15 giugno 2017 - laboratorio

Constraint Programming

Prof. Marco Gavanelli

15 giugno 2017

Descrizione problema

Un robot deve verniciare le carrozzerie delle automobili.

Le carrozzerie arrivano in una sequenza; per ogni carrozzeria viene indicato il colore di cui deve essere verniciata. Ad esempio, una sequenza potrebbe essere

rosso, *giallo*, *rosso*, *verde*, *giallo*.

All'interno della sequenza, è possibile fare dei piccoli spostamenti, fino ad un valore costante *MaxD*: se un'auto arriva in posizione *n*, allora la si può spostare nelle posizioni da *n*–*MaxD* a *n*+*MaxD*.

Se dopo aver verniciato una carrozzeria, il robot deve cambiare colore, allora deve effetturare una costosa operazione di pulizia degli ugelli.

Si trovi la sequenza ottima che soddisfa tutti i vincoli e che minimizza il numero di cambi di colore.

Nell'esempio riportato sopra, con *MaxD*=1 si può tenere la prima auto in posizione 1, la seconda spostata in posizione 3, la terza viene spostata in posizione 2 (in questo modo le due rosse sono vicine), la verde va in ultima posizione mentre la quinta auto viene anticipata alla posizione 4 (in questo modo le due gialle sono vicine). Il costo di questa soluzione è quindi 2 (un cambio di colore dal rosso al giallo ed uno dal giallo al verde).

CLP (10 punti)

Si risolva il problema usando ECL*i*PS*e*, scrivendo un predicato che prende almeno due parametri:

- *Sequenza*: è la sequenza di ingresso; ad es *Sequenza*=[*rosso*, *giallo*, *rosso*, *verde*, *giallo*]
- *MaxD*: (ad es. *MaxD* =2)

e fornisce il risultato nel formato che si ritiene più opportuno.

ASP (10 punti)

Si risolva il problema in Aswer Set Programming. La sequenza di auto in ingresso è data da un predicato `car(N,Col)` , che indica che l' `N` -esima auto in ingresso va colorata del colore `Col` . La distanza massima *MaxD* è riportata in un predicato `maxd(MaxD)` .

Si utilizzi l'istanza riportata nel file `istanza.pl` .

MiniZinc (3 punti)

Si risolva, usando MiniZinc, il corrispondente problema di soddisfacibilità, ovvero il problema senza funzione obiettivo. È quindi una soluzione del problema qualunque sequenza in uscita in cui un'auto è spostata al massimo di *MaxD* posizioni.

I dati di ingresso vengono forniti in un file `datiSequencing.mzn` , che va copiato nel file .mzn da consegnare.

Tale file contiene la definizione delle seguenti costanti:

- `NumCars` : intero, rappresenta il numero di auto nella sequenza in ingresso
- `car` : array `[1..NumCars]` di interi. L'elemento `car[i]` rappresenta il colore dell' `i` -esima auto nella sequenza. I colori sono rappresentati come interi.
- `MaxD` : intero, è (come prima) la massima distanza di cui può essere spostata un'automobile rispetto alla posizione in cui era nella sequenza.

Facoltativo: Funzione Obiettivo MiniZinc (1 punto)

Anche in MiniZinc, si trovi l'ottimo usando la funzione obiettivo definita in precedenza.

Soluzione CLP

```
:- lib(fd).
:- lib(fd_global).

% Data la sequenza di ingresso Lin, produce una lista L
% che contiene le posizioni finali delle auto.
% Es sequencing([rosso, giallo, rosso, verde,
giallo],L,2) da`
% L =          [  1  ,   3   ,   2  ,   5  ,   4   ]
% che significa che la prima auto rossa rimane in
posizione 1
% la prima gialla va in posizione 3
% la seconda rossa in posizione 2, ecc.
sequencing(Lin,L,MaxD):-
    length(Lin,N),
    length(L,N),
    L :: 1..N,
    vincolo_dist(L,1,MaxD),
    fd_global:alldifferent(L),
    objective(L,Lin,Obj),
    minimize(labeling(L),Obj).

% Per tutte le coppie (non ordinate) di auto in ingresso
objective([_],_,0):-!.
objective([P|L],[Col|Lcol],Obj):-
    objective_loop(P,L,Col,Lcol,ObjLoop),
    objective(L,Lcol,Obj1),
    Obj #= ObjLoop+Obj1.

objective_loop(_,[],_,[],0).
objective_loop(P,[_|L],Col,[Col|Lcol],Obj):- !,
% Se nella coppia considerata ho due auto con lo stesso
colore,
% non c'e` contributo nella funzione obiettivo
    objective_loop(P,L,Col,Lcol,Obj).
objective_loop(P,[P1|L],Col,[Col1|Lcol],Obj):- !,
% Se nella coppia considerata ho due auto di colore
diverso,
% ho un contributo qualora le due posizioni siano vicine
    Col \= Col1,
    P-P1 #= 1 #<=> Bool1,
    P1-P #= 1 #<=> Bool2,
    Obj #= ObjLoop+Bool1+Bool2,
```

```
    objective_loop(P,L,Col,Lcol,ObjLoop).

% Ogni auto non puo` essere inserita in una posizione
piu`
% lontana di MaxD dalla posizione che aveva in ingresso
vincolo_dist([],_,_).
vincolo_dist([H|T],N,MaxD):-
    H-N #=< MaxD,
    N-H #=< MaxD,
    N1 is N+1,
    vincolo_dist(T,N1,MaxD).
```

Soluzione ASP

```
position(P):- car(P,_).

1 {pos(NewPos,OldPos,Col):position(NewPos)} 1 :-
car(OldPos,Col).

:- pos(NewPos,OldPos,_), |NewPos-OldPos| > MaxD,
maxd(MaxD).

:- pos(NewPos,OldPos1,_), pos(NewPos,OldPos2,_),
OldPos1 != OldPos2.

switch(P):- pos(P,_,Col1), pos(P+1,_,Col2), Col1 != Col2.

#minimize { 1,P:switch(P) }.
```

Soluzione MiniZinc

```
include "globals.mzn";

int: NumCars;
int: r;
int: b;
int: g;
array[1..NumCars] of int: car;
int: MaxD=2;
```

```
array[1..NumCars] of var 1..NumCars: p;
array[1..NumCars] of var 0..1: swap;

constraint forall(i in 1..NumCars)
  (p[i]-i <= MaxD);
constraint forall(i in 1..NumCars)
  (i-p[i] <= MaxD);

constraint alldifferent(p);

solve minimize
sum([p[i]=p[j]+1 | i in 1..NumCars, j in 1..NumCars where
i!=j
     /\ car[i]!=car[j]]);
```