: better results
  - local search
  - optimal solution of relaxations
    - Lagrangean relaxation
    - Assignment Problem
    - Minimum Spanning Arborescence
  - search strategies based on these relaxations
    - subtour elimination
- Addition of Time Windows in OR approaches requires to re-write major code parts while in CP comes for free.

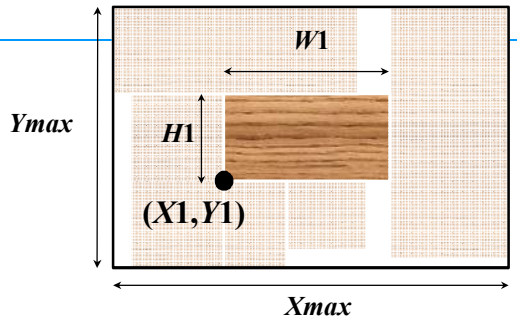
CUTTING & PACKING: problem definition

- Packing concerns the placement of a number of squares (of different sizes) in one or more larger boxes in such a way that squares do not overlap and minimizing the empty space
- Cutting is the problem of finding cuts of a master piece in order to obtain a given number of pieces with fixed dimensions, minimizing residues
  - Many variants:
    - strip packing
    - guillottine cuts
    - rotations allowed
    - 1 dimension - 2 dimensions - 3 dimensions - 4 dimensions



Packing

- *For each rectangle, 2 variables:*
  - *X position*
  - *Y position*
- *Domains:*
  - *X variables: from 0 to Xmax-W*
  - *Y variables: from 0 to Ymax-H*
- *Constraints:*
  - *Given 2 rectangles (X1,Y2) (X2,Y2) then:*
    - *either Rectangle 1 is left of Rectangle 2: X1 + W1 ≤ X2*
    - *or Rectangle 1 is right of Rectangle 2: X2+W2 ≤ X1*
    - *or Rectangle 1 is under Rectangle 2: Y1+H1 ≤ Y2*
    - *or Rectangle 1 is over Rectangle 2: Y2+H2 ≤ Y1*



2-D PACKING: CP model

- Constraints:
  - non overlapping constraints: given two squares whose coordinates are (`X1`, `Y1`) and (`X2`, `Y2`) and dimensions `D1`, `H1` and `D2`, `H2` respectively

- `X1+W1 ≤ X2 OR Y1+H1 ≤ Y2 OR X2+W2 ≤ X1 OR Y2+H2 ≤ Y1`

- Very hard form of disjunction: no propagation even after instantiation

- Redundant constraints can help:
- `cumulative(Xcoordinates,XDimension,Ydimension,H)`
  `cumulative(Ycoordinates,YDimension,Xdimension,D)`

PACKING: code

```
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`: creates a variable representing the spare space (or the number of bins if more than one is present)

- `label_squares` selects bigger squares first and assigns the coordinates in order to minimize spare space.

MODEL BASED VISION
VISUAL SEARCH: definition

- Visual Search in model based vision concerns the problem of recognizing an object in a scene given its model

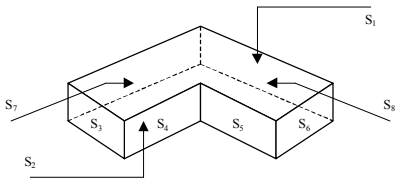
- How to describe the model
- How to perform the mapping

} *Both problems can be solved with constraints*

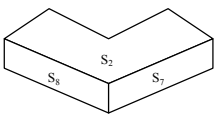
- MODEL: Constraint graph
  - Nodes: object parts
  - Arcs (constraints): their relations
- RECOGNITION: Constraint satisfaction

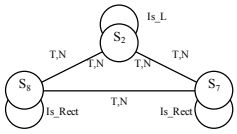
OBJECT RECOGNITION

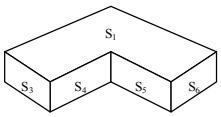
- 3-D OBJECT

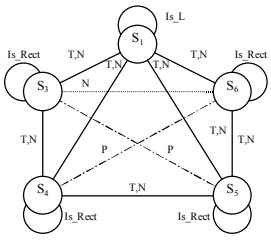


- Two views and the corresponding models









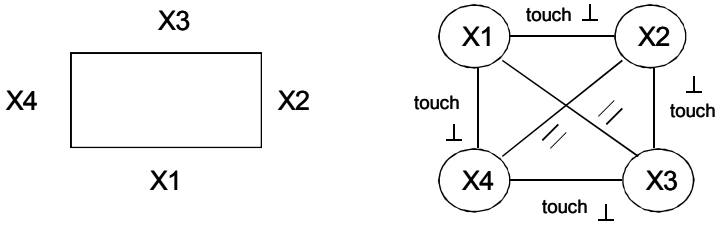
CONSTRAINT-BASED OBJECT RECOGNITION

- In order to recognize an object, a low level vision system should extract from the image some visual features (surfaces/edges)
- Constraint satisfaction techniques can be applied in order to recognize the object in the scene.
- The object is recognized if the extracted features satisfy the constraints contained in the model.
- Constraints allow to reduce the search space to be explored.

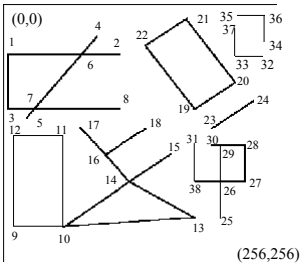
EXAMPLE

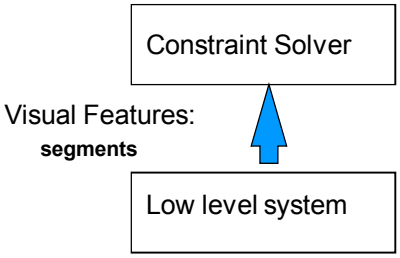
- Rectangle shape

  Four straight edges (variables), parallel two by two, which mutually touch themselves with a 90 degree angle ...



EXAMPLE (2)





rectangle CP model

```
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)
```


CP model: USER DEFINED CONSTRAINTS

rectangle CP model

```
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]).
```

- Give the declarative and operational semantics of the constraints: segments are described as facts: `segment(name,X1,Y1,X2,Y2)`
- In all CP languages there are tools that allow new constraints to be defined.
- An example in the CLP(FD) library of ECL^{i}PS^e

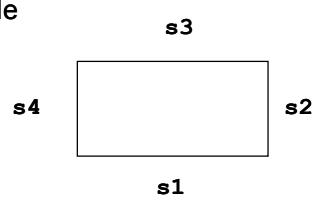
CP model: USER DEFINED CONSTRAINTS

```
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)
        ->    suspend(touch(X1,X2),3,[X1,X2]->fd:any)
        ;     true),
      wake.
```

- After the propagation, the constraint if not solved is suspended and awaked each time an event ***any of fd*** on one of the variables **(X1,X2)** happens.

CP model: SYMMETRIES

- Problem symmetries: arise when some permutations of the variables map a solution onto another solution.
- Four segments forming a rectangle
- One solution:
  - **X1 = s1**
  - **X2 = s2**
  - **X3 = s3**
  - **X4 = s4**



- Other identical solutions:
  - **X1 = s2 X1 = s3 X1 = s4**
  - **X2 = s3 X2 = s4 X2 = s1**
  - **X3 = s4 X3 = s1 X3 = s2**
  - **X4 = s1 X4 = s2 X4 = s3**

Time lost to look for already found solutions. Remove symmetries by imposing additional constraints

[Freuder AAAI91], [Puget ISMIS93], [Crawford et al. KR96], [Meseguer, Torras IJCAI99].

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 $P_1, \ldots P_n$ 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 $P_1 < P_2$, $P_2 < P_3$ $\ldots . P_{n-1} < P_n$ 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

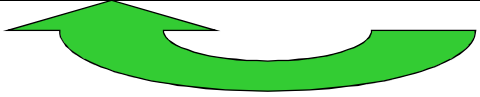
- *Le settimane sono simmetriche*
- *I Game sono simmetrici*

	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

- *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 game trovo una soluzione equivalente*

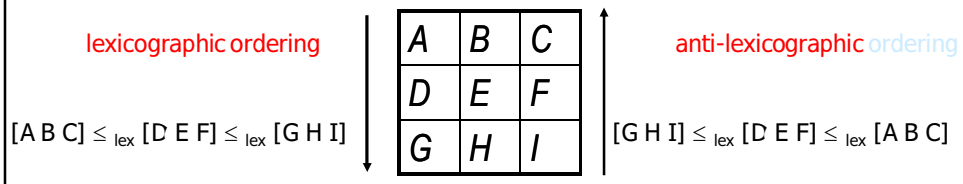
	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Game 1	0 vs 1	3 vs 7	4 vs 7	3 vs 6	0 vs 2	1 vs 5	2 vs 4
Game 2	2 vs	1 vs 4	0 vs 3	5 vs 7	1 vs 7	0 vs 6	5 vs 6
Game 3	4 vs 5	2 vs 6	1 vs 6	0 vs 4	3 vs 5	2 vs 7	0 vs 7
Game 4	6 vs	0 vs 5	2 vs 5	1 vs 2	4 vs 6	3 vs 4	1 vs 3

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 è 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
- In ECLiPSe: vincolo **`lexico_le`**



SIMMETRIE DI RIGA E COLONNA

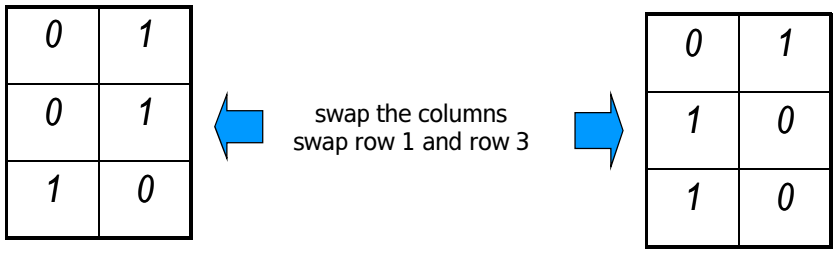
- Riusciamo a eliminare le simmetrie di riga e colonna singolarmente, ma se compaiono insieme, imporre gli ordinamenti su righe e colonne non elimina tutte le simmetrie.

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)*

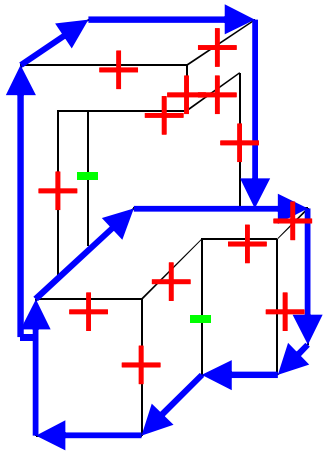
Bad news

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



MODEL BASED VISION (2)
OBJECT RECOGNITION

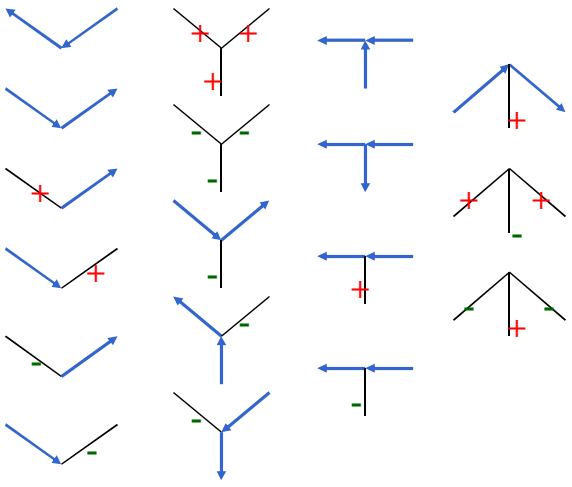
- *Viceversa, può essere richiesto di dare un significato ad un oggetto qualunque.*
- *Es, spiegare gli spigoli di una figura 3D*
  - *quali sono concavi - o convessi +*
  - *quali delimitano una figura*



OBJECT RECOGNITION

Waltz notò che le combinazioni sono in numero limitato

- *Variabili: segmenti nella figura*
- *Domini*: {+,-,←,→}
- *Vincoli: combinazioni possibili*



AI PLANNING: definition

- Planning concerns the problem of finding a chain of actions that achieve a given goal starting from a well known initial state. Actions are described with a set of preconditions (requirements) and postconditions (effects)

- Constraints
  - Plan constraints
    - temporal constraints (ordering)
    - designation and co-designation constraints
    - resource constraints
    - domain dependent constraints
  - Plan construction constraints
    - threats
    - open condition achievement

CONSTRAINTS ON THE PLAN

- Temporal constraints
  - qualitative ⟹ action A before action B
  - quantitative ⟹ action A in [10..30]

- Resource constraints
  - consumable resources
  - shared resources
  - renewable resources

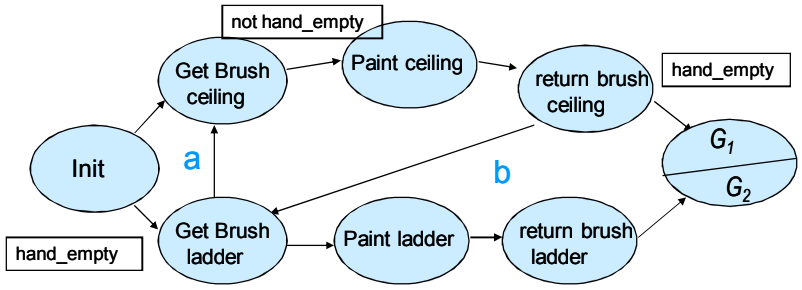
- Domain-dependent constraints
  - priority among resources
  - destructive actions on object A always after any other action on A

CONSTRAINTS ON PLAN CONSTRUCTION

- During the plan construction a set of decisions should be taken:
  - threat resolution
  - open condition achievement
- Good technique: least commitment. Decisions are delayed as much as possible in order to perform only consistent choices.
- Passive postponement vs. active postponement
- Each decision is represented by a variable whose domain contains possible solution to the pending decision. Constraints take into account interaction among decisions.

THREAT RESOLUTION

- Threat: conflict occurring when the effects of an action A threat the preconditions of an action B



Robot with one hand: threat

a promotion: getBrushLadder < getBrushCeiling

b demotion: returnBrushCeiling < getBrushLadder

THREAT RESOLUTION

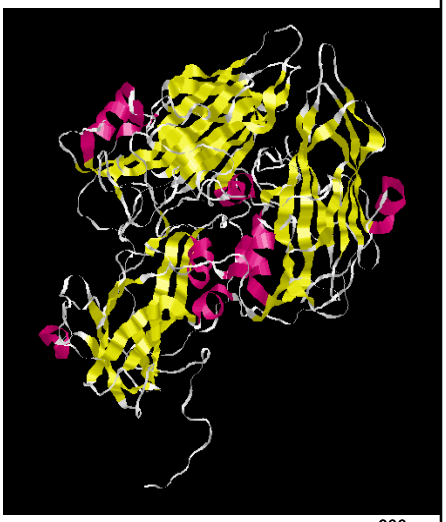
- Variable associated with the threat:
  - T1 :: [*getBrushLadder* < *getBrushCeiling*, *returnBrushCeiling* < *getBrushLadder*]
  - Threat variables linked by:
    - *incompatibility constraints*
    - *subsumption constraints*
  - Example: if a threat T2 can be solved only by the constraint *getBrushCeiling* < *getBrushLadder*, the only way of solving T1 becomes *returnBrushCeiling* < *getBrushLadder*
- Propagation of consequences of the decisions. Reduction of the search space and trashing avoided
- More complex solver: user defined constraints.

PLANNERS based on CP/CSPs

- ParcPlan ***[Lever, Richards, ISMIS94]***
- PlaNet ***[Barruffi, Milano ECAI98], [Barruffi et. Al. ECP99]***
- O-Plan ***[Tate, Drabble, Dalton, TR Univ. Edinburg 95]***
- Molgen ***[Stefik AIJ(16), 81]***
- DEVISER ***[Vere IJCAI85]***
- Descartes ***[Joslin PhD, 95], [Joslin, Pollac, EWSP96]***
- WatPlan ***[Yang, AIJ(58), 92]***
- SLNP ***[McAllester, Rosenblitt Nat.Conf AI 91]***
- FSNLP ***[Yang, Chan AIPS94]***
- Kambampati ***[TR Arizona State Univ. 96]***
- SIPE ***[Wilkins AI (22) 84]***

PROTEIN FOLDING

- *Una proteina può essere vista come una sequenza di amminoacidi*
- *Coppie di amminoacidi si attraggono o respingono*
- *La proteina si avvolge in una forma con energia minima*
- *Per trovare il modo in cui si avvolge una proteina, si devono svolgere analisi chimiche*



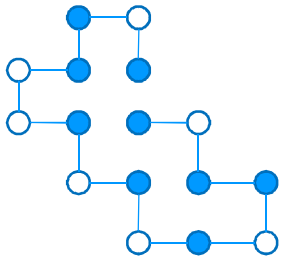
PROTEIN FOLDING

- *Le posizioni in cui un amminoacido si può trovare sono discrete*
- *Se due amminoacidi sono vicini, si attraggono o respingono a seconda dei valori di una tabella data*

	CYS	MET	PHE	ILE	LEU
CYS	-3.477	-2.240	-2.424	-2.410	-2.343
MET	-2.240	-1.901	-2.304	-2.286	-2.208
PHE	-2.424	-2.304	-2.467	-2.530	-2.491
ILE	-2.410	-2.286	-2.530	-2.691	-2.647
LEU	-2.343	-2.208	-2.491	-2.647	-2.501
…					

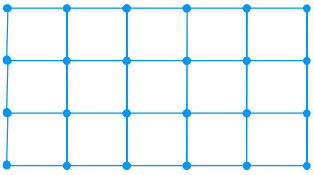
Simplified Example 2D

- Protein folds in the plane, possible positions for aminoacids are those with integer coordinates
- Aminoacids are of 2 types:
  - Type 0: does not attract, nor repulse,
  - Type 1: attracts aminoacids of type 1, but only if distance=1
- Example: protein
  [1,0,1,1,0,0,1,0,1,0,1,0,1,1,0,1]



CP Model

- Variables:
  - One variable for each aminoacid
- Domains:
  - Positions in the plane
- Constraints:
  - If two aminoacids are connected
    The second must be immediately up, down, left or right of the first
  - No two aminoacids can be in the same position
- Objective
  - Maximize number of neighboring aminoacids of type 1



Esempio in ECLiPSe

- *Consideriamo il caso 2D: gli amminoacidi si possono disporre in una griglia*
- *2 tipi di amminoacidi: 1 e 0. Gli 0 sono inerti, gli 1 si attirano, se sono in posizioni vicine*
- *Prendiamo in ingresso la lista che descrive la proteina (es. [1,0,0,1,0,1,1,0,1]).*

- *Forniamo in uscita una lista di uguale lunghezza che descrive la posizione di ogni amminoacido nello spazio*
  - *Ogni amminoacido ha una X ed una Y (intere)*
  - *Si può riassumere in un'unico valore K=Y*P+X (se P=Xmax+1)*

jP	*jP*+1		(*j*+1)*P*-1
…	…	…	…
2*P*	2*P*+1	…	3*P*-1
P	*P*+1	…	2*P*-1
0	1	…	*P*-1

Esempio di codice

```
protein(Lin,Lout) :-
   length(Lin,N), P is N+1,
   length(Lout,N),
   connected(Lout,Steps,P),    /*Un elemento è connesso
   al successivo. Steps=lista di differenze fra le pos
   degli amminoacidi */
   alldifferent(Lout),   /*Una pos occupata da 1 solo
   amminoacido*/
   func(Lout,Lin,Lbool,P),     /*Creo una lista di var
   bool per ogni coppia di amminoacidi: se gli
   amminoacidi sono vicini, allora il bool=1*/
   sumlist(Lbool,F),     F1 #= -F,
   minimize(labeling(Steps),F1).
```

Altri predicati ...

```
% func(Lout,Lin,Lbool,P)
func([_,_],_,[0],_) :- !.
func([H,H1,H2|T],[1,Xin1,Xin2|Tin],Lbool,P)
   :- !,
% Non serve mettere nella funzione obiettivo
   due successivi. Anche il primo ed il terzo
   non possono essere vicini
   cal_fun(H,T,1,Tin,L1,P),
   func([H1,H2|T],[Xin1,Xin2|Tin],L2,P),
   append(L1,L2,Lbool).
func([_|T],[0|Tin],Lbool,P) :-
   func(T,Tin,Lbool,P).
cal_fun(_,[],_,[],[],_).
cal_fun(X,[H|T],1,[1|Tin],[Bool|R],P) :-!,
   X #= H+1#<=>B1, X #= H-1 #<=> B2,
   X #= H+P#<=>B3, X #= H-P #<=> B4,
   sumlist([B1,B2,B3,B4],Bool), Bool::0..1,
   cal_fun(X,T,1,Tin,R,P).
cal_fun(X,[_|T],1,[0|Tin],R,P) :-
   cal_fun(X,T,1,Tin,R,P).
```

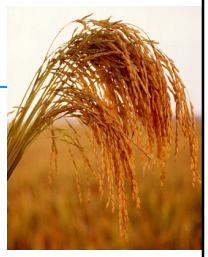
```
connected([_],[],_):- !.
connected([A,B|C],[Step|Steps],P):-
   MP is -P,
   Step :: [MP,-1,1,P],
   A #= B+Step,
   connected([B|C],Steps,P).
```


Computational Sustainability

- *New research field*
- *Many environmental problems are combinatorial in nature*
  - *Ecologic Corridors: Three national parks should be connected, buying land from owners. Which lands should be bought to minimize cost? [Gomes et al]*

www.computational-sustainability.org

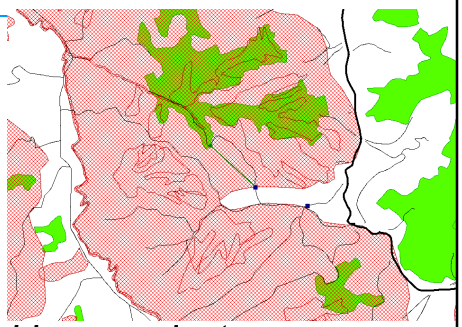
Biomass plant location



- Biomass power plants can use different types of fuel:
  - Hay
  - Wood chops, …
- Carbon neutral: biomass comes from plants that converted CO2 into O2
- But: we have to transport biomass to the plant, and transport produces CO2!
- Is it worth the while?

Biomass plant location

- Green = Forest
- Red = impossible location
- Black = roads



- Find the best location for a biomass plant, maximizing the difference between produced energy and energy used to build the plant/transportation

OTHER APPLICATIONS of CP

- Constraint Databases
- Spreadsheet
- Robotics/Control
- Diagnosis
- Test data generation
- Circuit Verification
- Natural Language
- Graphical Interfaces
- Graphical Editors
- Biology (DNA sequencing)
- Qualitative reasoning
- Temporal reasoning
- SAT
- Other LSCO problems

Timetabling

- *Si devono assegnare ad ogni corso un'aula ed un orario*
- *Ho aule di diversa capacità, con servizi (videoproiettore fisso, laboratori, ...)*
- *Ogni corso ha un docente e viene seguito da diversi gruppi di studenti.*
- *Non si devono sovrapporre i corsi di uno stesso docente, seguiti dagli stessi studenti o che si trovano nella stessa aula*
- *Per ogni corso so quante ore fa alla settimana*
- *I docenti possono esprimere preferenze sugli orari (nel limite del possibile). Alta priorità ai docenti a contratto*

Specifiche

- *Corsi*
  - *corso(sistemioperativi,stefanelli,130,[[info,2],[info,4],[ele,3],[tlc,3],[auto,4],[ele,5]],7,_,3,2,3,"Sistemi Operativi","http://www.ing.unife.it/informatica/SO-2/").*
  - *corso(strument_misure_elettr,corticelli,180,[[info,2],[auto,2],[tlc,2],[ele,2]],3,_,1,2,3,"Strumentazione e misure elettroniche",_).*
  - *corso(strument_misure_elettr_lab,corticelli,120,[[info,2],[auto,2],[tlc,2],[ele,2]],4,ele,1,4,4,"Strumentazione e misure elettroniche",_).*
  - *corso(inglese_turno1,inlingua,50,[[info,1],[auto,1]],4,_,2,1,2,"Inglese",_).*
  - *corso(inglese_turno2,inlingua,50,[[tlc,1],[ele,1]], 4,_,2,1,2,"Inglese",_).*
- *Aule*
  - *aula(1,250,n,_,s,s,_,_).*
  - *aula(lab_info,64, s,info,n,s, "http://www.ing.unife.it/sidi/cs_lab/cs_lab.htm","Laboratorio di Informatica Grande").*

Domini delle variabili Start

- *Attività fittizie*
  - *pausa pranzo*
  - *fine giornata*

		Lun	Mar	Mer	Gio	Ven
08:30	09:30	1	13	25	37	49
09:30	10:30	2	14	26	38	50
10:30	11:30	3	15	27	39	51
11:30	12:30	4	16	28	40	52
12:30	13:30	5	17	29	41	53
13:30	14:00	6	18	30	42	54
14:00	15:00	7	19	31	43	55
15:00	16:00	8	20	32	44	56
16:00	17:00	9	21	33	45	57
17:00	18:00	10	22	34	46	58
18:00	19:00	11	23	35	47	59
		12	24	36	48	60

Specifiche

	Lunedì	Martedì	Mercoledì	Giovedì	Venerdì
8.30 – 9.30	A	B	A	A	D
9.30 – 10.30	A	B	A	A	D
10.30 – 11.30	A	B	C	B	G
11.30 – 12.30	B	C	C	B	G
12.30 – 13.30	B	C	C	D	F
14 – 15	C	F	G	D	F
15 – 16	C	F	G	G	E
16 - 17	D	F	G	G	E
17 - 18	D	E	E	F	E
18 - 19	D	E	E	F	

- *Normalmente, i corsi devono stare in un "turno"*
- *In questo modo o due corsi si sovrappongono totalmente o non si sovrappongono*
- *Alcuni corsi sono in comunanza con altri CdL (Informatica a scienze, meccanica)*

Implementazione vincolo turni

- *Propia:*

```
turni_def(Turno,Ora1,
   O2,O3) infers fd.

turni_def(a,25,37,1).
turni_def(b,4,39,13).
turni_def(c,7,16,27).
turni_def(d,41,49,9).
turni_def(e,22,34,56)
turni_def(f,46,53,19)
turni_def(g,44,51,31)
```

		Lun	Mar	Mer	Gio	Ven
08:30	09:30	1	13	25	37	49
09:30	10:30	2	14	26	38	50
10:30	11:30	3	15	27	39	51
11:30	12:30	4	16	28	40	52
12:30	13:30	5	17	29	41	53
13:30	14:00	6	18	30	42	54
14:00	15:00	7	19	31	43	55
15:00	16:00	8	20	32	44	56
16:00	17:00	9	21	33	45	57
17:00	18:00	10	22	34	46	58
18:00	19:00	11	23	35	47	59
		12	24	36	48	60

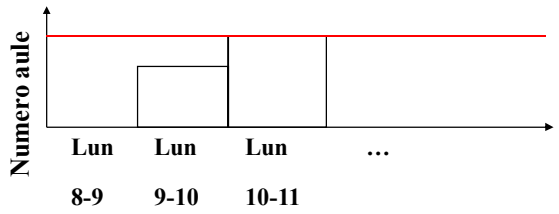
Timetabling

```
solve_ttturni(LSlot,N,LCosts):-
   crea_slot_turni(LSlot),
   imponi_vincoli(LSlot),
   break_symmetries(LSlot),
   obj_function(LSlot,N,LCosts),    N #< 10000,
   min_max(
    (search(LSlot,start of
    slot,most_constrained,indomain_random,complete,[]),
   aula_assignment(LSlot),
    save_solution(LSlot,N), print_solution(LSlot)
    )
   ,N).
```


Imposizione vincoli

```
imponi_vincoli(LSlot):-
   no_overlap_docente(LSlot),
   no_overlap_studente(LSlot),
   impose_cumulative(LSlot),
   other_constraints_courses(LSlot).
```

- *impose_cumulative impone il vincolo cumulativo in cui le aule sono risorse*



Vincoli sulle aule

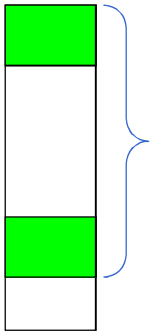
- *Per tutti i possibili sottoinsiemi di aule*
  - *Seleziona i corsi che possono stare **solo** in quel sottoinsieme di aule*
  - *Ciascun corso consuma 1 aula (1 risorsa)*
  - *Imponi il vincolo cumulativo su quelle risorse*
- *Es*
  - *Info1 □ 130 stud* *Aula 1 □ 250 posti*
  - *Analisi1 □ 160 stud* *Aula 5 □ 157 posti*
  - *Fisica 2 □ 100 stud* *Aula 7 □ 120 posti*
  - *Reti □ 100 stud*
- *Imponiamo:*
  - *Aula 1 □* `cumulative([Analisi1],…,1).`
  - *Aula 5 □* `cumulative([],…,1).`
  - *Aula 7 □* `cumulative([],…,1).`
  - *Aule 1,5 □* `cumulative([Info1,Analisi1],…,2).`
  - *Aule 1,5,7 □* `cumulative([Info1,Analisi1,Fisica2,Reti],…,3).`
  - *Aule 1,7 □* `[]` …

Euristiche

- *Ho vincolo di non sovrapposizione sulle aule, sugli studenti, sui docenti. Se decido che un gruppo di studenti stanno tutti nella stessa aula, soddisfacendo il no_overlap sull'aula, soddisfo automaticamente il no_overlap sugli studenti (può essere visto come least constraining principle)*
- *Quindi seleziono il gruppo di studenti che hanno più corsi ed assegno loro un'aula. Tolgo dal dominio degli altri corsi l'aula in questione.*
- *Alcune aule sono uguali dal punto di vista del modello. Questo introduce simmetrie: il labeling evita di assegnare l'aula simmetrica.*
- *Problema: io voglio imporre il vincolo cumulativo sui corsi assegnati ad una stessa aula. Però posso imporlo solo dopo che ho assegnato i corsi alle aule!*
- *Esistono dimostrazioni formali che ripartire daccapo è utile se si usa una selezione casuale (min_max va d'accordo con indomain_random)*

Funzione obiettivo

- *Minimizzare il numero di "buchi" di orario per gli studenti*
- *Allo stesso tempo, bisogna evitare che gli studenti abbiano giornate troppo "piene"*
- *Un giorno libero deve essere considerato positivo!*
- *Contano di più i gruppi di studenti che sono più numerosi*
- *Gli studenti che hanno molte scelte non vengono considerati*



Differenza fra la fine dell'ultima ora e l'inizio della prima

- *Aggiungo un valore negativo per ogni giorno libero*
- *Minimizzo la somma pesata (considerando il numero di studenti)*

Risultati

- *Minimizzazione delle sovrapposizioni:*
  - *AA 2003/04: 9 sovrapposizioni fra corsi obbligatori e facoltativi (dato calcolato su 2 trimestri)*
  - *AA 2004/05: 0 sovrapposizioni (su 3 trim)*
  - *AA 2003/04: 20 sovrapposizioni per recupero (su 2 trim)*
  - *AA 2004/05: 0 sovrapposizioni per recupero (su 3 trim)*

OVERVIEW

- Constraint Satisfaction (Optimization) Problems
- Constraint (Logic) Programming
  - *language and tools*
- AI Applications: modelling and solving
  - *Scheduling - Timetabling - Resource Allocation*
  - *Routing*
  - *Packing - Cutting*
  - *Graphics - Vision*
  - *Planning*
- Advantages and Limitations of CP: extensions

ADVANTAGES OF CP

- Easy problem modelling
- Constraints provide a natural way of implementing propagation rules
- Flexible treatment of variants of original problems:
  - easy introduction of new constraints
  - transparent interaction with other constraints
- Easy control of the search

LIMITATIONS of PURE CP

- Optimization side not very effective
- Over-Constrained problems:
  - no effective way of relaxing constraints
  - hard/soft constraints
- Dynamic Changes:
  - addition/deletion of variables
  - addition/deletion of domain values
  - addition/deletion of constraints

CP EXTENSION FOR OPTIMIZATION

- Integration of OR techniques in CP tools:
  - MP based solvers: 2LP ***[McAloon, Tretkoof PPCP94]***, OPL ***[Van Hentenryck, 99]***, Planner ***[ILOG Planner Manual]***
    - Integration of CPLEX and XPRESS in FD solvers
  - Integration of specialized algorithms for:
    - computing bounds
    - using reduced costs

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

CP EXTENSION FOR OVER-CONSTRAINED PROBLEMS

- HCSP:
  - Implementation of CP solvers exploiting Hierarchical CSP framework
  - Meta programming

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

CP EXTENSION FOR DYNAMIC CHANGES

- DCSP
- ATMS-based solvers
- Interactive Constraint Satisfaction

} Complex data-structures

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

TO KNOW MORE…..

- *Conferences:*
  - *International Conference on Principles and Practice of Constraint Programming (CP)*
  - *Logic programming conferences (ICLP - ILPS - JICSLP)*
  - *AI Conferences (ECAI - AAAI - IJCAI)*
  - *Operations research conferences (INFORMS - IFORS)*
  - *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR)*

- *Books:*
  - *Krzysztof R. Apt and Mark Wallace,* Constraint Logic Programming using ECL^{i}PS^e, *Cambridge*
  - *K. Marriott and P. Stuckey,* Programming with constraints: An Introduction, *MIT Press*
  - *Rina Dechter,* Constraint Processing, *Morgan Kaufmann*

TO KNOW MORE…..

- *Journals:*
  - *Constraint - An International Journal*
  - *AI - LP - OR Journals*
- *Industrial Applications:*
  - *COSYTEC, ILOG, CrosscoreOptimization, SIEMENS, BULL*
- *News group:* **`comp.constraints`**
- *Mailing lists:* **`CPWORLD@gmu.edu`**
- *Constraint Archive:* **`http://4c.ucc.ie/web/archive/`**