UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Ottimizzazione

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Ottimizzazione

- Un CSP è un problema in cui le soluzioni sono tutte equivalenti
- In molti problemi reali non è così: alcune soluzioni sono preferibili ad altre
- Spesso la discriminazione viene fatta tramite una funzione obiettivo da minimizzare o massimizzare
- Il valore della funzione obiettivo viene collegata alle variabili decisionali tramite vincoli
- Passare da minimizzazione a massimizzazione è banale: basta cambiare il segno. Spesso ci focalizzeremo su funzioni da minimizzare, senza perdere generalità

Esempio

- *Fra tutte le soluzioni del seguente CSP:*

```
X::1..10, Y::1..10, X*Y #> 5
```

- *Se ne calcoli una che minimizza* `X+Y`

findall/3

- Il meta-predicato `findall` raccoglie le soluzioni di un goal

```
findall(Termine, Goal, Lista)
```

- Trova tutte le istanze di `Termine` che soddisfano il `Goal` e le fornisce nella `Lista`

findall/3

- *Es:* `findall(X,p(X,Y),L).`

```
L = [1, 1, 2, 2]
Yes
```

```
p(1,2).
p(1,3).
p(2,3).
p(2,2).
```

- `findall(p(X,Y),p(X,Y),L).`

```
L = [p(1, 2),p(1, 3),p(2, 3),p(2,2)]
Yes
```

```
findall(q(X,Y),
          (p(X,Y), X<Y),
        L).
L = [q(1, 2), q(1, 3), q(2, 3)]
Yes
```


Modifica di un programma a run-time

- Se si desidera salvare dei dati che sopravvivono al backtracking, in Prolog si possono usare `assert` e `retract`

`assert(Clausola)`

- Aggiunge al programma la Clausola
- Es: `assert((p(X):- q(X)))` aggiunge al programma la clausola
  `p(X):-q(X).`
- Es: `assert(f(2))` aggiunge al programma il fatto
  `f(2).`
- Le clausole aggiunte sopravvivono al backtracking
- Per eliminare una clausola: `retract`.
- Es: `retract(f(2))`.

Trovare l'ottimo

- Per trovare la soluzione ottima di un CSP, si potrebbe
  - utilizzare la `findall` per trovare tutte le soluzioni
  - A posteriori, selezionare quella che ha il valore minimo di funzione obiettivo
- `csp(L):- imponi_vincoli(L), labeling(L).`
- ```
  soluzione_ottima(S):-
     findall(X,csp(X),L),
        seleziona_migliore(L,S).
  ```
- Richiede di calcolare tutte le soluzioni del CSP → molto inefficiente

findall/3

- Il meta-predicato `findall` raccoglie le soluzioni di un goal

`findall(Termine, Goal, Lista)`

- Invoca il Goal, quando si arriva ad una soluzione la salva, poi impone un backtracking al goal
- Può essere implementato così:
  - ```
    findall(Termine,Goal,Lista):- call(Goal),
    salva(Termine), fail.
    findall(_,_,Lista):- dati_salvati(Lista).
    ```
  - ```
    salva(Term):- retract(salvato(OldList)),
    append(OldList,[Term],NewList),
    assert(salvato(NewList)).
    ```
  - `salvato([]).`
  - `dati_salvati(L):- retract(salvato(L)).`
    ```

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Ottimizzazione

min-max

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

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
- Sarebbe più efficiente utilizzare i vincoli, per eliminare a priori soluzioni che non sono promettenti. Consideriamo un problema di minimizzazione, in cui una variabile C rappresenta il costo, ed è legata alle variabili decisionali tramite vincoli (Es: C #= A+B*X)

1) Risolvi il CSP: trova una soluzione, sia C* il suo costo
2) Aggiungi al CSP un nuovo vincolo C < C*
3) Se il nuovo CSP ha soluzione
   - Salta al passo 1
   - Altrimenti hai trovato l'ottimo

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*
- *Uso tipico:*
- `cop(L):-`
  `imponi_vincoli(L),`
  `C #=` *espressione_funzione_obiettivo*`(L),`
  `minimize(labeling(L), C).`

Es: `C #= X+Y-Z`

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*
- *Uso tipico:*
- ```
  cop(L):-
      imponi_vincoli(L),
      funzione_obiettivo(L,C),
      minimize(labeling(L), C).
  ```
  ```

min_max

- L'algoritmo usato da `min_max` prevede di far ripartire la ricerca da capo dopo aver trovato una soluzione
- Schema: salviamo in un predicato `best(C)` il miglior costo via via trovato

```
min_max(G,C):-
    assert(best(+∞)),
    min_max2(G,C).
min_max(G,C):- best(C).
min_max2(G,C):-
    repeat,
    best(B),
    C #< B,
    (call(G)
      ->  retract(best(B)), assert(best(C)), fail
      ;   !, fail).
```

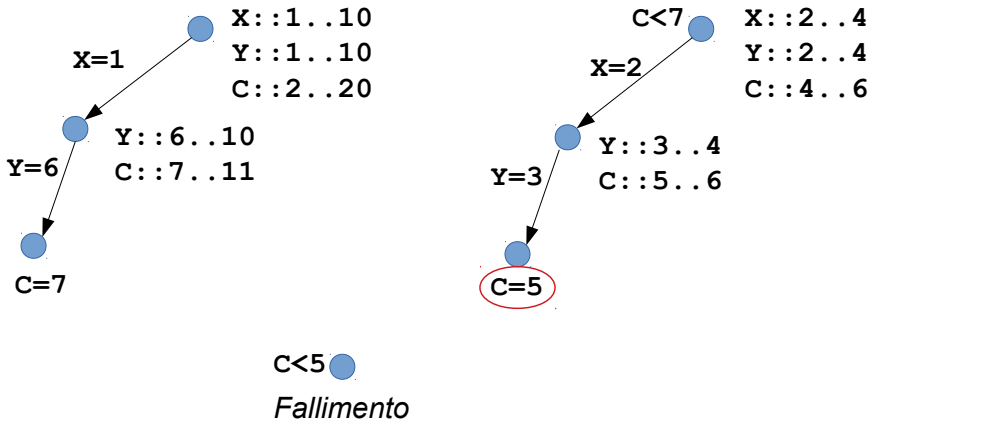
`repeat.`
`repeat:- repeat.`
Genera infiniti punti di scelta

Torna al punto di scelta della `repeat`

Dentro la `->` c'è un cut, quindi il punto di scelta della `call(G)` viene tagliato

Ottimizzazione: Esempio min_max

`[X,Y]::0..10, X*Y#>5, X+Y#=C, min_max(labeling([A,B]),C).`



Ottimizzazione: Esempio (min_max)

`[X,Y]::0..10, X*Y#>5, X+Y#=C, min_max(labeling([X,Y]),C).`

1. *Impone i vincoli e propaga:*
   X{[1..10]}, Y{[1..10]}, C{[2..20]}
2. *esegue labeling e trova una soluzione:*
   X=1, Y=6, C=7
3. *backtracking: torna alla situazione al passo 1*
4. *impone nuovo vincolo C #< 7 e propaga:*
   X{[2..5]}, Y{[2..5]}, C{[4 .. 6]}
5. *esegue labeling e trova una soluzione:*
   X=2, Y=3, C=5
6. *backtracking: torna alla situazione al passo 1*
7. *impone nuovo vincolo C #< 5 e propaga:*
   fallimento

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

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Ottimizzazione

minimize

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Min-max

- *La strategia min-max fa ripartire da capo la search*
- *Quindi il nuovo vincolo imposto C<C* fa propagazione subito*
- *D'altro canto, far ripartire da zero la search rischia di far riesplorare più volte le stesse parti dell'albero di ricerca*

minimize

- L'algoritmo usato da **`minimize`** prevede di far continuare la ricerca aggiungendo un nuovo vincolo
- Schema: salviamo in un predicato **`best(C)`** il miglior costo via via trovato

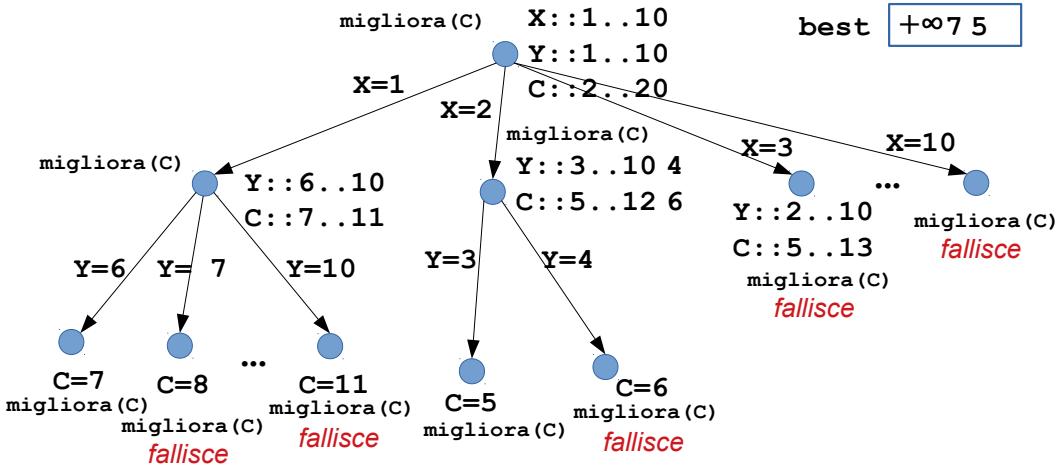
```
minimize(G,C):-
    assert(best(+∞)),
    minimize2(G,C).
minimize(G,C):- best(C).
minimize2(G,C):-
    migliora(C),
    call(G),
    retract(best(_)), assert(best(C)), fail.
```

```
migliora(C):-
   best(B), B1 is B-1,
   dvar_remove_greater(C,B1),
   suspend(migliora(C),3,
           [C->min,C->max]
          ).
```

Torna al punto di scelta della `call(G)`

Ottimizzazione: Esempio minimize

```
[X,Y]::0..10, X*Y#>5, X+Y#=C, minimize(labeling([X,Y]),C).
```



UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Ottimizzazione

minimize vs min-max

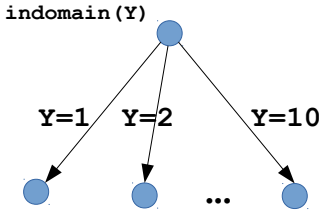
Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

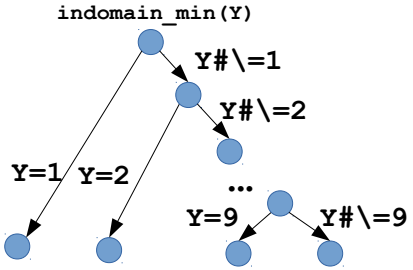
Prof. MARCO GAVANELLI

Minimize vs min-max

- Nell'esempio che abbiamo visto, **`minimize`** sembra meno efficiente di **`min-max`**, non è detto che sia sempre così
- Dipende anche da come viene effettuata la search:
  - se invece di **`indomain`** si usa un predicato che elimina il valore dal dominio prima di provare il valore successivo, il vincolo **`migliora/1`** viene attivato più spesso

```
indomain_min(Y):-
    dvar_domain(Y,D),
    dom_range(D,Min,_),
    (   Y=Min
      ; y #> Min
        indomain_min(Y)
    ).
```





Minimize vs min-max

- Molto dipende da quanta propagazione riesce a fare il vincolo C<C* verso le variabili decisionali
- Es: **`C #= X+Y+Z`** **`[X,Y,Z]:: -10..10`**
- Imponendo **`C #< 5`**
- → nessuna propagazione su **`X, Y, Z`**
- Es: **`C`** è il massimo fra **`X, Y`**, e **`Z`**
  `maxlist([X,Y,Z],C)`
- Imponendo **`C #< 5`**
- → I valori da 5 in su vengono eliminati dai domini di tutte le variabili **`X, Y, Z`**!
- Se il vincolo C<C* riesce a fare molto pruning, in genere conviene **`min-max`**,
- se riesce a fare poco pruning, conviene **`minimize`**.

Minimize vs min-max

- *In ogni caso, conviene provare sperimentalmente quale funziona meglio in un certo problema*

Minimize e min-max

- In ogni caso, eseguendo

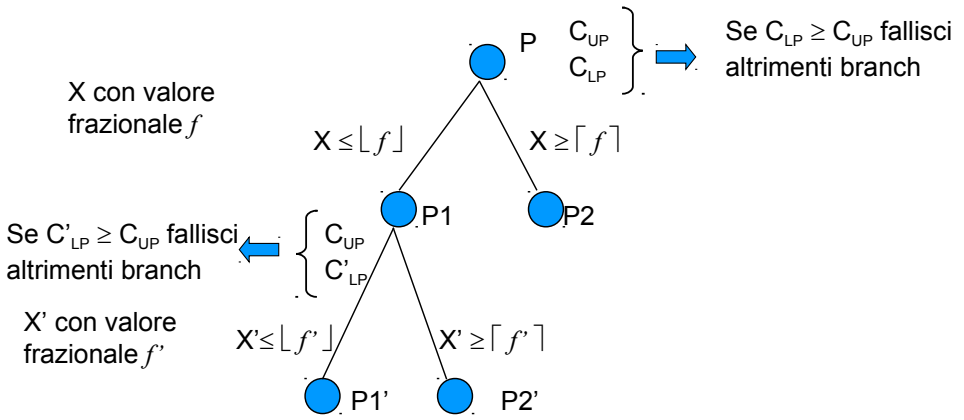
  `minimize(G,C)` oppure **`min_max(G,C)`**

  ci si aspetta che alla fine dell'esecuzione del goal G, C sia sempre ground.
- Questo risulta automaticamente vero se **`C`** dipende funzionalmente (è una funzione) dalle variabili che vengono istanziate in **`G`**
- Altrimenti, tipicamente si inserisce **`C`** all'interno della liste istanziate da **`G`**
- ```
  cop(L):-
      imponi_vincoli(L),
      vincoli_obiettivo(L,C),
      append(L,[C],T),
      minimize(labeling(T),C).
  ```
  ```

BRANCH AND BOUND

- Si usa una variabile *C* che rappresenta la funzione obiettivo e la si lega alle variabili decisionali, esempio somma di costi
- Se il legame tra *C* 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

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Scheduling

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

SCHEDULING - TIMETABLING

- 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

Set-up times

- *In alcuni problemi, esistono attività di tipo diverso; se due attività di tipo diverso vengono eseguite in sequenza sulla stessa macchina, è necessario un tempo di set-up*

Tipi di risorse

- *A volte uno stesso task può essere eseguito su macchine diverse.*
- *A macchine diverse possono corrispondere*
  - *Diversi tempi di elaborazione*
  - *Diversi costi di utilizzo*

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.

Exercise: obtain a list containing the same data using **`findall`**:

```
Data = [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)]
```

- Minimising the maximum ending time `End` is required.

SCHEDULING: Simple Example
COP model

- Decision variables:
  - A *Start* time for each task
  - A variable *End* represents the end of the schedule
- Constraints:
  - If $task_j$ must precede $task_i$: $Start_j + Dur_j \leq Start_i$
  - All tasks terminate in *End*: $\forall i$, $Start_i + Dur_i \leq End$
  - For each machine, a **`cumulative`** constraint with all the tasks running on that machine
- Objective: Minimising the maximum ending time *End*

SCHEDULING: Simple Example
CLP(FD) implementation

```
schedule(Data, End, TaskList):-
    makeTaskVariables(Data,End,TaskList),
    precedence(TaskList),
    machines(TaskList),
    extract_start(TaskList,LStart),
```

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

```
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)]
```

```
[task(j1,3,[],m1,Start1),
 task(j2,8,[],m1,Start2),
 task(j3,8,[j4,j5],m1,Start3),
 task(j4,6,[],m2,Start4),
 task(j5,3,[j1],m2,Start5),
 task(j6,4,[j1],m2,Start6)]
```

SCHEDULING: Simple Example

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

```
precedence(TaskList):- precedence2(TaskList,TaskList).
precedence2([],_).
precedence2([task(_,_,Prec,_,Start)|T],TaskList):-
  impose_after(Prec,Start,TaskList),
  precedence2(T,TaskList).
impose_after([],_,_).
impose_after([PrecJ|LPrec],StartI,TaskList):- !,
    get_task(PrecJ,TaskList,task(PrecJ,DJ,_,_,StartJ)),
    StartI #>= StartJ+DJ,
    impose_after(LPrec,StartI,TaskList).
get_task(J,[task(J,DJ,PJ,MJ,StartJ)|_],task(J,DJ,PJ,MJ,StartJ)):-!.
get_task(J,[_|TaskList],TaskJ):-
    get_task(J,TaskList,TaskJ).
```

SCHEDULING: Simple Example

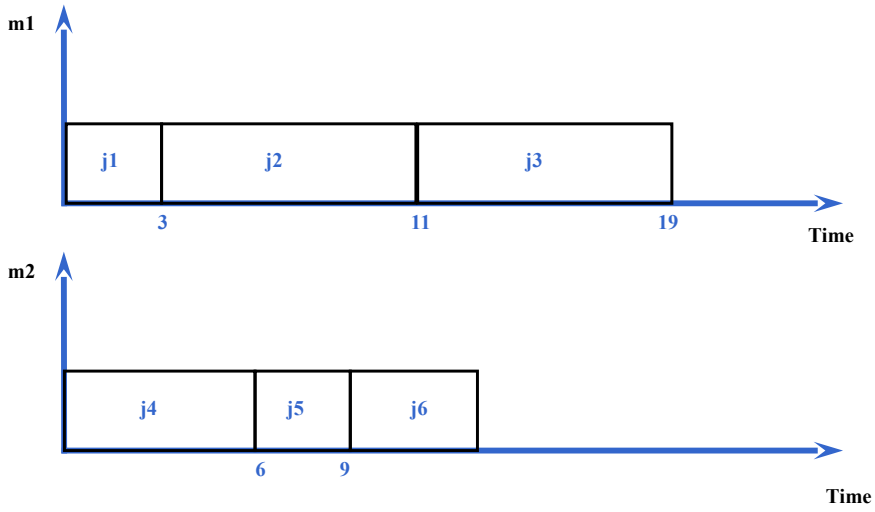
```
machines(TaskList):-
    findall(M,task(_,_,_,M),LM),
    remove_duplicates(LM,ListMachines),
    impose_cumulative(ListMachines,TaskList).
impose_cumulative([],_).
impose_cumulative([M|LM],TaskList):-
    select_same_machine(M,TaskList,LStart,LDur,LRes),
    cumulative(LStart,LDur,LRes,1),
    impose_cumulative(LM,TaskList).
```

Fissa ad 1 tutti gli elementi di **LRes**

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



UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

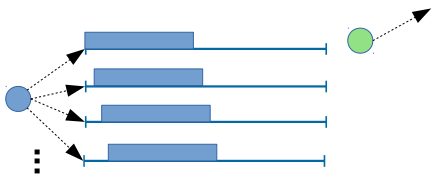
Scheduling – strategie di branching

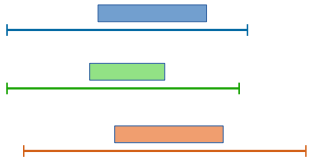
Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

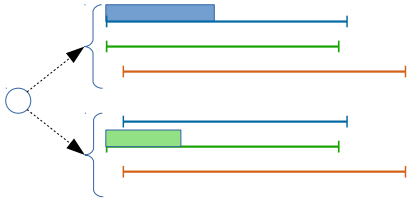
Search – minimizing makespan

- *Usual labeling:*





- *label_earliest*



SCHEDULING: Optimality

- **`minimize`**: finds the optimal solution (simple Branch & Bound)
- Minimization of the makespan: a heuristic that 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):-
    deletemin(First,TaskList,Others),
    mindomain(First,MinStart),
    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).
```


Altre strategie di branching

- *Ordering between activities*
  - Date due attività *i* e *j* che non devono sovrapporsi

$$S_i + D_i =< S_j \vee S_j + D_j =< S_i$$

- *Dichotomic search*
  - Ricerca binaria sulla funzione obiettivo

Scheduling Scientific Experiments on the Rosetta/Philae Mission

Gilles Simonin, Christian Artigues, Emmanuel Hebrard, and Pierre Lopez

CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Univ de Toulouse, LAAS, F-31400 Toulouse, France
{gsimonin,artigues,hebrard,lopez}@laas.fr

Abstract. The Rosetta/Philae mission was launched in 2004 by the European Space Agency (ESA). It is scheduled to reach the comet 67P/Churyumov-Gerasimenko in 2014 after traveling more than six billion kilometers. The Philae module will then be separated from the orbiter (Rosetta) to attempt the first ever landing on the surface of a comet. If it succeeds, it will engage a sequence of scientific exploratory experiments on the comet.

In this paper we describe a constraint programming model for scheduling the different experiments of the mission. A feasible plan must satisfy a number of constraints induced by energetic resources, precedence relations on activities, or incompatibility between instruments. Moreover, a very important aspect is related to the transfer (to the orbiter then to the Earth) of all the data produced by the instruments. The capacity of inboard memories and the limitation of transfers within visibility windows between lander and orbiter, make the transfer policy implemented on the lander's CPU prone to data loss. We introduce a global constraint to handle data transfers. The goal of this constraint is to ensure that data-producing activities are scheduled in such a way that no data is lost.

Thanks to this constraint and to the filtering rules we propose, mission control is now able to compute feasible plans in a few seconds for scenarios where minutes were previously often required. Moreover, in many cases, data transfers are now much more accurately simulated, thus increasing the reliability of the plans.

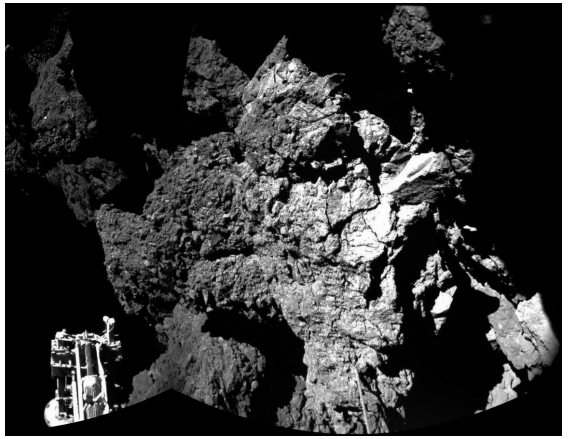
1 Introduction

The international Rosetta/Philae project is an European consortium mission approved in 1993, and is under the leadership of the German Aerospace Research Institute (DLR) and ESA[1]. The spacecraft was launched in 2004 by Ariane 5, and is set to travel more than six billion kilometers to finally reach and land on the comet 67P/Churyumov-Gerasimenko in 2014 in order to analyze the comet structure. It will follow a complex trajectory which includes four gravity assist maneuvers (3 x Earth, 1 x Mars) before finally reaching the comet and enter its orbit. Then, the lander Philae will be deployed and will land on the surface of the comet. Philae features ten instruments, each developed by a European laboratory, to accomplish a given scientific experiment when approaching, or once landed on the comet. For instance, *CIVA* and *ROLIS* are two imaging instruments, used to take panoramic pictures of the comet and microscopic images.

[1] European Space Agency.

M. Milano (Ed.): CP 2012, LNCS 7514, pp. 23-37, 2012.

Rosetta/Philae



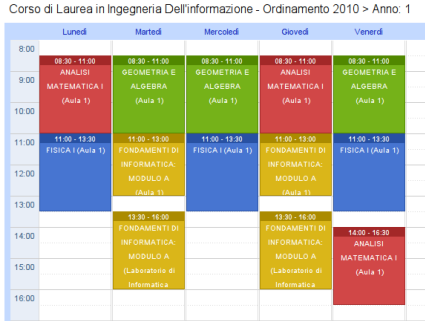
Credits: ESA (European Space Agency)

/home/marco/marco/dida/ia/AppliIA/Lucidi/2014-15/Philae's mission at comet 67P-YoqokoZkviM.mp4

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

- *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

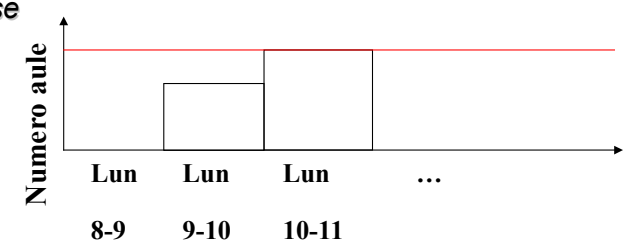
Timetabling

```
solve_ttturni(LSlot,N,LCosts):-
   crea_slot_turni(LSlot),
   imponi_vincoli(LSlot),
   break_symmetries(LSlot),
   obj_function(LSlot,N,LCosts),    N #< 10000,
   min_max(
    (    time_assignment(LSlot),
         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*

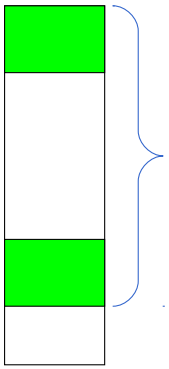


Vincoli sulle aule

- *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 =>* `[]` …

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)*